Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Softwarová komponenta pro 3D vizualizaci simulace silniční křižovatky

Plzeň 2013 Tomáš Sojka

Prohlášení

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

V Plzni dne 21. června 2013

Tomáš Sojka

Poděkování

Děkuji svému vedoucímu bakalářské práce Ing. Tomáši Potužákovi, Ph.D. za čas a cenné rady, které mi pomohly při zpracování bakalářské práce. Dále bych chtěl poděkovat své rodině za podporu během studia.

Abstract

The purpose of this bachelor thesis is to create a software component for 3D visualization of a Traffic Crossroad simulation. The component is part of application *CrossRoadControl* which is developed on the Department of Computer Science and Engineering University of West Bohemia in Pilsen. Application already contained the visualization in 2D view which is sufficient for operating a simulation and measuring. But it is not so suitable by visual side compared to 3D view that gives wide range of view on operation of crossroad and makes easier better imagination in real world. Considering that the whole component application is created in a programming language Java, the work deals with possibilities 3D displaying in Java. It can serve as inspiration for generation more application in Java making use 3D display.

Náplní této bakalářské práce je návrh a implementace softwarové komponenty, která poskytuje 3D vizualizaci simulace silniční křižovatky. Komponenta je součástí aplikace *CrossRoadControl*, která je vyvíjena na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni. Aplikace již obsahovala vizualizaci ve 2D pohledu, který je postačující pro ovládání simulace a měření. Ale už není tak vhodná po vizuální stránce oproti 3D zobrazení, které dává široké možnosti pohledu na provoz křižovatky a usnadňuje lepší představu v reálném světě.Vzhledem k tomu, že celá komponentová aplikace je vytvořená v jazyce Java, práce se zabývá možnostmi 3D zobrazování v Javě. Může tak sloužit jako inspirace pro vytváření dalších aplikací v Javě, využívajících 3D zobrazení.

Obsah

1 Úvod **1**

2 Komponentové programování **2**
2.1 Komponenta 2
2.2 Komponentové modely 2
2.2.1 OSGi 3
2.2.2 Spring 4
2.2.3 SpringDM 5

3 Java 3D **6**
3.1 Java 3D API 6
3.2 Graf scény 7
3.2.1 Hlavní třídy scény grafu 7
3.3 Postup pro psaní Java 3D programů 9
3.4 Vytváření vizuálních objektů 10
3.4.1 Geometrická primitiva 10
3.4.2 Třída Shape3D 11
3.4.3 Import vizuálního objektu ze souboru 12

4 Analýza **14**
4.1 Seznámení s komponentou TrafficCrossroad 14
4.1.1 Rozhraní komponenty 14
4.1.2 2D GUI 15
4.2 Specifikace požadavků 15
4.3 Případy užití 17
4.3.1 Diagram případů užití 17
4.3.2 Popisy případů užití 17
4.4 Návrh 3D zobrazení 18
4.4.1 Popis grafu scény 18
4.4.2 Vytvoření křižovatky 18

4.4.3 Vytvoření složitých objektů v křižovatce 19
4.4.4 Ovládání semaforů 21
4.4.5 Ovládání vozidel 21

5 Implementace **22**
5.1 Vývojové prostředí 22
5.2 Struktura aplikace CrossRoadControl 22
5.3 Struktura navrhované komponenty 23
5.3.1 Balík g3d 24
5.3.2 Balík gui 26
5.3.3 Balík utils 28
5.4 Vytvoření 3D modelů 29

6 Otestování a demonstrace komponenty **30**
6.1 Vizuální testy 31
6.1.1 Umístění vozidel 31
6.1.2 Nastavení semaforů 32
6.1.3 Umístění přechodů a značení pruhů 33
6.1.4 Korektní zobrazení 3D modelů 33
6.1.5 Souhrn výsledků 34

7 Závěr **35**

A Uživatelská dokumentace **38**
A.1 Požadavky pro spuštění 38
A.2 Instalace knihoven Java3D 38
A.3 Instalace workspace 38
A.4 Spuštění aplikace 39
A.5 Ovládání aplikace 39

B Import 3D modelů vozidel **41**

C Katalog 3D modelů vozidel **42**

1 Úvod

Cílem předkládané bakalářské práce je návrh a implementace softwarové komponenty, která bude poskytovat 3D vizualizaci simulace silniční křižovatky. Simulaci provádí komponentová aplikace *CrossRoadControl*. Tato aplikace je vyvíjena na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni a vytvořená komponenta pro 3D vizualizaci bude její součástí.

Vzhledem k tomu, že celá komponentová aplikace je vytvořená v jazyce Java, práce se zabývá možnostmi 3D zobrazování v Javě. Může tak sloužit jako inspirace pro vytváření dalších aplikací v Javě, využívajících 3D zobrazení.

Text je členěn do pěti hlavních kapitol. V první kapitole se dozvíte základní informace o komponentových technologiích a komponentě samotné. Dále zde budou uvedeny definice komponentových modelů a popis komponentových frameworků, které využijeme při vývoji navrhované komponenty.

Další kapitola se zabývá možnostmi vykreslování 3D objektů v Javě. Získáte zde informace o postupech psaní Java 3D programů a budování scény. Dále se seznámíte s možnostmi vytváření vizuálních 3D objektů.

V následující kapitole se provede analýza návrhu komponenty. Dočtete se zde jak vypadá dodaná komponenta, jaké budou požadavky na komponentu a jak bude vypadat scéna samotné vizualizace křižovatky. Také zde budou popsány způsoby ovládání semaforů a vozidel při běhu simulace.

Předposlední kapitola se zabývá samotnou implementací komponenty a popisem aplikace. Jsou zde popsány jednotlivé třídy komponenty a vytváření modelů pro 3D vizualizaci.

V poslední kapitole bude popsáno testování vizualizace na dvou křižovatkách. Uvidíte zde screenshoty porovnávající stávající 2D vizualizaci s 3D vizualizací.

2 Komponentové programování

Komponentově orientované programování se v posledním desetiletí, díky svým vlastnostem, stále více rozvíjí. Předností tohoto stylu programování je rozdělení softwaru na více menších funkčních softwarových komponent. Tyto komponenty jsou poté volány přes rozhraní a zároveň jejich implementace zůstane skrytá. Takový způsob přístupu přináší tyto výhody[1]:

- Urychlení vývoje aplikací.
- Znovupoužití již existujících a otestovaných komponent.
- Předvídatelnost chování komponent.

Ale i nevýhody[1]:

- Velká režie při přechodu na způsob komponentového programování.
- Větší počet frameworků, který brzdí rozvoj jednoho společného.
- Nutnost sledovat vývoj komponent třetích stran, které se rychle vyvíjejí.

2.1 Komponenta

Každá komponenta by měla být navržena pro co největší znovupoužití a využívání třetí stranou. Návrh komponenty by měl být založen na principu black-boxového modelu[1]. To znamená, že by neměla být implementace komponenty z vnějšku vidět a pro komunikaci mezi komponentami bude sloužit rozhraní[2].

2.2 Komponentové modely

Komponentový model určuje, jak budou jednotlivé komponenty vypadat, spolupracovat mezi sebou a jak se budou chovat. Modely také určují, zda jsou

[1] černá skříňka

složeny z jednoho nebo více typů komponent. Komponentový framework je pak konkrétní implementace jednoho komponentového modelu[3].

V současné době existuje mnoho komponentových modelů a ještě více komponentových frameworků, neboť od jednoho komponentového modelu může existovat více implementací, tedy komponentových frameworků. Některé z nich se používají pouze v oblasti výzkumu, jiné jsou využívány i v průmyslu (např. OSGi[2], viz dále)[3].

V následujících kapitolách budou popsány modely *OSGi*, *Spring* a *SpringDM*, které je důležité znát pro implementaci navrhované komponenty.

2.2.1 OSGi

OSGi framework implementuje dynamický komponentový model a nabízí služby programovacího jazyka Java. Poskytuje prostředí, ve kterém lze samotné komponenty vzdáleně instalovat, spouštět, aktualizovat a odinstalovávat bez nutnosti restartu. OSGi je průmyslový standard pro vývoj založený na komponentách. Využívá se v mnoha oblastech, jako jsou mobilní telefony, automobily a jiné [4].

Komponenty v OSGi modelu/frameworku se nazývají *bundle*. Bundle jsou klasické `.jar` soubory, rozšířené o soubor *manifest*, který popisuje vztahy mezi jednotlivými bundly[4].

Komunikace mezi bundly probíhá díky službám, které umožňují volat metody jiného bundlu. Bundle může poskytovat služby prostřednictvím svého rozhraní a zároveň požadovat v OSGi frameworku služby, které jsou poskytovány dalšími instancovanými bundly. Služby jsou jednou z nejdůležitějších částí OSGi frameworku. Komponentově orientované aplikace jsou tedy vytvářeny z bundlů spojených těmito službami. Služby jsou registrovány přímo ve zdrojovém kódu bundlu. OSGi bundl potřebuje znát kontext bundlu, ke kterému přistupuje bundle zvaný *bundle activator* ve svém vstupním bodu. Příklad jednoduchého zaregistrování služby je možné vidět ve výpisu 2.1 [4].

Kromě vzájemného volání služeb mohou jednotlivé bundly využívat pro komunikaci zasílání zpráv. Pro tuto komunikaci je třeba získat službu od bundlu *EventAdmin* (viz výpis 2.2) a zaregistrovat *listenery*, které budou

[2]Open Services Gateway initiative

zpracovávat události. Každý listener musí implementovat rozhraní `EventHandler`[5].

Každý OSGi bundle může poskytovat své třídy a rozhraní tak, že je zveřejní. Poté je mohou využívat i ostatní bundly ve frameworku. Tato funkcionalita je jiná forma komponentové interakce (mimo komunikaci přes služby) a dá se srovnat se sdílenými knihovnami v operačních systémech [4].

```
IMgGen msgGen = new MsgGenImpl();
 context.registerService("cz.zcu.kiv.cosi.msgtalk.IMsgGen",
   msgGen, new Hashtable("type", "MsGenImpl"));
```

Výpis 2.1: Příklad registrace služby v OSGi. Převzato z [3].

```
this.messageService = (EventAdmin) context.getService(
 context.getServiceReference(
   "org.osgi.service.event.EventAdmin"));
```

Výpis 2.2: Získání služby *EventAdmin*. Převzato z [5].

2.2.2 Spring

Spring framework nabízí řadu funkcí pro podporu vývoje rozsáhlých aplikací. Mezi nejvíce charakteristické funkce Spring frameworku patří aspektově orientovaný přístup programování, datový přístup, řízení transakcí, IoC[3] a vzdálený přístup [4].

Základem Springu je konfigurační soubor, který obsahuje XML[4] jmenný prostor pro beany. Tím usnadňuje konfiguraci elementů. Beany ve Springu v podstatě představují implementace tříd a jsou definované v konfiguračním souboru Springu jako elementy. V konfiguračním souboru lze definovat závislosti mezi komponentami, příklad je uveden ve výpisu 2.3[4].

```
<beans>
  <bean id="foo" class="x.y.Foo">
    <constructor-arg ref="bar" />
    <property name="baz" ref="baz" />
  </bean>
```

[3]Inversion of Control

[4]Extensible Markup Language

```
  <bean id="bar" class="x.y.Bar" />
  <bean id="baz" class="x.y.Baz" />
</beans>
```

Výpis 2.3: Příklad definice beanů ve frameworku Spring. Převzato z [3]

2.2.3 SpringDM

SpringDM[5] představuje rozšíření pro OSGi framework. SpringDM umožňuje OSGi komponentám využívat vlastností Spring frameworku a tím vylepšuje správu OSGi služeb[4].

SpringDM je distribuováno jako několik OSGi bundlů, které jsou spuštěny v cílovém OSGi frameworku. Jeden z bundlů kontroluje nově přidané bundly do frameworku, zda jsou to SpringDM bundly nebo ne[4].

