View metadata, citation and similar papers at core.ac.uk
brought to CORE by
provided by DSpace at VSB Technica

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Karetní hra Mariáš v internetovém prostředí
Card-game Marriage in internet environment

2009 Anna Brožková

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě dne 21.8.2009

……..……

Anna Brožková

Poděkování

Děkuji panu Ing. Petru Olivkovi z katedry informatiky za cenné rady, připomínky a konzultace, se kterými mě dovedl až k závěru mé bakalářské práce.

Abstrakt

Tématem této práce je síťová realizace karetní hry Mariáš. Hru hrají tři hráči proti sobě.
Aplikace má tedy dvě části – server (vytvořený pomocí Java servletu) a klient (vytvořený pomocí Java appletu).
Bylo třeba vytvořit vhodné grafické rozhraní, síťovou vrstvu a server starající se o udržování stavu hry a zajišťující komunikaci mezi klienty. Pro tento účel jsem musela vybrat vhodnou technologii. Další část práce spočívala v implementaci herní logiky, aby bylo zajištěno dodržování pravidel hry.
První kapitola je úvodní. Druhá kapitola srovnává různé serverové technologie a popisuje, proč jsem vybrala Java servlet. Třetí kapitola se týká technologií na straně klienta a zdůvodňuje použití Java appletu. Další kapitola obsahuje způsob řešení samotného programu. Po popisu použitých programů následuje uživatelská dokumentace. Jsou přiložena pravidla hry.

Klíčová slova

Java, servlet, applet, síť, hra, Mariáš, karty, Swing, server, klient

Abstract

This thesis is about creation of card-game Marriage in internet environment. The game is played by three players. The application has two parts: the server (implemented as Java Servlet) and the client (implemented as Java Applet).
First I had to create a graphical user interface, the network layer and the server to take care of the game status and to handle communication with clients. I had to choose an appropriate technology for that purpose. The next part of the work was about implementing the game logic to enforce game rules.
The first chapter contains introduction. In the second chapter, various server-side technologies are compared and reasons why Java Servlet was chosen are given. The third chapter discusses client-side technologies and the choice of Java Applet. Next chapter describes the programming part.
The description of software used is then followed by user documentation. The game rules are included in Appendix A.

Keywords

Java, Servlet, Applet, network, game, Marriage, cards, Swing, server, client

Obsah

1 Úvod

Tato práce se zabývá implementací hry Mariáš.

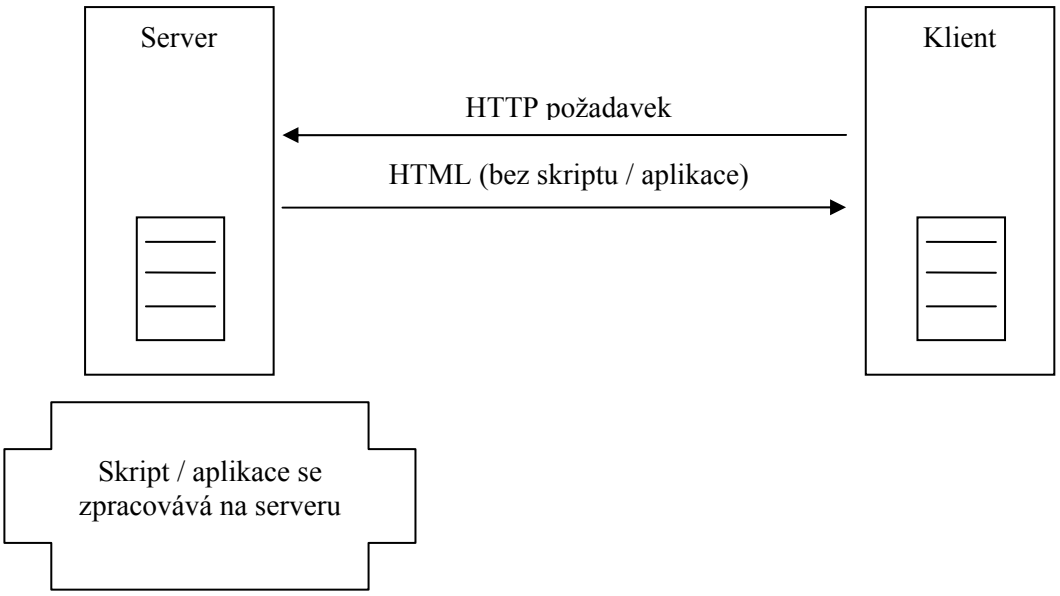
Tohle téma jsem si vybrala, protože ho považuji za zajímavé a chtěla jsem se věnovat práci se sítí v programovacím jazyce Java.

