VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

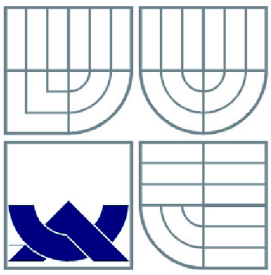
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KLIENT PRO 3D KLIENT/SERVER ŠACHY

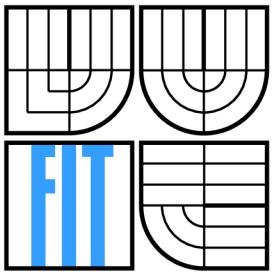
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE TOMÁŠ STARKA
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

Klient pro 3D klient/server šachy

Client for 3D Client/Server Chess

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ STARKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2009

Abstrakt

Bakalářská práce pokládá základní kamen projektu, který se zabývá hraním deskových her přes internet. Nastíněna je problematika deskových her, a pro jejich část je navržen systém postavený na aplikačním protokolu BGame, který byl za tímto účelem vytvořen. Jako testovací hra byly zvoleny Šachy a implementace je nad platformou Java.

Abstract

Bachelory work is a corner-stone for project which employs itself with playing board games via internet. It outlines some of board games issues and for a fraction of them is designed a system that stands on an application protocol called BGame which was created for that purpose. The Chess has been chosen as a testing game. Whole implementation is on Java platform.

Klíčová slova

Šachy, Desková hra, BGame protokol, sítě

Keywords

Chess, Board Game, BGame protocol, networking

Citace

Tomáš Starka: Klient pro 3D klient/server šachy, bakalářská práce, Brno, FIT VUT v Brně, 2009

Klient pro 3D klient/server šachy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Tomáš Starka

Datum (1. 5. 2009)

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu Ing. Adamu Heroutovi, Ph.D. za to, že mi umožnil dělat práci která mě bavila. Dále rodině a kamarádům za všemožnou pomoc a podporu.

Obsah

1 Úvod

Bakalářská práce Klient pro 3D klient/server šachy má za úkol vytvořit program pro hraní šachů mezi dvěma lidmi přes internet za pomoci šachového serveru a prozkoumat možnosti hraní deskových her přes internet s živými hráči. Jedná se o jednoduchý klient, který má za úkol zprostředkovat připojení k serveru, registraci hráče pod zvoleným jménem (přezdívkou, nickem), vyzvání jiného volného hráče a vzájemnou interpretaci tahů přes server. Zobrazování herní situace probíhá v 3D zobrazení vlastními zdroji, časem by však měl být schopen podporovat i externí renderování přes server.

Z rozsahu zadání vyplývá, že implementovat klientskou aplikaci nebude stačit. Aby bylo možné celý systém provozovat, je potřeba i serveru, který bude obsluhovat naše požadavky a udržovat povědomí o hraných hrách a hráčích. Zároveň je na serveru, aby posuzoval správnost prováděných tahů, průběžně vyhodnocoval stav hry a v případě jejího konce to zúčastněným oznámil.

Pokud máme i klient i server, potřebujeme navíc nejaký komunikační protokol, který by se dal v případě potřeby rozšiřovat i mimo šachy na ostatní deskové hry. Dále bychom rádi, aby byl v rámci možností spolehlivý, na druhou stranu hra není kritická aplikace jako například autopilot u letadla.

Dalším z požadavků je jednoduchá dostupnost a nenáročnost. Aby si uživatel nejlépe nemusel nic instalovat, aplikaci spustil přímo z okna nebo přímo v okně svého prohlížeče, a přitom se mu dostalo hry příjemného vzhledu ve 3D zobrazení.

Strukturu zprávy jsem zvolil trochu netradičně. Hlavním důvodem, proč se nedržet zaběhaného vzoru analýza a návrh, implementace a závěr, bylo usnadnit postup člověku, který by chtěl navázat některou z částí. Pokud by jej například zajímala pouze síťová část, přečte si pouze kapitolu, která se zabývá sítí. Kdo by chtěl například doplnit systém o externí renderování, bohatě mu postačí vědět jak funguje protokol, a která část GUI vykresluje scénu a jak scéna vypadá, ale už nepotřebuje vědět nic o deskových hrách, natož o implementaci herní logiky. Takto to je vymezeno již kapitolami samotnými.

2 Šachy jako desková hra

Z množství deskových her byly vybrány šachy pro svou netriviální složitost a značnou rozšířenost i popularitu po světě. V práci ani tak nejde o přesnou implementaci šachových pravidel, jako pokusit se navrhnout jednoduchý systém, který by se časem dal rozšířit o další deskové hry, či různé typy klientů, ať už textových nebo třeba pro malá mobilní zařízení.

2.1 Deskové hry

Deskovými hrami obvykle myslíme stolní hry, které se hrají na hracím plánu za pomocí figurek. Jsou tahové, tedy hráči se postupně střídají ve hře. Hrací plán bývá většinou pouze dvourozměrný a je rozdělen na hrací pole. Obecně se u deskových her zanedbává skutečná vzdálenost mezi figurkami a ty se pohybují po polích v diskrétních skocích.

K tomu, aby mohl být vůbec vytvořen nějaký univerzální systém na hraní deskových her, je potřeba si tyto nějak zanalyzovat a rozdělit. Jedno z užitečných rozdělení může zahrnovat herní principy, tedy co všechno v sobě hra kombinuje, a může vypadat například nějak takto:

- čistě deskové hry – používají pouze hrací plán a figurky. Typickým představitelem jsou právě Šachy, Dáma a nebo třeba Go),
- deskové hry s prvkem náhodnosti – kromě hracího plánu a figurek ještě potřebují nějaký generátor náhodných čísel, nejčastěji šestistěnou kostku. Typickým představitelem je Člověče, nezlob se!
- deskové hry „karetní“ – hrací plán se skládá z postupně vykládaných polí – kartiček. Mezi tyto hry patří Domino a Carcassonne,
- deskové hry kombinované – Tyto hry jsou nejsložitější variantou. Kromě kostek většinou potřebují i různé karty, žetony, počítadla apod. Patří mezi ně hry jako jsou Osadníci z Katanu, Starcraft board game, Arena a stovky dalších.

Další věcí, na kterou je třeba se podívat, je hrací plán. Hrací plán je nějaká množina *polí*. Jelikož však potřebujeme zachytit stavy hry, budeme za hrací plán považovat množinu stavů polí, podle jejich obsazení hracími „kameny“. Každé pole má minimálně 2 stavy. Jeden je prázdné pole a druhý je plné pole. Stavů může být samozřejmě více, například podle typu figurek. Abychom mohli efektivně kontrolovat tahy a abychom množinu stavů polí přiblížili skutečnému hracímu plánu, bylo by dobré prohlásit tuto množinu za uspořádanou n-tici. Třeba hra „Čověče, nezlob se!“, pokud bychom neuvažovali „domečky“, je vlastně cyklus. Můžeme ji tedy uspořádat jako n-tici polí od startovní pozice libovolného z hráčů po směru pohybu. Vypadalo by to následovně:

$$\begin{gathered} Hrací\ plán = (pole_1, pole_2, \ldots, pole_n) \\ pole_1, pole_2, \ldots pole_n \in \{prazdné, červený, modrý, zelený, žlutý\} \end{gathered} \tag{1}$$

Kde *n* je počet herních polí, barvy zobrazují figurky jednotlivých hráčů a *prázdné* značí pole neobsazené žádnou figurkou. *Hrací plán* nám udává přímo nějaký *stav*. Ten reflektuje rozmístění figurek na skutečném hracím plánu. Tohle nám však k reprezentaci celkového stavu hry nestačí. Kupříkladu v „Člověče, nezlob se!“, nám jedno pravidlo říká, že hráč který ještě od začátku hry nenasadil (nedostal svoji figurku na hrací plán), může házet až třikrát. K reprezentaci herního stavu je tedy potřeba ještě další matematické struktury, která by v sobě zahrnovala *Hrací plán* a informace o stavu hry jako celku. Budeme předpokládat, že něco takového máme k dispozici a nazveme to *herním stavem*.

Hra samotná sestává z tahů. Pokud máme aktuální herní stav, pak můžeme *tah* hráče pokládat za změnu tohoto stavu na stav jiný. Tedy bychom mohli psát:

$$Herní\ stav_x \Rightarrow Herní\ stav_y$$

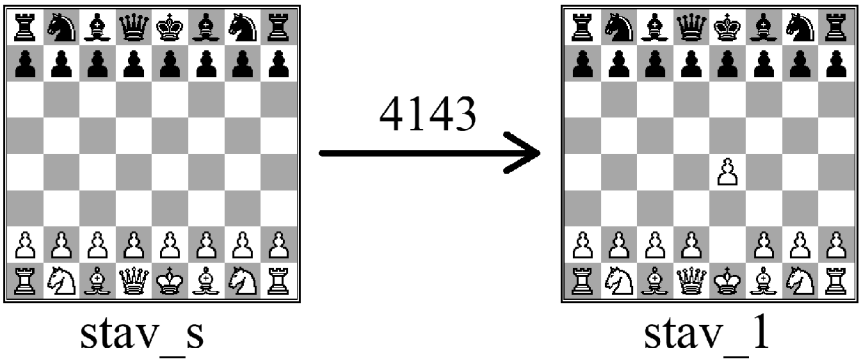
Dvojitá šipka značí přechod z jednoho stavu na druhý podobně jako by tomu bylo u stavových automatů či gramatik. Celý průběh takové partie čistě deskových her, by se dal definovat jako stavový automat *M*.

$$M = (Q, A, R, s, F)$$

Kde:

- *Q* je konečná množina herních stavů.
- *A* je vstupní abeceda, kde každý symbol reprezentuje nějakou akci, tah. Z implementačního hlediska je, u čistě deskových her, vhodné použít číselné vyjádření počátečního a cílového pole. Na šachovnici například „2123“ – jako xy počáteční a xy koncové pozice. Symbol vstupní abecedy lze také považovat za popis rozdílů mezi dvěma herními stavy.
- *R* je množina pravidel. Určuje, které tahy můžeme provést v daném herním stavu a jaký bude stav výsledný.
- *s* je počáteční stav, tedy rozmístění figurek na hracím plánu před prvním tahem.
- *F* je množina koncových stavů. Stavy kdy hra končí, např. v Dámě, když jeden hráč pozbyde všech svých figur, nebo se již nemůže pohnout platným tahem.