Aby se stal z OSGi bundlu SpringDM bundl, musí se rozšířit XML konfiguračním souborem, který je uložen v meta-informacích `JAR` souboru bundlu. Pro podporu OSGi funkcí musí XML konfigurační soubor obsahovat jmenný prostor Spring-OSGi[4]. Příklad registrace OSGi služby je uveden ve výpisu 2.4.

```
<bean id="EnvironmentService"
        name="environment"
        class="cz.zcu.kiv.crossroadcontrol.trafficcrossroad.
           elements.DefaultEnvironment"
        init-method="start"
        destroy-method="stop"
</bean>

<osgi:service id="EnvironmentServiceOSGi"
                ref="EnvironmentService"
                interface="cz.zcu.kiv.crossroadcontrol.
                   trafficcrossroad.IEnvironment" />
```

Výpis 2.4: Příklad registrace služby ve *SpringDM*

[5]Spring Dynamic Modules

3 Java 3D

V následujících kapitolách bude popsáno Java3D API[1] a vytváření vizuálních objektů.

3.1 Java 3D API

Java 3D API umožňuje psaní programů k zobrazení a interakci s trojrozměrnou grafikou. API poskytuje kolekci vysokoúrovňových konstrukcí, které umožňují vytvářet rozdílné scény s texturami, světly apod. Java 3D API je distribuováno jako volitelný balíček ke standardní distribuci, který je možné stáhnout ze stránek Oracle `http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html`. Všechny třídy jsou popsány v dokumentaci na [6]. Součástí balíčku jsou tři knihovny [7]:

- j3dcore.jar;
- j3dutils.jar;
- vecmath.jar

Jádro tvoří knihovna *j3dcore.jar*, obsahující balíček *javax.media.j3d*, který definuje stovky tříd pro práci s 3D grafikou. Detaily vykreslování jsou obsluhovány automaticky. Díky možnostem práce s vlákny v jazyce Java je jádro 3D Javy schopné vykreslit grafiku paralelně. Také automaticky optimalizuje vykreslování a tím tak zvyšuje výkon [7].

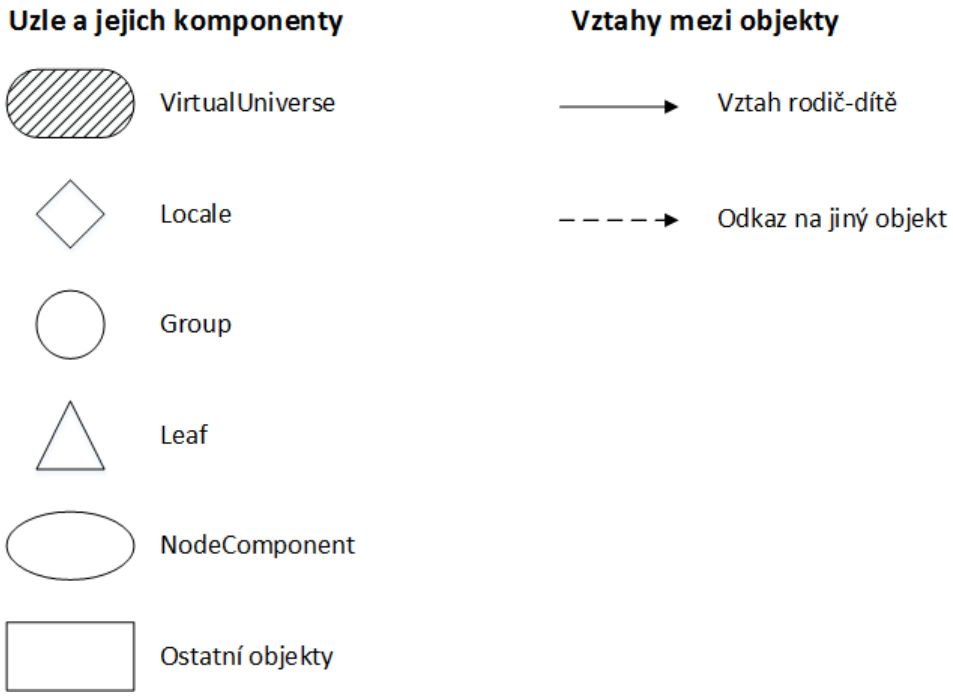
Knihovna *j3dutils.jar*, ve které je balíček *com.sun.j3d.utils*, který obsahuje pomocné třídy pro Javu 3D. Pomocné třídy se využívají například pro načítání objektů, vytváření geometrických útvarů a vytváření grafu scény [7].

Knihovna *vecmath.jar* , ve které je balíček javax.vecmath, který obsahuje matematické třídy pro vytváření vektorů, bodů, matic a jiných matematických objektů [7].

[1] Application Programming Interface

3.2 Graf scény

Java 3D virtuální vesmír je vytvořen z grafu scény. Tento graf scény je vytvořen instancemi tříd Java a je sestaven z objektů definujících geometrii, zvuky, světla, umístění, orientaci a vzhled objektů. Datová struktura grafu je tvořena z uzlů a referencí. Nejběžnějším vztahem mezi uzly je rodič-dítě, přičemž uzel může mít jakýkoliv počet dětí a jednoho rodiče. Pro grafické zobrazení grafu scény se využívají symboly uvedené na obrázku 3.1.



Obrázek 3.1: Symboly používané pro kreslení grafu scény. Převzato z [7].

Symboly na levé straně představují objekty používané v grafu scény. Plná šipka značí vztah rodič-dítě, čárkovaná šipka značí odkaz na jiný objekt[7].

3.2.1 Hlavní třídy scény grafu

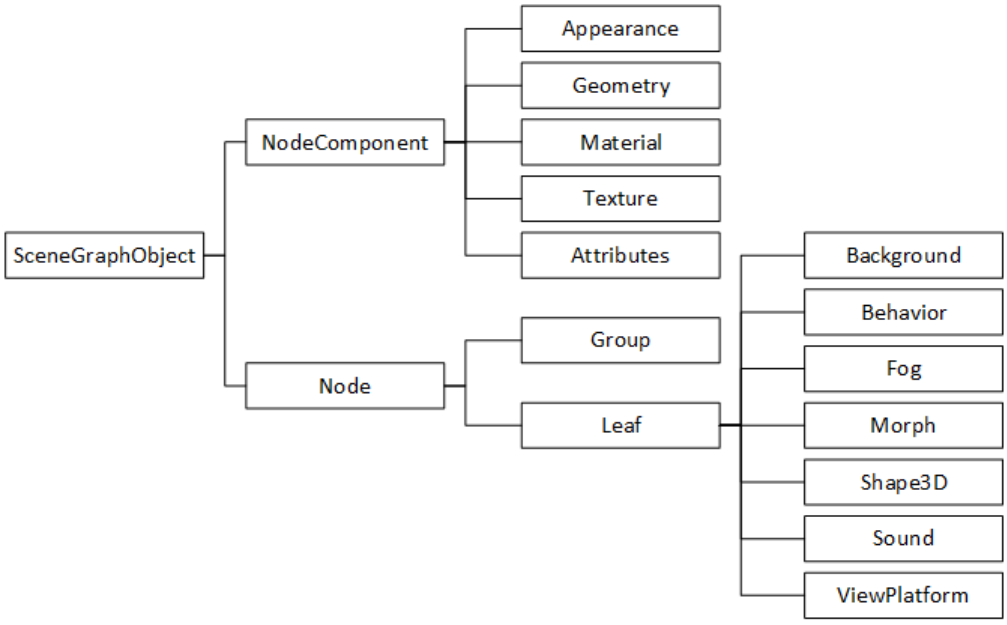
Třída `VirtualUniverse` je kontejner nejvyšší úrovně ve scéně grafu. Může odkazovat na více objektů třídy `Locale`, ale ve většině programů má jen jeden objekt `Locale`[8].

Třída `Locale` představuje ve virtuálním vesmíru referenční bod, který určuje umístění vizuálních objektů, a slouží jako kořen subgrafů grafu scény[8].

Třída `Group` je určena pro specifikaci polohy a orientaci vizuálních objektů ve virtuálním vesmíru. Má dvě podtřídy a těmi jsou `BranchGroup` a `TransformGroup`. `BranchGroup` se užívá pro tvorbu grafu scény a je to jediný objekt, který může být dítě objektu `Locale`. `TransformGroup` uchovává informace o posunu nebo rotaci vizuálního objektu. Posun nebo rotace se vytváří pomocí objektu `Transform3D`, který není součástí grafu scény[8].

Třída `Leaf` se využívá pro specifikaci tvaru, zvuku a chování objektů ve virtuálním vesmíru. Objekty této třídy nemají žádné potomky, ale mohou odkazovat na objekty třídy `NodeComponent`. Příklad podtříd je `Shape3D`, `Light`, `Behavior` a `Sound`[8].

Třída `NodeComponent` slouží pro popsání geometrie (třída `Geometry`), vzhledu (třída `Appearance`), textury (třída `Texture`) a vlastností materiálu (třída `Material`) objektů třídy `Shape3D`. Instance třídy `NodeCompoment` není součástí grafu scény, ale je na ní odkazováno z jednoho nebo více objektů třídy `Shape3D`[8].



Obrázek 3.2: Část hiearchie Java 3D API obsahující třídu `NodeComponent` a její potomky. Převzato z [8].

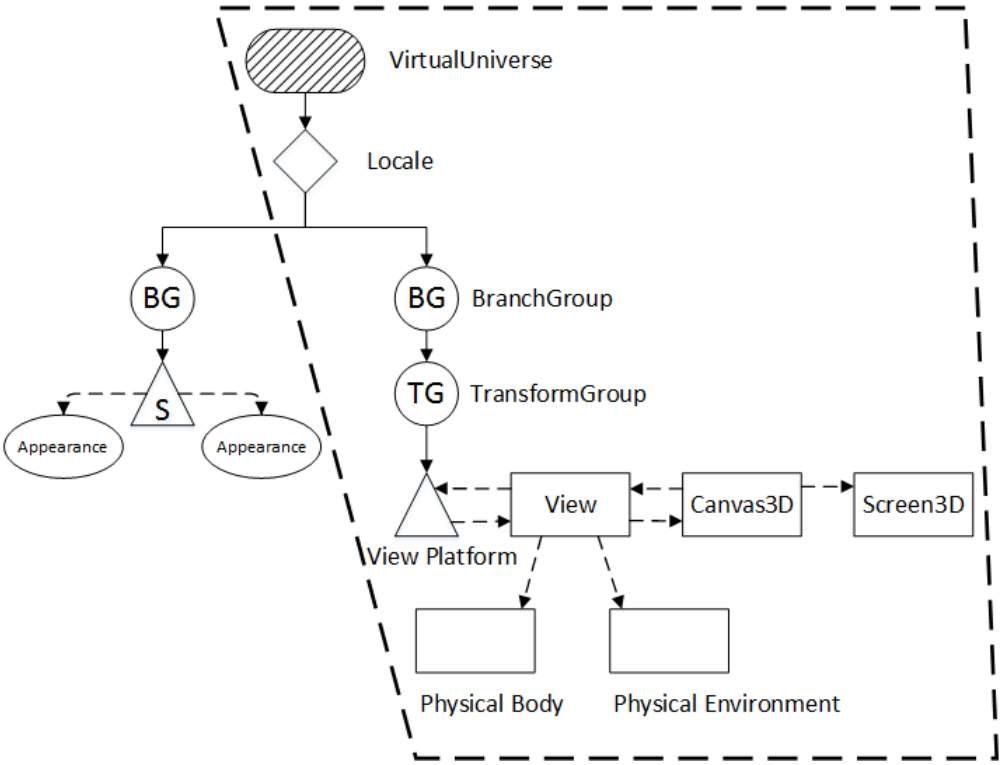
3.3 Postup pro psaní Java 3D programů

Tento postup může být použit pro sestavení programu v Java 3D[7]:

1. Vytvoření objektu `Canvas3D`.
2. Vytvoření objektu `VirtualUniverse`.
3. Vytvoření objektu Locale a připojení k objektu `VirtualUniverse`.
4. Zkonstruování grafu vizuální větve.
   (a) Vytvoření objektu `View`.
   (b) Vytvoření objektu `ViewPlatform`.
   (c) Vytvoření objektu `PhysicalBody`.
   (d) Připojení `ViewPlatform`, `PhysicalBody`, `PhysicalEnvironment` a `Canvas3D` objektů k objektu `View`.
5. Zkonstruování grafu obsahové větve.
6. Přeložení grafu obsahové větve.
7. Vložení subgrafu do `Locale`.