Cílem bylo vytvoření serveru pomocí Java Servletu, rozhraní pro klienta pomocí Java Appletu a navržení způsob komunikace mezi serverem a klienty. Klientská aplikace musí garantovat, že vstup od uživatele nebude porušovat herní pravidla, aby nedošlo k narušení stavu hry.

Práce se zabývá pouze jednou variantou hry (tzv. „Volený mariáš“). Hra je možná pouze mezi lidmi, program neobsahuje hráče řízené počítačem.

2 Technologie na straně serveru

Na straně serveru (například na webovém serveru) se zpracovávají operace, které potřebují přístup k datům, které nejsou přístupné na straně klienta (viz obr. 1).



Obr. 1 - Technologie na straně serveru

2.1 Java servlety

Servlety jsou programy napsané v jazyku Java, které běží na straně serveru. Zpracovávají HTTP dotazy, provádějí různé činnosti na straně serveru a generují HTML stránky.

V paměti je vždycky jenom jedna instance daného servletu, každý požadavek se zpracovává v samostatném vlákně. Data tedy mohou zůstat mezi jednotlivými dotazy v paměti.
To je výhoda oproti technologiím, které hodnoty proměnných uchovávají pro každý požadavek nezávisle, a musí je tedy pokaždé načítat například z databáze, což znamená časovou zátěž navíc.

Stejně jako ostatní programy v Javě i servlety jsou přenositelné – lze je spouštět na různých platformách, přičemž je stačí zkompilovat pouze jednou. Programy v Javě totiž při kompilaci nevytváří přímo binární kód, ale pouze tzv. bajtový kód (*bytecode*), který při spuštění programu interpretuje Java Virtual Machine (JVM).

Zkompilovaný servlet (stejně jako běžný program v Javě) se skládá ze souborů s příponou *.class*, které jsou obvykle spojeny do jednoho Java archivu (přípona *.jar*).

Stejně jako pro každou jinou webovou stránku i pro spuštění servletů potřebujeme webový server. Ne všechny servery však servlety podporují. O serverech použitých při vytváření této práce je více napsáno v částech 5.2 Apache Tomcat a 5.3 Resin.

2.2 JSP

Java Servlet Pages je technologie, která se servlety souvisí. Z JSP dokumentů se vytváří servlety. Jejich využití je vhodné při vytváření stránek, které jsou z větší části statické. Potom není nutné psát ručně celý kód pro servlet, takže zdrojový kód je přehlednější, a tím tedy je i jeho napsání často jednodušší.

(Většina informací o Java Serletech a JSP pochází z [12] a [15]).

2.3 ASP

Active Server Pages je technologie na straně serveru od společnosti Microsoft. Původně to byl add-on k IIS (Internet Information Services – dnes druhý nejpoužívanější server podle [1]).

Většina ASP stránek je napsáno v jazyku VBScript (Visual Basic Scripting Edition), ale jsou podporované i jiné jazyky.

Typická koncovka souborů je .asp, někdy je nastavena i pro .htm a .html).

ASP je dnes vnímáno jako zastaralé, většinou je nahrazováno technologií ASP .NET.

2.4 ASP .NET

ASP. NET je nástupce ASP. Kód, který lze napsat v jakémkoli jazyku, který je v .NET Framework podporován, se překládá do CIL (Common Intermediate Language) (obdobně jako se zdrojový kód Javy překládá do bajtového kódu).

Soubory v jazyku ASP .NET mají obvykle příponu .aspx.

Při tvorbě aplikací pomocí na platformě .NET máme přístup k rozsáhlé knihovně funkcí.

Nevýhodou ASP .NET, stejně jako všech ostatní technologií založených na .NET Framework, je omezenost na platformy společnosti Microsoft. Pro spuštění totiž všechny .NET aplikace vyžadují nainstalovaný .NET Framework runtime, který existuje pouze pro systémy Microsoft Windows. I když je nutno přiznat, že existuje open source iniciativa, projekt jménem Mono, umožňující spouštění .NET aplikací kromě ve Windows i na platformách Linux, OS X a BSD.

Je přímým konkurentem JSP.

2.5 CGI

Standard CGI (Common Gateway Interface) umožňuje spouštění libovolných programů na webovém serveru [10]. Spustitelné soubory se obvykle ukládají do adresáře *cgi-bin*. Při každém požadavku se program musí znovu zavést do paměti (což může být u více zatížených serverů problém).

CGI se dnes používá už relativně málo. Jeden z důvodů je to, že společnosti poskytující webhosting zpravidla na jeden serverový stroj umísťují data mnoha uživatelů. Každému uživateli je tedy třeba zabránit v přístupu k datům ostatních uživatelů. Proto nelze umožnit spouštění libovolných programů, protože spustitelné soubory od uživatelů je obtížné omezit tak, aby je nebylo možné zneužít k získání kontroly nad celým serverovým počítačem.