To, co nás nejvíce zajímá při návrhu implementace hry, jsou právě pravidla, často označovaná termínem *herní logika*. Můžeme si je vybavit jako trojici (počáteční herní stav, symbol abecedy, koncový herní stav). Pokud tedy v množině pravidel *R* nalezneme odpovídající trojici, můžeme prohlásit tah za platný a změnit aktuální herní stav z počátečního na koncový. Takovou množinu však při implementaci nemáme, a proto musíme vymyslet mechanismus, který nám určí, pro danou trojici, zda-li je, či není, pravidlem množiny *R*. Tímto se zabývají skutečná pravidla konkrétních her, která v co největší míře zobecňují pohyb figur a další náležitosti týkající se tahu. Nám je pak zbývá jen vhodně formalizovat.



Obr. 1: Příklad dvou stavů automatu.

Udělat systém pro hraní všech typů deskových her by bylo velmi složité. My se však omezíme pouze na hru v Šachy.

2.2 Šachy

Pravidly Šachů se zde zabývat nebudeme. Jako desková hra spadá do první kategorie z předchozího dělení. Díky svému rozšíření a popularitě už v dnešní době existuje velké množství různých počítačových implementací. Ať už jen jako hra pro ukrácení volného času nebo profesionální šachový stroj určený pro zdatnější hráče, jako například populární Rybka [1], kterou vytvořil V.Rajlich. My se však budeme zabývat pouze hrou dvou živých hráčů, protože úkolem práce není zkoumat možnosti umělé inteligence na poli deskových her. Spokojíme se se základní implementací herní logiky, jako jsou platné tahy, zjištění ohrožení krále, tzv. šachu, a konce hry, tzv. šachmatu. Druhů remíz je také více, my se spokojíme s remízou, která vyvstane ze vzájemné dohody mezi oběma hráči. Dále zanedbáváme různé šachové varianty. Tyto hry jsou šachům podobné. Mohou se lišit například: jiným hracím plánem (jiná velikost, jiný tvar apod.), fairy pices (zvláštní šachové figury s vlastními pravidly) a různě změněná další pravidla [2]. Stejně tak se spokojíme s časově neomezenou hrou.

Z hlediska hracího plánu z (1), můžeme pro pohodlnost pokládat šachovnici za matici stavů 8×8, idexujeme od 0 a osy nazvěme tradičně x a y. Úprava nám velmi zjednoduší testování platnosti tahů. Na platnosti předchozích úvah nic nemění, jelikož můžeme nadefinovat ekvivalentní zobrazení matice na uspořádanou n-tici *Hrací plán* za pomocí vhodné mapovací funkce. Stavy jednotlivých polí zde budou reprezentovat klasické šachové zkratky.

$$pole \in \{X, r, b, n, k, p, R, B, N, K, P\} \qquad (2)$$

Kde X je prázdné pole a dále klasicky r pro věž, b pro střelce, n pro koně, p pro pěšce, k pro krále a q pro dámu. Velká písmena pro bílého hráče a malá pro černého. Jako formalismus pro zjištění správnosti tahu, byla, díky převodu plánu na matici, zvolena závislost x na y, indexů koncového a počátečního pole. Například pro střelce platí: $|\Delta y| = |\Delta x|$. Dále třeba pro věž musí platit $(\Delta y = 0 \wedge \Delta x \in \mathbb{Z}) \vee (\Delta y \in \mathbb{Z} \wedge \Delta x = 0)$. Je samozřejmé, že pro každý tah je nutné zkontrolovat, zda figurka skončila na šachovnici. Tedy že koncové: $x \in \langle 0,7 \rangle \wedge y \in \langle 0,7 \rangle$, z čehož se dá vyvodit: $|x - 3.5| \leq 3.5 \wedge |y - 3.5| \leq 3.5$. Dalším nutným

kritériem je, aby políčka po cestě byla prázdná s tím, že na posledním poli může být figurka soupeřova. Vyjímkou je samozřejmě figura koně a rošáda.

2.3 Implementace herní logiky

O reprezentaci hry a herní logiku se stará jediná třída `ChessGame` v balíku j3dchess. Uvnitř se nachází de facto všechny věci, které jsme rozebrali v kapitole 2 a návazně pak v podkapitole 2.2. Aktuální stav hracího plánu reprezentuje proměnná board jako dvourozměrné pole znaků, které odpovídá znakům z *pole* definovaného v (2), kde znak X je nahrazen ‘x‘. `ChessGame` tedy reprezentuje kompletní herní stav, včetně možností rošády a en passant. Dále nám třída poskytuje nástroje jak kontrolovat zprávnost tahů. Metoda `isValidTurn(Move)` nám kompletně zhodnotí správnost provedeného tahu, tedy jeli možné cílového pole dosáhnout, zdali není na konci pohybu ohrožen náš král apod. Tato metoda vrací `true` pokud je tah validní podle pravidel. Naimplementovány jsou všechny druhy pohybů včetně rošád a braní mimochodem (en passant). Výměna pěšce za figuru, při dosažení opačného konce hracího plánu, zůstala nedořešena. Dále je potřeba provést daný tah metodou `move(Move)`, která změní stav hracího plánu. Konečně zavoláním metody `nextPlayerTurn()`, změníme hráče, který je na tahu. Další podrobnosti viz. zdrojové kódy a dokumentace.

3 Výběr platformy

Základním rozhodnutím snad každého projektu je, kterou platformu nebo platformy bude podporovat. V mém případě se jednalo zároveň i o volbu programovacího jazyka. S přihlédnutím na rozsah a nároky projektu, padla volba na objektově orientovaný přístup. Uvažoval jsem jazyk C++, se kterým mám relativně velké zkušenosti. Co se týče 3D knihovny, zajímal jsem se o Ogre. Dalším jazykem, který jsem chtěl blíže poznat, byla Java. Oba jazyky mají, dle mého názoru, dobrou podporu. Jedná se ať už o dokumentaci nebo o tutoriály, které jsou volně k dispozici na internetu. Nicméně při zvážení požadavku, aby byla aplikace jednoduše distribuovatelná přes webové stránky, považuji za výhodnější použít právě druhý zmíněný jazyk.

3.1 Java

Java je jazyk pro stejnojmenou platformu vytvořený a udržovaný firmou Sun Microsystems. Hned na začátku bych zmínil, že jsem projekt vyvíjel pod verzí Java Standard Edition Version 6 Update 7. Java nabízí pěknou kompaktní platformu, která by měla být nezávislá na operačním systému, a která si s sebou už ve standardní edici nese spoustu užitečních balíčků a utilit, které jsou dobře zdokumentované a na jednom místě. K dispozici je volně i zdrojový kód těchto balíků. Zároveň jsou na stránkách firmy Sun dostupné anglicky psané tutoriály, které postihují většinu věcí ze standardní edice a jsou poměrně aktuální. Jedním z negativních dopadů snahy o přenositelnost Javy je její neschopnost kompilace kódu pro platformu procesoru. Java se kompiluje do bytekódu, který je poté interpretován tzv. Virtuální mašinou (*JVM*). Célé to znamená, že chod programů bývá pomalejší a paměťově náročnější než u běžných aplikací psaných například v jazyce C. Dále také nutí uživatele instalovat si na počítač prostředí pod kterým běží. V dnešní době je už ale Java natolik rozšířená, že spousta uživatelů ji už nainstalovanou má. Ta se pak sama stará o aktualizace. Dále se podíváme blíže na některé její součásti, které jsem použil nebo zvažoval

3.1.1 J3D API

Toto API slouží jako nádstavba nad *AWT*, což je jsou balíky pro práci s 2D grafikou, a poskytuje nám jakýsi 3D engine pro Javu. Původně bylo J3D vyvíjeno uvnitř firmy Sun, nyní ale běží jako komunitní projekt na **http://java3d.j3d.org/**, kde se také dá pořídit poslední verze ke stažení i se zdrojovými kódy. J3D pak k samotnému renderingu používá buďto OpenGL nebo DirectX, podle toho co nalezne v operačním systému. J3D API již není součástí standardní edice Javy a je nutné jej doinstalovat zvlášť z výše uvedeného zdroje. V práci jsem použil verzi 1.5.2.