Pro zjednodušení výše uvedeného návodu je k dispozici objekt `SimpleUniverse`. Ten provádí kroky 2, 3 a 4 automaticky a zastupuje funkce všech objektů v čárkovaném rámečku v grafu třídy (viz Obrázek 3.3). Použitím této třídy se redukuje čas potřebný k vytvoření vizuální větve a programátor se může více věnovat obsahové stránce. Postup je pak zredukován na[7]:

1. Vytvoření objektu `Canvas3D`.
2. Vytvoření objektu `SimpleUniverse`, který odkazuje na `Canvas3D`.
   - Úprava objektu `SimpleUniverse`.
3. Zkonstruování obsahové větve.
4. Přeložení grafu obsahové větve.
5. Vložení grafu obsahové větve do `Locale` v `SimpleUniverse`.



Obrázek 3.3: Graf třídy `SimpleUniverse`. Převzato z [7].

3.4 Vytváření vizuálních objektů

V Java 3D existují tři způsoby vytvoření objektu. Jelikož se všechny tři způsoby použijí v navrhované komponentě, budou popsány v následujících kapitolách.

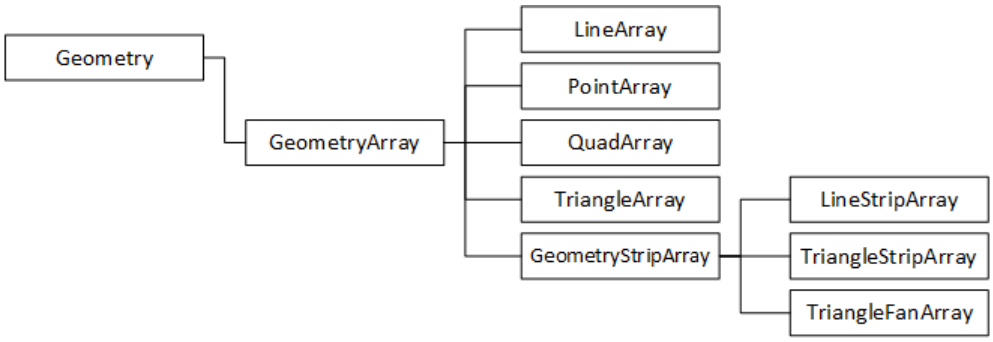
3.4.1 Geometrická primitiva

Nejjednodušším způsobem je použití geometrických primitiv jako je hranol, kužel, válec a koule. Tato primitiva se vytvářejí pomocí tříd `Box`, `Cone`, `Cylinder` a `Sphere` a nacházejí se v balíčku *com.sun.j3d.utils.geometry*. Detaily tříd lze nalézt v dokumentaci[6]. Velikost objektu může být určena pouze při jejich vytváření. Vzhled objektu lze měnit pomocí odkazu na instanci třídy `Appearance`. Umístění lze měnit pomocí `TransformGroup` [8].

3.4.2 Třída Shape3D

Dalším způsobem vytvoření vizuálního objektu je vytvoření instance třídy `Shape3D`. Samotný objekt `Shape3D` neobsahuje žádné informace o vzhledu ani o geometrii objektu. Ty jsou uloženy v objektech `NodeComponent`, na které objekt `Shape3D` odkazuje. Pro popis geometrie může objekt `Shape3D` odkazovat na jeden objekt třídy `Geometry` a pro popis vzhledu může odkazovat na jeden objekt třídy `Appearance`[8].

Geometrie objektu
Třída `Geometry` je abstraktní, tudíž objekt bude odkazovat na instanci jedné z jejích podtříd. V této práci bude uvedena pouze podtřída `GeometryArray`, která je také abstraktní. Proto budou pro vytváření vizuálních objektů použity její potomci. První čtyři potomky[2] (viz Obrázek 3.4) lze použít ke specifikování bodů, úseček a vyplněných polygonů (trojúhelníky a čtyřúhelníky). Tyto třídy ale neumožňují znovupoužití vertexů [8].

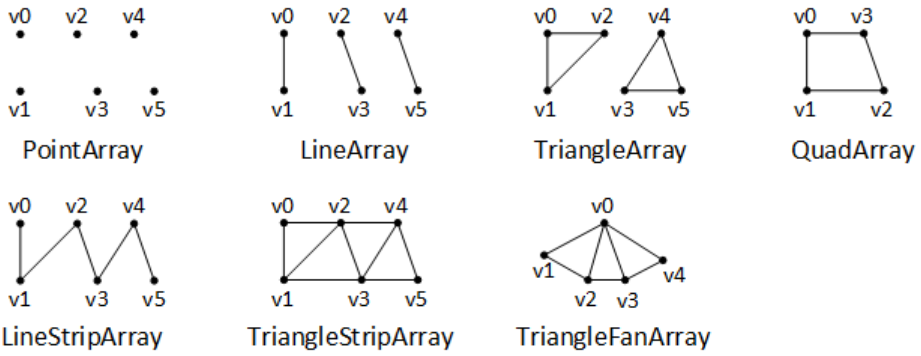


Obrázek 3.4: Část hiearchie Java 3D API obsahující třídu GeometryArray a její potomky. Převzato z [8].

Poslední potomek `GeometryStripArray` je abstraktní třídou, od které jsou odvozeny třídy, které vytvářejí pásová primitiva[3]. Tyto třídy umožňují znovupoužít vložené vertexy a spojit je do jedné křivky nebo plochy. Tím se zlepší výkon při vykreslování. `LineStripArray` vykresluje spojitou čáru. `TriangleStripArray` vytváří trojúhelníky sdílející strany, čili znovupoužívá poslední vykreslený vertex. `TriangleFanArray` znovupoužívá vždy první vertex v pásu každého trojúhelníku [8].

[2] `PointArray`, `LineArray`, `TriangleArray` a `QuadArray`
[3] pro vytváření křivek a povrchů

Všechny výše popsané způsoby vykreslování u jednotlivých tříd jsou znázorněny na obrázku 3.5.



Obrázek 3.5: Způsoby, jakým se vykreslují jednotlivé podtřídy třídy `GeometryArray`. Převzato z [8].

3.4.3 Import vizuálního objektu ze souboru

Posledním způsobem jak vytvořit vizuální objekt je použití loaderu k načtení geometrie ze souboru. Balíček *com.sun.j3d.loaders* poskytuje třídy, které umožňují načítat obsahy jiných grafických souborů, jež se vytvořili pomocí programů pro modelování 3D grafiky. Pro použití loaderu se postupuje takto[9]:

1. Nalézt příslušný loader pro určitý formát.
2. Naimportovat loader.
3. Naimportovat ostatní důležité třídy.
4. Deklarovat odkaz na objekt `Scene` (nepoužívat konstruktor).
5. Vytvořit objekt loaderu.
6. Načíst soubor v *try blocku* a přiřadit výsledek na vytvořený odkaz objektu `Scene`.
7. Vložit objekt `Scene` do scény grafu.

Dle bodu 1 je nutné nalézt vhodný loader. Java3D SDK obsahuje loader načítající soubory formátu `.obj`. Je jím třída `ObjectFile` distribuována v balíčku *com.sun.j3d.loaders*. Po načtení souboru do objektu `Scene` můžeme

získat `BranchGroup` se všemi částmi[4] vizuálního objektu pomocí metody *getSceneGroup()*.

Další užitečnou metodou třídy `Scene` je metoda *getNamedObjects()*, pomocí které můžeme získat tabulku třídy `Hashtable`. Tato tabulka umožňuje rychlý přístup k částem vizuálního objektu podle klíče, kterým je název určité části objektu.

Po těchto krocích můžeme k částem vizuálního objektu přistupovat a upravovat jejich vlastnosti[5][9]. Ukázku popsaného postupu lze vidět ve výpisu 3.1.

```
BranchGroup modelBG = null;
Scene s = null;
ObjectFile f = new ObjectFile();
f.setFlags(ObjectFile.TRIANGULATE | ObjectFile.STRIPIFY);
try {
  s = f.load(fileNameObject);
} catch (FileNotFoundException e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
} catch (IncorrectFormatException e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
} catch (ParsingErrorException e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
}

Hashtable<?, ?> namesTable = s.getNamedObjects();
modelBG = s.getSceneGroup();
modelBG.setUserData(namesTable);
```

Výpis 3.1: Import vizuálního objektu ze souboru

[4]objekty třídy `Shape3D`

[5]např. měnit vzhled (barva, textura), pozici, rotaci, atd.

4 Analýza

Před samotným návrhem a vytvořením komponentové aplikace pro trojrozměrné zobrazení křižovatky, bylo nutné se seznámit s prostředím, ve kterém bude projekt vyvíjen a s prací s komponentami v modelu Spring DM. Způsob komunikace mezi komponentami bude s využitím rozhraní.

Dále bylo nutné se seznámit s dodanou komponentou *TrafficCrossroad*, která poskytuje rozhraní potřebná k ovládání silniční křižovatky a poskytuje veškeré informace pro její vizualizaci. Pro vyzkoušení a testování byly dodány dvě mapy křižovatek.

4.1 Seznámení s komponentou TrafficCrossroad

Pro návrh komponenty bylo nutné se seznámit s dodadanou komponentou *TrafficCrossroad*. Komponenta poskytuje rozhraní, které implementuje a grafické uživatelské rozhraní. Toto 2D *GUI*[1] zobrazuje silniční křižovatku v pohledu ze shora.

4.1.1 Rozhraní komponenty

Zde budou popsána základní rozhraní, která budou potřeba pro chod navrhované komponenty.

Základním rozhraním dodané komponenty je `IEnvironment`, které umožňuje přístup k ovládání simulace silniční křižovatky a poskytuje přístup ke všem elementům[2] silniční křižovatky. Takto se dostane navrhovaná komponenta ke všem parametrům[3] potřebným pro vizualizaci křižovatky.

Rozhraní `ITrafficCrossroad` poskytuje veškeré informace o celé křižovatce a obsahuje seznamy všech elementů.

[1] Graphical User Interface

[2] např. ramena křižovatky, pruhy, semafory, vozidla, přechody atd.

[3] např. souřadnice umístění, velikost a natočení objektů, stavy semaforů, atd.

Rozhraní `ITrafficArm` poskytuje seznam příjezdových a odjezdových pruhů. Dále obsahuje informaci o umístění a natočení ramene.

Rozhraní `IOutgoingTrafficLane` poskytuje velikost odjezdového pruhu stejně jako rozhraní `IIncomingTrafficLane`, u kterého se navíc bude zjišťovat směr jízdy pro vykreslení značení[4].

Rozhraní `ITrafficLight` poskytuje typ semaforu, směr jízdy a aktuální stav semaforu, který je důležitý pro ovládání světel semaforu.

Rozhraní `IVehicle` poskytuje typ, barvu a rozměr pro vytvoření vozidla a pro ovládání vozidla poskytuje aktuální pozici a informaci, zda vozidlo bude při příštím kroku simulace na konci cesty.

4.1.2 2D GUI

V horní části okna se nachází tlačítka pro ovládání simulace silniční křižovatky, které budou obsaženy i v navrhované komponentě *TrafficCrossroad3D* aby se nemusela simulace vykreslovat ve 2D i 3D zároveň. Ve spodní části se dále uvádí informace o uběhlém času simulace a nastavení zpoždění křižovatky. Uprostřed je panel vykreslující křižovatku a běh simulace. Díky tomu bude možné vizuálně ověřit správné umístění a pohyb objektů ve scéně navrhované komponenty. Náhled okna je v obrázku 4.1.

4.2 Specifikace požadavků

Komponenta musí umožňovat následující:

1. Výběr silniční křižovatky, která se bude simulovat.
2. Ovládání simulace silniční křižovatky.
3. Vizualizace silniční křižovatky.
4. Změna pohledu a pohyb kamery.