2.6 PHP

PHP (zkratka pochází z anglického výrazu „PHP : Hypertext Preprocessor“) [9] je skriptovací jazyk, který se používá pro vytváření dynamických webů.

PHP lze použít pro vytvoření serverové části her, nicméně pro složitější projekty je jednodušší použít jazyk, který je přímo zaměřený na tvorbu webových aplikací.

2.7 Srovnání

Servlety mají oproti CGI výhodu, že zůstávají v paměti mezi jednotlivými dotazy – takže není nutné stav hry pokaždé ukládat do databáze.

ASP .NET má podobné výhody jako Java, ale kvůli větší přenositelnost mezi platformami byla zvolena Java.

PHP není podle mě vhodné na tvorbu složitých aplikací, protože neobsahuje přísnou typovou bezpečnost a není v něm nutné deklarovat proměnné. To sice umožňuje rychle psát jednoduché programy, ale při tvorbě složitých programů může způsobovat problémy.

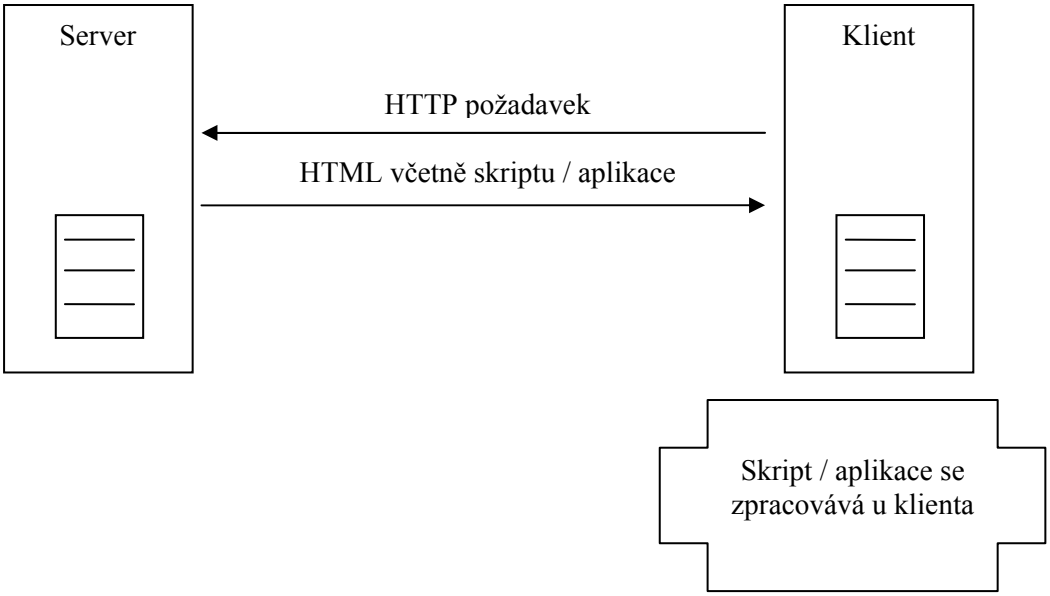
Běžné ASP je dnes už považováno za zastaralé.

Rovněž použití technologie založené na Javě na straně klienta i na straně serveru umožňuje použít stejný kód pro společné části (například kontrola dodržování pravidel hry).

Použití skriptů JSP nebylo zapotřebí, protože webová prezentace obsahuje jenom minimální množství statického HTML kódu.

3 Technologie na straně klienta

Na straně klienta (například ve webovém prohlížeči) probíhají činnosti, které zpracovávají vstup od uživatele a starají se o zobrazování (viz obr. 2).



Obr. 2 - Technologie na straně klienta

3.1 Java applet

Applety jsou programy v Javě, které jsou spouštěny prohlížečem na straně klienta. V uživatelském počítači musí být nainstalováno Java Runtime Enviroment.

Zkompilovaný applet (stejně jako servlet nebo jakýkoli běžný program v Javě) se rovněž skládá ze souborů s příponou *.class*, které jsou obvykle spojeny do jednoho Java archivu (přípona *.jar*). Tyto soubory musí být vloženy na webovou stránku, to lze učinit pomocí značek <*applet*>, <*embed*> nebo <*object*>.

Podle W3C je značka <*applet*> zastaralá a je doporučeno ji nahradit značkou <*object*> [2]. Nicméně společnost Sun doporučuje používat pořád <*applet*>, protože tag <*object*> není vždy funkční ve všech prohlížečích (nemusí fungovat například v Mozilla Firefox) [3].

Používání občas znepříjemňuje poměrně dlouhé načítání JVM. Prohlížeče se mnohokrát při otevření stránky s appletem na několik sekund zastaví a chvíli nereagují na žádný vstup od uživatele.