3.1.2 Java Web Start

Díky této technologii, mohou být jakékoliv aplikace psané pod Javou spouštěny přes web, v podstatě jedním kliknutím. Tato technologie je přímo součástí *JRE*. Podmínkou je, aby byla daná aplikace zabalená do *JAR* formátu. Zároveň je potřeba vytvořit *JNLP* soubor, který upřesňuje co má být odkud staženo, jakou verzi Javy potřebuje a další věci, viz. stránky firmy Sun. Na tento soubor pak jednoduše odkážeme na webové stránce pomocí HTML tagu nějak takto:

```
<a href="cesta/k/souboru.jnlp">Spusť aplikaci</a>
```

Také bývá potřeba webovému serveru říct, jak má s .jnlp souborem zacházet, kupříkladu pro Apache servery toho docílíme přidáním příkazu buďto do konfiguračního souboru mime.types:

```
application/x-java-jnlp-file jnlp
```

nebo pak do souboru .htaccess:

```
AddType application/x-java-jnlp-file JNLP
```

Po stáhnutí a spuštění takového souboru se automaticky aktivuje Java, která podle informací v něm obsažených sama stáhne a spustí aplikaci. Ta pak běží pod tzv. *sandboxem*, který takto spuštěnou aplikaci omezuje z bezpečnostních důvodů. Mezi tyto omezení patří například nemožnost přístupu k lokáním diskům apod. Při vytváření nových síťových spojení přímo z aplikace, je uživatel upozorněn a dotázán, zda-li chce spojení povolit či nikoliv. Pokud bychom z nějakého důvodu potřebovali, aby aplikace fungovala normálně bez těchto omezení, musíme jar soubor podepsat. Při spuštění je pak uživatel dotázán zda podpisu věří. Toto ale není náš případ. J3DChess si nese vše potřebné zabalené už v samotném jar archivu.

3.1.3 Applet

Další zvažovanou technologií byl Java Applet. Takto se dá program napsaný v Javě spouštět přímo uvnitř stránky. Jakmile webový prohlížeč, který podporuje Javu, narazí na tag `<APPLET>`, přenese kód programu na klientský počítač a sputí ho podobně jako Web Start. Problémem však je, že applet běží uvnitř prohlížeče a tedy musí na něj spoléhat co se týče např. vykreslovaní a dalších věcí. Je tedy hodně závislý na konkrétním prohlížeči, a to snižuje jeho přenositelnost. Kupříkladu v Opeře verze 9.27 se nerenderovala séna vytvořená pomocí J3D API, zatímco v Internet Exploreru verze 6.0 ano. Z tohoto důvodu jsem od tohoto jinak elegantního řešení upustil a použil raději Java Web Start, který prohlížeč používá pouze jako prostředek k stáhnutí konfiguračního JNLP souboru a běh pak svěří už samotné JVM[3].

4 Síťová část

Nyní se konečně blíže podíváme na aplikaci samotnou a začneme síťovou částí. Vzhledem k tomu, že hra funguje přes internet, je síťová komunikace důležitým aspektem celé aplikace. Vezmeme-li v potaz klasické požadavky jako jsou: bezpečnost, spolehlivost, včasnost, přišlo mi za transportní protokol vhodné zvolit TCP. Tento protokol se nám sám stará o spolehlivost, a tak odpadá nutnost obsluhy na vyšší vrstvě, což je ovšem stále možné, ale o tom až dále. Nic nám ovšem do budoucna nebrání postavit komunikaci nad například nespojovým UDP protokolem.

4.1 Aplikační protokol

Pro komunikaci klientů se serverem je potřeba stanovit nějaký protokol na aplikační vrstvě ISO/OSI, podle kterého si budou obě strany vzájemě vyměňovat informace. Při zkoumání šachových protokolů jsem nejčastěji narážel na dvojici XBoard(WinBoard) a UCI. Tyto protokoly však slouží ke komunikaci GUI s tzv. šachovým enginem. Což je program, který obstarává hru, tedy zpracovává naše tahy, umožňuje pracovat s umělou inteligencí apod. Sice se takovýto engine v něčem podobá našemu hernímu serveru, nicméně do budoucna by nám nepostačoval. Zvláště pokud bychom chtěli hrát i jinou hru než šachy nebo například pokud by se server staral i o ono externí renderování scény.

Bylo tedy potřeba navrhnout vlastní jednoduchý protokol. Tento jsem pojmenoval BGame protokol (z angl. Board Game), přičemž jsem měl na mysli jeho rozšiřitelnost pro pojetí více deskových her zároveň. Analýza deskových her však nebyla předmětem práce, a proto jsem vytvořil podmnožinu protokolu, zaměřenou na hru Šachy. Tento nese název J3DChess protokol, podle názvu hry samotné. Tedy to, co by platilo pro BGame, platí i pro J3DChess protokol. Udělat dobrý protokol se vším všudy by bylo časově i jinak náročné, proto je tento protokol spíše návrhem, jak by to v budoucnu mohlo vypadat. I přes své nedostatky našemu účelu postačuje. Protokol ve většině kritických situacích spoléhá na protokoly nižších vrstev. Například ve věcech doručení zprávy. Proto byl implementován nad spojovým protokolem.

Celkům BGame protokolu budeme říkat *příkazy*. Ty se skládají z hlavičky, zprávy a přídavných informací.

4.1.1 Hlavička

Obsahem hlavičky by měl být název protokolu (tedy „BGame“) a jeho verze. Dále pak sekvenční číslo, identifikátor hry a typ (název) hry.

Sekvenční číslo
Umožňuje kotrolu pořadí příkazů na aplikační vrstvě. Zatím není využíváno. Může být však velmi užitečné, pokud bychom používali externí rendering. Jednoduše bychom si pamatovali číslo poslední žádosti o rendering, které by nám pak server zopakoval v odpovědi. Renderování může být zdlouhavé a pravděpodobně bude mít delší odezvu. Tak se může například stát, že uživatel pohne s kamerou scény, ještě než se mu starý pohled vrátí vyrenderovaný. Toto se může stát několikrát, uživatel třikrát pootočí kamerou a zároveň přijde soupeřův tah. Teprve potom se vrátí vyrenderovaná scéna, která už však neodráží

aktuální stav. Sekvenční číslo nám pak umožní toto zjistit a opožděné rendery, které by již nekorespondovaly se scénou u klienta, zahodit.

Identifikátor hry
Jedinečný identifikátor rozehrané partie. Mělo by stačit, pokud bude jedinečný v rámci herního typu. V budoucnu může slouži jako identifikace hry při různých příležitostech jako například: Ukládání rozehrané hry na serveru, při použití nespojových protokolů bude-li možnost aby jeden hráč hrál více her současně apod.

Typ hry
Jednoznačný identifikátor hry. Bude sloužit jako součást identifikace partie. Zároveň se podle toho pozná, které hry a jejich varianty server podporuje. Proto musí být standardizovány v rámci protokolu. Například to může být „chess“ pro šachy či „draughts“ pro dámu apod. Pro J3DChess používám typ „CHESS“.

4.1.2 Zprávy a dodatečné informace

Podstatou každého příkazu je typ zprávy a její kód, který nám určuje o jaký požadavek jde. Jedná se z části o model request-response (požadavek-odpověď), ale jsou zde i výjimky, například právě poslání soupeřovat tahu serverem není iniciováno žádostí klienta. Kód zpávy je právě to, co ve většině případů rozlišuje mezi žádostí, potvrzením a odmítnutím.

Kód zprávy
Kód zprávy je celé číslo (integer). Pokud nebude výslovně uvedeno jinak, paltí u zpráv obecně toto:

- 0 – je požadavek (request)
- 1 – je potvrzení (kladná odpověď tzv. ACK)
- -1 – je zamítnutí (záporná odpověď tzv. NAK)

Je samozřejmě možné v dalším vývoji prohlásit cokoliv kladného jako ACK a cokoli záporného jako NAK. Rozhodl jsem se nezobecňovat, jelikož se může časem ukázat, že v určité deskové hře, která bude třeba v budoucnu podporována, bude výhodné použít kód jako přídavnou informaci. Například u zprávy typu MOVE, může být užitečné považovat cokoliv kromě 0 a -1 jako ACK a podle toho se nějak zařídit.

Typy zpráv a dodatečné informace
Nyní se podíváme na samotné typy zpráv. Jsou to ty, které jsou implementovány a byly použity ke hře. Je to tedy jen nutný základ. U každé zprávy si rovnou popíšeme které dodatečné informace nese.

