ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra kybernetiky

3D zobrazení bytů / budov pro navigaci a simulaci jejich ovládání

Bakalářská práce

Studijní program: Kybernetika a robotika
Studijní obor: Robotika

Vedoucí práce: Ing. Petr Novák, Ph.D.

Martin Plaček

Praha 2012

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Martin P l a č e k

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: 3D zobrazení bytů / budov pro navigaci a simulaci jejich ovládání

Pokyny pro vypracování:

Práce je tematicky zaměřena na projekt, jehož cílem je snadná a přehledná lokalizace pacientů, personálu a případně vybavení a to zejména v nemocnicích.

1. Seznamte se s možnostmi 3D zobrazení bytů / budov (a podobných objektů) za účelem navigace na cílovou pozici (zejména z pohledu jejich obsluhy nebo dozoru). Dále se zaměřte na možnosti informačního zobrazení požadovaných objektů a osob pro jejich snadnou a rychlou lokalizaci. Rovněž prostudujte možnosti vizualizace některých běžných objektů (okna, dveře, TV, ...) za účelem simulace jejich činnosti / ovládání.
2. Navrhněte vhodné a přehledné 3D zobrazení bytů / budov pro účely zmíněné v bodě 1). Zaměřte se zejména na snadnost vytvoření podkladů (tj. mapy, objektů) ze kterých je budoucí 3D model konstruován. Rovněž navrhněte vhodné rozhraní tohoto modelu pro jeho univerzální využití v dalších budoucích projektech.
3. Implementujte navržené 3D zobrazení jako desktopovou nebo webovou aplikaci s vhodným ovládáním. Rovněž vytvořte základní testovací aplikaci sloužící zejména pro demonstraci schopností vytvořeného 3D zobrazení.

Seznam odborné literatury:

[1] Internetové stránky a propagační materiály (i firemní) týkající se tématu práce
[2] Knížky a manuály k potřebným / využitým SW nástrojům (.NET, C#, WPF, Silverlight, XMA, ...)
[3] Další podklady dodá vedoucí práce

Vedoucí bakalářské práce: Ing. Petr Novák, Ph.D.

Platnost zadání: do konce zimního semestru 2012/2013

L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 9. 1. 2012

Poděkování

Na tomto místě bych rád poděkoval především vedoucímu práce Ing. Petrovi Novákovi, Ph.D. za cenné rady a připomínky v průběhu zpracování této bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 24.5.2012 .....

.....
Podpis autora práce

Abstrakt

Tato práce se zabývá návrhem a implementací programové komponenty, která bude vytvářet snadno konfigurovatelný a dostatečně interaktivní trojrozměrný model od běžného bytu až po vícepodlažní budovu. Tento model bude sloužit především pro lokalizaci osob pohybujících se v bytu / budově a dále také pro ovládání některých objektů, které se v budově nacházejí.
Model komponenta vytváří ze snadno editovatelných bitmapových obrázků, ve kterých jsou v podobě mapy definovány objekty, ze kterých se skládají jednotlivá patra budovy. Jména a další vlastnosti, které nelze graficky (v podobě obrázku) snadno vyjádřit, jsou do komponenty načítány z textového souboru.

Abstract

This work concerns with design and implementation of a software component that creates easily configurable and interactive three-dimensional model of a flat or a house. This model will serve especially for localization of persons moving inside the flat / house and also for control of some objects located inside the building.
The component creates 3D model from easily editable bitmap pictures that contain a map of all objects creating one floor of the building. Each floor of the building has its own bitmap picture. Names and other properties, which cannot be expressed in a picture, are loaded from text file directly into the component.

Obsah

Seznam obrázků

1 Úvod

V současné době již sice existují projekty (SW) pro různá 3D zobrazení budov neboli objektů. Ty jsou však v podstatě dvou typů. První typ zahrnuje SW nástroje jako je AutoCad, 3D Studio a podobné. Ty jsou určeny pouze pro v podstatě dokonalé statické zobrazení 3D scény a neposkytují žádný stupeň interakce s uživatelem. Jedná se převážně o demonstrační zobrazení. Druhý typ zahrnuje sice již 3D zobrazení poskytující určitý stupeň interakce, ale nevýhodou je jejich tvorba přímo na míru danému použití. Zde jde převážně o snahu při vzdáleném ovládání některých objektů. Tyto SW výstupy však nejsou nikterak rozšiřitelné z pohledu uživatele a již vůbec vhodné pro experimentální nebo dokonce výzkumné účely.

Tato práce se tedy v podstatě snaží o vytvoření dostatečně univerzálního a snadno konfigurovatelného dostatečně interaktivního 3D modelu od libovolného běžného bytu až po více podlažní objekt. Bakalářská práce je však samozřejmě velmi ovlivněna projektem s názvem BOS (2011-2013) jehož je vedoucí mé práce spoluřešitel. Jde o projekt pro sledování pohybu nejen pacientů, ale i personálu a to nejen v malém domácím, ale i ve větším nemocničním prostředí.
V malém domácím prostředí jde v podstatě o vzdálený dohled nad převážně staršími osobami, které se v tomto prostředí libovolně pohybují. Jedním úkolem projektu je vhodná vizualizace polohy (případně záznamu pohybu) dané osoby. Nejlépe tedy formou webové aplikace a tím umožnit získat informace o aktuální poloze sledované osoby z libovolného místa. Jde tedy o zobrazení jednotlivých osob, například rozlišených barvou podle jejich typu (rezident, ošetřovatel). Toto je tedy jedním z hlavních požadavků na tuto bakalářskou práci. Druhý cíl projektu BOS (v této bakalářské přímo neobsaženým) spočívá v rozpoznání určitého průběhu / vzoru chování této osoby pro možnou detekci nestandardních situací. Takto rozpoznané „nestandartní“ situace jako například, dlouhé setrvání na netypickém místě (například WC, koupelna) musí být však rovněž vhodně indikováno. Z tohoto důvodu se tato bakalářská práce současně věnuje možnosti zobrazení různých varovných nebo přímo nebezpečných stavů jako je například speciální (barevná) indikace konkrétní osoby.
Ve větším, tedy více patrovém, komplexu je situace s dostatečnou lokalizací požadovaných osob poněkud složitější. Osoba se může nacházet nejen ve shluku několika dalších osob, ale rovněž v libovolném patře nebo dokonce průběžně přecházet z patra do patra. V tomto případě nemusí být barevná indikace hledané osoby dostačující a je nutno přistoupit k poněkud globálnějšímu zobrazení aktuální polohy například formou různých ukazatelů.

Dalším z projektů kde se počítá s využitím výstupů této práce, je snaha o tvorbu (ne zcela) simulovaného prostředí inteligentního bytu / domácnosti. Zde není 3D model určen pouze pro zobrazení aktuálního stavu, ale rovněž jako interakční vstupní (webový) terminál pro vzdálené ovládání některého inteligentního bytu / domácnosti. Je tedy nutné aby 3D model obsahoval možnost zobrazení nejen různých typů běžného stacionárního domácího vybavení (stoly, židle, …), ale současně základní komponenty jak pro zobrazení, tak i ovládání stavu tohoto prostředí (dveře, okna, televize …).

V poslední řadě se využití výstupů této práce uvažuje rovněž v projektu SPES (2011-2014), který je zaměřen nejen na dohled, ale současně i na pomoc osobám s různým handicapem v domácím prostředí. Ve většině případů se jedná o vozíčkáře nebo přímo ležící osoby. V mnoha případech tyto osoby nejsou schopny svépomocí otevřít okno, zapnout televizi nebo dokonce otevřít dveře příchozí návštěvě. Z tohoto důvodu je nutno tyto akce vykonávat z určitého „velínu / dozoru“, kde pověřená osoba právě pomocí například takovéhoto webového 3D modelu je schopna ve vzdáleném bytě / domácnosti uskutečnit požadovanou akci rezidenta. Komunikace se vzdálenou osobou je zajištěna pomocí různých mobilních sítí nebo interkonů. Pro tento projekt je tedy velmi vhodné zobrazení vícepatrových budov obsahujících samostatné byty / podlaží, protože se velmi často jedná o vice-bytové komplexy sdružující takovéto osoby (jedna osoba – jeden malý byt).

Dostatečně samostatnou částí, v podstatě nezávislou na všech popsaných (řešených) projektech, je vhodná komunikace aplikace pro 3D zobrazení scény se serverem, nebo jiným libovolným uzlem, který bude nejen poskytovat aktuální data pro zobrazení (polohy osob, stav komponent, …), ale rovněž bude přijímat akce uživatele (otevřít / zavřít dveře, …). Návrh takového komunikačního kanálu s dostatečně obecným protokolem není nikterak snadný.
Jelikož se jedná pouze o bakalářskou práci, tak nelze všechny dříve uvedené poznatky a požadavky zcela realizovat. Rovněž v průběhu řešení zmíněných projektů (časové období delší než čas vymezený této práci) vznikají stále další a další požadavky. Z těchto důvodů se práce zaměřuje na vytvoření určitého prototypu, avšak samozřejmě plně funkčního. Ten obsahuje zejména ty základní a běžné komponenty pro dostatečnou činnost (dveře, oka, výtah – nikoli již však jezdící schody) a rovněž základní komunikaci s externím uzlem (získání polohy osob, zaslání akcí uživatele). Další požadavky a komponenty lze snadno přidat pouze včleněním nových schopností / objektů nebo tzv. poděděním od stávajících.

2 Cíle

Cílem práce je navrhnout vhodné a přehledné zobrazení 3D bytu / budovy pro účely lokalizace a navigace osob, které se po bytu / budově pohybují. Aby bylo 3D zobrazení bližší realitě, budou do něj také zahrnuty některé běžné objekty, které budou schopné simulace činnosti / ovládání.
Během návrhu je nutné počítat s tím, že výslednou aplikaci bude používat v podstatě dozor těchto osob, proto je potřeba 3D zobrazení tomu vhodně přizpůsobit a případně přidat další komponenty s informacemi o požadovaných objektech nebo osobách.
Součástí návrhu je také vytvoření podkladů, ze kterých se bude 3D zobrazení konstruovat. U podkladů je kladen důraz především na jejich snadné vytváření. Vhodným podkladem může být například obrázek s mapou půdorysu objektů.
Pro 3D zobrazení je také potřeba navrhnout vhodné univerzální rozhraní, které umožní model využít i v dalších budoucích projektech.

Výstupem této práce bude implementace navrženého 3D zobrazení jako desktopové nebo webové aplikace a vytvoření základní testovací aplikace pro demonstraci schopností vytvořeného 3D zobrazení.

V práci se nevěnuji tvorbě různorodých modelů objektů s detailními texturami a například odrazy světla. 3D zobrazení bude mít hlavně informační charakter a k tomuto účelu budou modely také navrženy. Rovněž se v práci nevěnuji optimalizaci komunikace a výměně dat mezi aplikací pro 3D zobrazením a systémem skutečné budovy. V aplikaci jsou implementovány pouze základní komunikační prvky, které slouží převážně pro demonstraci možností prototypové aplikace.

3 Návrh řešení

3D scéna
Aplikace bude zobrazovat model celé budovu s jednotlivými patry ležícími nad sebou. Zdi budou ve všech patrech pouze do poloviny své výšky, aby bylo alespoň částečně vidět dovnitř budovy, tedy do příslušných pater. Mezi zobrazením pater se bude možné přepínat, tak aby patro, o které máme zájem, bylo dobře viditelné včetně objektů a osob, které se v požadovaném patře vyskytují.

Grafické objekty, ze kterých se bude 3D zobrazení skládat, budou celkem tří typů – statické, akční a doplňkové.
Statické objekty budou představovat části budovy / bytu, se kterými není možné interagovat a slouží pouze k vytvoření realističtějšího modelu skutečné budovy / bytu. Mezi statické objekty budou patřit například modely stolů, židlí, stěn nebo podlahy.
S akčními objekty bude v aplikaci možné interagovat a tedy tyto objekty ovládat a měnit jejich stav. Aby různé stavy byly ve 3D zobrazení patrné, budou modely akčních objektů měnit svojí polohu / barvu / tvar, tak aby názorně představovali stav ve kterém se nacházejí. Například model dveří se bude v pantech otáčet a tím představovat zavřené nebo otevřené dveře, nebo model televize bude měnit barvu obrazovky podle toho, jestli je televize zapnutá nebo vypnutá.
Doplňkové objekty nebudou představovat modely objektů z reálného světa, ale budou ve 3D zobrazení sloužit jako pomocné značky / ukazatele / barevné plochy, které budou uživateli aplikace pomáhat s ovládáním a orientací ve 3D modelu budovy / bytu. Doplňkovým objektem jsou například zóny zájmu nebo navigační ukazatel.

Modely různých typů akčních objektů budou mít různé způsoby pro změnu jejich stavu, aby ovládání odpovídalo možnostem skutečného objektu, který model reprezentuje. K dispozici budou ovládací prvky pro nastavování tří typů hodnot – dvoustavové (zapnuto/vypnuto, otevřeno/zavřeno), procenta (otevřenost okna) a číslo (nastavení teploty v bazénu).
Možnosti ovládání pro jednotlivé objekty se nastaví až při běhu aplikace podle informací, které poskytne server, který se stará o skutečnou budovu. Například když je okno možné pouze otevřít nebo zavřít, tak se v aplikaci použije dvoustavový ovládací prvek nebo pro jiné okno, které je možné otevřít v rozmezí od zavření až do úplného otevření se použije ovládací prvek procenta.

Všechny grafické objekty se budou skládat pouze z kvádrů s různou šířkou, délkou a výškou. Takto vytvořené modely objektů sice nebudou dokonale identické s předlohou z reálného světa, ale budou mít jiné přednosti, které budou pro výslednou aplikaci značně významné.
Především je nutno jmenovat menší náročnost na výkon počítače na kterém bude aplikace spuštěna. Pokud by modely byly dokonalé kopie skutečných objektů, museli by být tvořeny z velkého počtu ploch a vrcholů a tím pádem by i jejich vykreslování na obrazovku bylo výpočetně náročnější. Více ploch a vrcholů by také znamenalo větší datový objem pro jednotlivé modely, což by u webové aplikace mohl být značný problém při jejím spouštění v místech se špatným připojením k internetu (například na chatě v horách s připojením přes mobilní telefon).

Ovládání 3D zobrazení bude pomocí myši a klávesnice současně. Myší se bude otáčet s modelem, aby bylo možné se na budovu podívat z ideálního úhlu, a klávesnicí se bude modelem pohybovat. Modely akčních objektů bude možné ovládat klikáním myši do prostoru jejich zobrazení, nebo pomocí ovládacích prvků umístěných v seznamu všech akčních objektů.