Při startu komponenty *TrafficCrossroad* se načtou všechny dostupné křižovatky z XML souboru a vždy je aktivní právě jedna křižovatka. Navrhovaná

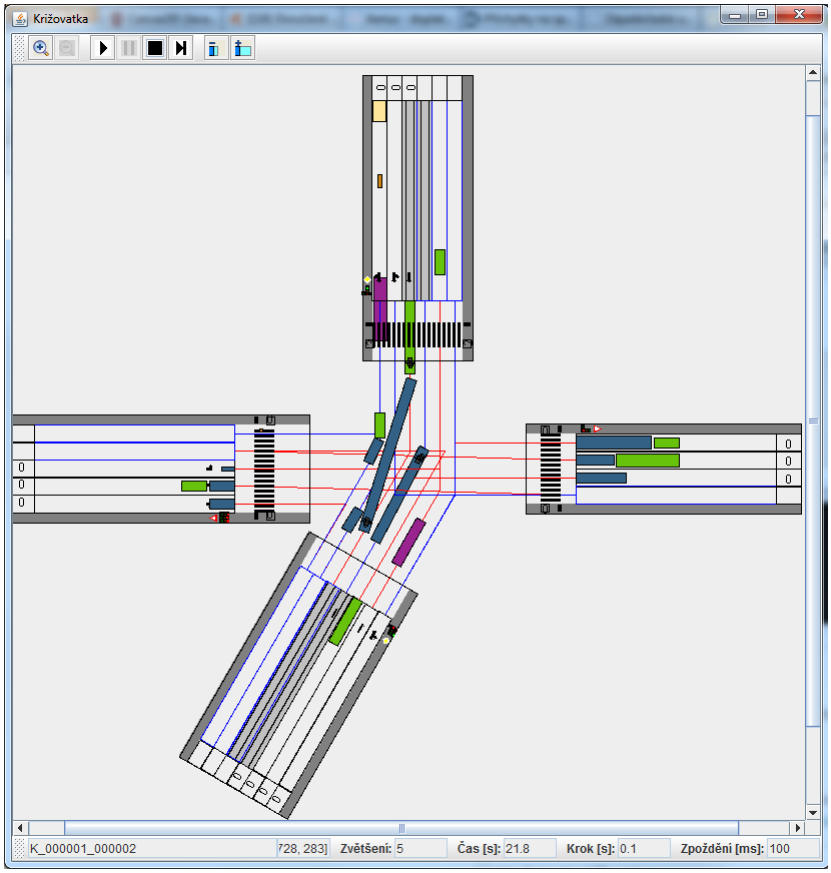
[4]šipky značící směr jízdy

komponenta *TrafficCrossroad3D* bude muset poskytnout seznam načtených křižovatek a umožnit výběr křižovatky, která bude aktivní.

Ovládání simulace poskytne zapnutí, pozastavení, zastavení a krokování simulace. Dále bude umožněno zvyšování a snižování zpoždění běhu simulace.

Vizualizace silniční křižovatky by se měla co nejvíce přibližovat reálnému pohledu. Pro vyšší přehlednost se nebudou okolo křižovatky zobrazovat žádné jiné objekty např. budovy, stromy, atd.

Pro pohyb kamery se bude využívat klávesnice a tlačítka s předdefinovanými pozicemi pro rychlejší změnu pohledu.

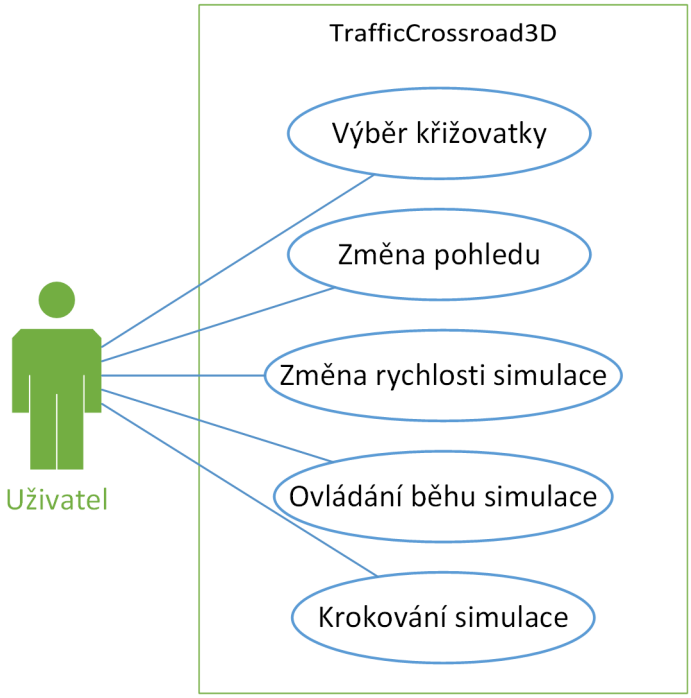


Obrázek 4.1: Grafické 2D rozhraní dodané komponenty.

4.3 Případy užití

Jednotlivé případy užití jsou znázorněny a popsány v následujících kapitolách.

4.3.1 Diagram případů užití



Obrázek 4.2: Diagram případů užití komponenty.

4.3.2 Popisy případů užití

Výběr křižovatky
Uživatel otevře dialogové okno s výběrem křižovatky. Výběrem a potvrzením se zruší stávající křižovatka a načte se nově zvolená.

Změna pohledu
Uživatel změní pohled pomocí tlačítek z devíti směrů[5] pro rychlé nastavení

[5]sever, severovýchod, východ, jihovýchod, jih, jihozápad, západ, severozápad a střed

pohledu kamery. Dále bude moci měnit pohled pomocí klávesnice.

Změna rychlosti simulace
Uživatel zvýší nebo sníží zpoždění běhu simulace. Pokud se zpoždění zvýší, bude běh simulace pomalejší.

Ovládání běhu simulace
Uživatel spustí, zastaví či pozastaví běh simulace.

Krokování simulace
Uživatel provede jeden krok simulace.

4.4 Návrh 3D zobrazení

Grafické uživatelské rozhraní komponenty bude obsahovat ovládací panel simulace silniční křižovatky, ovládací panel kamery pro změnu pohledu a panel s 3D vizualizací. Realizace panelu s 3D vizualizací bude provedena podle návodu uvedeného v kapitole 3.3. Obsahová větev bude realizována podle zjednodušeného grafu scény (viz obrázek 4.3).

4.4.1 Popis grafu scény

`SimpleUniverse` je instance třídy, do které se vloží obsahová větev viz kapitola 3.3. *Root scene graph* představuje kořenový uzel, do kterého se budou vkládat všechny elementy scény. `KeyNavigatorBehavior` je třída, která se stará o změnu pohledu kamery pomocí klávesnice.

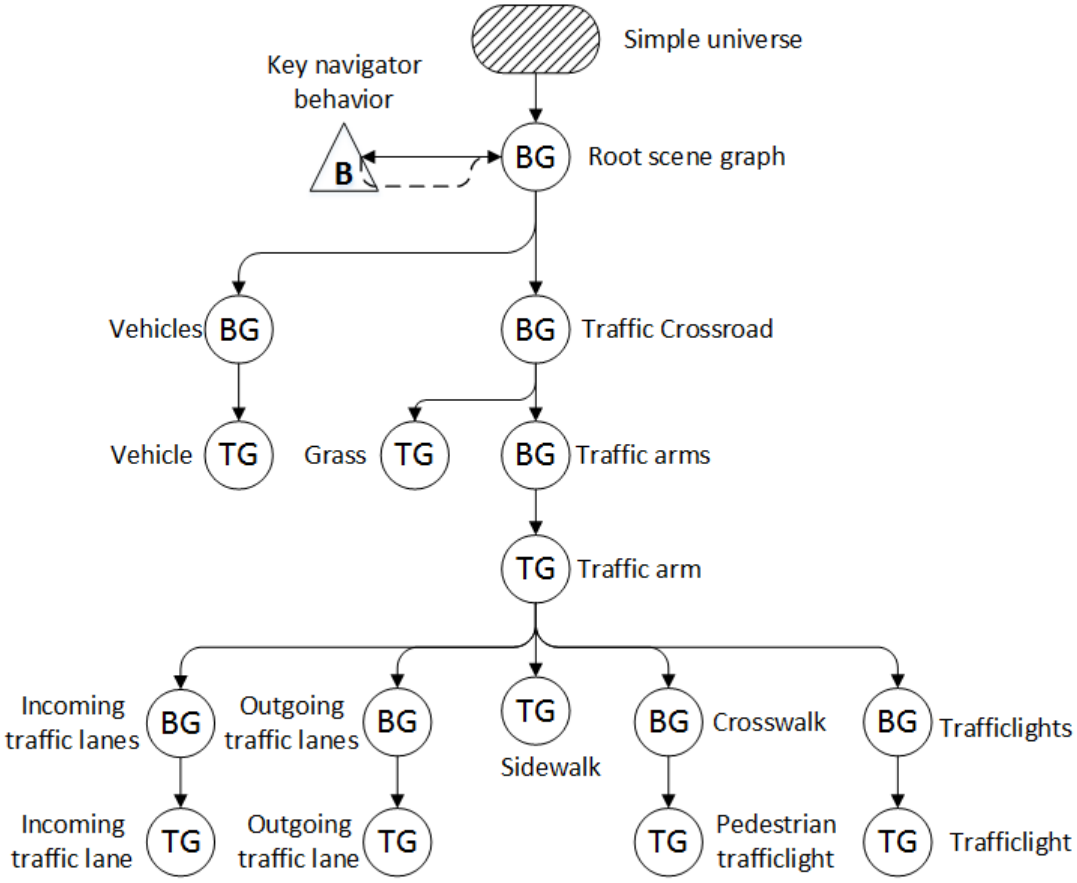
Uzel *Vehicles* bude obsahovat vozidla pohybující se v křižovatce. Kvůli skutečnosti, že se vozidla budou přidávat a odebírat za běhu komponenty, jsou objekty vozidel ukládány mimo uzel samotné křižovatky. Uzel *Traffic Crossroad* a jeho potomky popíši v následující kapitole.

4.4.2 Vytvoření křižovatky

Datová struktura podstromu tvořeného uzlem *Traffic Crossroad* a jeho potomky se skládá z uzlů typu `BranchGroup` a `TransformGroup`. Z grafu scény

lze vyčíst, že `BranchGroup` bude sloužit jako seznam určitých elementů křižovatky a každý tento element bude v `TransformGroup`, aby se mohl přemístit na svou pozici a případně natočit do svého směru.

Při vytváření křižovatky se budou postupně procházet všechny seznamy elementů, které jsou k dispozici prostřednictvím rozhraní dodané komponenty *TrafficCrossroad* popsané v kapitole 4.1.1.



Obrázek 4.3: Zjednodušený graf scény.

4.4.3 Vytvoření složitých objektů v křižovatce

Zatímco se jednoduché objekty[6] mohou vytvářet přímo v kódu Java 3D (viz kapitoly 3.4.1 a 3.4.2), objekty vozidel, semaforů a značek jsou příliš složité

[6]např. silniční pruhy, chodníky, roviny, atd.

a je potřeba je vytvořit v nějakém programu pro modelování 3D vizuálních objektů. Takto vytvořený model se naimportuje do komponenty podle návodu popsaného v kapitole 3.4.3. Dále je pro některé takto vytvořené objekty nutné vytvořit textury, které se na objekt namapují.

Semafory a značky
Objekty semaforů a značek se vytvoří tak, jak je popsáno výše a vloží se do `TransformGroup`. U té se nastaví příslušná pozice a natočení a vloží se do příslušného uzlu. Pro ovládání semaforů se z `BranchGroup` vyberou objekty třídy `Shape3D` představující světla semaforu a jejich odkazy se uloží do seznamu společně s odkazem na rozhraní semaforu z dodané komponenty.

Vozidla
Díky skutečnosti, že se geometrie a textury importují zvlášť, pro každý typ vozidla se naimportuje geometrie do objektu třídy `BranchGroup` jen jednou. Při vkládání vozidla do křižovatky se tato `BranchGroup` naklonuje a všechny části objektu se odkáží na instanci třídy `Appereance` s příslušnou texturou. Tato `BranchGroup` se vloží do `TransformGroup`, u té se nastaví pozice a natočení a vloží se do uzlu *Vehicles*. Pro ovládání vozidel se musí odkaz na `TransformGroup` a rozhraní vozidla z dodané komponenty uložit do kolekce `HashMap`. Ovládání vozidel bude detailněji popsáno v kapitole 4.4.5

Vzhledem k tomu, že barvy vozidel jsou v dodané komponentě generovány náhodně, musela by se měnit barva textury za běhu, což by bylo komplikované. Pro vyřešení tohoto problému bude pro každý typ vozidla připravena sada textur, které se budou lišit svou barvou. Při vkládání vozidla se objekt odkáže na náhodně vybranou instanci třídy `Appereance` s příslušnou texturou.