Jednou z dodatečných informací, která může být přítomna u všech typů zpráv, je tzv. textová část. Tato se používá jako prostředek k interakci s uživatelem. Pokud například server zamítne náš požadavek, může v textové části uvést důvod. Tuto součást by měl každý klientský program interpretovat uživateli například formou dialogu, který mu zobrazí.

register
Tato zpráva nám zprostředkuje registraci hráče na herním serveru. To znamená přihlášení se. Přídavné informace jsou samozřejmě nějaký jednoznačný identifikátor hráče. V našem případě je to přímo přezdívka (nick). V rozšíření se můžou vyskytovat třeba i autentizační údaje, pokud by byly účty trvalé, nebo nějaký presence status, jaký známe například z různých IM (Instant Messenger).

players
Zpráva, která má za úkol poskytnout hráči úplný seznam všech přihlášených/registrovaných hráčů, včetně něj samotného, pokud tak učinil. Zde jsem například udělal výjimku a jako kód zprávy v kladném případě posílám počet takovýchto hráčů zvýšený o jedna. Součástí zprávy je seznam hráčů. V našem případě si opět vystačíme s přezdívkami. V budoucnu je ale možné posílat i další informace, například různé statistiky.

invite
Ustavuje herní sezení mezi hráči. Z technického hlediska je to typický three-way handshake. Hostící hráč pošle pozvánku druhému hráči, což je zpráva s kódem 0. Ten buďto výzvu přijme nebo odmítne. Pokud přijme, je ještě na hostícím hráči, aby potvrdil. Hostící hráč samozřejmě může vzít pozvánku zpět dříve, než se pozvaný rozhodne, zda přijímá či nikoliv. Tento mechanizmus je zde proto, že hráči nemusí vždy reagovat svižně, a tak se může hostící hráč rozhodnout raději pozvat někoho jiného. Z implementačního hlediska je zrušení pozvánky příkaz invite s kódem -1. Součástí zprávy je samozřejmě identifikace hráče, kterému pozvánka nebo odpověď patří. Ta se na serveru změní v identifikaci hráče, který ji poslal. Do budoucna zbývá promyslet, jak bude probíhat ustavování sezení při hře více jak dvou hráčů a co se stane, pokud někteří přijmou a někteří odmítnou.

move
Konečně příkaz move slouží k odesílání tahů. Ve skutečnosti se spíše jedná o žádost k provedení tahu. Klient odešle tah na server, ten jej vyhodnotí a v případě, že je validní, odešle potvrzení všem zúčastněným hráčům. Pokud tah v pořádku není, pošle se zamítnutí odesilateli. Součástí je informace odkud kam se táhlo. J3DChess protokol toto rozšiřuje o stav hry, tedy: nic, šach nebo šachmat. Jedná se jen o experimentální rozšíření.

draw
Zprávu draw použijeme v případě, že chceme soupeřovi nabídnout remízu. Je samozřejmě vhodné napsat i důvod do textové části. V J3DChess se to postará vstupní dialog v GUI. Pokud protivník nabídku přijme, pak je hra u konce. V opačném případě hra normálně pokračuje dál. Tento typ zprávy nemá žádné speciální součásti.

system
Tento typ zpráv se používá jako mimoherní sdělení nějaké události. Například pokud váš oponent nečekaně ukončil spojení se serverem. Zde se kód zprávy využívá jako pomocný identifikátor, stejně jako třeba kódy zpráv v smtp protokolu. Vzhledem k velmi řídkému použití zatím nebyly tyto kódy nijak standardizovány. V implementaci J3DChess protokolu existuje pouze zpráva 501 značící právě odpojení oponenta od rozehrané partie. V číslování napodobuji smtp, kde zprávy 5xx značí permanentní chybu.

Opět je to spíše návrh do budoucna. Žádné další součásti zpráva nemá, ale měla by vždy využít textové části.

4.2 Prezentační protokol

Na otázku jak můžeme reprezentovat BGame protokol, je vícero odpovědí. Teď bych ve zkratce rozebral ty, nad nimiž jsem uvažoval.

Binární
Jako první z možností jsem zvažoval binární formu. Java má totiž velmi pokročilé serializační mechanismy, které dobře fungují bez jakéhokoliv nastavování. Stejně tak však umožňují téměř absolutní kontrolu nad serializčním procesem, a jak už bývá u Javy obvyklé, vše je pěkně popsáno. Takto bychom mohli udělat jednu velkou třídu Příkaz a celou ji poslat po síti. Problém však nastane, pokud jedna z applikací nebude napsaná v Javě. Toto není neřešitelné, jen znevýhodňuje aplikace psané v jiných jazycích, kde by se musela ať už serializace nebo deserializace doplnit tak, aby korespondovala s Javou. Toto je obecně problém posílání binárních dat reprezentující struktury.

Textová
Čitelný text má hned několik výhod. Dá se jednoduše ladit vzhledem k tomu, že stačí vypsat na konzoli to, co nám po síti přišlo. Dále například pokud by někdo v budoucnu napsal jednoduchý textový klient, dají se příkazy pro server psát do konzole. Problém pak nastává ve formálním definování takového protokolu. Bylo by zapotřebí gramatiky a navíc bychom si museli sami dopsat kontrolu korektnosti příkazů.

XML
Nakonec padla volba na XML. Je čitelné, i když kvůli značkám mírně obtížněji. Navíc je podporováno ve většině vyšších programovácích jazycích. Ty nám většinou nabízejí parsování do nějaké podoby DOM, se kterou můžeme dále pracovat. XML samo nabízí různé formalismy pro kontrolu správnosti, jako jsou DTD či XML schema. Tyto pak například Java dokáže využít ke kontrole správnosti, a tak nám ušetří spoustu práce.

K implementaci jsem si po prostudování materiálů na internetu vybral JAXP, což je jedno z API pro práci s XML, které vytvoří objektovou strukturu podobnou DOM stromu a umožňuje nám ji pak procházet a měnit. Jsou i další možnosti jak parsovat XML v Javě a další jak serializovat. Vzhledem k neúplnosti protokolu, nebylo XML specifikováno nijak formálně. Nyní se však pokusím přiblížit jeho podobu a na příkladu vysvětlit. V úvodu se sluší zdůraznit, že je parsování XML dokumentu **implementováno case-sensitive**. Je tedy důležité dodržet přesně názvy značek a atributů včetně hodnot, tak jak jsou zde uváděny. Za aplikačního programátora toto však samy dělají třídy balíku j3dchess.net. Teď k jednotlivým značkám:

- `?xml` – Úvodní značka tradičně udává kódování a další parametry XML dokumentu.
- `proto` – Hlavním tagem který obepíná celý dokument je `<proto>`. Ten má následující povinné parametry – name a version. Vypadat by měl nějak takto: `<proto name="BGame" version="0.1">`.

- `header` – Uvnitř je jediný tag <header> reprezentující hlavičku protokolu, viz. 4.1.1. S parametry `game-type` (typ hry, pro nás CHESS), `seq-num` (sekvenční číslo, které se ale zatím k ničemu nepoužívá a je možné jej vynechat) a `gameID` (sem se při vytváření herního sezení zapíše identifikátor hry, jinak je prázdné).
- `message` – Vnořený v header, je tag `<message>`. Jež má dva atributy: type (typ zprávy, tak jak jsou vypsány v 4.1.2.) a code (ten reprezentuje kód, celé číslo integer, opět podle 4.1.2). Navíc může obsahovat textovou část tedy text mezi `<message>` a `</message>`.
- Další tagy jsou přidávány na úroveň tagu message a nesou doplňující informace, viz 4.1.2. Budou uvedeny rovnou jako příklad:
  - `<player nick="nick" />` – tag player má pouze jeden, zato povinný atribut nick, kde uvedeme přezdívku hráče, která zároveň slouží i jako jednoznačná identifikace. Pokud bude systém rozšířen o jinou formu identifikace, bude nutné protokol v tomto místě pozměnit.
  - `<move from="45" to="46" state="NONE">` – toto je příklad rozšíření BGame o `state` používaný v J3DChess. State může nabývat pouze hodnot, jaké určuje implementace enumu j3dchess.Move.State, tedy přesně: `NONE`, `CHECK`, `CHECKMATE`. Správně by se před připojením klienta mělo takovéto rozšíření nejprve dohodnout, náš případ toto pro zjednodušení zanedbává. Povinné parametry pro BGame jsou pouze `from` a `to`, které by měly obsahovat řetězec jasně identifikovatelný rozhraním `j3dchess.Movable`, tak aby ukazovaly odkud kam byl tah proveden.

Nyní uvedu pro ilustraci příklad dotazu a odpovědi na registraci hráče na serveru.

Žádost

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<proto name="BGame" version="0.1">
     <header game-type="CHESS" gameID="" seq-num="1">
          <message code="0" type="register"/>
          <player nick="pepa"/>
     </header>
</proto>
```

Odpověď

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<proto name="BGame" version="0.1">
     <header game-type="CHESS" gameID="" seq-num="1">
          <message code="1" type="register">You have been
               successfuly registered.
          </message>
          <player nick="pepa"/>
     </header>
</proto>
```

4.3 Implementace síťové části

Implementace všech pomocných tříd, které souvisejí se síťovou částí naleznete v balíku `j3dchess.net`.