Značka applet může mít následující atributy (podle [14]):

Název atributu	Popis
code	Název třídy s appletem.
object	Odkaz na serializovaný applet (alternativní možnost spouštění appletu).
align	Nastavení obtékání textem.
alt	Alternativní text.
archive	Název případného JAR souboru s appletem.
codebase	Cesta k souborům tříd (když není uvedena, předpokládá se aktuální adresář).
height	Výška appletu.
hspace	Vodorovná mezera okolo appletu.
name	Název appletu (používá se ve skriptech).
vspace	Svislá mezera okolo appletu.
width	Šířka appletu.

Tab. 1 - Attributy značky applet

Povinně musí být přítomna značka code nebo object, a potom značky width a height.

Dovnitř elementu <*applet*> taky můžeme vkládat parametry pomocí značky <*param*>. Hodnoty, které takto uvedeme, můžeme používat přímo v kódu appletu (pomocí metody *getParameter()*).

Kód pro vložení appletu může tedy vypadat například takhle:

```
<applet width="950" height="800"
      code="bro130.marias.network.client.ClientApplet"
      archive="Marias.jar">
  <param name="servletURL" value="http://localhost:8080/Marias">
  Ke spuštění appletu je třeba mít nainstalovanou Javu.
</applet>
```

3.2 Adobe Flash

Používá se běžně na webových stránkách, obzvláště k zobrazování animované reklamy. Přestože je to jazyk původně určený pro vytváření animací, pomocí ActionScript (objektový jazyk určený pro Flash) se dají vytvářet i komplexní aplikace.

Výhodou je velká rozšířenost, většina uživatelů webu dnes má Flash nainstalovaný (podle společnosti Adobe [4] až 99%).
Flash se běžně užívá pro tvorbu jednoduchých webových her, k tomu přispívá jednoduchost tvorby animované grafiky a vkládání zvuků.
Nicméně Flash není vhodný na vytváření složitějších aplikací.

3.3 Javascript a AJAX

Jde o nejpoužívanější skriptovací jazyk webu.

Přestože se názvem podobá jazyku Java, jedná se o odlišný programovací jazyk.

Javascript je interpretovaný objektový jazyk, který lze vložit přímo do kódu webové stránky. Prováděn je přímo prohlížečem, (což je nevýhoda tohoto jazyka, protože různé prohlížeče jej v některých ohledech interpretují odlišně).

AJAX (Asynchronous JavaScript and XML) je souhrný název pro technologie vývoje aplikací měnících svůj obsah bez toho, aby bylo třeba stránku znovu načítat [11]. Je možné kontaktovat server a dostat odpověď v XML formátu. Toto může sloužit pro tvorbu příjemnějších uživatelských prostředí.

Nevýhodou je nutnost použití moderního prohlížeče, obsahujícího všechny potřebné technologie (i když většina dnes používaných prohlížečů už požadavky AJAXu splňuje).

Další nevýhodou je nemožnost otevřít spojení směrem ze serveru na klienta – je to nutné řešit např. opakovaným dotazováním ze strany klienta, nebo udržováním jednoho stálého otevřeného spojení.

3.4 Srovnání

Bylo potřeba technologie, která by umožňovala snadnou síťovou komunikaci.

Programovat složitější aplikace pomocí Javascriptu stěžuje mimo jiné i různé odlišnosti mezi prohlížeči.

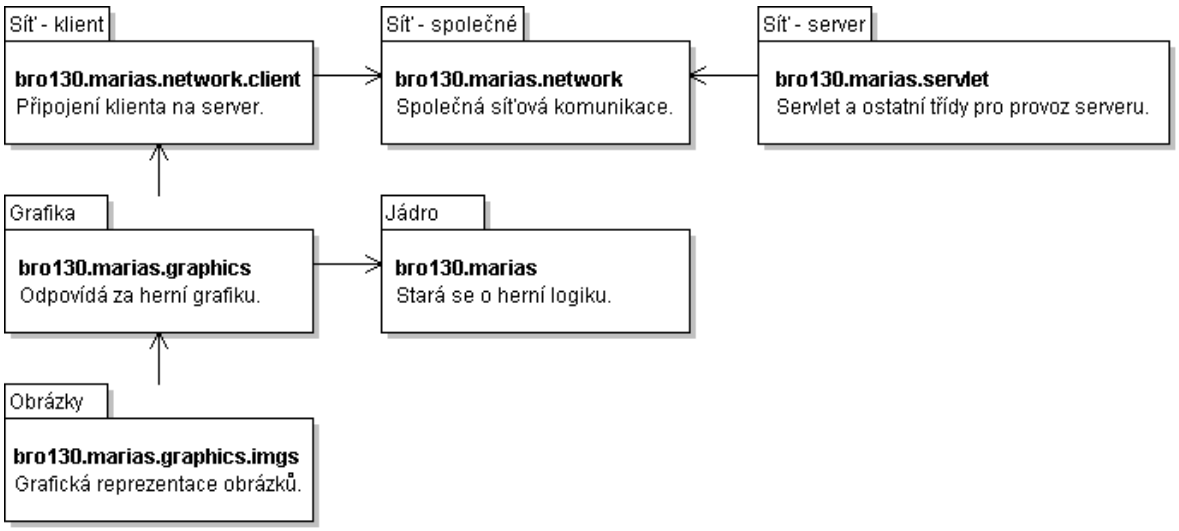
Použití technologie Flash by sice ulehčilo vytváření grafického rozhraní a zvukových efektů, ale implementaci herní logiky by učinilo složitější. Navíc nejpoužívanější studio Adobe Flash CS4 Professional je komerční.