V případě, že pro daný typ vozidla nebude k dispozici model nebo textura, komponenta neskončí chybou, ale vytvoří se obyčejný kvádr (viz kapitola 3.4.1) s příslušnými rozměry a barvou.

Pro snadné doplnění typu vozidel, nových modelů nebo textur, bude umístění těchto modelů uloženo v `XML` souboru. Tento soubor se při startu komponenty načte a poté se načtou všechny modely a jejich textury. Struktura `XML` souboru a umístění všech souborů je uvedeno v příloze B.

4.4.4 Ovládání semaforů

Při aktualizaci křižovatky se projde seznam semaforů. U každého semaforu se přes uložené rozhraní dodané komponenty zjistí, jaké barvy světel jsou aktuálně nastaveny, a na tyto barvy se světla nastaví.

4.4.5 Ovládání vozidel

Ovládání vozidel bude řešeno na podobném principu jako ovládání semaforů. Rozdíl je v tom, že se vozidla přidávají a odstraňují za běhu simulace. Pro zajištění rychlého přístupu k vozidlu pro změnu pozice, musí být `TransformGroup` s vozidlem uložena v kolekci `HashMap`, kde je klíčem ID vozidla.

Při každé aktualizaci se musí projít všechna vozidla přes rozhraní dodané komponenty *TrafficCrossroad* a zjistit, zda se vozidlo nachází v kolekci. Pokud zde vozidlo není obsaženo, vytvoří se nové vozidlo, vloží se do `TransformGroup` a ta se vloží do uzlu grafu scény *Vehicles* a do kolekce. Pokud vozidlo je již v kolekci obsaženo, nastaví se u jeho `TransformGroup` nová pozice a natočení. Dále se kontroluje, zda vozidlo nebude příští krok na konci cesty. V tomto případě se vozidlo vymaže z kolekce i z uzlu *Vehicles*.

5 Implementace

V této části bude popsána výsledná implementace komponenty pro 3D zobrazení křižovatky.

5.1 Vývojové prostředí

Pro vývoj softwarové komponenty bylo použito prostředí "Eclipse IDE for Java EE Developers" pro J2EE ve verzi "Helios Service Release 2", do kterého byl doinstalován nástroj "SpringSource Tool Suite". Pro použití *SpringDM* bylo nutné naimportovat do workspace bundly z balíku "Spring Dynamic modules".

Pro podporu Java3D bylo nutné do projektu komponenty naimportovat knihovny z Java 3D API uvedené v kapitole 3.1.

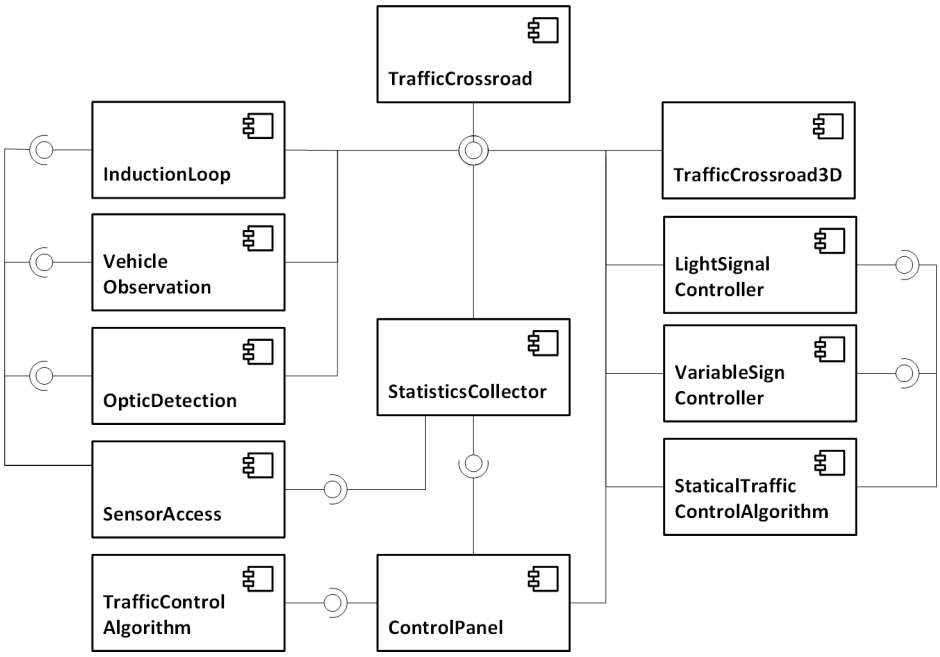
Pro vytváření složitějších vizuálních objektů[1] byl použit program "Autodesk 3ds Max 2012, Release 14, student". Pro vytváření textur k těmto objektům byl použit program "Adobe Photoshop CS6 Extended, verze 13.0".

5.2 Struktura aplikace CrossRoadControl

V této kapitole je uveden diagram komponent celé aplikace (viz obrázek 5.1). Je na něm znázorněna komunikace mezi dodanými komponentami a navrhovanou komponentou pro 3D zobrazení *TrafficCrossroad3D*.

Z diagramu vyplývá, že *TrafficCrossroad3D* využívá jen rozhraní komponenty *TrafficCrossroad* a neposkytuje ostatním komponentám žádné služby.

[1] např. vozidla, semafory, značky, atd.

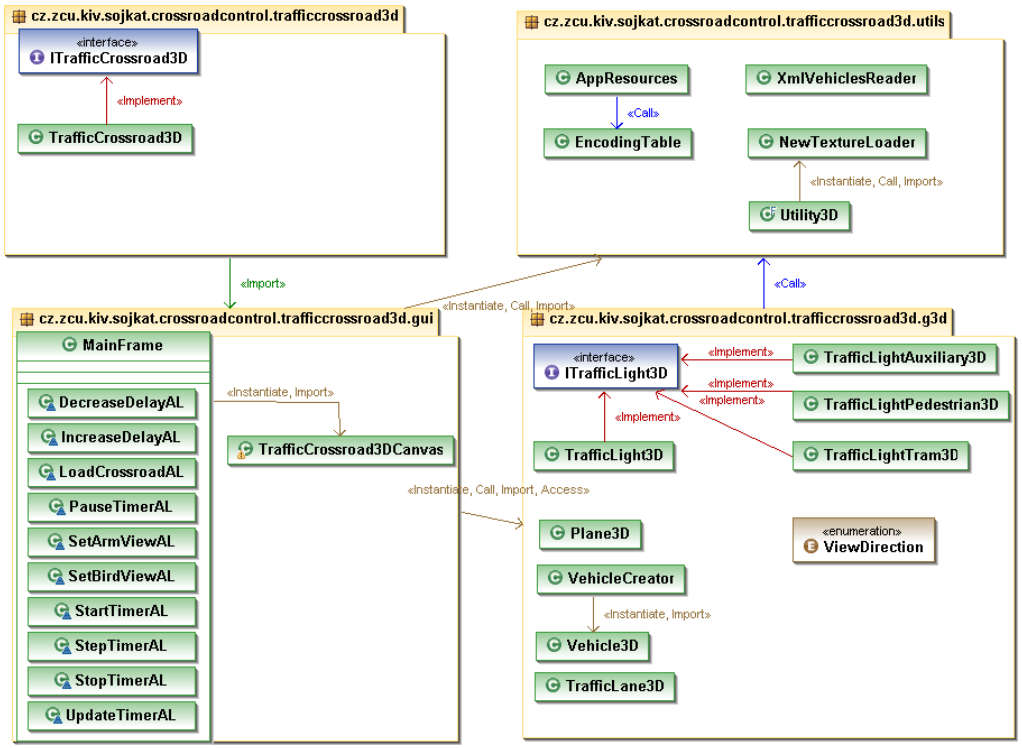


Obrázek 5.1: Diagram napojení komponent.

5.3 Struktura navrhované komponenty

Komponenta se skládá ze tří balíků *g3d, gui a utils*. V těchto balících jsou umístěny všechny třídy komponenty kromě `TrafficCrossroad3D`. Tato třída je umístěna mimo balíky a implementuje rozhraní `ITrafficCrossroad3D`, které zajišťuje spuštění komponenty ve frameworku `SpringDM` viz kapitola 2.2.3.

V následujících kapitolách budou popsány jednotlivé balíky a třídy v nich obsažené. Zjednodušený UML diagram je na obrázku 5.2.



Obrázek 5.2: UML diagram komponenty.

5.3.1 Balík g3d

Tento balík obsahuje všechny třídy související s vytvářením grafických 3D objektů a stavbou grafické scény. Následně budou popsány jednotlivé třídy.

ITrafficLight3D
Rozhraní, které musí implementovat všechny třídy, které vytváří objekt pro ovládání semaforu během simulace. Definuje barvy světel a metody pro nastavení základních barev světel a nastavení aktuálního stavu semaforu. Dále definuje metody pro nastavení schopnosti objektů světel tak, aby se daly měnit za chodu aplikace a pro nastavení ovladače semaforu.

TrafficLight3D
Třída implementující rozhraní `ITrafficLight3D`, která slouží k ovládání klasického semaforu se třemi světly[2].

[2]červená, žlutá a zelená

TrafficLightAuxiliary3D
Třída implementující rozhraní `ITrafficLight3D`, která slouží k ovládání semaforu, který signalizuje odbočení vpravo. Tento semafor má jen jedno světlo[3].

TrafficLightPedestrian3D
Třída implementující rozhraní `ITrafficLight3D`, která slouží k ovládání semaforu pro chodce, tento semafor má dvě světla, které vyobrazují chodce[4].

TrafficLightTram3D
Třída implementující rozhraní `ITrafficLight3D`, která slouží k ovládání semaforu pro tramvaje. Tento semafor se liší od ostatních nejvíce. Obsahuje čtyři žlutá světla, která se rozsvicují nejen podle stavu semaforu, ale i podle směru pruhu, pro který je semafor určen.

Plane3D
Třída je oddděděná od třídy `Shape3D` a slouží k vytvoření roviny[5]. Geometrie je tvořena pomocí pásového primitiva `TriangleFanArray`, viz kapitola 3.4.2. Při vytváření roviny se musí předat pole bodů, kde první bod je střed roviny a další body jsou kolem středu v pořadí proti směru hodinových ručiček.

TrafficLane3D
Tato třída je rovněž oddděna od třídy `Shape3D`. Primárně třída slouží k vytvoření silničních pruhů, ale dá se využít k vytvoření ostatních rovin, které jsou definované délkou a šířkou. Při vytváření instance třídy se dá ovlivnit posunutí roviny a opakování textury.

Vehicle3D
Třída, která slouží pro ovládání vozidel během simulace. Obsahuje `TransformGroup` vozidla, pomocí které se dá měnit pozice a ovladač vozidla, tj. objekt implementující rozhraní `IVehicle`, přes které se dá zjistit aktuální pozice vozidla.

Důležitou metodou je `updatePosition()`, která zjistí přes ovladač, jakou má vozidlo aktuální pozici a tuto pozici nastaví u `TransformGroup` vozidla.

[3] zelená šipka doprava
[4] červená a zelená
[5] např. střed křižovatky

Další metodou, která by se měla uvést, je `isOnEnd()`. Ta zjistí, zda se vozidlo nachází v odjezdovém pruhu[6] a zda je na konci cesty.

VehicleCreator
Tato třída slouží k vytváření instancí třídy `Vehicle3D` a obsahuje všechny dostupné modely a textury vozidel.

Třída poskytuje metody pro načtení 3D modelů z `.OBJ` souborů podle typu vozidel, které poté uloží do *hashmapy*, kde bude klíčem typ vozidla a hodnotou bude třída `BranchGroup` s vozidlem.

Další metodou je metoda pro načtení textur, které se uloží rovněž do *hashmapy*. Klíčem je typ vozidla a hodnotou je třída `List<Appereance>`. Tento seznam bude uchovávat vzhled s určitou texturou pro daný typ vozidla.

Nejdůležitější metodou třídy je `getVehicle(IVehicle vehicle)`, která vrátí objekt třídy `Vehicle3D`. Algoritmus funguje tak, že nejdříve zjistí, zda je k dispozici pro daný typ vozidla 3D model i textura. Pokud jeden z těchto prvků není k dispozici, tak se místo modelu vozidla vytvoří kvádr o rozměrech daného typu vozidla a přiřadí se mu jeho barva. V opačném případě se vytvoří klon 3D modelu pro daný typ vozidla, náhodně se vybere ze seznamu textura a přiřadí se k modelu.