4.3.1 Protokol

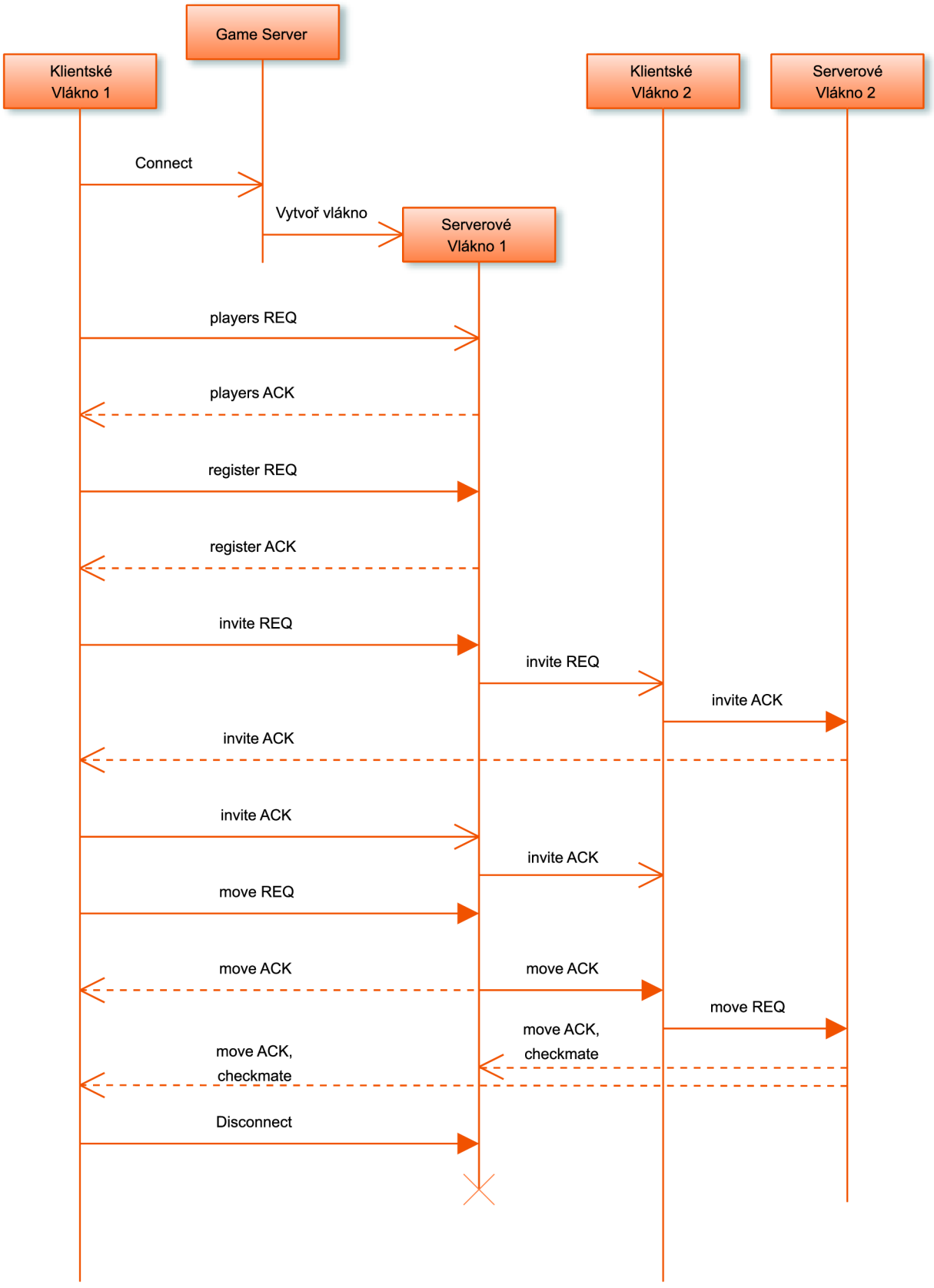
Samotný protokol je rozdělen do několika pomocných tříd a rozhraní. Jedna část se stará o výstupní formát a druhá o parsování vstupního.

Výstup obstarává třída `J3DChessProto`, která obsahuje soustavu metod pro transformaci dat. Ty následně putují přímo do socketu. Každý příkaz je zde reprezentován metodou, která sestaví požadavek jako XML dokument, serializuje jej jako řetězec a vloží do `ByteArrayOutputStream`, a ten posléze vrátí. Navíc také obsahuje metodu která nám z požadavku players vrátí `Vector<String>` vrácených jmen hráčů ze serveru. Objekt této třídy v sobě uchovává část dokumentu, která se mění jen zřídka. Ušetříme tak čas, který by byl zapotřebí k znovuvytváření celého Document objektu.

Vstupní část je dělena poněkud složitěji. Přijímání dat obstarává metoda doc`FromChannel()` třídy `ChannelXMLReader`. Je to statická metoda a slouží pouze k pročištění kódu. Tato, spíše funkce, čeká až po síti přijde celé jedno XML jakožto řetězec, a z něj pomocí `DocumentBuilder` vytvoří `Document`, který následně vrátí. Pokud nastane nějaká chyba, funkce vrátí null. Při zjištění ukončení spojení vzdálenou stranou uzavře navíc spojení. K parsování dokumentu slouží třída `J3DChessProtoXMLParser`. Tato třída má jediný úkol, a to rozpoznat příkaz a zavolat příslušnou metodu *callback* objektu a jí předat parametry. V budoucnu by bylo lepší, aby, pokud bude více parserů, tedy více prezentačních formátů, vzniklo nějaké standardizované rozhraní. Pravděpodobně by obsahovalo jen metodu `parse(Object)`. Teď již k samotnému callbacku, jedná se o implementaci `BGameMessageProcessor` rozhraní a z něj oddědeného `J3DChessMessageProcessor`, které je pak implementováno třídou `J3DChessMessageProcessorAdapter`. Adaptér je pouze prázdnou implementací všech metod rozhraní. Jeho jediným účelem je být předkem skutečné implementace, která si pak může vybrat a naimplementovat pouze to, co se hodí. Tak například, když se do protokolu přidají nějaké další příkazy, aplikační programátor v podstatě změnu nepocítí.

4.3.2 Server

Server je jednoduchá konzolová aplikace tvořená třídou J3DChessServer. Není součástí hlavní aplikace. Spouští se bez parametrů a příchozí spojení čeká na portu 5600. Server je konkurentní, pro každé příchozí spojení vytvoří nové vlákno, které obsluhuje spojení. Server zároveň spravuje hru, což nejčastěji znamená, že kontroluje správnost tahů a zaznamenává změny na šachovnici. Vytváření hry probíhá, jak už bylo popsáno, pomocí tzv. three-way handshake, který je podrobněji vidět na obrázku 2.



Obr. 2: Příklad životního cyklu serverového vlákna.

Na obrázku jsou vidět dvě vlákna. To proto, že server vytváří jedno vlákno pro každé sezení. Ve chvíli, kdy na server dorazí pozvánka, najde objekt vlákna referenci na vlákno soupeřovo. Tak rovnou přistupuje k oponentově socketu, který je zapouzdřený třídou `SocketChannel`, a zároveň i k dalším položkám. Pokud je sezení ukončeno je reference vymazána. Vlákno se ukončí jakmile dojde k odpojení klienta.

4.3.3 Klient

Síťový klient sestává z vlákna vnořeného ve vlastní hlavní třídě aplikace `J3DChessMainFrame`. Vlákno je reprezentováno třídou `J3DChessClientThread`. Tato třída obsluhuje síť a hru samotnou. Vlákno je vytvořeno se startem aplikace a spuštěno při prvním pokusu o připojení. Jeho hlavní smyčka se stará, stejně jako je tomu u serveru, o síťové spojení a příchozí data. Odesílání zabezpečuje hlavní vlákno aplikace skrze metody klientského vlákna. Třída `J3DChessCallback` se stará o reakci na zprávy přijaté ze serveru. Je opět vnořená v `J3DChessMainFrame`, kvůli lepší přístupnosti nejrůznějších prvků, jako například GUI.

5 J3DChess Aplikace

Spustitelnou třídou aplikace je `J3DChessApp`. Ta však pouze vytvoří instanci `J3DChessMainFrame`, která má na svědomí běh celé klientské části. Tato třída spojuje dohromady celou aplikaci. Jednotlivé části budou podrobněji popsány v následujících podkapitolách.

Jelikož je `J3DChessMainFrame` základní třídou okenní aplikace, musí dědit z jedné ze tříd schopných vytvořit okno. Tyto třídy jsou tři. Jsou to tzv. *Top-Level containers* [4], a byla z nich použita třída `JFrame`. Okno je pomyslně rozděleno na tři části [obr-deleni], které rozebereme v podkapitole 5.1. Zde jenom ve zkratce uvedeme části, ze kterých je třída složena. Jedná se především o třídy vnořené a metody, které nám inicializují aplikaci, nebo jinak pomáhají k jejímu běhu. Mezi již zmíněné vnořené třídy patří `J3DChessClientThread` a `J3DChessCallback`, které jsou rozebrány v kapitole 4.3.3. Mezi další patří třídy obsluhující GUI komponenty odvozené z AbstractAction. Tyto budou popsány v 5.1.3. Dále pak obsahuje metody, které inicializují 3D scénu, propojují a vytváří některé GUI prvky. Bližší náhled poskytuje dokumentace a zdrojové kódy.