Lokalizace osob a vybavení budovy

Pro lokalizaci osob a objektů bude k dispozici seznam, ve kterém budou uvedeny všechny akční objekty a přítomné osoby, které se v budově vyskytují. U každé položky bude ovládací prvek, kterým se daný objekt nebo člověk najde a ve 3D zobrazení se vhodně graficky zvýrazní. K tomu bude sloužit tzv. navigační ukazatel, což je v podstatě výrazný 3D grafický objekt, který se zobrazí na místě hledaného objektu nebo na toto místo bude ukazovat.

Jako navigační ukazatel jsem vybral tří přímky, které se protínají v místě, kde se nachází hledaný objekt. Přímky jsou navzájem kolmé a vzhledově připomínají osy pravoúhlého souřadného systému s počátkem v místě hledaného objektu.

Alternativou pro tento navigační ukazatel by mohla být například šipka, která by se ukázala nad objektem a ukazovala by něj, nebo barevně výrazná „aura“, která by se ukázala okolo objektu. Tyto ukazatele, by ale nemuseli být moc vhodné například pro rozsáhle budovy, protože při oddálení kamery tak, aby byla vidět celá budova by značka byla příliš malá a splývala by s okolními objekty. Navržené, tři protínající se, přímky tímto neduhem netrpým, jelikož procházejí celou zobrazovanou 3D scénou a místo jejich průniku je snadno viditelné i s oddálenou kamerou.

Pro zjednodušení dohledu nad obyvateli bytu / budovy budou do aplikace implementovány tzv. zóny zájmu. Tyto zóny v podstatě představují určité oblasti budovy, které jsou nějakým způsobem důležité.

Například v nemocnici bude oblast chodby mezi pokoji pacientů označena jako zóna zájmu. Pokud některý z pacientů vyjde z pokoje na chodbu, tak se dostane do této zóny. Aplikace tuto skutečnost rozpozná a pomocí textové zprávy nebo grafických prvků o tom bude informovat dozor, který se o pacienty stará.

Podkladové materiály

Podkladové materiály pro tvorbu 3D zobrazení budou dvojího typu – nekomprimovaný obrázek (PNG) a textový soubor (XML). Inspirací při výběru podkladových materiálů mi byla diplomová práce A. Pošusty [5], ve které se taktéž věnuje 3D grafice tvořené z obrázkových podkladových materálů.

Obrázek bude v podstatě představovat mapu všech objektů, které se v patře nacházejí a rozmístění jenotlivých objektů v obrázku bude odpovídat rozmístění v reálné budově. Objekty se budou rozlišovat barevně a graficky. Barvy budou sloužit k určení typu objektu (například stůl bude mít hnědou barvu a okno tmavě modrou barvu) a grafická úprava bude sloužit pro větší přehlednost v obrázku a také k definování velikostí objektů.

Objekty, které v reálném světě můžou mít různé velikosti, jako například stoly, okna nebo bazény, budou do obrázku kresleny jako obdélníky vyplněné barvou příslušící k danému objektu (například stůl bude kreslen jako hnědý obdélník / čtverec). Velikost tohoto obdélníku bude v obrázku odpovídat velikosti reálného objektu.

Objekty, které se v reálném světe vyskytují v jedné typické velikosti, nebo jejich velikost není zásadní a jejich 3D model se může trochu odlišovat, jsou do obrázku kresleny jako značky ve tvaru písmene „T“. Na rozdíl od značení pomocí obdélníku může značka navíc definovat i orientaci objektu a proto jsou tímto způsobem označeny například televize nebo židle.
Pokud by bylo potřeba vytvářet model objektu, který může mít různé velikosti a zároveň záleží na jeho orientaci, kreslil by se do obrázku jako kombinace obou výše zmíněných typů značení. Velikost a typ by se definovaly jako velikost a barva obdélníku a orientace by se definovala jako otočení značky. Tato značka by se nakreslila dovnitř obdélníku, aby bylo patrné, ke kterému objektu patří.

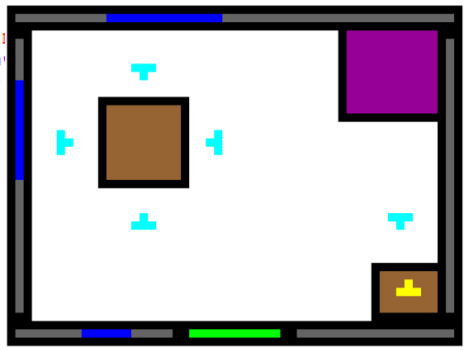
Všechny objekty kreslené do obrázku, které nejsou typu značka, musejí být ve tvaru obdélníku, nebo čtverce a jejich strany musejí být rovnoběžné s hranami obrázku. Tyto požadavky vycházejí ze způsobu dekódování obrázku, kdy je použit algoritmus, který v podstatě vyhledává pouze rohy nakreslených objektů a pomocí nich následně vytváří správně velké i umístěné 3D modely.

Některé vlastnosti objektů, nebo přímo celé objekty není možné efektivně definovat přímo do obrázku, proto je zaveden druhý typ podkladového materiálu – textový soubor.
V tomto souboru se budou definovat zóny zájmu a oblasti pro světelné zdroje. Oba „objekty“ budou typicky rozsáhlé oblasti a v obrázku by zabírali příliš velkou plochu, proto se budou definovat v textové podobě.
Jako text se také budou definovat jména objektů. Pro určení, které jméno patří ke kterému objektu, budou v textu uvedeny vždy dvojce jméno-souřadnice, kde souřadnice udává pozici, kde je objekt nakreslen v obrázku a tedy ke kterému objektu jméno patří.

Na obrázku [Obrázek 1] je ukázka, jak budou podkladové soubory vypadat.

Implementace
Pro imlementaci jsem vybral technologii Silverlight [1],[7],[9], která umožní, aby aplikace byla spustitelná ve webovém prohlížeči a nebylo nutné ji instalovat do všech zařízení, ze kterých budeme chtít budovu / byt sledovat a ovládat.

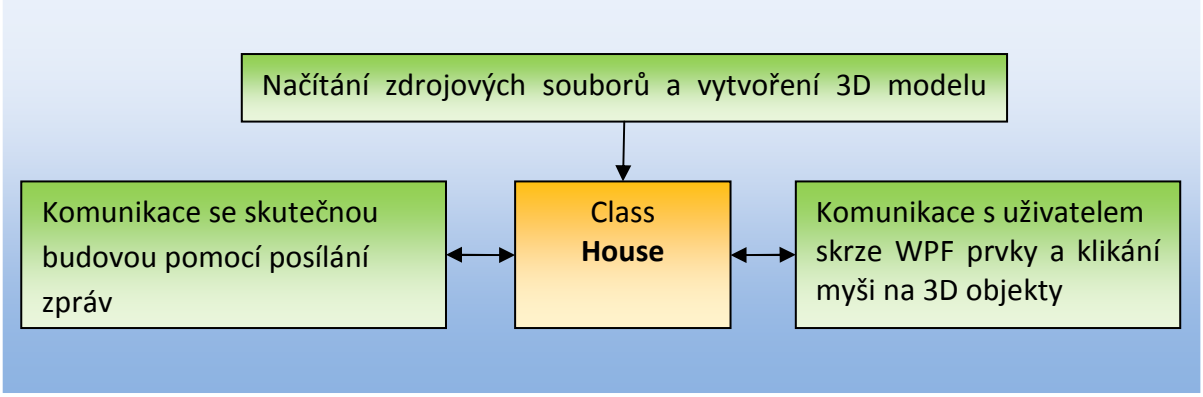
```
<?xml version="1.0" encoding="utf-8"?>
<ParametersFromFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <light>
    <LightDefinition Name="chodba1" UpL_X="29" UpL_Y="79" DowR_X="61" DowR_Y="141" />
    <LightDefinition Name="chodba2" UpL_X="29" UpL_Y="141" DowR_X="61" DowR_Y="175" />
  </light>
  <zones>
    <ZoneDefinition Name="chodba" UpL_X="26" UpL_Y="76" DowR_X="63" I
    <ZoneDefinition Name="ložnice" UpL_X="63" UpL_Y="76" DowR_X="120
    <ZoneDefinition Name="pokoj" UpL_X="26" UpL_Y="175" DowR_X="120"
  </zones>
  <pointNames>
    <PointName Name="vchodové" X="40" Y="76" />
    <PointName Name="na chodbě" X="100" Y="190" />
    <PointName Name="na chodbě na stole" X="105" Y="203" />
    <PointName Name="v komoře" X="99" Y="167" />
    <PointName Name="v kuchyni" X="73" Y="245" />
    <PointName Name="v pokoji" X="26" Y="200" />
    <PointName Name="bazén" X="160" Y="240" />
  </pointNames>
</ParametersFromFile>
```



Obrázek 1 – XML soubor definující světla, zóny zájmu a jména objektů (vlevo), obrázek patra s barevně odlišenými objekty (vpravo)

4 Popis řešení

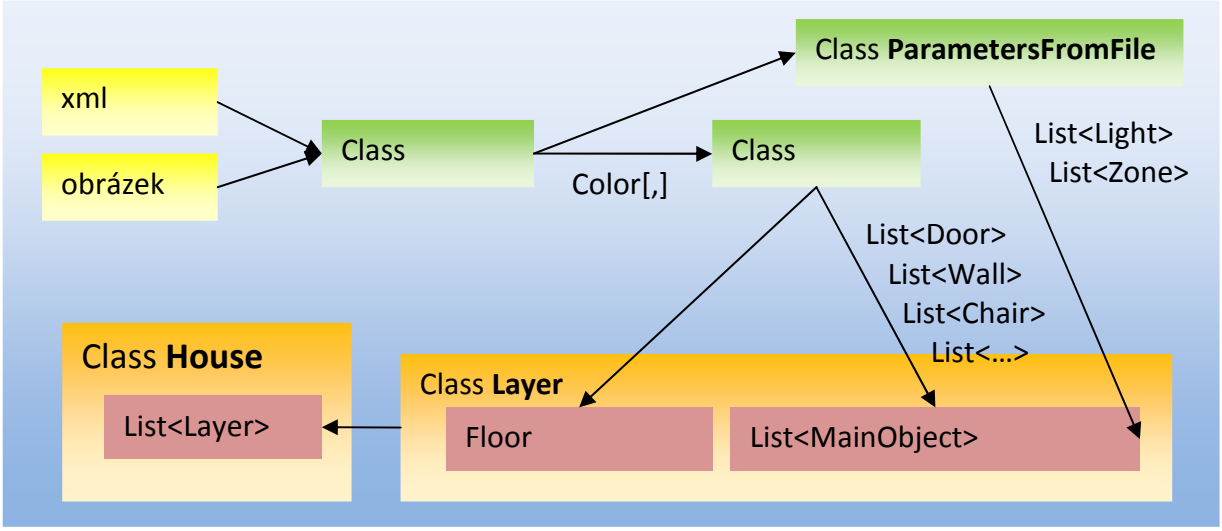
V následujícím textu se věnuji vnitřní struktuře aplikace a popisu jednotlivých tříd. Pro větší přehlednost jsem celou implementaci rozdělil do tří logických bloků, které jsou naznačeny na obrázku [Obrázek 2]. Každému bloku je věnována jedna podkapitola, ve které jsou detailně popsány třídy a komponenty, které do daného bloku patří.



Obrázek 2 – Rozdělení aplikace do tří logických bloků

4.1 Model budovy

Celý model budovy je tvořen z pater, které se po jednom načítají ze vstupních souborů. Každé patro je představováno instancí třídy **Layer** a všechna patra jsou uložena v *List*u ve třídě **House**, jak je znázorněno ve schématu [Obrázek 3].
Vytváření patra je uskutečněno ve dvou krocích. Nejdříve se načte obrázek a podle něj se vytvoří modely potřebných vybavení, stěny a podlaha. Pokud se jedná o patro v přízemí je navíc vytvořena současně i zahrada. Ve druhém kroku se načte XML soubor, podle kterého se modelům vybavení přiřadí jména a vytvoří se modely světel a zón zájmu.



Obrázek 3 – Schéma vytváření modelů objektů z podkladovým souborů

Třída **LoadFile**

Tato třída je statická a pouze poskytuje metody pro načítání bitmap a XML souborů.

Bitmapy je možné načítat do dvou typů proměnných, do dvojrozměrného pole tříd **Color** nebo do **Texture2D**. Pole tříd **Color** má velikost totožnou s velikostí načtené bitmapy a každá buňka obsahuje barevnou informaci o jednom pixelu obrázku. Toto pole barev se dále předá do třídy **Patterns** kde se obrázek převede na trojrozměrné modely objektů. Načtení do **Texture2D** se používá pro získání textur modelů.

Z načítaných XML souborů se pomocí *xmlSerializeru* přímo vytvářejí instance tříd **HouseDefinition** a **ParametersFromFile**.
Třída **HouseDefinition** obsahuje seznamy jmen zdrojových souborů a doplňující informace o budově. Třída **ParametersFromFile** obsahuje tři seznamy s definicemi světel, zón zájmů a jmen ovladatelných objektů.

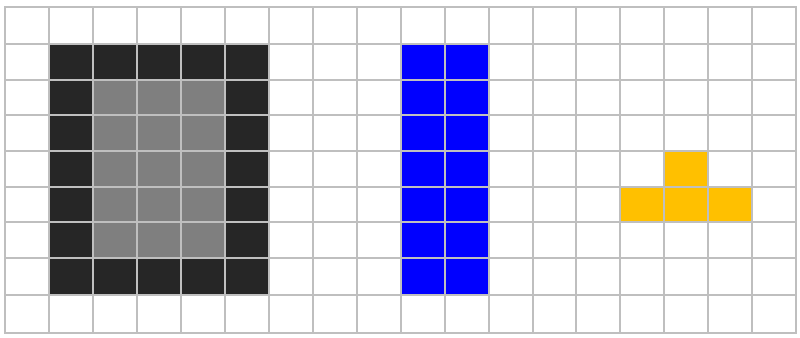
Třída **ParametersFromFile**

Tato třída obsahuje pouze tři seznamy. První dva seznamy obsahují instance tříd **LightDefiniton** a **ZoneDefinition**, ve kterých jsou definované pozice, rozměry a jména světel a zón.
Třetí seznam obsahuje instance třídy **PointName**, které obsahují 2D souřadnice bodu a jméno, které k bodu patří. Po vytvoření všech 3D objektů se pro každý objekt hledá v tomto seznamu bod, jehož souřadnice odpovídá pozici objektu. Pokud je takový bod nalezen, je objekt pojmenován podle jména jaké má bod.

Třída **Patterns**

Vstupem do této třídy je dvojrozměrné pole tříd **Color**, které obsahuje obrázek s podklady pro vytvoření 3D modelu budovy. Přímo v konstruktoru se provede základní analýza obrázku a připraví se data pro rozpoznávání objektů. Poté je možné volat jednotlivé metody *GetWall*, *GetDoor*, *GetChair*, atd. a získat seznamy instancí tříd daného objektu.

V obrázku se můžou vyskytovat 3 typy označení objektů – s okrajem, bez okraje a značka. Jak jednotlivé typy vypadají a jaký je mezi nimi rozdíl je patrné z obrázku [Obrázek 4].