Takže bylo využito Java Appletu.

4 Popis řešení programu

Projekt se dělí na dvě části – klientskou a serverovou.

Pro přehlednost je zdrojový kód rozdělen do šesti balíčků. Dva jsou společné pro obě dvě části (herní jádro a společné síťové třídy), tři balíčky používá pouze klientská část projektu (balíčky týkající se vykreslování grafiky, načítání obrázků a připojení na server) a jeden balíček (obsahující serverovou část síťování včetně třídy servletu) pro serverovou část projektu (viz obr. 3 a tab. 2).

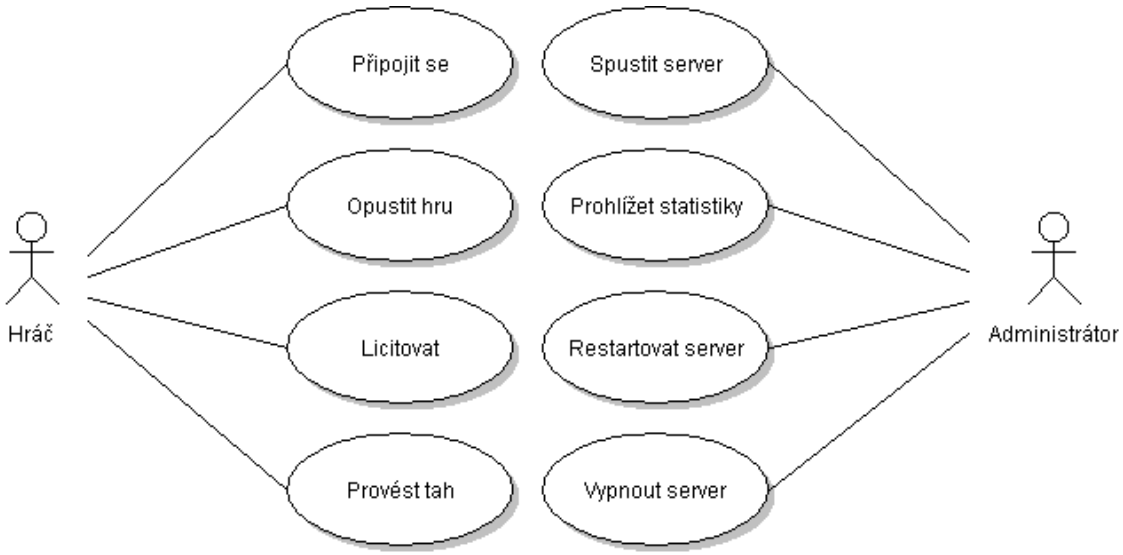


Obr. 3 - Diagram balíčků

bro130.marias	Hlavní balíček, obsahuje třídy, které se starají o herní logiku.
bro130.marias.graphics	Balíček, který se stará o herní grafiku.
bro130.marias.graphics.imgs	Balíček, který obsahuje grafickou reprezentaci obrázků a třídu pro jejich načítání.
bro130.marias.network	Obsahuje třídy pro síťovou komunikaci společné pro server i pro klienta.
bro130.marias.network.client	Sdružuje třídy potřebné k připojení klienta na server.
bro130.marias.servlet	Tento balíček obsahuje třídu servletu a ostatní třídy potřebné pro provoz serveru.

Tab. 2 - Balíčky zdrojového kódu

Diagram případů užití je na obr. 4.