Takto připravený model[7] se vloží do `TransformGroup`, která se společně s ovladačem použije při vytváření instance `Vehicle3D`.

ViewDirection
Třída výčtového typu, která poskytuje hodnoty pro 9 zeměpisných směrů. Tyto směry jsou potřeba při změně pohledu na křižovatku.

5.3.2 Balík gui

Balík *gui* obsahuje třídy `MainFrame` a `TrafficCrossroad3DCanvas`, které vytváří grafické uživatelské rozhraní pro ovládání simulace a vizualizaci.

MainFrame
Třída oddděděná od třídy `JFrame`, která vytváří hlavní okno komponenty s

[6] pruh je instancí `IOutgoingTrafficLane`
[7] kvádr nebo načtený 3D model vozidla

ovládacími prvky a panelem pro 3D vizualizaci, viz níže. Většina metod slouží pro vytváření ovládacích prvků, které není nutné více rozebírat.

Třída dále obsahuje vnitřní třídy, které zapouzdřují implementaci rozhraní pro obsluhu událostí. Všechny tyto vnitřní třídy implementují rozhraní `ActionListener`.

Třída obsahuje časovač, který při každém tiknutí posune simulaci o jeden krok a zaktualizuje vizualizaci křižovatky. Pro obsluhu časovače slouží tyto vnitřní třídy:

- `UpdateTimerAL` - obsluha časovače, která při každém tiknutí zaktualizuje stav simulace.
- `StartTimerAL` - spustí časovač.
- `StopTimerAL` - zastaví časovač.
- `PauseTimerAL` - pozastaví časovač.
- `IncreaseDelayAL` - zvýší prodlevu časovače.
- `DecreaseDelayAL` - sníží prodlevu časovače.

Dále třída obsahuje vnitřní třídy pro nastavení pohledu a načtení nové křižovatky. U nastavení pohledu se zavolá příslušná metoda u instance třídy `TrafficCrossroad3DCanvas` se zvolenými parametry. U načtení křižovatky se vytvoří dialogové okno a po výběru křižovatky se zavolá příslušná metoda u instance třídy `TrafficCrossroad3DCanvas`.

- `SetBirdViewAL` - nastavení ptačího pohledu.
- `SetArmViewAL` - nastavení pohledu z ramene křižovatky.
- `LoadCrossroadAL` - načte nově zvolenou křižovatku.

TrafficCrossroad3DCanvas
Tato třída je oddděděna od třídy `Canvas3D`, která vytváří panel pro 3D vizualizaci. Metody vracející objekt třídy `TransformGroup` nebo `BranchGroup` představují jednotlivé uzle grafu scény, viz kapitola 3.2. Tyto metody není třeba popisovat. Následně budou popsány metody, které zajišťují ovládaní křižovatky za běhu simulace. Pro popsání těchto metod je důležité zmínit, že třída obsahuje seznam semaforů, který obsahuje objekty implementující rozhraní `ITrafficLight3D`. Dále obsahuje *hashmapu* s vozidly, u které je klíčem ID vozidla a hodnotou objekt třídy `Vehicle3D`.

Metoda `updateTraffic()` zajišťuje nastavení aktuálního stavu křižovatky a provede tyto tři kroky:

1. Projde seznam se semafory a u každého semaforu zavolá metodu pro nastavení aktuálního stavu světel.
2. Skrze rozhraní komponenty *TrafficCrossroad* dostane seznam všech silničních pruhů a projde ho. U každého pruhu projde seznam vozidel, kde pro jednotlivá vozidla zavolá metodu `updateVehicle(IVehicle vehicle)`. Tato metoda je popsána níže.
3. Opět skrze rozhraní komponenty *TrafficCrossroad* dostane seznam všech cest. Tyto cesty spojují přijezdové a odjezdové pruhy, tzn. že vozidla v nich obsažená se nacházejí uprostřed křižovatky. Dále se opět pro každé vozidlo zavolá metoda `updateVehicle(IVehicle vehicle)`.

Metoda `updateVehicle(IVehicle vehicle)` zajišťuje nastavení aktuální pozice vozidla v křižovatce. Nejdříve podle ID zjistí, zda je vozidlo již umístěno v křižovatce. Pokud ano, zavolá se u vozidla metoda `updatePosition()` pro aktualizaci pozice. Pokud vozidlo obsaženo není, vytvoří se nové vozidlo pomocí třídy `VehicleCreator`, které se vloží do seznamu vozidel. Dále se `TransformGroup` vozidla vloží do grafu scény.

Metoda `setView()` je určena pro nastavení pohledu. Vzhledem k tomu, že metoda je přetížená, může metoda nastavit pohled podle těchto kritérií:

1. `setView(ViewDirection direction)` nastaví pohled z požadovaného směru ke středu křižovatky.
2. `setView(Coordinate coordinate)` nastaví pohled z požadovaného bodu ke středu křižovatky.
3. `setView(Point3d positionTraffic, Point3d positionObserver)` nastaví pohled požadovaného bodu *positionObserver* k požadovanému bodu *positionTraffic*.

Metoda `changeTrafficCrossroad(ITrafficCrossroad activeTC)` má funkci pro načtení nové křižovatky. V prvním kroku odstraní všechny uzle z grafu scény a poté zavolá metodu pro vytvoření grafu scény pro novou křižovatku.

5.3.3 Balík utils

Balík *utils* obsahuje tyto pomocné třídy:

Utility3D
Třída `Utility3D` poskytuje statické metody pro výpočty potřebné při umisťování 3D objektů v grafu scény. Dále obsahuje metody pro načítání objetků a textur ze souboru.

XmlVehiclesReader
Tato třída je určena pro parsování XML souboru, ve kterém jsou uloženy cesty k 3D modelům a texturám pro určité typy vozidel.

5.4 Vytvoření 3D modelů

Další důležitou částí vývoje komponenty bylo vytvoření modelů vozidel a jiných 3D objektů, které jsou třeba k vizualizaci křižovatky. Jednoduché objekty[8] jsou vlastní tvorby.

Část modelů vozidel byla stažena z volně přístupné knihovny 3D modelů na `http://www.turbosquid.com/`. Jedná se o tyto typy vozidel:

- MOTORCYCLE
- PASSENGER_VEHICLE
- VAN
- BUS

Tyto modely byly následně upraveny na rozměry odpovídající rozměrům vozidel v aplikaci. Dále bylo nutné na všechny modely namapovat textury a vytvořit textury ve více barevných variantách.

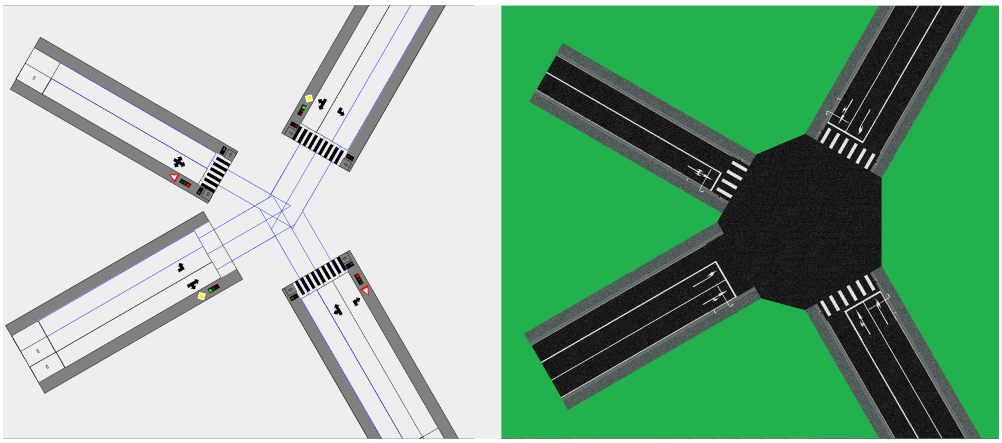
Zbývající část modelů vozidel je vlastní tvorby. Modely se vytvářely podle nějaké šablony, např. výkresy vozidel nebo papírové modely.

Vzhled všech typů vozidel je uveden v příloze C

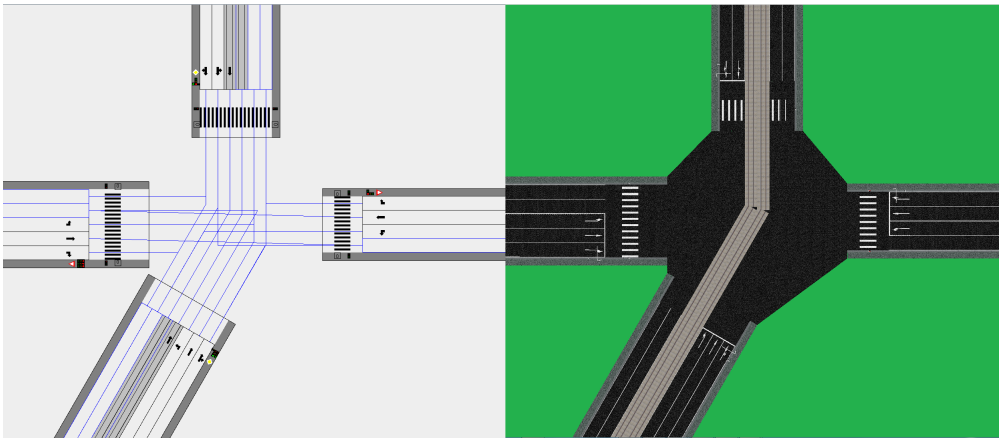
[8] semafory a dopravní značky

6 Otestování a demonstrace komponenty

Komponenta se testovala na dvou dodaných křižovatkách. První křižovatka, kterou budeme značit K1, byla klasická křižovatka se světelnou signalizací se čtyřmi rameny, viz obrázek 6.1. Druhá křižovatka K2 má také světelnou signalizaci a čtyři ramena. Od první křižovatky se liší tím, že má více pruhů a obsahuje tramvajový pás, viz obrázek 6.2.



Obrázek 6.1: Křižovatka K1 ve 2D a 3D pohledu.

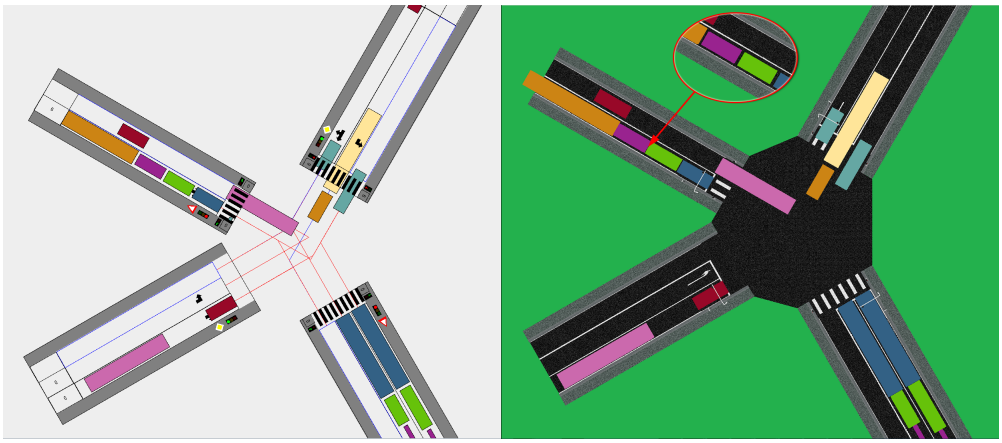


Obrázek 6.2: Křižovatka K2 ve 2D a 3D pohledu.