Obrázek 4 – Typy označení objektů (zleva) typ s okrajem, typ bez okraje, typ značka

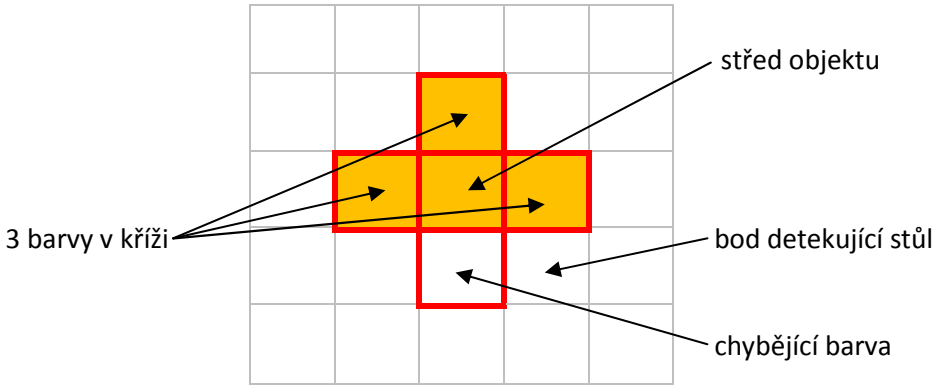
Typ značka

Tímto typem jsou označeny pouze objekty, které mají jednu standardní velikost a je u nich potřeba definovat orientaci. Toto označení používají židle a televize.

Hledání těchto objektů má dvě fáze, nejprve je nalezena pozice umístění objektu a poté je určena orientace.

Postupně se prochází celé pole barev, a pokud se narazí na barvu, která patří hledanému objektu, zkontrolují se barvy sousedících buněk. Pokud právě 3 buňky v ramenech kříže (jak je vyznačeno na obrázku [Obrázek 5]) obsahují stejnou barvu, je nalezený bod označen jako pozice objektu. Orientace objektu se poté určí z chybějící barvy v ramenu kříže. Pokud barva chybí dole, je objekt orientován nahoru. Obdobně pro zbylé tři směry.

U objektu televize je navíc rozlišováno, zda se nachází na stole nebo na podlaze. Určení se provede pomocí bodu vpravo dole od středového bodu. Je-li barva bodu totožná s barvou stolu, je televize umístěna na stůl, v opačném případě je umístěna na podlahu.

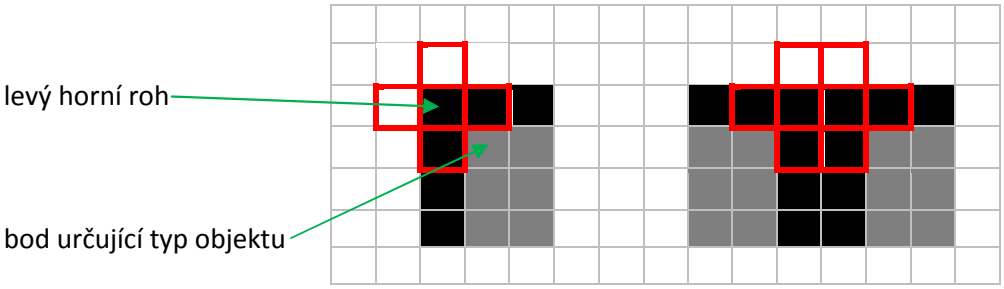


Obrázek 5 – Princip využití kříže (červeně) pro vyhledávání objektů typu značka

Typ s okrajem

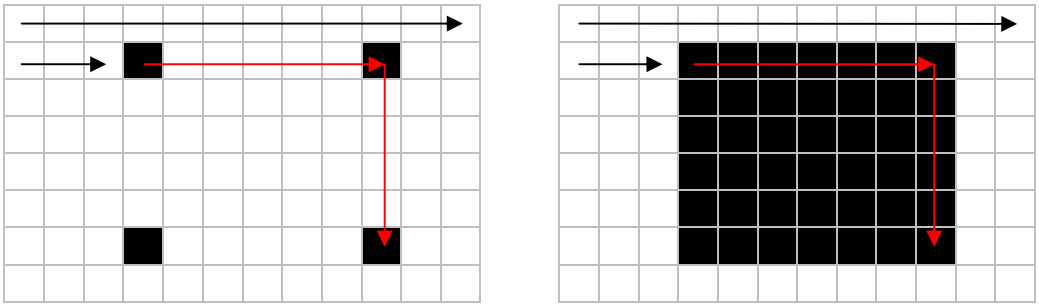
Typ s okrajem je použit pro objekty, které můžou mít libovolnou velikost a nezáleží na jejich orientaci. Tímto typem jsou označeny stěny, stoly, výtah a dveře.

Objekty typu s okrajem, jsou vyhledávány pomocí detekce rohů. Pokud je při procházení pole barev nalezen černý bod, otestuje se kříž kolem tohoto bodu. Obsahují-li ramena kříže pouze černou nebo bílou barvu, je bod označen jako roh. Tento způsob vyhledávání pracuje polehlivě, i když na sebe objekty těsně navazují, jak je zobrazeno na obrázku [Obrázek 6]. Typ objektu se následně určí z barvy bodu, který leží vpravo dole od levého horního rohu.



Obrázek 6 – Využití kříže (červeně) pro vyhledávání rohů objektů, dva objekty typu s okrajem, které na sebe těsně navazují (vpravo)

Pro zjednodušení vyhledávání objektů jsou všechny rohy nalezeny současně a je vytvořena maska v podobě dvojrozměrného pole hodnot *boolean*. Pole má shodné rozměry jako vstupní obrázek a tam, kde byl nalezen roh má vytvářené pole hodnotu *true*, jinak *false*.
Pole se prochází po řádcích shora dolů. První nalezená hodnota *true* představuje levý horní roh objektu. Na stejném řádku se pokračuje dále pro nelezení pravého horního rohu. Od tohoto bodu se dále pokračuje ve sloupci dolů a nalezne se pravý dolní roh. Ze vzdáleností těchto bodů se získá velikost objektu. Všechny rohy k danému objektu jsou poté vymazány, aby nedošlo k jejich přimíchání do jiného objektu. Vzhled pole typů *boolean* a princip jeho procházení je zobrazeno na obrázku [Obrázek 7] vlevo.



Obrázek 7 – Procházení 2D pole booleanů, (šipky) směr procházení, (černá) hodnota true, (bílá) hodnota false, (vlevo) typ s okrajem, (vpravo) typ bez okraje

Typ bez okraje
Tento typ jsem zavedl pouze s důvodu, aby bylo možné také vytvářet o rozměru pouze 1 pixel. U typu s okrajem je minimální rozměr 3 pixely (2 pixely pro okraje a 1 pixel pro výplň objektu), což byl problém je pro označení například oken. Pozice oken se kreslí přímo na stěnu, a pokud by okno bylo tlusté 3 pixely, musela by stěna být tlustá minimálně 5 pixelů.
Objekty, které jsou typu bez okraje, se hledají nejsnadněji. Projde se celé pole barev a vytvoří se dvojrozměrná maska typu *boolean*. V místě, kde barva bodu odpovídá barvě hledaných objektu je v poli hodnota *true*, jinak *false*. Poté se pole prochází po řádcích shora dolů, jak je znázorněné na obrázku [Obrázek 7] vpravo. První nalezená hodnota *true* představuje levý horní roh objektu. Na stejném řádku se pokračuje dále směrem doprava, dokud mají prvky v poli hodnotu true. Poslední hodnota true, tedy ta nejvíce vpravo, představuje pravý horní roh objektu. Od tohoto bodu se dále postupuje obdobným způsobem ve sloupci dolů. Poslední hodnota true, tedy ta nejníže, je pravý dolní roh. Ze vzdáleností a pozic těchto rohů se vytvoří správně veliký objekt na správném místě.

Třída **MainObject**

MainObject je bázová třída od které jsou poděděny všechny třídy, které představují modely objektů. Jediná vyjímka je třída **Floor**, která představuje model podlahy, ale není od **MainObject** poděděna.

Třída se skládá ze dvou pomyslných částí. Jedna část poskytuje prostředky a metody pro vytváření trojrozměrného modelu a druhá část udržuje vlastnosti modelu a zajišťuje správné zobrazení ovládacích prvků.

Část pro 3D model

Základem pro vytváření modelů objektů je metoda *GetBlockVertices*, která vrací seznam vrcholů pro vytvoření kvádru. Jejím vstupem je pozice a velikost požadovaného kvádru a výstupem je seznam vrcholů, podle kterých je grafická karta schopná kvádr vykreslit na obrazovku.
Každý vrchol je instancí třídy **VertexPositionNormalTexture**, tedy obsahuje informace o pozici v trojrozměrném prostoru, normále a pozici pro mapování textury. Normála je definovaná k ploše, která je vrcholem tvořena, respektive k ploše trojúhelníku, který je tvořen třemi vrcholy.
Vrcholy jsou v seznamu v takovém pořadí, aby je bylo možné přímo zapsat do **VertexBufferu** a poté jako *PrimitiveType.TriangleList* vykreslit.

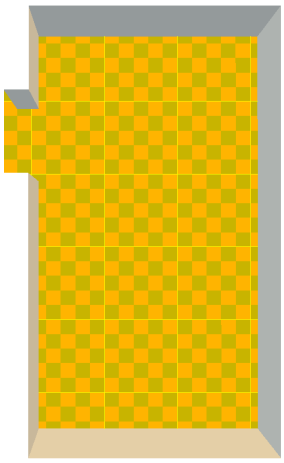
Pro ovládání objektů klikáním myši a zobrazování plovoucích názvů jsou implementovány metody *GetPointsForBox* a *GetBoundingBox*.
Obě metody jsou úzce spjaty a spolu vytvářejí pro daný objekt **BoundingBox**, který je poté použit, pro zjištění nad kterým objektem se kurzor myši nachází nebo na který objekt uživatel klikl. Postup jakým způsobem se objekt vybírá je podrobně popsán pod třídou **House** u metody *MousePicker*.

Ke změně barvy objektu, nebo alespoň části objektu, slouží metody *ChangeColor* a *ChangeColorToOriginal*.
ChangeColor vytvoří jednopixelovou texturu v barvě předané parametrem a nechá ji namapovat na objekt, čímž dojde ke změně barvy objektu. *ChangeColorToOriginal* vrátí texturu a tedy i barvy do původního stavu.
Tyto metody jsou použity například při zpuštěném alarmu, kdy objekt periodicky mění barvu ze své původní na červenou a z červené zpět na původní.

Vykreslování objektů na obrazovku probíhá v metodě *Draw*. V **MainObject** je tato metoda označena jako virtual a má prázdné tělo. Všechny poděděné třídy tuto metodu přetěžují, tak aby vykreslování probíhalo podle potřeb jednotlivých modelů.

Část pro vlastnosti a ovládací prvky
Tato část je využívána především modely akčních objektů (objekty které například interagují s uživatelem), pro potřeby jejich ovládání. Při vytváření nového modelu akčního objektu se zjistí, o jaký typ objektu se jedná, a podle toho se zobrazí správné ovládací prvky. Například pro model okna se v ovládacím panelu zobrazuje *TextBox* s informací o otevřenosti okna a *Slider*, pomocí kterého je možné okno otevírat na požadovanoi úroveň.

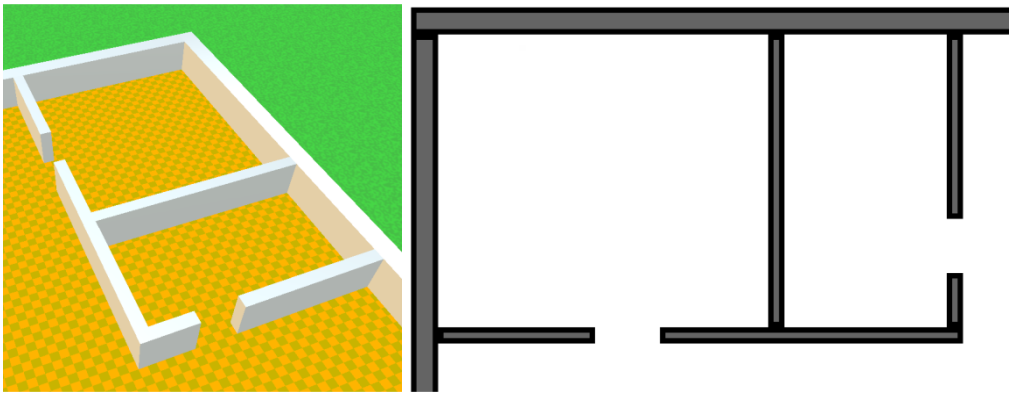
Třída **Floor** (podlaha)



Obrázek 8 – Pohled zhora na podlahu se zapnutou mřížkou (žluté přímky)

Podlaha se zobrazuje jako jedna celistvá plocha, na kterou je namapována textura se šachovnicovým vzorem.
Pro lepší navigaci a lokalizaci lidí ve 3D zobrazení je možné na podlaze zobrazit mřížku, jak je vidět na obrázku [Obrázek 8]. K tomu slouží metoda *LoadGrid*, která má vstupní parametr velikost požadované mřížky. V metodě se vytvoří seznam vrcholů, které je následně možné vykreslit jako *PrimitiveType.LineList* a tím vytvořit na podlaze navigační mřížku.

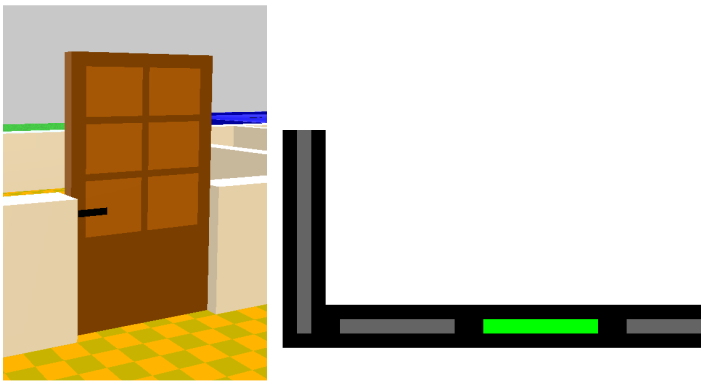
Třída **Wall** (zeď)



Obrázek 9 – Systém zdí jednoho z pater (vlevo) a podklady pro tyto zdi (vpravo)

Jedna zeď je reprezentována jedním kvádrem. Celý systém zdí je tvořen jednotlivými zdmi, které na sebe těsně navazují a tím vytvářejí efekt jednolité zdi. Podkladový obrázek a vygenerovaný 3D systém zdí je zobrazen na obrázku [Obrázek 9].