Obr. 4 - Diagram případů užití

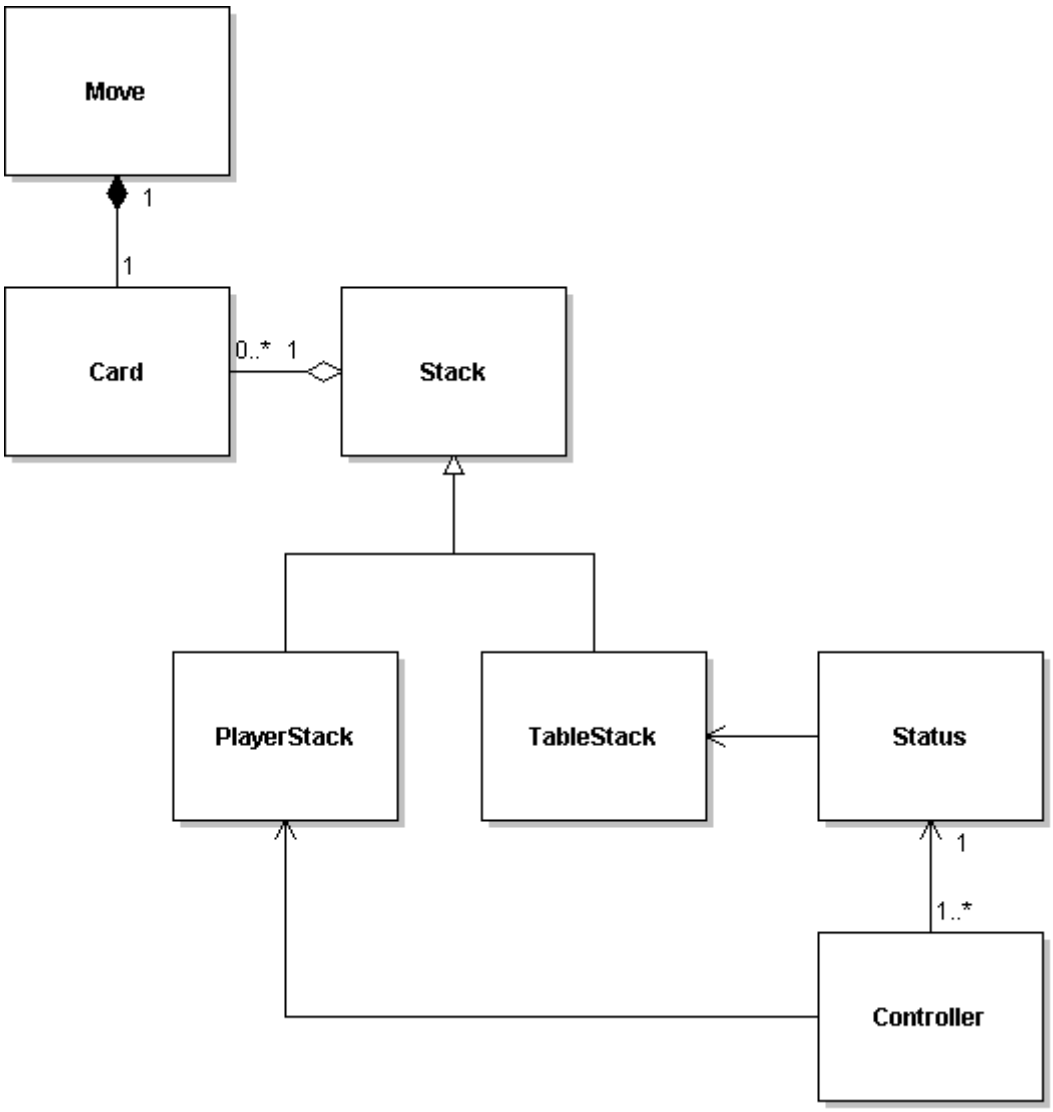
4.1 Herní logika

Důležitá částí je herní logika (zdrojový kód je v hlavním balíčku).

Většina herních událostí probíhá tak, že mezi sebou komunikují třídy **Status** a **Controller**. Instance třídy **Status** je na serveru, instance třídy **Controller** na klientovi. Podpora síťování je vyřešena tak, že na straně klienta je i instance třídy **ServletConnectionStatus** a na straně serveru instance třídy **AppletController**, které slouží k posílání zpráv přes síť.

Například pokud je čas na provedení tahu, tak server pošle výzvu hráči, který je na řadě. Potom klient odpoví kartou, kterou chce táhnout. Server tuto volbu oznámí i ostatním hráčům, potom vyzve k provedení tahu dalšího hráče (pokud se nejednalo o poslední tah).

Na obr. 5 je třídní diagram hlavních tříd.



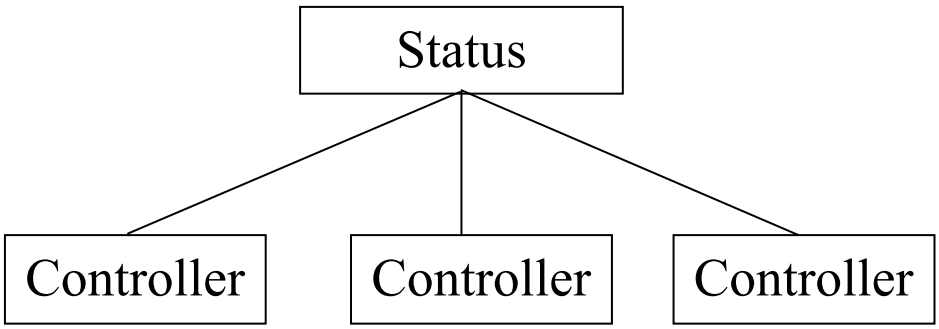
Obr. 5 - Třídní diagram důležitých tříd z hlavního balíčku

Status

Jádrem programu je třída **Status** (Stav), která obsahuje všechny informace o aktuálním stavu probíhající hry.