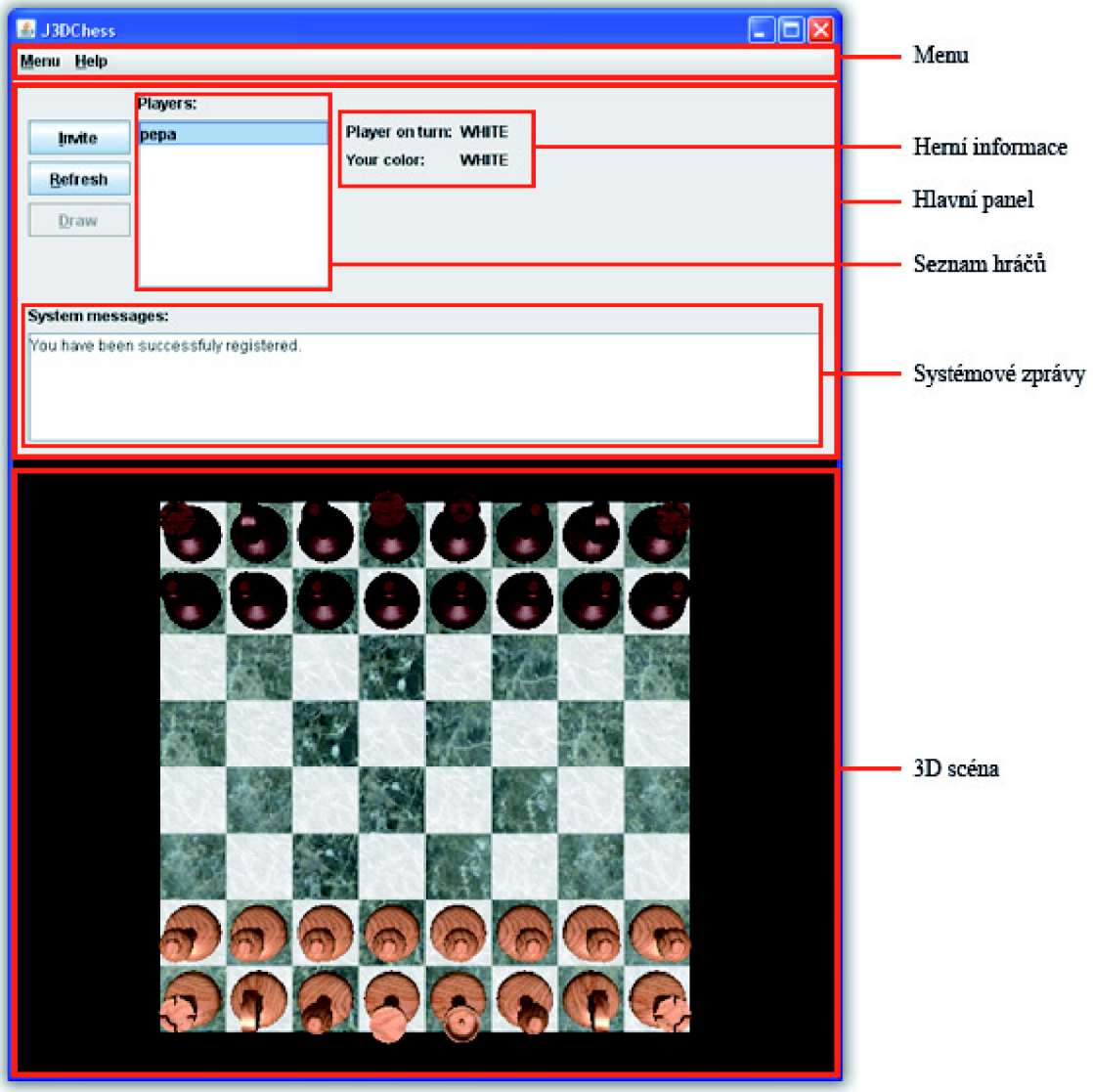
5.1 GUI

Pro GUI máme v Javě v podstatě dvě možnosti. Buďto se spolehneme na klasické *AWT* [5] nebo zvolíme *Swing*. O AWT (Abstract Window Toolkit) se často hovoří jako o heavy-wight rozhraní. Tvoří základ i pro Swing, Ten je často referován jako light-weight a je součástí JFC (Java Foundation Classes), což je skupina balíků pro sestavení GUI a grafické funkcionality a interaktivity do aplikací psaných v Javě [6]. Rozhodl jsem se pro swing, jelikož poskytuje více možností, a je takovou propracovanější a novější verzí AWT.

Jak již bylo řečeno, hlavní okno aplikace reprezentuje třída `J3DChessMainFrame`, která se však sama stará o další věci. Kromě hlavího okna a příležitostných popup dialogů stojí za zmínku Connection Dialog. Dialog který umožňuje uživateli se připojit, odpojit a registrovat na serveru. Funkčnost prvků zajišťují tzv. *Actions* [7]. Celé GUI používá výchozí styl Javy, což je *Java look and feel*. Zároveň je nutné vzpomenout, že GUI, stejně jako zdrojové kódy, jsou v angličtině, která je univerzálnější. Téměř u všech použitelných komponent se po najetí myši zobrazí kratičký popisek, který uživateli napovídá, co která komponenta dělá.

5.1.1 Hlavní okno

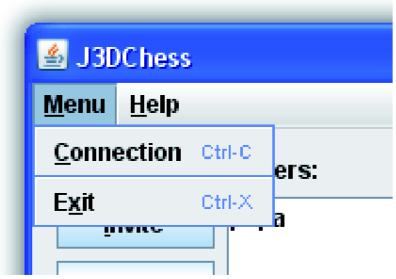
Hlavní okno se skládá ze tří částí: Menu, Hlavího panelu a 3D scény. Pro lepší názornost následuje obrázek. V hlavním okně byl použit `GridbagLayout` [8].



Obr. 3: Obrázek části hlavního okna

Menu je jednoduché. Vytváří jej metoda `createMenu` třídy `J3DChessMainFrame` . Obsahuje dvě hlavní položky:

- Menu
  - Connection, které zobrazí Connection dialog, viz. 5.1.2. Lze také použít klávesovou zkratku Ctrl+C
  - Exit, která ukončí aplikaci. Klávesová zkratka je Ctrl+X
- Help
  - How to, tato položka zobrazí malou nápovědu, jak zahájit hru.



Obr. 4: Obrázek menu

Krom klávesových zkratek, které jsou uvedeny v Menu, fungují ještě tzv. *Key Accelerators*. Tedy klasické použití klávesy Alt a písmenka které je podtržené v názvu/popisku komponenty, jak to známe z ostatních GUI systémů.

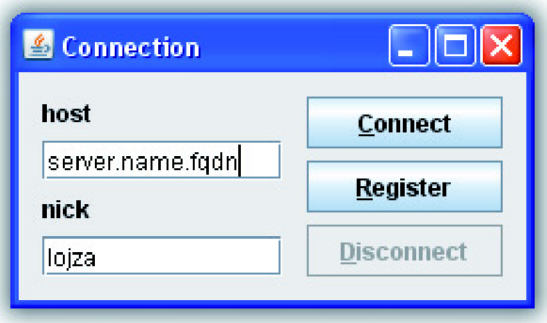
Hlavní panel je implementován jako potomek `JPanel` uvnitř balíku `j3dchess.gui` s názvem `MainJPanel`. Jako layout panelu byl zvolen `GroupLayout` pro svoji univerzálnost, i když za ni platíme zvýšenou obtížností [9]. K jednotlivým komponentám:

- Tlačítka – jsou implementovány jako `JButton` a jejich funkčnost obstarává vždy přidělená akce, s podobným názvem. Akce se komponentám přiřazují v metodě connectGUI uvnitř třídy `J3DChessMainFrame`.
  - Invite – klient pozve hráče vybraného ze seznamu hráčů. Pokud není žádný hráč vybrán, nebo se akce nezdaří, je na to uživatel upozorněn. Tlačítko odesílá příkaz *invite*, viz. 4.1.2. U hostujícího hráče se toto tlačítko změní na *Cancel Invite*. Tímto může vzít uživatel pozvání zpět.
  - Refresh – Toto tlačítko umožňuje obnovit seznam hráčů.
  - Draw – Jakmile je navázána hra mezi dvěma hrači, je možné aby jeden z nich nabídl druhému remízu. Po stisknutí tlačítka bude hráč vyzván k vložení textu, který se druhému zobrazí. Nejčastěji by to mělo být zdůvodnění, například odkaz na nějaké pravidlo apod. Je však možné jej ponechat prázdný.
- Seznam hráčů – JList komponenta, uvádějící seznam hráčů, které stáhneme ze serveru po připojení nebo při použití tlačítka *refresh*. Jedná se o reakci na zprávu *players*, viz. 4.1.2.
- Textová oblast – jedná se o místo, kam se zobrazují důležité zprávy, například tah, který byl proveden apod. Je reprezentována objektem sys`TextArea` a je umístěna v dolní části panelu.
- Popisky (text labels) – Mají informační charakter.

Poslední částí obrazovky je místo, kam se renderuje 3D scéna šachovnice. Tvoří ji komponenta `PicCanvas3D`, která je odvozená od `Canvas3D`. Sem se vykresluje to, co vidí kamera. Stavba scény je detailněji popsána kapitolou 5.2.

5.1.2 Connection Dialog

Toto dialogové okno slouží k připojení a registraci na serveru. Je implementováno třídou `ConnectionDlg`, která je potomkem `JFrame` a je umístěna v balíku `j3dchess.gui`. Vyvoláme jej buďto pomocí položky v menu, nebo zmáčknutím Ctrl+C.



Obr. 5: obrázek connection dialog.