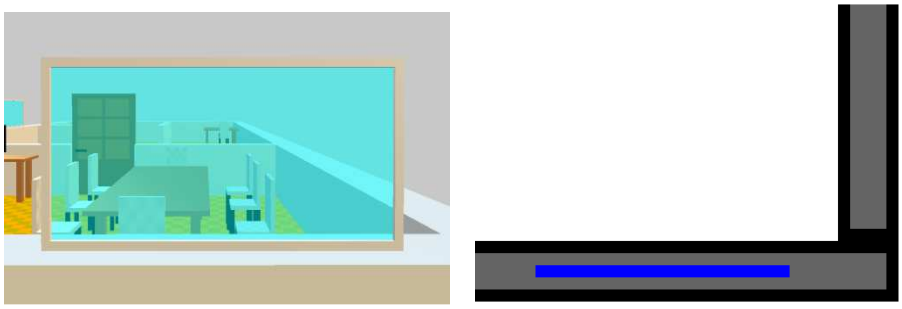
Třída **Door** (dveře)



Obrázek 10 – Model zavřených dveří (vlevo), zeleně definované dveře v obrázku (vpravo)

Model dveří je tvořen jedním kvádrem, jehož požadovaná velikost a orientace se předává v konstruktoru. Velikost je ještě upravena tak, aby tloušťka dveří byla jeden pixel.
Kvádr má kolem dokola jednobarevný rám a na přední a zadní stěně texturu, která se načítá z bitmapového souboru. Jak textura vypadá je vidět na obrázku [Obrázek 10]
Dveře je možné otevírat a zavírat, čímž mění svojí polohu. Proto bylo nutné přepsat bázovou metodu *GetBoundingBox*, tak aby získaný **BoundingBox** odrážel otevřený resp. zavřený stav dveří.

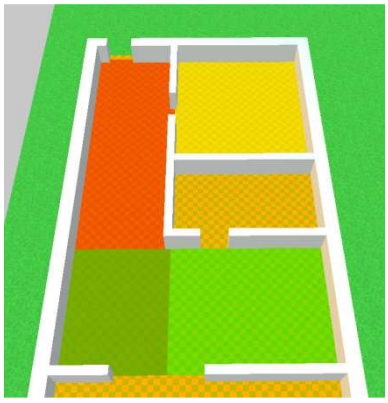
Třída **MyWindow** (okno)



Obrázek 11 – Model zavřeného okna (vlevo), modře definované okno v obrázku (pravo)

Model okna se skládá z bílého rámu a půhledné skla. Rám je tvořen 4 kvádry, jejichž vrcholy se získají bázovou metodou *GetBlockVertices*. Vrcholy pro vytvoření skla se definují přímo ve třídě.
Vizualizace otevřeného a zavřeného okna se uskutečňuje posouváním skla dolů a nahoru. Pokud je okno zavřené vyplňuje sklo celou plochu v rámu, pokud je okno otevřené, je sklo zasunuté dolů do zdi. Jak vypadá zcela zavřené okno je vidět na obrázku [Obrázek 11].

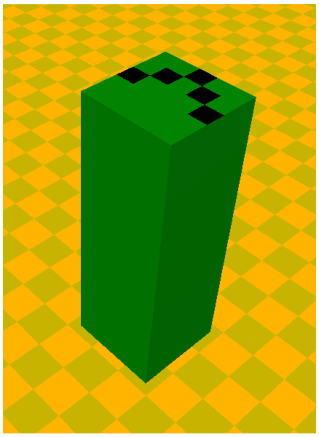
Třída **Zone** (zóna zájmu)



Obrázek 12 – Pohled na 3 zóny zájmu

Zóny zájmu se zobrazují jako barevné poloprůhledné plochy [Obrázek 12]. Barva jakou má zóna mít se v hexadecimální podobě definuje v podkladovém XML souboru.
Různé zóny se vykreslují v lehce odlišné výšce nad podlahou, aby nedocházelo k nehezkým grafickým artefaktům.

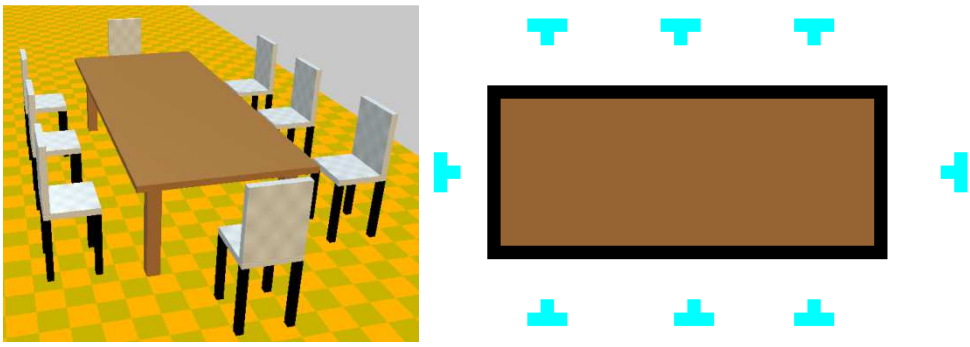
Třída **Person** (člověk)



Obrázek 13 – Model člověka

Model člověka se na rozdíl od ostatních modelů vytváří vždy v počátku souřadné soustavy a poté se teprve pomocí transformačních matic posouvá, aby se zobrazoval na správném místě. Všechny ostatní modely objektů se vytváří přímo na místě, kde mají být, protože se s nimi již nebude dále hýbat. Lidé se ale hýbat budou a vytvářet nový model člověka pokaždé, když změní svoji pozici, by bylo značně neefektivní, proto se model vytvoří pouze jednou a poté se s ním posunuje pomocí transformací.
Aby bylo možné člověka kdykoli přesunout, je implementována metoda *Move*, jejíž vstupní parametry jsou 2D souřadnice určující pozici kam se má člověkem posunout a rovněž parametr na jaký úhel se má člověk natočit.
Pro zobrazení jakým směrem je člověk otočený, má model člověka na hlavě namapovanou texturu se šipkou (znázorněné na obrázku [Obrázek 13]), která ukazuje směr, kam se člověk dívá.

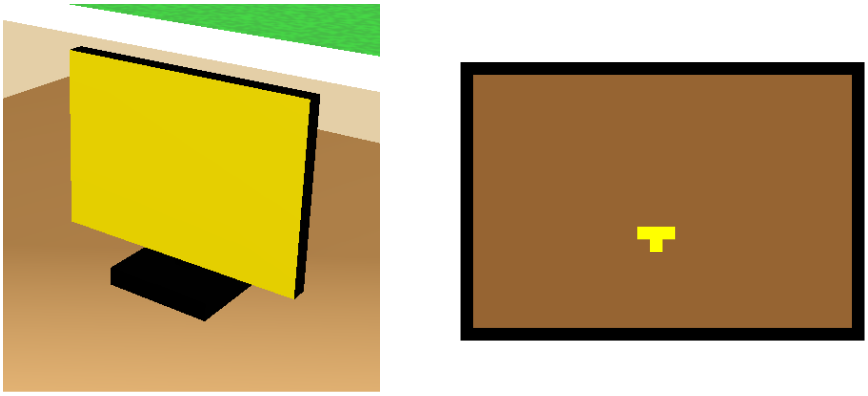
Třída **Chair** (židle) a Třída **Table** (stůl)



Obrázek 14 – Pohled na stůl a židle (vlevo), podkladový obrázek (vpravo)

Modely stolů a židlí jsou pouze statické objekty vytvořené z několika kvádrů. Židle mají navíc na sedáku a opěradle namapovanou jednoduchou texturu se vzorem šachovnice [Obrázek 14].

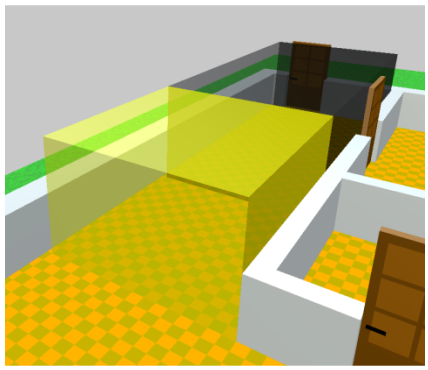
Třída **TV** (televize)



Obrázek 15 – Pohled na zapnutou televizi (vlevo), podkladový obrázek pro televizi na stole (vpravo)

Televize se skádá ze tří kvádrů poskládaných tak, aby tvořily podstavec, stojánek a vlastní obrazovku. Čelní stěna představující obrazovku může měnit texturu a tím simulovat stav televize. Momentálně je možné nastavit pouze dva stavy – vypnuto a zapnuto, reprezentované černou a žlutou barvou (model zapnuté televize je na obrázku [Obrázek 15]). V případě potřeby více stavů (například pro přepínání televizních kanálů) lze snadno třídu upravit a přidat nové textury pro další stavy.

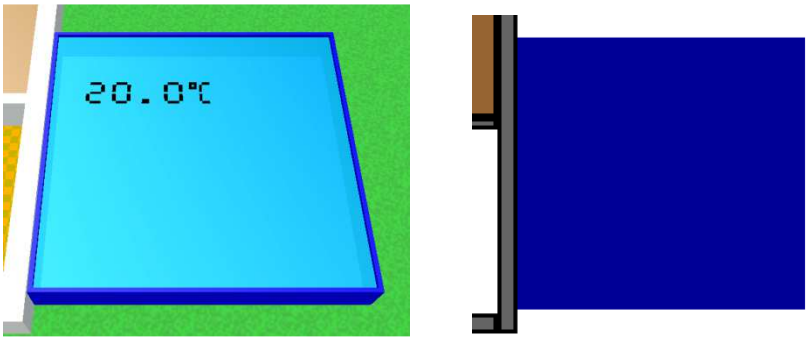
Třída **Light** (světlo)



Obrázek 16 – Zapnuté světlo (vpředu), vypnuté světlo (vzadu)

Světlo je zobrazováno jako poloprůhledný kvádr, který může rovnoměrně měnit barvu od černé do žluté. Černá znamená zhasnuté světlo a žlutá rozsvícené. Plně rozsvícené a plně zhasnuté světlo je zobrazené na obrázku [Obrázek 16].

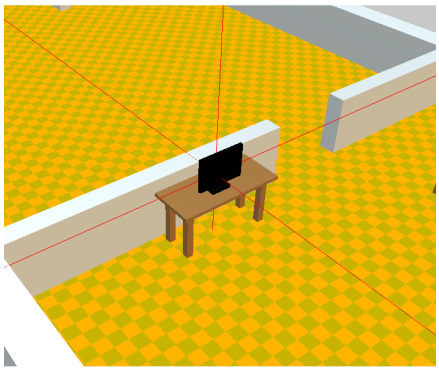
Třída **Pool** (bazén)



Obrázek 17 – Model bazénu

Bazén se skládá z pěti hranolů poskládaných tak, aby tvořili nádobu s otevřenou vrchní částí. Voda v bazénu je imitována poloprůhlednou, modře zabarvenou plochou, která je umístěna kousek pod vrchem. Těsně nad vodou je další plocha, na kterou se mapuje textura s napsanou teplotou.
Zobrazená hodnota teploty může být od -99.9°C do +99.9°C s přesností na jedno desetinné místo.

Třída **Marker** (navigační ukazatel)



Obrázek 18 – Hledaná televize zvýrazněná značkou

Třída Marker představuje značku, která se používá pro lokalizaci objektů ve 3D zobrazení. Značka se skládá ze tří přímek, které jsou navzájem kolmé a protínají se v místě, kde leží hledaný objekt (jak přímky vypadají, je patrné z obrázku [Obrázek 18]). V případě, že se jedná o pohyblivý objekt, je pozice značky upravována tak, aby odpovídala aktuální pozici hledaného objektu.

Třída **House**

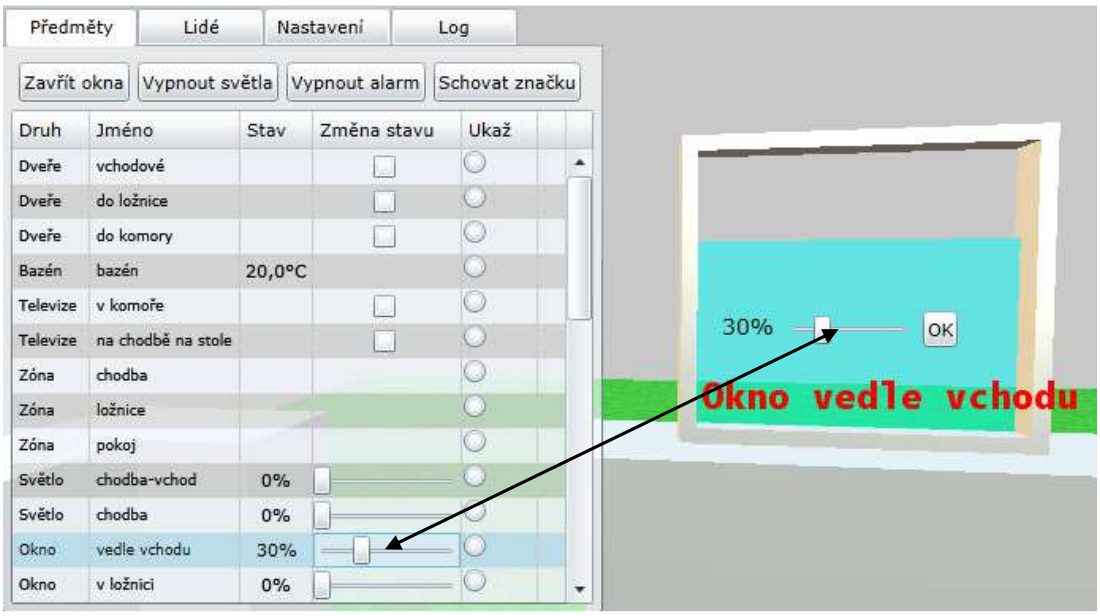
Třída obsahuje 3 důležité seznamy objektů – seznam pater, seznam lidí a seznam objektů, které mají aktivní alarm.
V seznamu pater jsou uloženy instance tříd **Layer**, které představují jednotlivá patra. Každé patro obsahuje seznam všech objektů, které se v patře vyskytují a tak je možné získat odkaz na instanci jakéhokoli objektu, který se v budově vyskytuje.
Odkazy na instance lidí (tříd **Person**) jsou uloženy jednak v seznamu objektů pro příslušné patro, ve kterém se člověk nachází, tak i v souhrnném seznamu lidí v celé budově, tedy ve třídě **House**. Odkaz v seznamu patra slouží ke správnému vykreslení člověka ve 3D scéně. Odkaz v seznamu lidí slouží pro efektivnější vyhledávání, protože při požadavku na změnu pozice člověka není potřeba hledat daného člověka v seznamech objektů přes všechny patra, ale stačí projít jen seznam lidí, který je mnohonásobně kratší.
Do seznamu alarmů se dávají odkazy na instance objektů, které mají aktivován alarm. Všem objektům v tomto seznamu jsou periodicky volány metody *ChangeColor* a *ChangeColorToOriginal*, čímž objekty mění barvu z červené na původní a z původní zpět na červenou. Při přidání objektu do tohoto seznamu se také u daného objektu zobrazí červený obdélník v ovládacím panelu. Při odebrání z tohoto seznamu se červený obdélník zase odstraní.