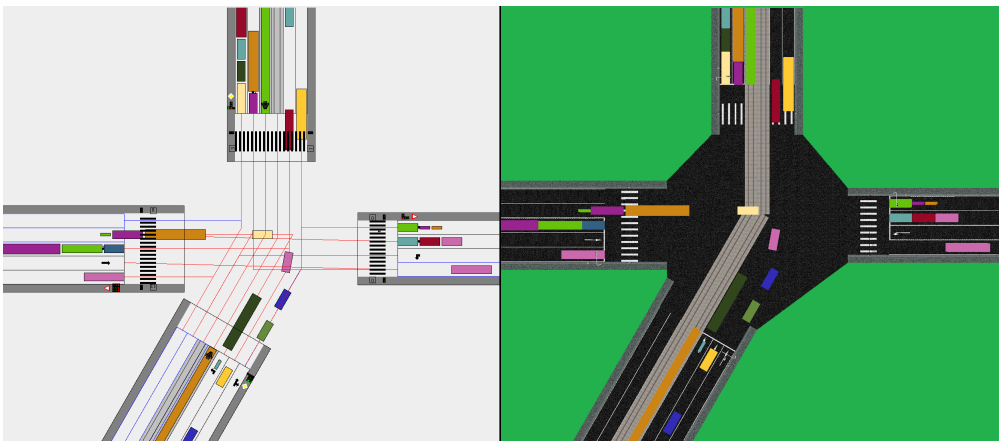
6.1 Vizuální testy

Tato forma testování bude testovat správné umístění objektů v křižovatce. Jako vzor, podle kterého se bude porovnávat 3D vizualizace bude sloužit 2D GUI dodané komponenty. Pro lepší porovnání s 2D zobrazením nebudou v 3D komponentě naimportovány žádné modely vozidel. To bude mít za důsledek to, že se místo reálných modelů vozidel budou zobrazovat kvádry se stejnou velikostí a barvou, jako ve 2D zobrazení.

6.1.1 Umístění vozidel



Obrázek 6.3: Testování správného umístění vozidel u mapy K1.

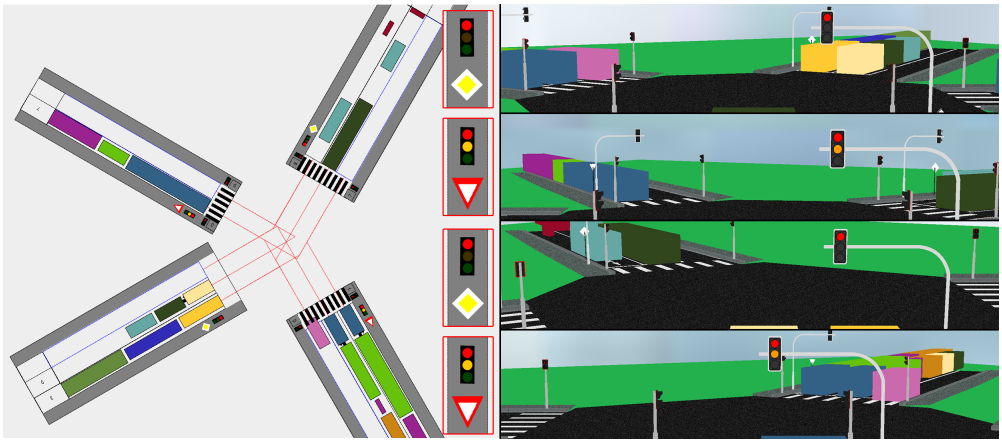


Obrázek 6.4: Testování správného umístění vozidel u mapy K2.

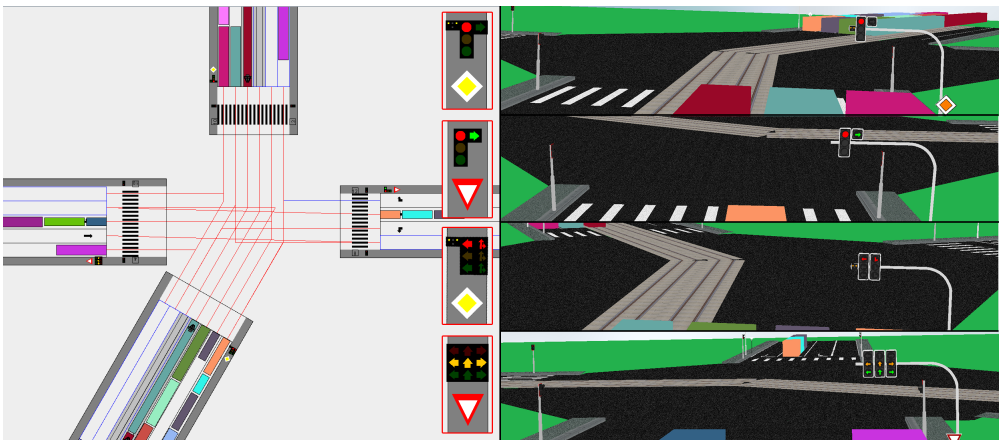
Na obrázcích 6.3 a 6.4 lze vidět, že jsou vozidla umístěna správně. U 3D zobrazení lze vidět, že mezi vozidly nejsou takové mezery jako u 2D zobrazení. To je způsobeno tím, že vozidla mají nějakou výšku a pozorujeme je z nějakého úhlu. Kdybychom jsme se přesunuli kolmo nad tato vozidla, mezeru bychom už viděli, viz bublina na obrázku 6.3

6.1.2 Nastavení semaforů

Z obrázků 6.5 a 6.6 je patrné, že vizualizace semaforů odpovídá stavu ve 2D zobrazení dodané komponenty.



Obrázek 6.5: Nastavení semaforů u mapy K1.



Obrázek 6.6: Nastavení semaforů u mapy K2.

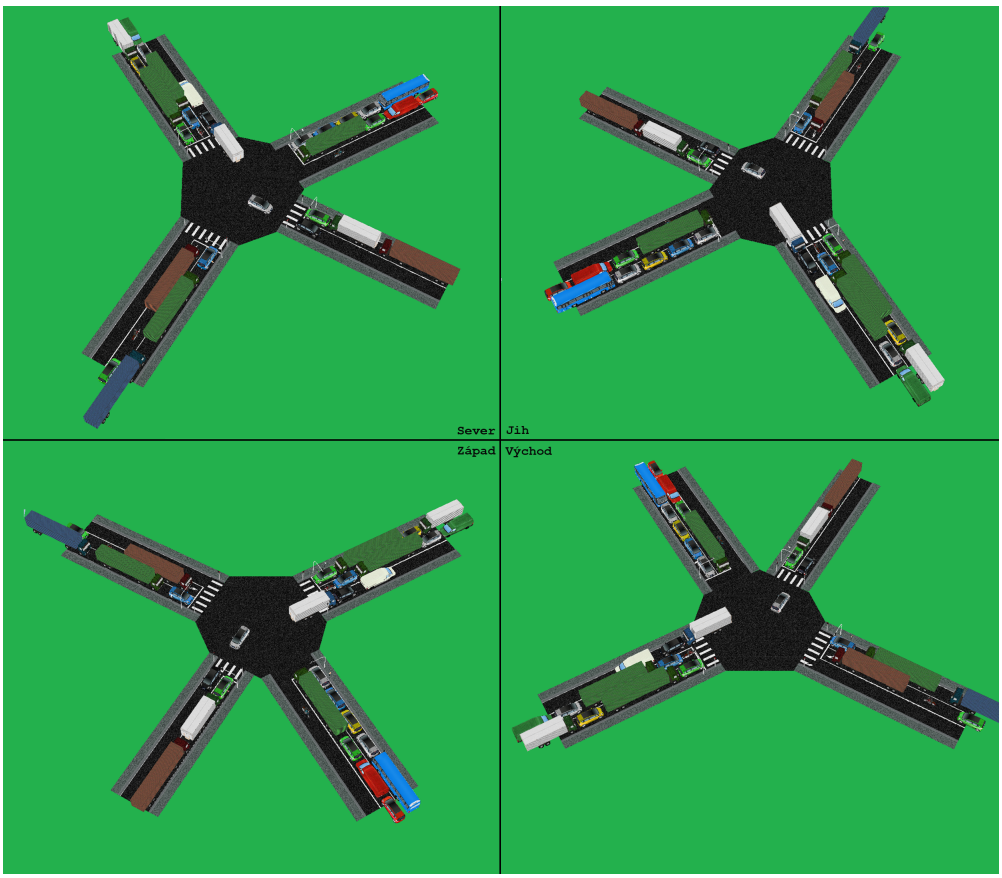
Kontrolu semaforů již nešlo provádět z pohledu shora, proto je 3D zobrazení rozděleno do čtyřech částí. V každé části jsou vyfoceny semafory pro jedno rameno křižovatky a u nich je pro lepší čitelnost zobrazeno zvětšení semaforu ve 2D zobrazení.

6.1.3 Umístění přechodů a značení pruhů

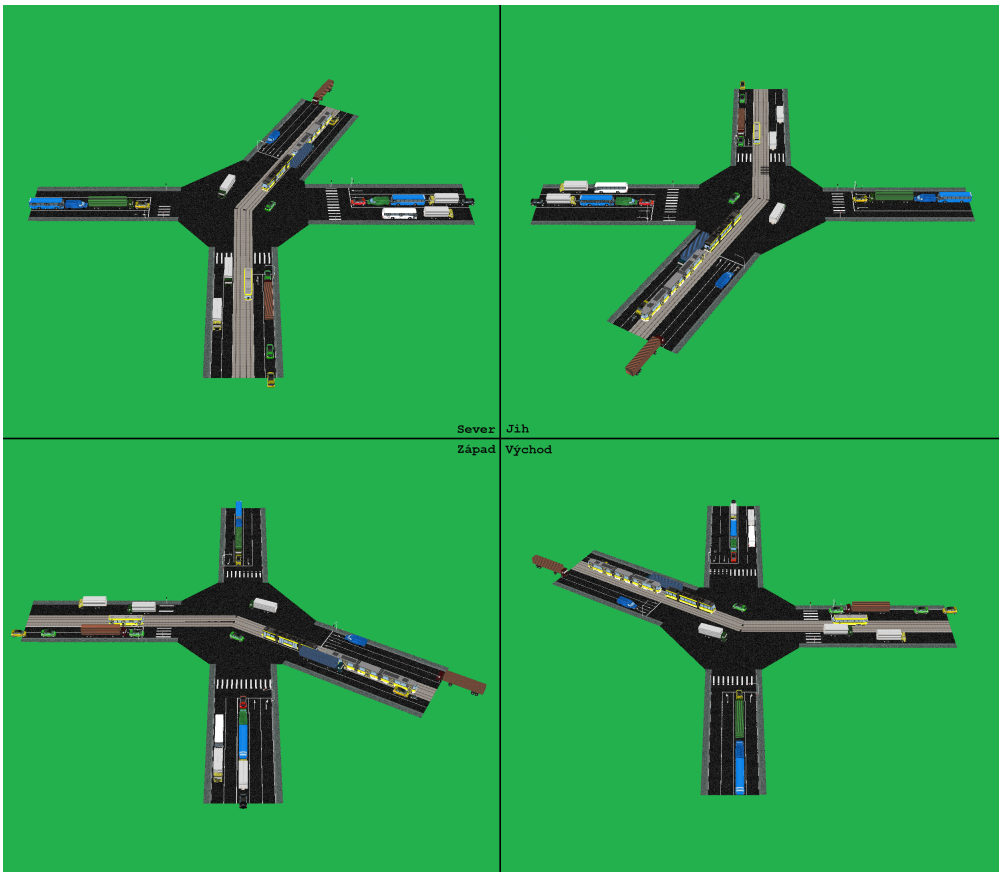
U obou křižovatek jsou přechody a značení pruhů umístěny správně.

6.1.4 Korektní zobrazení 3D modelů

Závěrečným testem je otestování reálných 3D modelů vozidel. Na obrázku 6.7 lze vidět, že se 3D modely vozidel vykreslují ze všech stran stejně.



Obrázek 6.7: Pohled na křižovatku K1 ze všech stran.



Obrázek 6.8: Pohled na křižovatku K2 ze všech stran.

Test proběhl tak, že se nechala simulace nějakou dobu běžet, aby se křižovatka naplnila vozidly. Poté se simulace pozastavila a zkontrolovalo se postavení vozidel ze čtyřech směrů[1].

6.1.5 Souhrn výsledků

Všechny provedené testy proběhly úspěšně a nebyl nalezen žádný nedostatek v umístění objektu nebo jeho vykreslení.

[1] sever, jih, západ a východ

7 Závěr

Hlavním cílem bakalářské práce byl návrh a implementace komponenty, která bude určena pro 3D vizualizaci simulace silniční křižovatky.