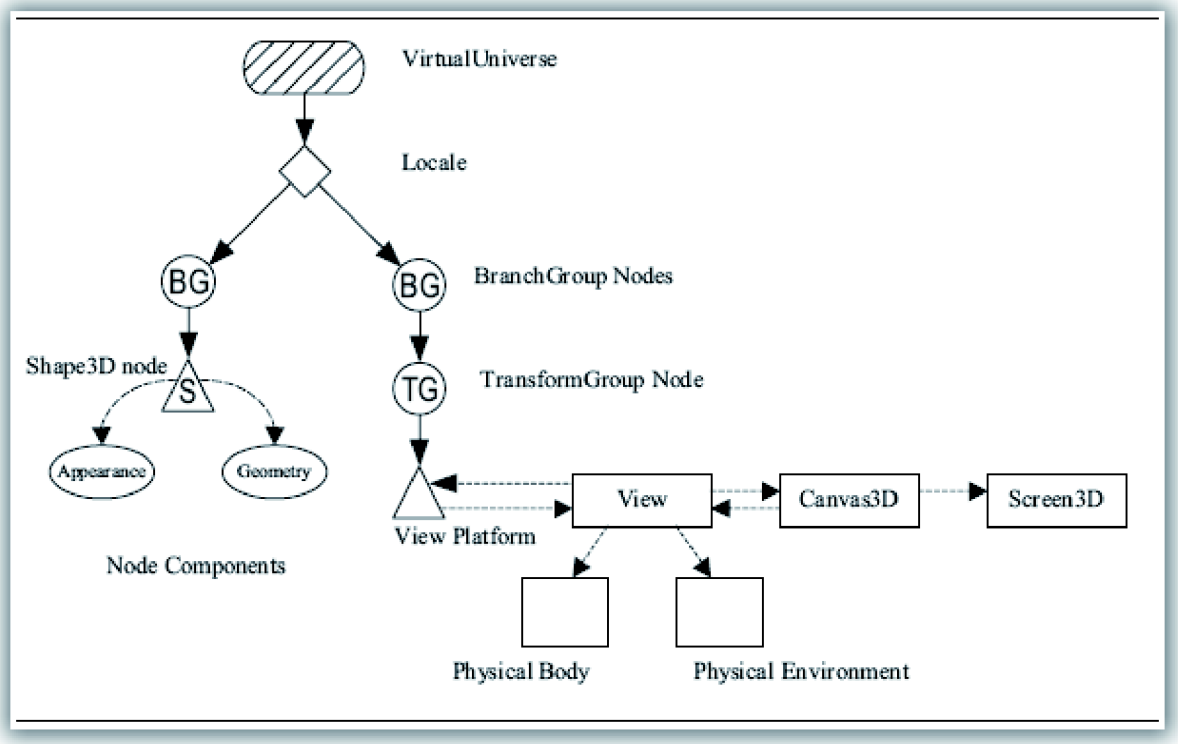
Jak je vidět na obrázku, okno sestává ze dvou textových položek: jméno serveru (doménové jméno serveru nebo IP adresa) a přezdívky, pod kterou se chce hráč registrovat. Pro tlačítka platí: connect připojí hráče k zadanému serveru , register hráče připojí (pokud není doposud připojen) a pokusí se jej zaregistrovat, disconnect odpojí hráče od serveru. Zavřením dialogu jej pouze zneviditelníme, tak jsou všechy vyplněné údaje a stavy tlačítek zachovány. Při úspěšné registraci dialog sám zmizí a příslušná zpráva se zobrazí v textové oblasti. V opačném případě se objeví popup dialog s upřesněním chyby, zároveň je na výstup směrován výpis výjimky.

5.1.3 Akce komponent

Každé tlačítko nebo položka menu, vyjma dekorací oken, má k sobě přiřazenou tzv. akci. Akce je třída, která implementuje reakci na aktivaci komponenty a zároveň ji dokáže přiřadit popisek, klávesovou zkratku, ikonu apod. Díky tomu můžeme oddělit funkčnost od komponenty. A tak implementovat chování jednou i pro více různých komponent [7]. Třídy akcí jsou definovány opět uvnitř `J3DChessMainFrame` kvůli lepší přístupnosti ke GUI. Jednotlivé akce se totiž navzájem ovlivňují a tak má smysl je dávat blízko k místu, kde jsou definované. Jednotlivé akce je zbytečné popisovat. Více opět v dokumentaci a zdrojovém kódu. Akce jsou pojmenovány tak, aby odrážely svůj účel. Například `ShowRegisterDlgAction` zobrazí dialog připojení.

5.2 3D scéna

V J3D API je scéna reprezentována stromovým grafem na jehož vrcholu je `Locale`, reprezentující systém souřadnic. Na něj se pojí většínou třídy `BranchGroup`, které pak nesou buďto rovnou objekty nebo nejdříve `TransformGroup` a pak teprve další věci. Součástí jedné `BrachGroup` je i aparát kamery. Naše scéna ovšem vychází s předdefinované třídy `SimpleUniverse`, a tak máme tyto věci již udělané. Celá scéna která je v `Locale` a má přiřazený `Viewer` a `Canvas` se nazývá *living* . To co „vidí“ viewer se pak kreslí na zmíněný kanvas [10].



Obr. 6: obrazek příklad grafu scény

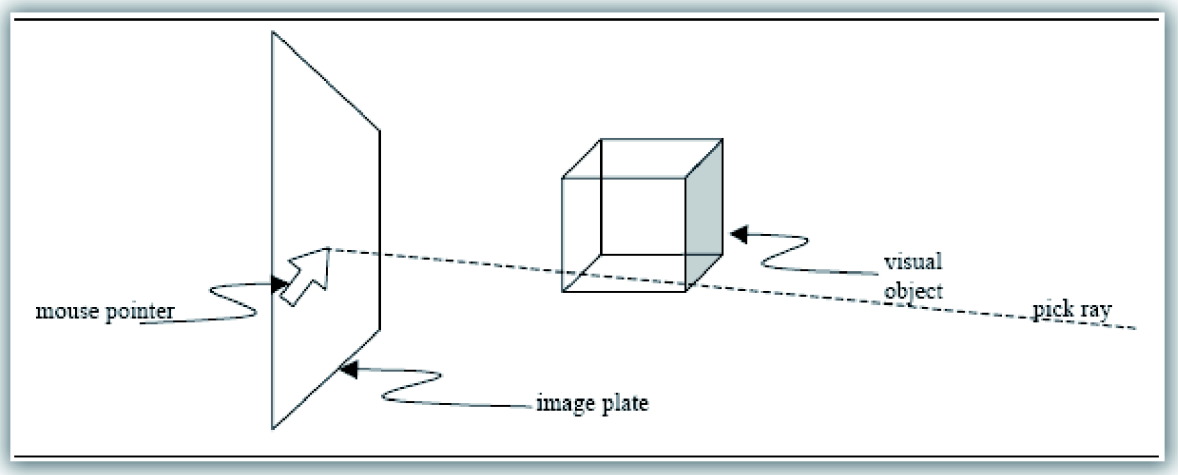
Teď bude následovat popis celků, které do scény přidáváme nebo je nějak významě modifikujeme.

5.2.1 Kamera

Kamera je reprezentována nekolika třídami. Nás bude zajímat `ViewingPlatform`, protože nad ní je soustava `TransformGroup`, tříd, pomocí nichž můžeme kamerou hýbat, viz. Obr. 6. Jedním z parametrů konstruktoru `SimpleUnivers`, je právě počet za sebe napojených grup. K reakci na události myši, která způsobí rotaci, posun či zoom (posun dopředu), máme třídy odvozené z `MouseBehavior`. Například vhodnou kombinací `MouseRotate`, jsme dosáhli sférického pohybu kamery. Ten se děje v reakci na tažení (drag) myši přes kanvas pomocí levého tlačítka. Kamera rotuje kolem středu šachovnice. Při tažení, za pomoci pravého tlačítka, se kamera posouvá relativně po své x-ové a y-ové ose. Střed rotace se ale nemění. Pokud táhneme levým talčítkem a máme stisknutý alt, pak zoomujeme, tedy pohybujeme kamerou po její z-ové ose (dopředu a dozadu).

5.2.2 Picking

Picking je výraz pro vybírání objektů ve scéně. V našem případě je potřeba, aby uživatel mohl ukázat, kterou figurkou se chce pohnout a kam. K tomu nám v J3D slouží třída `PickMouseBehavior`, jejímž oděděním jsme získali `PickSelect`, který více vyhovuje našim potřebám. Instanci behavior třídy musíme přidat, stejně jako ostatní, do grafu scény. Tento objekt je pak aktivován kliknutím myši (nikoli tažením) nad kanvasem a začně provádět picking. Do scény je vržen pomyslný paprsek z pozice ukazatele myši paralelně s projekcí. Poté se počítá průnik s objekty ve scéně a ten, který je pozorovateli nejblíže, je vrácen [11].



Obr. 7: Obrázek Projekce paprsku do scény

Picking samozřejmě můžeme různě nastavit a upravovat. Naše třída `PickSelect` si nechá vrátit celou cestu grafem od počátku až k vlastnímu objektu třídy `Shape3D`. Obsahem cesty jsou pouze uzly jež mají povolený picking. Ze jména uzlu poznáme, zdali jde o šachovnici nebo figurku, a podle toho se zařídíme. Většinou nějakým způsobem upravíme scénu. V našem případě to znamená, že pokud právě uživatel dodal dostatek informací k provedení pohybu, tedy označil jednotku kterou chce táhnout a pole na které chce táhnout, provedeme kontrolu tahu. Tato kontrola je nepovinná, jelikož ji stejně ještě provede i server. My ale necheme server zatěžovat zbytečnými požadavky. Pokud si tedy myslíme, že je tah validní podle pravidel, které byly implementovány, pak jej pošleme na server jako požadavek. Jediný případ, kdy ve skutečnosti při pickingu ovlivňujeme graf scény, je označení a odznačení jednotky. Při této operaci vytvoříme tzv. marker. Ten je reprezentován třídou `CellMarker`. Jedná se o potomka třídy `Shape3D`, který na určeném místě vykreslí čtverec zvolené velikosti a barvy, mírně posunutý v kladném směru osy z. Konkrétně je použitá barva RGBA(1.0, 0, 1.0, 0.3) a marker překrývá celé políčko.