Ve třídě House je implementována metoda *MousePicker*, která slouží pro ovládání objektů pomocí klikání myši na jejich modely ve 3D zobrazení. Vstupem do této funkce je pozice kurzoru myši na obrazovce a informace zda bylo zmáčknuto pravé tlačítko myši.
Z pozice kurzoru se vytvoří paprsek (instance třídy **Ray**), který vede kolmo k obrazovce směrem dozadu skrz 3D model budovy. Poté se prochází všechny objekty, které se vyskytují v aktivním patře, a testuje se, zda se jejich **BoundingBox** protíná s takto běžícím paprskem. Ze všech objektů, kterými prochází paprsek, se vybere ten, který je nejblíže ke

kameře a odkaz na něj se předá dále. Z takto vybraného objektu se vybere jeho jméno a zobrazí se jako plovoucí text vedle kurzoru myši. V případě, že bylo zmáčknuto pravé tlačítko myši, zavolá se metoda, která způsobí změnu stavu vybraného objektu (pokud objekt není pouze dvoustavový, zobrazí se ovládací prvek, kterým je možné objekt přepínat mezi více stavy).

4.2 Komunikace s uživatelem

Pro interakci s uživatelem využívá aplikace dvě cesty znázorněné na obrázku [Obrázek 19], uživatel může klikat na objekty ve 3D zobrazení nebo klikat na ovládací prvky vytvořené na dialogu aplikace (jako jsou například *Button*, *Checkbox*, *Slider*,…). Obě cesty jsou navzájem propojené, takže změna stavu objektu (např. otevření dveří) kliknutím na prvek dialogu (v případě dveří *CheckBox*) se okamžitě projeví i ve 3D zobrazení. V opačném směru toto funguje naprosto identicky, při kliknutí myši na objekt ve 3D zobrazení se okamžitě změní i stav ovládacího prvku na dialogu.



Obrázek 19 – Ovládací prvky v seznamu ovladatelných objektů (vlevo), model okna se zobrazeným SliderHandler (vpravo), (červeně) plovoucí text se jménem objektu

Třída **SliderHandler**

SliderHandler slouží jako ovládací prvek pro ovládání oken a světel. Celý prvek se skládá z textu, posuvníku a tlačítka jak je vidět na obrázku [Obrázek 19].
Při kliknutí myší na 3D model okna nebo světla se zobrazí tento ovládací prvek a umožní spojitě měnit stav daného objektu. Ke změně stavu slouží posuvník, jehož hodnota se navíc na levé straně zobrazuje v podobě textu. Na pravé straně je tlačítko, kterým se potvrzuje nastavená hodnota a po jeho kliknutí celý ovládací prvek zmizí.

4.3 Komunikace se skutečnou budovou

Tato kapitola popisuje návrh a možnosti komunikace mezi vytvořenou webovou aplikací a vzdáleným serverem, ze kterého budou přenášena data pro zobrazení a na který budou zasílány povely od uživatele.

Aplikace zobrazující pouze stav prostředí, tedy například vybavení budov a pozice osob by nebyla příliš užitečná bez skutečné interakce s okolím. Za tímto účelem byla do aplikace implementována schopnost komunikace se vzdáleným „serverem“ ze kterého může přijímat aktuální stav skutečného objektu a rovněž mu poskytovat akce od uživatele. Vzdálený komunikační uzel nemusí být přímo server, ale libovolný běžící program poskytující aktuální údaje a zpracovávající požadavky uživatele. Princip komunikace mezi vlastní aplikací a vzdáleným serverem bude následující:

1) Webová aplikace se připojí na vzdálený server, který je chopen připojit současně i několik takovýchto webových aplikací pro možnost nejen sledovat, ale i řídit objekt z několika míst současně.
2) Webová aplikace si od serveru vyžádá potřebné informace pro nastavení schopností interakce pro uživatele. Tedy například jaké objekty jsou určeny pouze pro zobrazení aktuálního stavu a jaké objekty může uživatel ovládat. Možnosti interakce pro uživatele může být rovněž záviset na lokalitě odkud je kontrolní aplikace spouštěna. Pokud je například spuštěna ze zařízení umístěného přímo v budově / bytu tak mohou být pro uživatele zpřístupněny mnohem větší možnosti, než když je aplikace spuštěna například z internetové kavárny.
3) Server aplikaci poskytne všechny potřebné informace pro budoucí interakci s uživatelem a rovněž si zapamatuje, jaké schopnosti byly této aplikaci přiděleny. Tato skutečnost může být vhodná zejména z možného „nalomení“ vzdálené komunikace třetí stranou. Server bude ignorovat všechny příchozí povely, které budou přesahovat nastavený typ povolené interakce.
4) Webová aplikace si vyžádá stav všech objektů, pro které má poskytovat zobrazení jejich aktuálního stavu. Objekty schopné interagovat s uživatelem jsou označeny jako „akční“ a ostatní pouze jako „statické“.
5) V určitých intervalech bude webová aplikace po vzdáleném serveru požadovat aktualizaci stavu zobrazených objektů. Pro omezení komunikace je vhodné přenášet pouze změny stavu příslušných objektů. Webová aplikace se tedy vždy po daném časovém intervalu dotáže serveru, zda jsou nějaké změny a podle nich aktualizuje 3D zobrazení.
   Tento postup však není vždy nejvýhodnější, jelikož zde záleží na tytu spojení mezi aplikací a vzdáleným serverem. V nejběžnějším případě bude využíváno HTTP přenosu pomocí funkce GET. Ve složitějších případech je vhodné využívat spojení přímo pomocí SOCKETu. První varianta pomocí GET je univerzální, ale není přímo určena pro samočinné předávání dat ze serveru do webové aplikace. V tomto případě si webová aplikace všechny informace ze serveru vyžaduje, což je vhodné pro zpracovávání akcí uživatele serverem z webové aplikace, ale je nevhodné například pro okamžité zobrazení varovných / havarijních stavů detekovaných serverem. Druhá varianta pomocí SOCKETu je mnohem využitelnější, ale vyžaduje jak na straně webové aplikace, tak i na straně serveru mnohem více úsilí a

programového kódu. Velkým přínosem je však vytvoření pevného spojení mezi webovou aplikací a serverem a možnosti zasílání libovolných informací ze serveru přímo do webové aplikace i bez jejich vyžádání.

6) Server bude poskytovat webové aplikaci aktuální hodnoty objektů podle jejich skutečného stavu. Bude tedy poskytovat pouze změny na příslušných objektech.
   Pod pojmem „server“ si lze představit jakýkoli SW program komunikující s webovou aplikací. K Tomuto „serveru“ nemusí být přímo připojeny HW komponenty představující vybavení domu / bytu. Ten může obsahovat pouze databázi jejich aktuálního stavu a tyto stavy poté poskytovat webové aplikaci. Pokud dojde k aktualizaci hodnoty v databázi na serveru, tak je u příslušné položky nastaven příznak „aktualizace“ a tím je server informován o nutnosti zaslání tohoto nového stavu do webové aplikace.
7) Pokud uživatel na webové aplikaci vykoná nějakou činnost, které se vztahuje k příslušné komponentě v objektu, tak webová aplikace tento požadavek zašle do serveru. Server je v podstatě zodpovědný za skutečné vykonání akce od uživatele. Zde již velmi záleží na činnosti serveru, zda požadavek na nastavení stavu reálného objektu uloží pouze do databáze s nějakým příznakem pro jiný program, který někdy později požadovanou činnost „skutečně“ vykoná, nebo server sám tento program vyzve, aby tuto činnost vykonal.

Nyní se budou postupně opakovat body 5) a 6). Pokud uživatel vykoná nějakou akci, tak se provede bod 7).

Informace o alarmech a havarijních stavech je potřeba do webové aplikace přenést co nejrychleji. V případě SOCKETového spojení je dosažení tohoto požadavku v podstatě snadné. V případě HTTP GET spojení je situace složitější.
Při využití HTTP GET spojení webové aplikace se serverem, lze princip dotazování rozdělit na několik úrovní:

Nejvyšší - Celkem častý stručný dotaz, ale pouze na alarmové nebo havarijní stavy.
Střední - Méně častý dotaz na činnosti jako je například otevření / zavření dveří / okna.
Nízká - Dotaz s delším intervalem na komponenty, které svůj stav již z principu nemohou
měnit rychle, jako například teplota vody v bazénu.

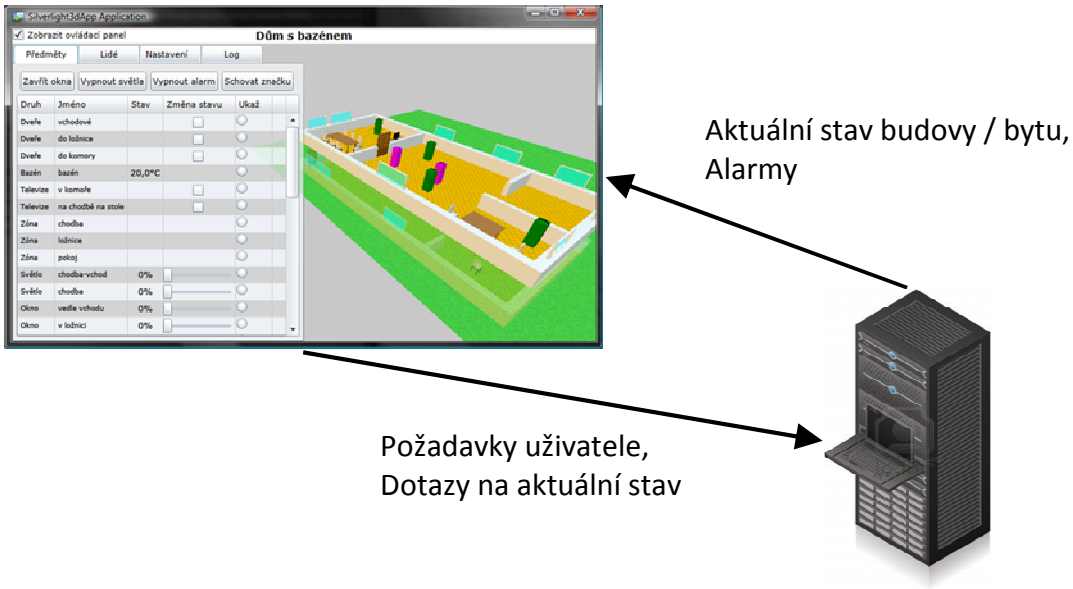
Tímto způsobem lze vytvořit kompromis v dotazovacím systému mezi frekvencí dotazování a počtem přenášených dat.

Pokud bude uživatel uvnitř bytu / budovy a bude využívat pro přístup například tablet nebo smartphone, tak se jedná o zcela jednoznačné zařízení, s pevně danou IP adresou, a lze tedy výhodně použít komunikaci pomocí SOCKETu. Komunikace bude zcela oboustranná a tedy velmi rychlá. Všechny alarmy a havarijní stavy budou zobrazeny v podstatě okamžitě. Toto má význam, protože se uživatel v daném objektu přímo pohybuje a může tedy na tyto skutečnosti okamžitě reagovat. Naopak pokud se uživatel připojuje vzdáleně (kontrola objektu z dovolené) tak je vhodná komunikace pomocí HTTP GET. Ta bude sice pro aktualizaci zobrazeného stavu pomalejší, ale vzhledem ke vzdálenosti od objektu to nemusí být velkou nevýhodou.

4.3.1 Typy přenášených informací

Typy a formát přenášených informací a dat je nezávislý na typu komunikace mezi aplikací a serverem, což velmi zjednodušuje celý systém. Nejprve jaké hlavní typy zpráv budou přenášeny:

- Inicializační pro zahájení komunikace mezi aplikací a serverem.
- Dotazovací z aplikace na server ohledně zjišťování aktuálního stavu.
- Informační o aktuálním stavu ze serveru do aplikace.
- Řídící z aplikace na server za účelem vykonání požadavku od uživatele.
- Alarmy a jiné důležité informace ze serveru do aplikace.



Obrázek 20 – Schéma zasílání zpráv mezi aplikací a serverem, šipkami je naznačen směr kterým se dané typy zpráv posílají

Na první pohled se sice jedná o významově rozdílné zprávy, ale ve skutečnosti budou obsahovat velmi podobné informace. Například zpráva s informací o otevření okna ze serveru do aplikace, bude podobná té, kterou aplikace zašle na server o akci uživatele vyžadující otevření okna. Každá zpráva bude obsahovat následující položky:

- Datum a čas (datum a čas platnosti zprávy)
- Typ zasílané zprávy neboli informace.
- Identifikaci objektu, ke kterému se zprava / informace vztahuje.
- Data neboli hodnota obsažená ve zprávě a vztahující se k objektu.
- Platnost zde uvedených dat / hodnot.
- Libovolná textová poznámka s doplňujícími informacemi pro uživatele.

V následující ukázce zdrojového kódu je patrné, jak jsou jednotlivé složky posílané zprávy implementovány do aplikace.

```
  // datum a čas poslání zprávy
[XmlAttribute("DateTime")]
public String Date
{
     // získání data a času ze systémových hodin
   get { return dateTime.ToString(DateTimeFormatWithMillis); }
   set { dateTime = DateTime.ParseExact(value, DateTimeFormatWithMillis, null); }
}

  // typ zprávy resp. jaké informace se ve zprávě přenášejí
[XmlAttribute("TypeMessage")]
public TypeMessage type;

  // identifikační jméno objektu, ke kterému se zpráva vztahuje
[XmlAttribute("Name")]
public virtual string NameXML
{
   get { return name; }
   set { name = value; }
}

  // platnost dat posílaných ve zprávě
[XmlAttribute("StatusType")]
public TypeStatus typeStatus = TypeStatus.None;

  // textová poznámka s doplňujícími informacemi
[XmlAttribute("StatusInfo")]
public string statusInfo = null;
```

Typ zasílané zprávy
Typ zasílané zprávy představuje v podstatě výčet několika hodnot (jedná se o principiální přehled):

- INIT – Inicializace komunikace z aplikace na server. V podstatě žádost o registraci (nově) spuštěné aplikace na serveru. Každé spuštěné aplikaci může být přiděleno unikátní identifikační číslo a rovněž oprávnění (jak již bylo uvedeno).
- STATE_ASK_INIT – Žádost z aplikace na server o informace o všech dostupných objektech a jejich stupni interakce z hlediska uživatele.
- STATE_ANSWER_INIT – Odpověď ze serveru do aplikace obsahující informace o všech dostupných objektech a stupni jejich interakce pro uživatele.
- STATE_ASK – Žádost z aplikace na server zda byla změna na některém sledovaném objektu.
- STATE_ANSWER – Odpověď ze serveru do aplikace o aktuálních změnách objektů.
- USER_ACTION – Požadavek z aplikace na server pro vykonání akce uživatele.
- PERSON_MOVE – Aktualizace polohy zobrazované osoby / předmětu.
- ALARM – Aktivace / deaktivace varovného stavu / zobrazení na některém objektu.