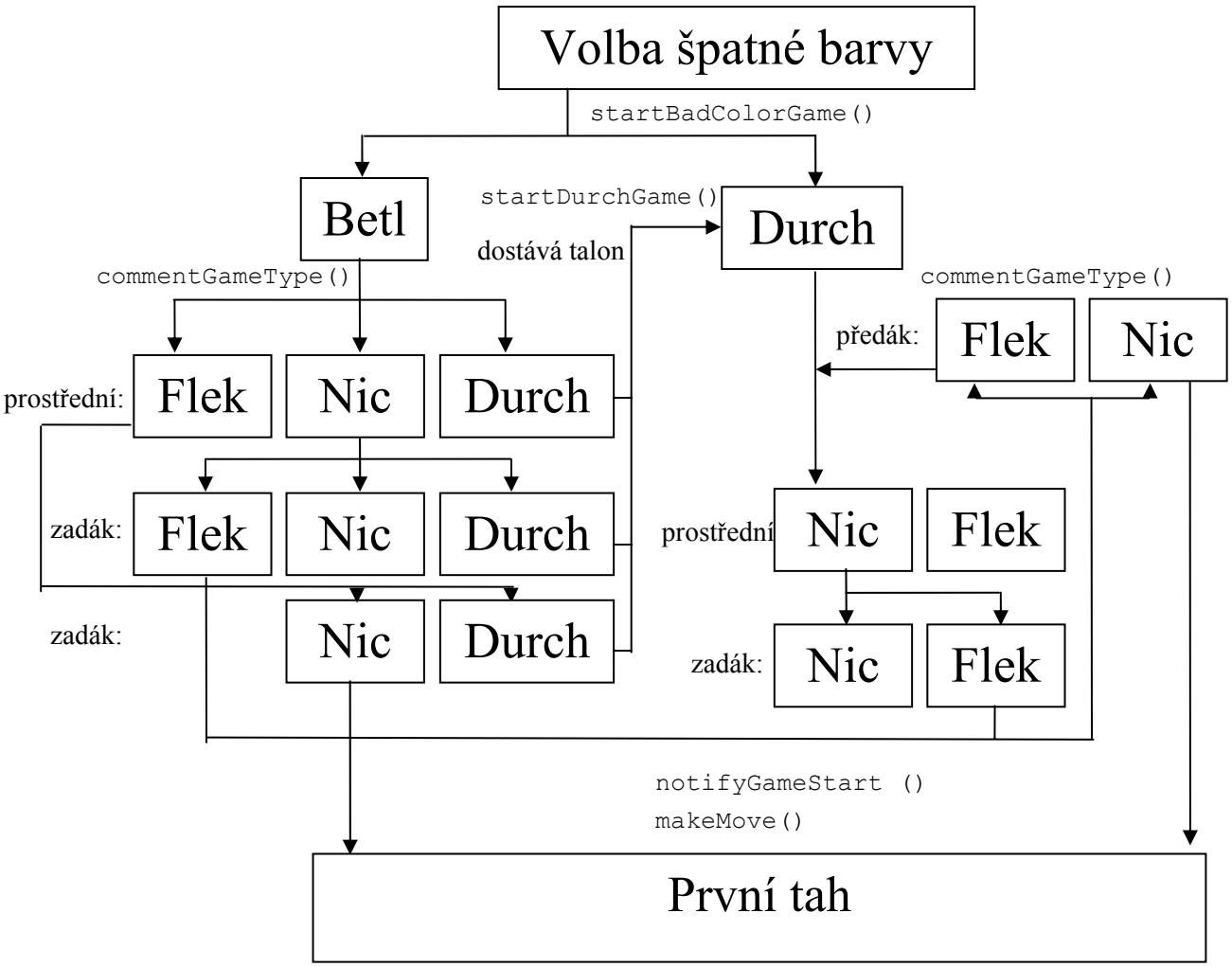
Obsahuje především informace o kartách, které mají v ruce, i o těch, které jsou na stole. (Pomocí instancí tříd **Stack**, **TableStack** a **PlayerStack**). Dále uchovává informace o tom, kdo je na tahu, a ukládá také výsledek licitace.

Každý status komunikuje se třemi objekty typu Controller – jeden za každého hráče (viz obr. 6).