5.2.3 Modely

Všechny modely byly vytvářeny pomocí šablonování. Většinou je jednoduchý profil rotovaný kolem středu figurky. Následně byly modely polygonizovány, vhodně otočeny (aby vyhovovaly umístění ve scéně) a byla na ně aplikována textura. To vše přímo v modelovacím programu. Různé figurky používají různý způsob mapování. Modely bylo potřeba převést do formátu, který umí J3D načíst. Načítání modelů probíhá pomocí tzv. Loaderů. `Loader` je rozhraní, které nám umožní načíst soubor, ve kterém je nějaká 3D scéna. S jeho pomocí dostaneme `Scene`, což je rozhraní, které nám umožní přistoupit k načtené scéně jako stromovému grafu, který používá J3D. Odtud pak dostaneme nejvrchnější uzel, pomocí `getSceneGroup()`. Ten připojíme do hlavního grafu. Loader sám se postará o to, abychom ze souboru dostali všechno, včetně textury, kterou jsme aplikovali. S modely pak můžeme normálně pracovat, jako bychom je vytvořili přímo v J3D.

Modely jsou umístěny ve složce objects. Jsou ve formátu .obj a .mtl, což jsou původní formáty z Wavefront Advanced Visualizer™, viz. online dokumentace. Obrázky textur jsou v tradičních formátech PNG a JPEG. V současnosti jsou samotným J3D podporovány ze základu jen dva formáty (Wavefront

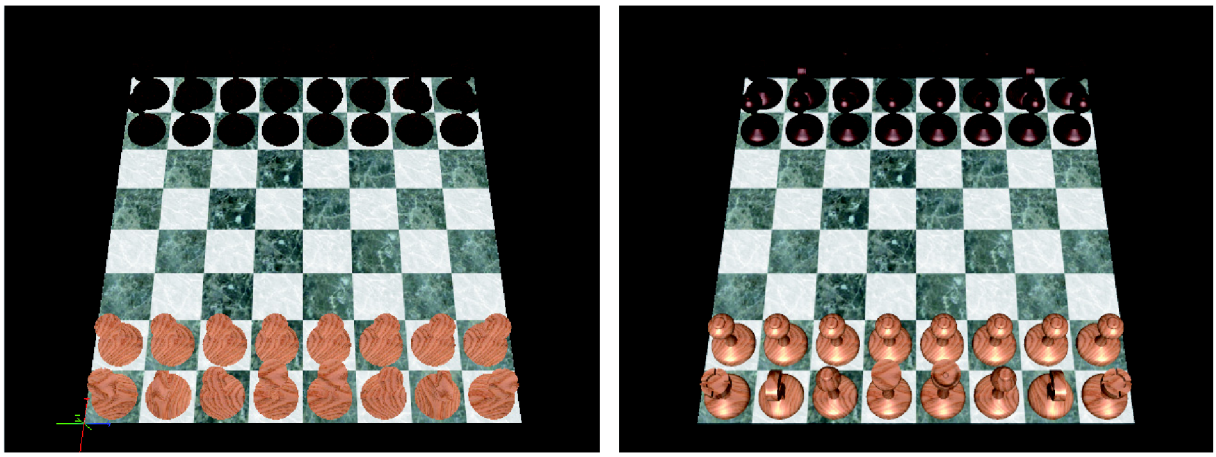
a Lightwave 3D). Na internetu je však možné najít zdarma i další, např. pro 3D Studio Max. Jeden takový jsem vyzkoušel. Stačí změnit jen velmi málo, jeden objekt, aby byl schopný načíst jiný formát. Nicméně jsem zůstal u původní verze, jelikož ta nepotřebuje dodatečné balíky.

Abychom měli pohodlnější práci s modely figur, které se na rozdíl od šachovnice musejí pohybovat, bylo vhodné vytvořit jim obalovou třídu. Takovou třídou je `Unit`, ze které pak dědí každý typ figurky. Máme například třídu `Pawn`, `King` atd. V třídě `Unit` je definováno základní chování. Pomocí metody move měníme pozici, metou hide figurky odstraníme ze scény. Ale stále existuje jako objekt, aby mohla být později opět zobrazena atd. Také je potřeba figurky někde uchovávat. K tomuto účelu slouží `UnitContainer`. Tento v sobě nese dva zřetězené seznamy, jeden pro bílé a druhý pro černé figury. Implementuje také množství vyhledávacích operací nad nimi. Pokud bychom třeba potřebovali zjistit, jaká jednotka je na určitém políčku, použijeme metodu `whatUnitIsOn`. Třídy `Unit` i `UnitContainer` jsou vzájemě provázané, tedy každá figurka má odkaz na `UnitContainer`, do kterého je umístěna. Tyto třídy jsou součástí balíku `j3dchess.unit`. Další součástí tohoto balíku je i enum `PColor`, který určuje barvu hráče. Podrobnější informace jsou v dokumentaci.

5.2.4 Osvětlení

Aby bylo ve scéně vůbec možné osvětlení využít, bylo potřeba změnit vlastnosti načtených modelů. Ty jsou totiž ze základu nastaveny tak, aby se osvětlení nekombinovalo s texturou, a tedy i pokud do scény nějaká světla umístíme, nepoznáme rozdíl.K tomu slouží metoda `remakeAppearance` třídy Unit, ve které se modely načítají. Tato metoda doplní a pozmění určité informace potřebné k tomu, aby osvětlení fungovalo správně. Co se týče aplikace světla na texturu, byla vybrána vlastnost MODULATE, která vynásobí barvu textury s barvou světla a výsledek pak aplikuje na objekt.

Světla samotná jsou ve scéně pouze dvě. Jedno ambientní a druhé směrové. Ambientní je reprezentováno objektem třídy `AmbientLight` a nese barvu `Color3f(0.5f,0.5f,.5f)`. Směrové světlo je tvořeno objektem třídy `DirectionalLight`. Krome barvy `Color3f(0.6f,0.6f,.6f)`. Je potřeba také udat směr, kterým má svítit. Světlo je posazeno jakoby za bílého hráče: `Vector3f(0.f,.45f,-1.f)`. Stíny nejsou ve scéně renderovány.



Obr. 8: Obrázek s bez osvětlení a 0×AA (vlevo), s osvětlením 2×AA (vpravo)

6 Závěr

Díky této práci jsem se seznámil především s platformou Java. Ta už přímo v základu oplývá většinou potřebných věcí k běhu aplikace. Mimo Javu jsem se též seznámil s J3D API, které mi umožnilo jednoduše vytvořit interaktivní 3D scénu.

Podařilo se mi položit základ systému pro hraní deskových her přes internet a na ukázku jej aplikovat na hru Šachy. Ta bohužel neimplementuje všechna dostupná pravidla a módy hry. Asi nejdůležitější součástí systému je aplikační protokol, který, pokud bude dobře rozšířen, může podporovat i jiné než jen čistě deskové hry. Systém je ovšem z tohoto hlediska spíše ve stádiu návrhu a na jeho dokončení by bylo potřeba více času a zkušeností.

Dalších postupů může být několik. Může se například více zobecnit systém, tedy rozšířit objektový návrh a rozšířit stávající protokol o další množinu příkazů. Může být naimplementována další čistě desková hra nad stávajícím systémem, tředa Dáma a jiné. Nebo se můžeme zabývat pouze vylepšením grafické části šachů, případně implementací dalších pravidel nebo začleněním nějakého stroje, který by zastal úlohu jednoho hráče.

Celý projekt jsem vyvíjel s pomocí svn serveru assembla. Je tedy veřejně dostupný k shlédnutí na URL: **http://www.assembla.com/wiki/show/j3d-chess** .

Literatura

[1] Rybka.[online][cit.2009-04-14]. <URL: http://www.rybkachess.com>

[2] Wikipedia. *Chess variant*. [online] poslední úprava 4/19/2009 [cit.2009-04-20]. URL: <http://en.wikipedia.org/wiki/Chess_variant>

[3] Sun Microsystems, Inc. *Applets*. [online] [cit.2009-04-15]. URL: <http://java.sun.com/applets/>

[4] Sun Microsystems, Inc. *Using Top-Level Containers*. [online] poslední úprava 2/14/2008 [cit.2009-04-20]. URL: <http://java.sun.com/docs/books/tutorial/uiswing/components/toplevel.html>

[5] Sun Microsystems, Inc. *Short Course: Graphical User Interfaces*. [online] [cit.2009-04-20]. URL: <http://java.sun.com/developer/onlineTraining/awt/contents.html>

[6] Sun Microsystems, Inc. *About the JFC and Swing*. [online] poslední úprava 2/14/2008 [cit.2009-04-20]. URL: <http://java.sun.com/docs/books/tutorial/uiswing/start/about.html>

[7] Sun Microsystems, Inc. *How to Use Actions*. [online] poslední úprava 2/14/2008 [cit.2009-04-20]. URL: <http://java.sun.com/docs/books/tutorial/uiswing/misc/action.html>

[8] Sun Microsystems, Inc. *How to Use GridBagLayout*. [online] poslední úprava 2/14/2008 [cit.2009-04-20]. URL: <http://java.sun.com/docs/books/tutorial/uiswing/layout/gridbag.html>

[9] Sun Microsystems, Inc. *How to Use GroupLayout*. [online] poslední úprava 2/14/2008 [cit.2009-04-20]. URL: <http://java.sun.com/docs/books/tutorial/uiswing/layout/group.html>

[10] Sun Microsystems, Inc. *Java 3D API Tutorial*. [online] [cit.2009-04-20]. URL: <http://java.sun.com/developer/onlineTraining/java3d/>

[11] Sun Microsystems, Inc., Dennis J Bouvier. *Getting Started with the Java 3D™ API: Interaction*. [online] [cit.2009-04-21]. URL: <http://java.sun.com/developer/onlineTraining/java3d/j3d_tutorial_ch4.pdf>

Seznam příloh