Identifikace objektu
Další položku tvoří identifikace objektu. Ta je složena ze tří částí a to:

- Patra, ve kterém se objekt nachází (0, 1, 2, …).
- Typy jakého objekt je (dveře, okno, …).
- Unikátního jména objektu (hlavní dveře, okno v ložnici, …).

Z těchto tří položek je sestaven jeden řetězec představující unikátní identifikaci komponenty v rámci celé budovy / bytu. Takto stanovený typ může současně předurčovat některé činnosti komponenty.

Data
Tato položka obsahuje data neboli přenášenou hodnotu a je v podstatě ze všech obsažených položek nejsložitější. Ne však co do významu, ale co do jejího zápisu. Při komunikaci je třeba uvažoval, nejen rozmanitost operačních systémů, ale rovněž i programovacích jazyků. Jak aplikace, tak i server mohou běžet nejen na jiném operačním systému, ale mohou být napsány i v jiném programovacím jazyce. Některé programovací jazyky nemusí obsahovat například typ „boolean“ pro dvoustavové hodnoty a z tohoto důvodu je potřeba vytvořit takové typy, které by byly pro programy napsané v různých jazycích pochopitelné. Pro datové hodnoty byly navrženy následující datové typy:

- OffOn – Pro dvoustavové hodnoty, obdoba typy boolean (dveře otevřít / zavřít, světlo svítí / nesvítí)
- Percentage – Celočíselná hodnota z určitého rozsahu, který je vhodné převést na rozsah 0 až 100% (míra otevření okna, míra rozsvícení světla, míra zatažení závěsu)
- Integer – Pro libovolné celočíselné hodnoty, jak kladné tak i záporné (počet litrů vody v nádrži na déšť)
- Float – Pro libovolné desetinné hodnoty, jak kladné tak i záporné (teplota)
- String – Pro hodnoty, které lze pomocí čísel velmi těžko zapsat (nastavení TV kanálu podle názvu)

Zda programovací jazyk například v případě hodnoty typu „OffOn“ využije datový typ „boolean“ nebo „integer v rozsahu 0-1“ je zcela na jeho možnostech nebo na uvážení programátora.
Většina přenášených dat (stav, nastavení, …) bude obsahovat pouze jednu hodnotu, jako jsou stav okna nebo dveří. Někdy je však potřeba přenést dvě nebo i více hodnot jako například poloha a směr otočení osoby. Pro jednoduchost budou všechny datové hodnoty přenášeny jako řetězce. Aby bylo možné snadno detekovat i několik hodnot v přenášených datech, byl navržen zápis pomocí „klíče“ a „hodnoty“. Zde je uvedeno několik příkladů přenášených dat a v nich obsažených hodnot:

Otevření běžného okna: Type=OffOn;Value=On
Nastavení svitu světla na polovinu: Type=Percentage;Value=50
Umístění osoby na pozici s otočením: Type=Float;X=12;Y= 34;Angle=56
Uvedený typ pro jednoduchost udává typ všech hodnot v datech. Všechny datové položky mají tedy stejný typ (v současné verzi).

Platnost dat
Další z položek s názvem „platnost zde uvedených dat“ je velmi důležitá při reálném provozu systému a může nabývat těchto hodnot:

- Actual / Valid – Uvedená hodnota je zcela aktuální (k danému datu a času). Server tuto hodnotu obdržel jako aktuální (její stáří není delší než nějaký limit) a poskytl ji zobrazovací aplikaci.
- Last – Aktuální hodnotu nebylo možno získat (například z důvodu přerušené komunikace se senzorem) a proto je zde uvedena poslední známá / zaznamenaná hodnota. Tuto hodnotu server poskytne pouze, pokud čas pro aktualizaci této veličiny nepřekročí určitý maximální limit.
- InValid – Hodnota není platná nebo spíše se ji nepodařilo v daném maximálním časovém limitu zjistit. V tomto případě hodnota nemusí být vůbec uvedena nebo může být uvedena poslední známá.
- Error / Fail – Komunikace se senzorem / komponentou selhala a tudíž požadovanou hodnotu nelze poskytnout.

Zobrazovací aplikace by samozřejmě při zpracování hodnoty měla k této položce přihlédnout a podle ní indikovat stav příslušné komponenty uživateli.

Textová poznámka
Poslední položku tvoří „textová poznámka“. Tato položka bude ve většině případů zřejmě nepoužita. Jsou však okolnosti, kdy může být velmi užitečná, například v situaci, kdy uživatel zapne výpust nádrže a čerpadlo se odmítne zapnout, protože v nádrži není přítomna žádná voda (jde o chytré čerpadlo se senzorem výšky hladiny). V tomto okamžiku řídící mechanismus čerpadla místo potvrzení „OK“ zašle například zprávu „NoWaterToPump“ a jeho stav zůstane stále v „Off“. Pokud by komunikace mezi serverem a zobrazovací aplikací poskytovala přenos pouze stavu čerpadla (Off - On), tak by byl uživatele zmaten, protože jeho povel v podstatě nebyl vykonán. Tato položka tedy řeší tuto situaci.

4.3.2 Struktura přenášené zprávy

Řešení komunikace lze velmi často oddělit do tří složek:

- Obsažené informace (jaké položky, jejich hodnoty, rozsah) se budou přenášet.
- Jakou strukturu budou mít přenášená data, jak budou ve zprávě organizována (snadné vložení, přidání, vyzvednutí).
- Jak budou hodnoty / data skutečně uložena neboli zakódována při vlastním přenosu (textové, binární)

První a druhý bod byly již v podstatě popsány v pojednání o významu a organizaci přenášených dat / hodnot. Do tohoto popisu není vhodné se pouštět příliš podrobněji, protože se jedná o návrh a určitý prototyp komunikace a v současné době nejsou známy všechny požadavky.

Binární kódování

Jako první možnost lze uvést binární kódování neboli i tzv. „serializaci“ (převod datového objektu na posloupnost byte s přesně daným pořadím / významem). Tento způsob má některé výhody jako minimální objem přenášených dat, pro některé programové jazyky je velmi snadný a zejména rychlý převod do a z binárního tvaru. Velká nevýhoda ale spočívá v závislosti na programovacím jazyku. Mezi dvěma odlišnými jazyky bývá komunikace na binární úrovni velmi složitá (kódování desetinných čísel, pořadí bytů nižší / vyšší) a proto se v podstatě nevyužívá. Naopak pokud bude server i (všechny) aplikace vytvořeny ve stejném programovacím jazyce je tento typ komunikace velmi přínosný. Poslední nevýhodou binárního stylu komunikace je velmi malá variabilita zpětné kompatibility. Pokud server do zprávy vloží novou položku, se kterou aplikace nepočítá, dojde téměř jistě na straně aplikace k selhání při pokusu vyzvednout i známá data. Od tohoto typu komunikace bylo ustoupeno.

Všechny další možnosti v podstatě představují textově orientovaný přenos. Některé lze zmínit a uvést jejich výhody a nevýhody.

Prostý řetězec

Všechny položky a hodnoty zprávy lze zapsat do prostého řetězce s nějakým vhodným oddělovačem, který se jinde nevyskytuje. Tento způsob není moc vhodný, protože vyžaduje přesnou znalost pořadí všech položek a v případě přidání nové položky může dojít ke stejnému typu selhání jako u binárního přenosu. Tento nedostatek lze celkem snadno odstranit vytvořením tzv. dvojit „Klíč=Hodnota“. Zpráva tedy bude strukturována například: „Typ=Stav;Název=Okno;Hodnota=Otevřeno;…“. Tento formát je univerzálnější, protože jednotlivé položky (v podstatě dvojice) jsou vyzvedávány zásadně podle „klíče“, který je v řetězci vždy unikátní. Přidání nové dvojice (s unikátním klíčem) na straně serveru a při její neznalosti na straně aplikace komunikaci neohrozí, protože aplikace tuto položku ignoruje. Při volbě vhodných názvů pro klíče (výstižnost, krátkost) může jít o velmi efektivní přenos, který lze snadno uskutečnit přes mnoho typů kanálů včetně mailu nebo SMS.

Příklad zprávy ve formátu klíč-hodnota:
Name=Dveře:do pokoje:1;TypeMessage=Status;StatusType=Valid;TypeValue=OffOn;Value=On

XML formát

Při dostupnosti dostatečně kapacitního přenosového kanálu, zle s výhodou využít přenos dat pomocí XML struktury. Jedná se o jistou obdobu předchozího „řetězce“, která má některé výhody a nevýhody:

(+) Přenášené informace jsou v podstatě ve stromové struktuře, která je velmi variabilní pro vkládání dalších informací.
(+) Skoro všechny současné programovací jazyky obsahují přímou podporu pro práci s XML formátem.
(+) Mnoho editorů poskytuje přehledné zobrazení obsahu XML struktury, což je při vývoji velmi vhodné.
(-) Položky jsou uzavřeny v tzv. tagách a ty navyšují celkový přenášený objem zprávy. Každá položka je příslušným tagem uvedena a rovněž tímto tagem ukončena.

Existují i některé rozšířené varianty XML formátu, ale pro naše účely je základní formát zcela postačující. Pro jeho dobrou podporu ve všech významných programovacích jazycích byl zvolen jako hlavní formát pro přenos zpráv. Jedná se tedy o zcela textový formát, dobře čitelný a v současné době zřejmě nejrozšířenější. Také poskytuje vhodnou variabilitu pro budoucí rozšiřování přenášených položek / dat.

Příklad zprávy ve formátu XML:

```
<Message>
 <Data>
  <MessageDevice>
        <Name>Dveře;do pokoje;1</Name>
        <TypeMessage>Status</TypeMessage>
        <StatusType>Valid</StatusType>
        <TypeValue>OffOn</TypeValue>
        <Value>On</Value>
  </MessageDevice>
  <MessageDevice>
        <Name>Okno;v ložnici;0</Name>
        <TypeMessage>Status</TypeMessage>
        <StatusType>Valid</StatusType>
        <TypeValue>Percentage</TypeValue>
        <Value>50</Value>
  </MessageDevice>
 </Data>
</Message>
```

JSON formát

Jde o formát pocházející z jazyka JAVA, který je XML formátu principiálně podobný, ale v mnoha směrech poskytuje přidané hodnoty. Byl vytvořen přímo pro přenos strukturovaných dat uložených v programových objektech. Oproti XML formátu není tak „upovídaný“ neboť tagy jsou v podstatě nahrazeny závorkami. V současné době však ne všechny programovací jazyky tento formát dostatečně podporují a proto nebyl ve vytvářeném prototypu využit. Přechod z formátu XML na JSON byl neměl být později žádný problém.

Příklad zprávy ve formátu JSON:

```
{"Message": {
   "Data": {
      "MessageDevice": [
         "Name": "Dveře;do pokoje;1",
         "TypeMessage": "Status",
         "StatusType": "Valid",
         "TypeValue": "OffOn",
         "Value": "On"},
```

```
        { "Name": "Okno;v ložnici;0",
          "TypeMessage": "Status",
          "StatusType": "Valid",
          "TypeValue": "Percentage",
          "Value": "50"}
        ]
  }
}
```


3.3.2 Implementace komunikace

Jelikož stěžejní částí práce byla webová aplikace poskytující 3D zobrazení, tak vlastní komunikace mezi serverem a aplikací byla vytvořena pouze na úrovni prototypu. Vlastní server byl „simulován“ pomocí běžné aplikace vytvořené v Microsoft .NET (WPF). Ten obsahoval jednoduché GUI pro simulaci reálných domácích komponent (zadání jejich stavu), zasílání jejich stavu do webové aplikace a rovněž zobrazení změn, která z webové aplikace přišli.
Ve webové aplikaci byl rovněž implementován prototyp komunikace se serverem. Jelikož webová aplikace již obsahuje vlastní databázi grafických komponent / objektů, tak komunikační část v podstatě pouze vyzvedává přijaté informace a aktualizuje grafické stavy a rovněž transformuje akce uživatele z grafických objektů na zprávy zasílané do serveru. Princip komunikace a struktura zpráv i přenášených informací již byly popsány a proto nemá význam toto zde popisovat znova.

3.3.3 Realizace

Jako zcela nepovinná část práce bylo experimentálně odzkoušeno napojení vytvořeného systému na reálný svět. Za tímto účelem byl využit již částečně instalovaný domácí řídicí systém sestavený zejména z komponent od firmy Papouch [10]. Jedná se o různé komponenty poskytující vstupy, výstupy, měření teploty atd. a komunikující pomocí LAN, RS232, RS485 atd. Jeden z těchto modulů (4 vstupy, 4 výstupy, teplota) byl napojen na vytvořený server, který z tohoto modulu získával reálně vstupy, teplotu a zasílal na něho akce uživatele (z webové aplikace).

5 Některé programové části vytvořené aplikace

V této kapitole se věnuji některým významným komponentám, v podstatě programovým částem, které jsou v aplikaci implementovány.

Příprava pro použití XNA

Aby bylo možné využívat všechny funkce, které XNA Framework nabízí, je potřeba mít v projektu implementovaných několik nezbytných komponent.