Vývoj komponenty by se dal rozdělit na dvě části. V první části vývoje jsem se seznámil se základy komponentového programování a s dodanou komponentou, která poskytuje navrhované komponentě veškeré informace ohledně simulace.

V druhé části vývoje jsem musel nejdříve nastudovat principy programování v Java3D a způsoby vytváření 3D objektů. Prvotním cílem bylo vytvořit vizualizaci pomocí barevných kvádrů, které budou představovat vozidla. Po dostatečném seznámení s Java3D jsem zjistil, že je možné importovat 3D objekty vytvořené v grafickém programu. Tímto způsobem jsem mohl vytvořit 3D modely vozidel reálného vzhledu.

Dále se mi podařilo vytvořit efektivní způsob přidávání dalších modelů vozidel, který nevyžaduje zásah do zdrojového kódu komponenty. Toto je důležité v případě, že se aplikace bude rozšiřovat o další typy vozidel.

Všechny testy v kapitole 6 proběhly úspěšně. Vytvořená komponenta poskytuje 3D zobrazení křižovatky, které odpovídá 2D zobrazení dodané komponentové aplikace. Práce tak zcela splňuje zadání.

Seznam zkratek

API	Application Programming Interface
GUI	Graphical User Interface
IoC	Inversion of Control
JAR	Java Archive
JDK	Java Development Kit
JRE	Java Runtime Environment
OSGi	Open Services Gateway initiative
SpringDM	Spring Dynamic Modules
XML	Extensible Markup Language

Literatura

[1] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Professional, New York, 2nd edition, 2002.

[2] C. Szyperski, D. Gruntz, and S. Murer. *Component Software – Beyond Object-Oriented Programming*. ACM Press, New York, 2000.

[3] T. Potuzak, J. Snajberk, R. Lipka, and P. Brada. *Design of a Component-based Simulation Framework for Component Testing using SpringDM*. Department of Computer Science, University of West Bohemia, Univerzitni 8, 306 14 Plzen, Czech Republic.

[4] D. Rubio. *Pro Spring Dynamic Modules for OSGiTM Service Platform*. Apress, USA, 2009.

[5] http://wiki.kiv.zcu.cz/UvodDoKomponent/ParkovisteOsgi. Zasílání zpráv v osgi, 17.1.2013.

[6] http://download.java.net/media/java3d/javadoc/1.5.2/index.html. Dokumentace java 3d 1.5.2, 22.2.2013. Sun Microsystems, Inc.

[7] Dennis J. Bouvier. *Getting Started with the Java 3D^{TM} API, Chapter 1*. Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A, 2000.

[8] Dennis J. Bouvier. *Getting Started with the Java 3D^{TM} API, Chapter 2*. Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A, 2000.

[9] Dennis J. Bouvier. *Getting Started with the Java 3D^{TM} API, Chapter 3*. Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A, 2000.

A Uživatelská dokumentace

A.1 Požadavky pro spuštění

Pro spuštění aplikace je třeba mít:

1. Nainstalovanou *Java JDK*[1] nebo *Java JRE*[2].
2. Knihovny pro *Javu3D*.

Instalační soubory pro Javu jsou na CD ve složce `instalace/Java`. Po instalaci je nutné nainstalovat knihovny Java3D, návod je popsán v následující kapitole.

A.2 Instalace knihoven Java3D

Na CD jsou ve složce `instalace/Java3D` umístěny dva instalační soubory. Podle toho jakou máme nainstalovanou verzi[3] Javy zvolíme instalaci Java3D. U 64-bitové verze je třeba provést tyto dva kroky navíc:

1. Zkopírovat všechny `.dll` knihovny z:
   `.../Program Files/Java/Java3D/1.5.1/bin`
   do `...Program Files/Java/jre7/bin`
2. Zkopírovat všechny `.jar` knihovny z:
   `.../Program Files/Java/Java3D/1.5.1/lib/ext`
   do `.../Program Files/Java/jre7/lib/ext`

A.3 Instalace workspace

Přiložený *Eclipse* a *workspace* na CD ve složce `instalace` rozbalíme na pevný disk. Před spuštěním *Eclipse* je třeba upravit soubor eclipse.ini, který se nachází v kořenovém adresáři programu. Na řádce:

[1] Java Development Kit
[2] Java Runtime Environment
[3] 32-bit nebo 64-bit

```
-vm
C:\Program Files (x86)\Java\jdk1.7.0_13\bin
```

je třeba upravit správnou cestu k Javě. Poté by již neměl být problém prostředí *Eclipse* spustit.

Po spuštění vybereme workspace a provedeme další úpravu. V *Package Explorer* nalezneme projekt *TrafficCrossroad3D* a zvolíme *Properties*. Poté se nám otevře dialogové okno, kde zvolíme *Java Build Path -> Libraries*. V seznamu knihoven uvidíme *j3dcore.jar*, *j3dutils.jar* a *vecmath.jar*, u kterých musíme pomocí tlačítka *Edit* upravit cestu ke knihovnám.

Poslední úpravu provedeme v projektu *TrafficCrossroad*, kde ve složce `/TrafficCrossroad/META-INF/resources/config/` otevřeme konfigurační soubor `TrafficCrossroadConfig_cs.properties`. Na řádce 24[4] upravíme správnou cestu k workspace.

A.4 Spuštění aplikace

Pro spuštění aplikace stačí otevřít kontextové menu u projektu *TrafficCrossroad3D* a zvolit `Run As -> OSGi Framework`.

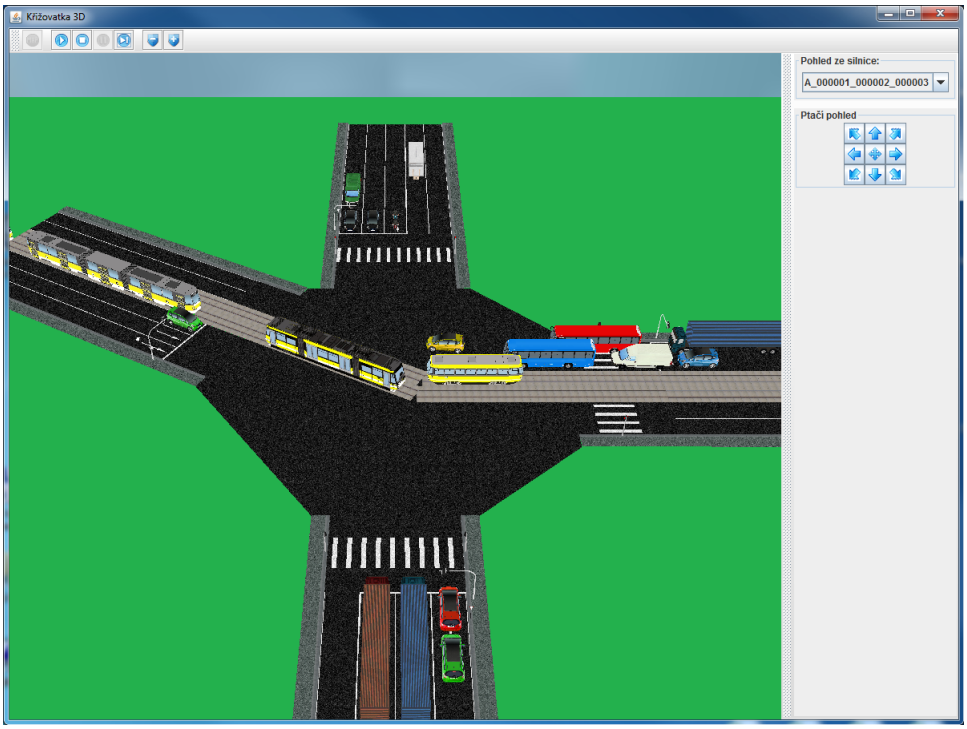
A.5 Ovládání aplikace

Ovládání aplikace probíhá pomocí tlačítek v GUI komponenty, viz obrázek A.1. Před spuštěním simulace je třeba zvolit křižovatku.

V horním panelu jsou tlačítka pro ovládání simulace v tomto pořadí: výběr křižovatky, spuštění, zastavení, pozastavení, krokování, snížení rychlosti a zvýšení rychlosti simulace.

V bočním panelu je rozbalovací seznam s rameny křižovatek, který slouží pro změnu pohledu v daném rameni. Dále jsou k dispozici směrová tlačítka, pomocí kterých se dá měnit pohled ze všech světových stran.

[4] `environment.trafficCrossroadsFile`



Obrázek A.1: GUI komponenty.

Pokud klikneme do panelu s vizualizací, můžeme pohled ovládat pomocí klávesnice. Zde je seznam kláves:

Klávesa	**Akce**	**Akce s klávesou Alt**
←	rotace doleva	posun doleva
→	rotace doprava	posun doprava
↑	pohyb dopředu	
↓	pohyb dozadu	
Pg Up	rotace nahoru	posun nahoru
Pg Dn	rotace dolů	posun dolů

B Import 3D modelů vozidel

Všechny 3D modely jsou uloženy ve složce projektu `/TrafficCrossroad3D/-META-INF/resources/objects` a všechny textury jsou ve složce `/Traffic-Crossroad3D/META-INF/resources/textures`.

Ve výpisu B.1 lze vidět jak vypadá XML soubor, který se využívá pro uložení cest ke 3D modelům a texturám všech vozidel. Soubor je uložen ve složce projektu `/TrafficCrossroad3D/META-INF/resources/vehicles/vehicle-s.xml`.

U každého vozidla (element `vehicle`) je třeba zadat parametr `type`, který určuje typ vozidla. U objektů a textur je třeba zadat parametr `name`, který určuje cestu k souboru.

```
<vehicles>
  <vehicle type="PASSANGER_VEHICLE">
    <objectFile name="vehicles/WT/Vehicle5mWT.obj"/>
    <textureFile name="vehicles/vehicle5m/vehicle5m_red.jpg"/>
    <textureFile name="vehicles/vehicle5m/vehicle5m_blue.jpg"/>
    <textureFile name="vehicles/vehicle5m/vehicle5m_green.jpg"/>
  </vehicle>
  <vehicle type="TRAM_MEDIUM">
    <objectFile name="vehicles/WT/tram21mWT.obj"/>
    <textureFile name="vehicles/tram21m/tram21m_yellow.jpg"/>
  </vehicle>
  <vehicle type="TRAMS_LONG">
    <objectFile name="vehicles/WT/tram32mWT.obj"/>
    <textureFile name="vehicles/tram32m/tram32m_yellow.jpg"/>
  </vehicle>
</vehicles>
```

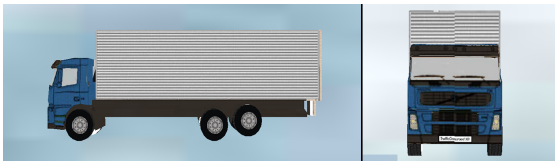
Výpis B.1: Ukázka struktury XML souboru pro import vozidel.

C Katalog 3D modelů vozidel

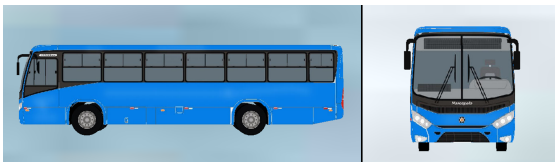
Obrázek C.1: Typ vozidla MOTORCYCLE.

Obrázek C.2: Typ vozidla PASSANGER_VEHICLE.

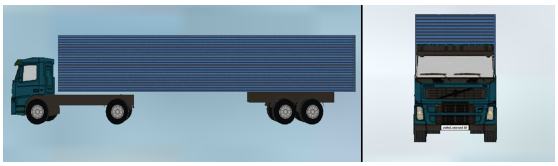
Obrázek C.3: Typ vozidla VAN.



Obrázek C.4: Typ vozidla MINIBUS.

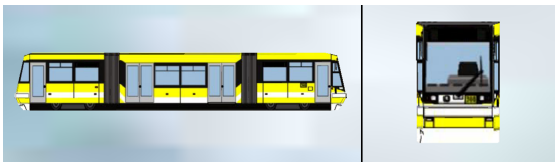


Obrázek C.5: Typ vozidla BUS.



Obrázek C.6: Typ vozidla LORRY.

Obrázek C.7: Typ vozidla TRAM_SHORT.



Obrázek C.8: Typ vozidla TRAM_MEDIUM.

Obrázek C.9: Typ vozidla TRAM_LONG.