- Pro získání přístupu k procesoru grafické karty je nutné do souboru Silverlight3dAppTestPage.aspx přidat parametr `enableGPUAcceleration` a nastavit jeho hodnotu na `true`.

```
<param name="source" value="ClientBin/Silverlight3dApp.xap"/>
<param name="onError" value="onSilverlightError" />
<param name="background" value="white" />

<param name="enableGPUAcceleration" value="true" />

<param name="minRuntimeVersion" value="5.0.60401.0" />
<param name="autoUpgrade" value="true" />
```

- Prvek *DrawingSurface* umístěný v **MainPage.xaml** definuje plochu, na kterou je vykreslován 3D obsah. Musí obsahovat událost *Draw*, která se spustí vždy, když budeme požadovat překreslit obsah v *DrawingSurface*. Událost *Loaded* se spustí pouze jednou na začátku a slouží jako signál k načtení dalších komponent, které budou používány v průběhu spuštěné aplikace.

```
<DrawingSurface Name="DrawSurf"  Loaded="OnLoaded" Draw="OnDraw" />
```

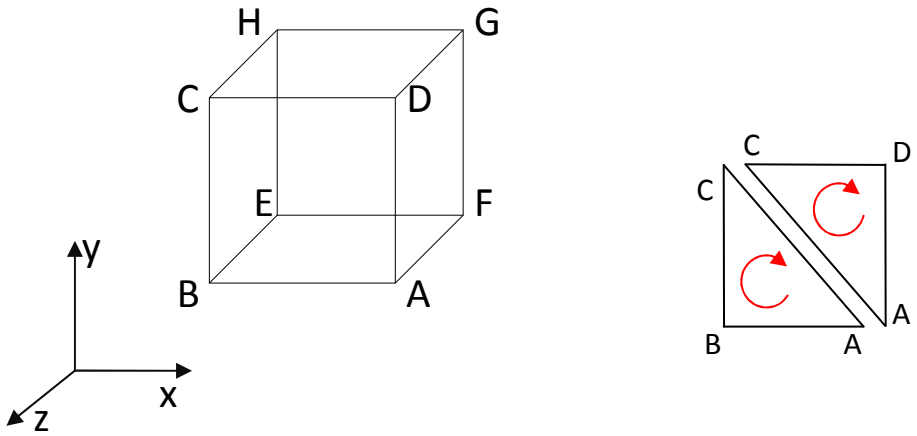
- Před začátkem vykreslování 3D scény je nutné mít instanci třídy **GraphicsDevice**, která má vykreslování nastavené na akcelerované grafickým procesorem (*RenderMode.Hardware*). Pokud by akcelerace GPU nebyla k dispozici, nebylo by možné vykreslovat 3D grafiku a aplikace by nefungovala.
  Neakcelerované vykreslování bývá často způsobené špatně nastaveným Silverlight pluginem, proto jsem implementoval krátkou zprávu, která uživateli naznačí kde hledat chybu.

```
GraphicsDevice graphicsDevice;

  // tato metoda se provede hned po startu aplikace
private void OnLoaded(object sender, RoutedEventArgs e)
{
     // získání instance GraphicsDevice
    graphicsDevice = GraphicsDeviceManager.Current.GraphicsDevice;

     // pokud není GPU akcelerace k dispozici zobrazí se okno se zprávou
    if (GraphicsDeviceManager.Current.RenderMode != RenderMode.Hardware)
    {
         MessageBox.Show("Na stránce pluginu Silverlight nastavte (3D grafika: použít
             blokované ovladače zařízení) na Povolit,
             "Warning", MessageBoxButton.OK);
    }

    …
}
```

Metoda GetBlockVertices

Tato metoda je klíčová při vytváření modelů všech 3D objektů, protože všechny modely, které se ve 3D zobrazení vyskytují, jsou složené z více či méně hranolů (např. zeď z jednoho nebo židle ze šesti). *GetBlockVertices* vytváří kvádr ve formě, podle které dokáže grafické zařízení vykreslit kvádr na obrazovku. Vstupními parametry této metody jsou pozice kvádru a velikost, výstupem je List vrcholů.

Všechny objekty vykreslované ve 3D jsou vykreslovány použitím trojúhelníků [2] (dokonce i koule může být poskládána z trojúhelníku, pokud se jich použije dostatečné množství). Každý trojúhelník je definován třemi body a každý bod je definován vektorem, který určuje jeho X, Y a Z souřadnice. Někdy znalost pouze pozice bodu nemusí stačit (např. když chceme definovat i barvu bodu) a je potřeba použít složitější strukturu tzv. vertex, který udrží více vlastností než jen pozici bodu.
Pro vytváření 3D objektů používám instance tříd **VertexPositionNormalTexture**, což je vertex, který uchovává pozici bodu, normálu a UV souřadnice pro mapování textury. Normála je definována k ploše trojúhelníku, kterého je bod součástí (např. pro trojúhelník z bodů A, B, C bude normála směřovat podél osy z a všechny tři body budou mít souřadnice normály (0,0,1)). Důvodem zavedení normál je použití světelných efektů ve 3D zobrazení. Bez znalosti o natočení plochy v prostoru, by nebylo možné správně spočítat dopadající a odražené světelné paprsky.
Pro vytvoření jedné plochy kvádru je potřeba definovat 4 vrcholy (rohy čtverce) a vytvořit dva trojúhelníky. Jelikož je ale každý trojúhelník tvořen 3 vrcholy, bude do seznamu uloženo dohromady 6 vrcholů. Trojúhelníky mají vrcholy definované po směru hodinových ručiček, vzhledem ke kameře. Této informace se využívá při vykreslování, aby se nevykreslovala zadní strana trojúhelníku (bude zevnitř kvádru – nebude vidět).



Obrázek 21 – Označení vrcholů kvádru a zobrazení souřadného systému používaného XNA (vlevo), vytvoření jedné stěny ze dvou trojúhelníků (vpravo)

```
public List<VertexPositionNormalTexture> GetBlockVertices(Vector3 point, Vector3 size)

{
   // parametr point je pozice levého, dolního, zadního rohu kvádru – na obrázku bod E
   // parametr size je velikost kvádru size=(podel osy x, podel osy y, podel osy z)

   // inicializace Listu do kterého se uloží vrcholy
 List<VertexPositionNormalTexture> vertices = new List<VertexPositionNormalTexture>(36);

   // připravení vrcholů kvádru, podle pozice a velikosti předané v parametrech
 Vector3 A = new Vector3(point.X + size.X, point.Y, point.Z + size.Z);
 Vector3 B = new Vector3(point.X, point.Y, point.Z + size.Z);
 Vector3 C = new Vector3(point.X, point.Y + size.Y, point.Z + size.Z);
 Vector3 D = new Vector3(point.X + size.X, point.Y + size.Y, point.Z + size.Z);
 Vector3 E = new Vector3(point.X, point.Y, point.Z);
 Vector3 F = new Vector3(point.X + size.X, point.Y, point.Z);
 Vector3 G = new Vector3(point.X + size.X, point.Y + size.Y, point.Z);
 Vector3 H = new Vector3(point.X, point.Y + size.Y, point.Z);

   // new VertexPositionNormal( souřadnice vrcholu, normála, souřadnice UV )

   // vytvoření přední strany
 vertices.Add(new VertexPositionNormalTexture(A, new Vector3(0, 0, 1), new Vector2(1,
1)));
 vertices.Add(new VertexPositionNormalTexture(B, new Vector3(0, 0, 1), new Vector2(0,
1)));
 vertices.Add(new VertexPositionNormalTexture(C, new Vector3(0, 0, 1), new Vector2(0,
0)));
 vertices.Add(new VertexPositionNormalTexture(A, new Vector3(0, 0, 1), new Vector2(1,
1)));
 vertices.Add(new VertexPositionNormalTexture(C, new Vector3(0, 0, 1), new Vector2(0,
0)));
 vertices.Add(new VertexPositionNormalTexture(D, new Vector3(0, 0, 1), new Vector2(1,
0)));

   // vytvoření zadní strany
 …
   // vytvoření horní strany
 …
```

```
   // vytvoření spodní strany
 …
   // vytvoření pravé strany
 …
   // vytvoření levé strany
 vertices.Add(new VertexPositionNormalTexture(B, new Vector3(-1,0, 0), new Vector2(1,
1)));
 vertices.Add(new VertexPositionNormalTexture(E, new Vector3(-1,0, 0), new Vector2(0,
1)));
 vertices.Add(new VertexPositionNormalTexture(H, new Vector3(-1,0, 0), new Vector2(0,
0)));
 vertices.Add(new VertexPositionNormalTexture(B, new Vector3(-1,0, 0), new Vector2(1,
1)));
 vertices.Add(new VertexPositionNormalTexture(H, new Vector3(-1,0, 0), new Vector2(0,
0)));
 vertices.Add(new VertexPositionNormalTexture(C, new Vector3(-1,0, 0), new Vector2(1,
0)));

 return vertices;
}
```

Vytvoření modelu zdi

Při vytváření nové instance třídy **Wall** (představující 3D model zdi) se do konstruktoru předává odkaz na **GraphicsDevice** a instance třídy **Rect** (představuje 2D obdelník). Získání odkazu na **GraphicDevice** je zásadní pro všechny 3D modely objektů, protože do této třídy se následně posílají vrcholy objektů a **GraphicsDevice** se postará o jejich vykreslení na obrazovku. Instance **Rect** v sobě udržuje pozici a rozměry obdélníka, který představuje podstavu zdi.

```
public Wall(GraphicsDevice graphicsDevice, Rect rect)
{
    base.graphicsDevice = graphicsDevice;

       // uložení předaného obdélníku pro další použití
    base.outline = rect;

       // zeď je statický objekt, proto je hodnota false
       // (akční objekty např. televize maji hodnotu true)
       // podle této proměnné se dále rozhoduje, jestli se objekt ukáže v seznamu
       // objektů v ovládacím panelu
    base.hasAction = false;

       // z Enumu ObjectType se vybere typ objektu, který tato třída reprezentuje
    base.Type = ObjectType.Wall;

      // inicializace Listu, ve kterém jsou uloženy vrcholy, ze kterých se skládá model
zdi
      // vrcholů bude dohromady 36 (3*2*6=36)
      // 3 vrcholy na trojúhelník, 2 trojúhelníky na stěnu, 6 stěn na kvádr
    base.vertices = new List<VertexPositionNormalTexture>();

       // souřadnice [x,y,z] levého horního rohu zdi
    float pointX = (float)rect.X;
    float pointZ = (float)rect.Y;
    float pointY = 0;

       // velikost objektu ve třech rozměrech
    float sizeX = (float)rect.Width;
    float sizeZ = (float)rect.Height;
    float sizeY = 10;
```

```
        // získání vrcholů zdi zavoláním bázové metody GetBlockVertices
    vertices.AddRange(base.GetBlockVertices(new Vector3(pointX, pointY, pointZ),
        new Vector3(sizeX, sizeY, sizeZ)));

        // vytvoření textury, která se bude mapovat na povrch zdi
        // jelikož je zeď jednobarevná, stačí vytvořit texturu o velikosti jeden pixel
        // proměnná color je instance třídy Color, která definuje barvu zdi
    base.texture = new Texture2D(graphicsDevice, 1, 1, false, SurfaceFormat.Color);
    texture.SetData<Color>(new Color[1] { color });

        // vytvoření Listu bodů ze kterých se bude vytvářet BoundingBox
    base.points = base.GetPointsForBox(rect, 0, sizeY);

        // vytvoření nového zásobníku vrcholů
        // tento VertexBuffer se při vykreslování posílá do GraphicsDevice
    base.vertexBuffer = new VertexBuffer(graphicsDevice,
        VertexPositionNormalTexture.VertexDeclaration, vertices.Count, BufferUsage.None);
    vertexBuffer.SetData(vertices.ToArray());

    basicEffect = new BasicEffect(graphicsDevice);
        // předání informace, že na 3D objekt se bude mapovat textura
    basicEffect.TextureEnabled = true;
        // předání textury, která se bude mapovat na 3D objekt
    basicEffect.Texture = texture;
        // zavolání metody, která nastaví základní světelné efekty
    basicEffect.EnableDefaultLighting();
}
```

Metoda *Draw* slouží pro vykreslení 3D modelu zdi na obrazovku. Jelikož se tato metoda volá při každém požadavku na překreslení obrazovky, což může být až 60 krát za sekundu, obsahuje metoda pouze nutné minimum operací, aby zbytečně nezatěžovala procesor výpočty, které není nutné provádět opakovaně.

```
public override void Draw(Matrix world, Matrix view, Matrix projection)
{
        // pokud je visible nastaveno na false, přeskočí se celé vykreslování
        // a zeď se na obrazovce neukáže
    if (visible)
    {
            // podle požadavku na průhlednost objektu se vybere BlendState
            // AlphaBlend pro průhledný objekt a Opaque pro neprůhledný
        if (base.Transparent) graphicsDevice.BlendState = BlendState.AlphaBlend;
        else graphicsDevice.BlendState = BlendState.Opaque;

            // matice world, view a projection slouží pro výpočet finální pozice vrcholů
            // objektu
        basicEffect.World = world;
        basicEffect.View = view;
        basicEffect.Projection = projection;

            // do GraphicsDevice se pošlou vrcholy, které budeme chtít vykreslit
        graphicsDevice.SetVertexBuffer(vertexBuffer);

        foreach (EffectPass effectPass in basicEffect.CurrentTechnique.Passes)
        {
            // aplikování efektů nastavených v basicEffect
          effectPass.Apply();

            // vykreslení zdi
            // TriangleList znamená, že vertexbuffer obsahuje list trojúhelníků
            // začít brát vrcholy od indexu 0 a vykreslit 12 trojúhelníků
          graphicsDevice.DrawPrimitives(PrimitiveType.TriangleList, 0, 12);
        }
    }}
```

6 Uživatelský manuál pro webový 3D model

Tato kapitola popisuje jak aplikaci používat a jak vytvářet podkladové soubory, aby jim aplikace rozuměla a správně podle nich vytvořila 3D model budovy / bytu.

6.1 Vytváření podkladových souborů

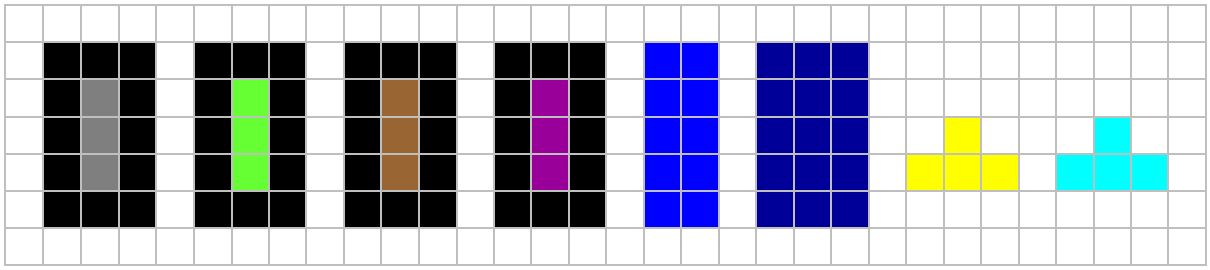
Při vytváření podkladových souborů je nutné dodržet zásady správných barev a typů značení objektů:

- Pro nakreslené objekty používat naprosto identické barvy, jaké jsou uvedeny v tabulce [Tabulka 1]. Aplikace porovnává jednotlivé barevné složky (R, G, B) a nemá nastaven žádný rozptyl, tedy pokud se jednotlivé složky zcela neshodují, nedokáže aplikace objekty správně rozpoznat.

Typ objektu	**typ značení**	**barva**
stěna	s okrajem	šedivá R=100 G=100 B=100
dveře	s okrajem	zelená R=0 G=255 B=255
stůl	s okrajem	hnědá R=150 G=100 B=50
výtah	s okrajem	fialová R=150 G=0 B=150
okno	bez okraje	modrá R=0 G=0 B=255
bazén	bez okraje	modrá R=0 G=0 B=150
televize	značka	žlutá R=255 G=255 B=0
židle	značka	modrá R=0 G=255 B=255

Tabulka 1 – přehled značení objektů

- Pro objekty používat správné typy značení, jak jsou uvedeny v tabulce [Tabulka 1] a zobrazené na obrázku [Obrázek 22].



Obrázek 22 – (zleva) stěna, dveře, stůl, výtah, okno, bazén, televize, židle

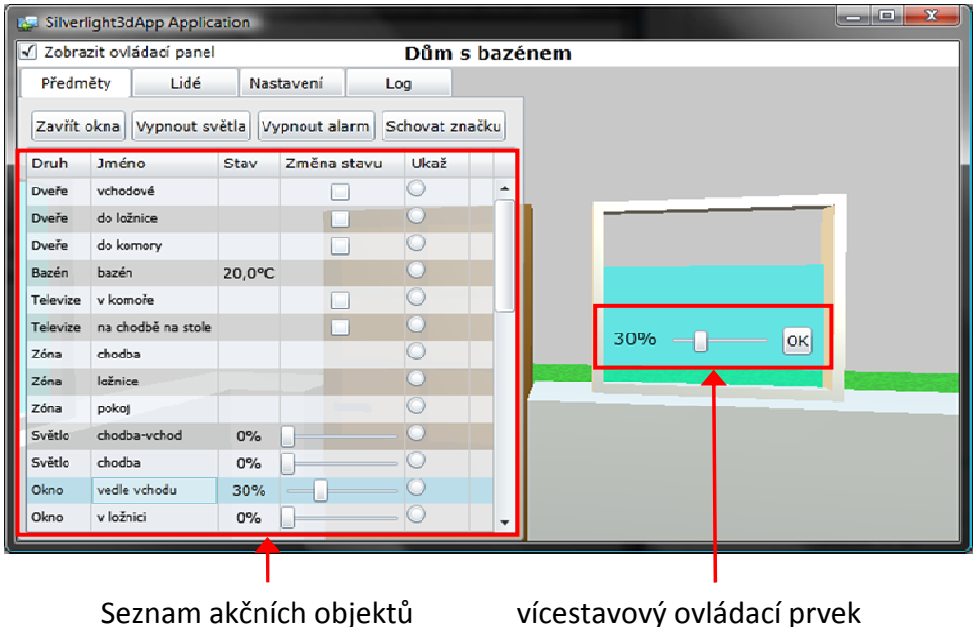
6.2 Ovládání aplikace

Ovládání aplikace jsem se snažil udělat co nejjednoduší a intuitivní, aby aplikace byla snadno pochopitelná a ovladatelná i pro netechnicky zdatné uživatele. I přes veškerou snahu se pravděpodobně najdou ovládací prvky, jejichž význam nemusí byt hned všem patrný a proto v této podkapitole uvedu manuál jak aplikaci používat.

Základní ovládání pohybu a otáčení 3D modelu budovy / bytu se provádí pomocí myši a klávesnice:

- otáčení modelu – pohybem myši zároveň se zmáčknutým pravým tlačítkem
- přiblížení / oddálení – kolečkem myši
- pohyb modelu v rovině – šipky na klávesnici
- pohyb modelu nahoru / dolu – se zmáčknutým „shift“ šipky nahoru / dolu
- otáčení kolem středu budovy – se zmáčknutým „shift“ šipky doleva / doprava
- přepínání aktivního patra – „PageUp“ / „PageDown“

Záložka Předměty
Akční objekty (okna, dveře,…) se ovládají kliknutím pravého tlačítka myši na daném objekt ve 3D scéně, nebo kliknutím na ovládací prvek v seznamu akčních objektů. Při kliknutí na objekt ve 3D scéně se buď hned změní stav objektu (dvoustavové objekty – zavřeno / otevřeno), nebo se nad objektem zobrazí ovládací prvek, kterým je možné objekt přepínat mezi více stavy.
V záložce jsou také tlačítka pro současné zavření všech oken v budově / bytu, vypnutí všech světel v budově / bytu, vypnutí alarmu a schování navigačního ukazatele.

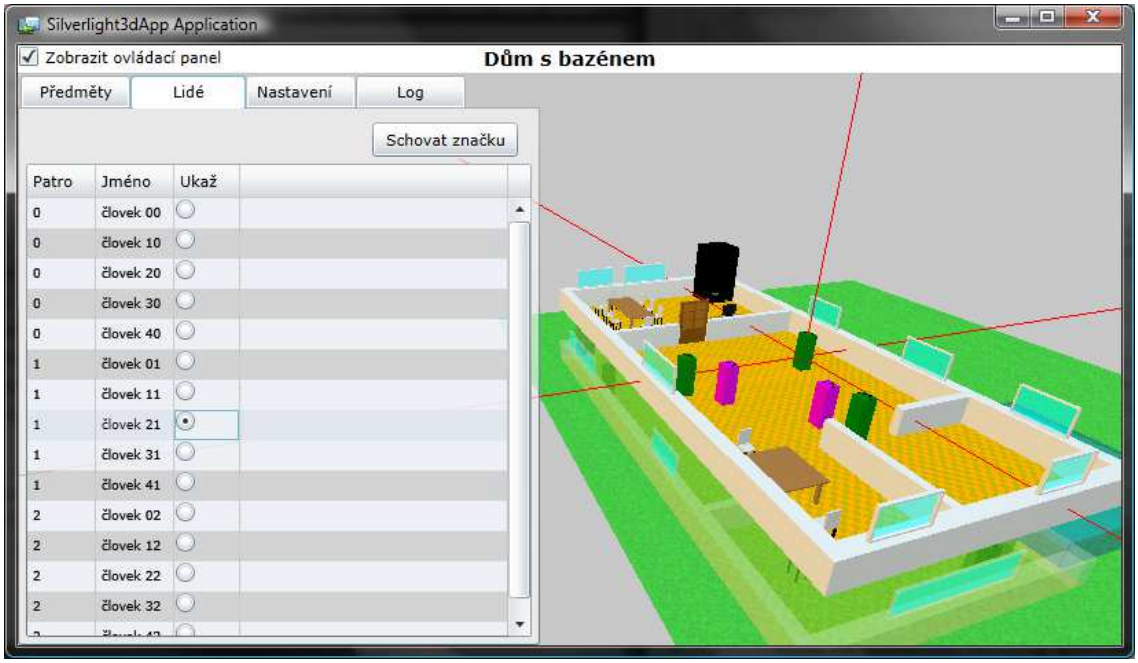


Obrázek 23 – Screenshot aplikace s otevřenou záložkou „Předměty“

Záložka Lidé

Tato záložka slouží k ovládání resp. lokalizaci lidí. Je zde uveden seznam všech residentů / ošetřovatelů a u každé položky je uvedeno jméno, patro ve kterém se nachází a zapínač ukazatele pro lokalizaci člověka ve 3D scéně. Na obrázku [Obrázek 24] je vidět, jak lokalizace hledané osoby vypadá.

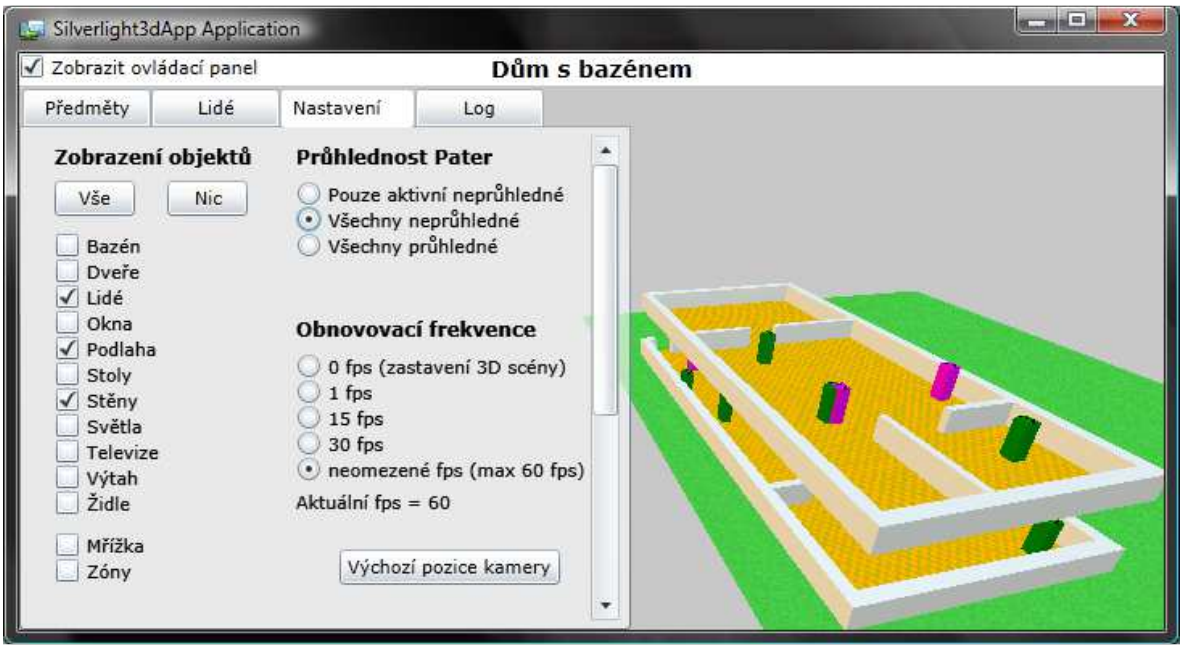
Tlačítko „Schovat značku“ slouží pro schování navigačního ukazatele.



Obrázek 24 – Screenshot aplikace s otevřenou záložkou „Lidé“ a lokalizací osoby

Záložka Nastavení
Slouží pro nastavení 3D scény.

- Zobrazení objektů – pouze zatržené objekty se budou zobrazovat
- Průhlednost pater – nastavení průhlednosti jednotlivých pater
- Obnovovaní frekvence – pro snížení zatížení počítače, je možné snížit obnovovací frekvenci 3D scény, nebo scénu úplně zastavit
- Tlačítko Výchozí pozice kamery – budova se zobrazí na stejném místě a pod stejným úhlem jako byla při zapnutí aplikace



Obrázek 25 – Screenshot aplikace s otevřenou záložkou „Nastavení“

7 Závěr

Hlavním cílem této práce bylo vytvoření webové aplikace pro nejen 3D zobrazení domů a budov, ale současně i jejich vybavení a pohybu osob, které se mohou v objektu nacházet. V tomto směru byl vytyčený cíl splněn a výstupem práce je tedy mnohostranně využitelná aplikace.

Mezi hlavní výhody jmenujme schopnost vytvoření plnohodnotného 3D modelu pouze z půdorysů, v podstatě map, jednotlivých poschodí. Každý půdorys může současně obsahovat i několik typů běžných zařízení jako jsou nejen dveře a okna, ale rovněž stůl a židle, nebo televizní přijímač. Tato schopnost aplikaci předurčuje skutečně pro široké využití a ne pouze pro graficky a technicky zaměřené uživatele. Další neopomenutelná výhoda spočívá v možnosti (téměř libovolné) interakce s uživatelem. Tomu je dovoleno nejen nastavit zobrazení 3D scény podle konkrétních požadavků (určité patro, určité vybavení), ale současně pomocí této aplikace pozorovat a ovládat reálné (vzdálené) okolí.

Právě poslední zmíněná výhoda bude využita v několika projektech. Například v projektu BOS pro zobrazení pohybu a tedy i sledování osob v budovách. Aplikace je schopna nejen zobrazit požadovanou osobu, tj. její pozici, ale současně ji tzv. zaměřit z pohledu uživatele / dozoru. To znamená zobrazit pomocné informace pro její nalezení / identifikaci (souřadnicové čáry).
Další projekt kde aplikace najde využití je snaha o vzdálené ovládání inteligentní domácnosti. Zde lze skutečně využít poskytovanou interakci pro uživatele. Aplikace bude tedy nejen poskytovat informace o aktuálním stavu, ale rovněž přijímat povely od uživatele za účelem změnu tohoto aktuálního stavu.

Pro základní ověření použitelnosti vytvořené aplikace pro dva zmíněné projekty, byl vytvořen prototypový server poskytující potřebnou komunikaci a částečně simulaci dat z reálného prostředí. Jako skutečnou simulaci můžeme považovat „náhodný“ pohyb osob a zobrazení jejich pozic. Naopak za skutečný příklad lze považovat testovací napojení na reálný domácí systém poskytující některé aktuální hodnoty a schopný přijímat požadavky od uživatele.
Jelikož se jedná o velmi praktickou práci, tak samozřejmě nelze pokrýt všechny a to zejména budoucí požadavky. Z tohoto důvod vznikl seznam poznatků získaných ze současného řešení, který bude vhodným základem pro budoucí vylepšování a rozšiřování. Jedním z hlavních úkolů do budoucna je bezesporu grafická úprava 3D scén neboli zobrazených předmětů. V průběhu práce byl kladen důraz zejména na princip a ověření navržených řešení než na skutečnou grafickou úpravu scény a v ní zobrazených objektů.

8 Seznam použité literatury

[1] Matthew MacDonald, *Pro Silverlight 5 in C#,*
Apress 2012, ISBN 978-1-4302-3480-7

[2] Aaron Reed, *Learning XNA 4.0*,
O'Reilly 2010, ISBN 978-1-449-39462-2

[3] Karli Watson et al. , *Beginning Visual C# 2010*,
Wiley Publishing Inc. 2010, ISBN 978-0-470-50226-6

[4] Ján Hanák, *Praktické objektové programování v jazyce C# 4.0*,
Artax 2009, ISBN 978-80- 87017-07-4

[5] Antonín Pošusta, *3D model pro vizualizaci / simulaci a ověřování řídících systémů*,
Diplomová práce, ČVUT FEL 2011

[6] *MSDN Library* [online], květen 2012
http://msdn.microsoft.com/en-us/library/

[7] Umesh Patel, *Basic 3D in Silverlight* [online], květen 2012
http://silverlight.bayprince.com/tutorials.php

[8] Christian Moser, *WPF Tutorial* [online], květen 2012
http://wpftutorial.net/

[9] VŠB-TU Ostrava, *Výukový kurz Microsoft Silverlight* [online], květen 2012
http://silverlight.cs.vsb.cz/

[10] společnost Papouch s.r.o. [online], květen 2012
http://www.papouch.com/cz

A Seznam použitých zkratek

XNA	vývojářská platforma společnosti Microsoft pro vývoj her v jazyce C# a VB
WPF	Windows Presentation Foundation, technologie pro vytváření uživatelského rozhraní
GPU	Graphic Procesor Unit, grafický procesor
HW	Hardware, veškeré fyzické technické vybavení počítače
XML	Extensible Markup Language, obecný značkovací jazyk
PNG	Portable Network Graphics, formát pro bezeztrátovou kompresi grafiky
SW	Software, veškeré programové vybavení počítače

B Obsah CD

\Silverlight3dApp.xap – zkompilovaná aplikace

\Silverlight3dAppTestPage.html – soubor pro spuštění aplikace

\3D zobrazení budov (BP2012).pdf – tato práce v elektronické podobě

\resources* – podkladové soubory, ze kterých se vytváří 3D model budovy

\setup\Silverlight.exe – instalační soubor pro Silverlight plugin