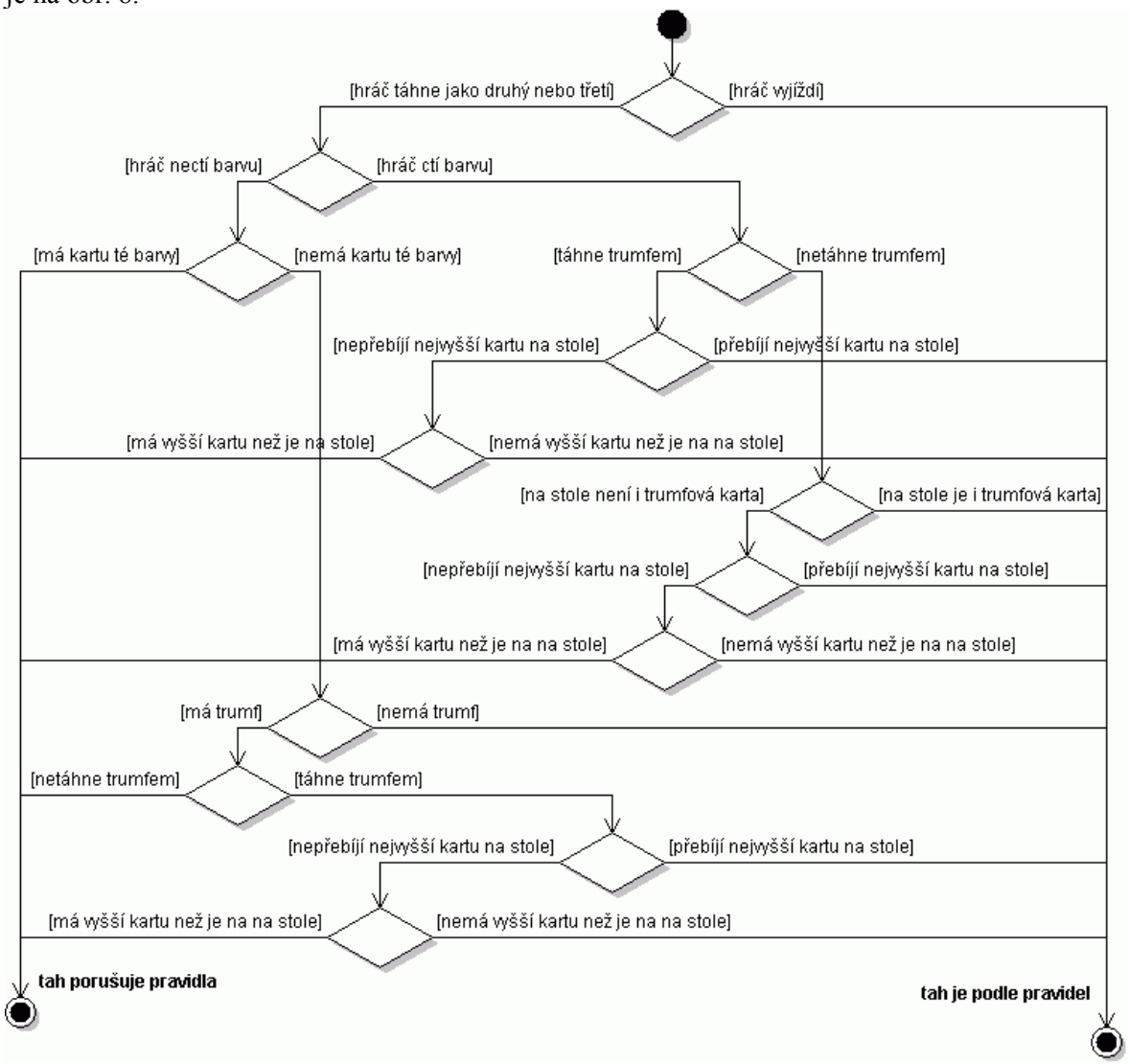
Obr. 6 - Použití objektů Controller ve třídě Status

Grafický nákres možností volání metod tříd **Status** a **Controller** v jedné hře při zvolení špatné barvy hry po první tah je na obr. 7. (pro vysvětlení pojmů viz Příloha A – Pravidla hry).



Obr. 7 - Metody třídy Status při špatné barvě hry

Třída **Status** se také stará o kontrolu dodržování herních pravidel. Aktivitní diagram pro kontrolu je na obr. 8.



Obr. 8 – Aktivitní diagram kontroly herních pravidel

Seznamy důležitých metod a proměnných jsou uvedeny v příloze B.

Controller

Třídy implementující toto rozhraní slouží ke komunikaci objektu Status s jedním hráčem.

V běžícím programu existuje vždy jedna instance třídy LocalPlayerController a dvě instance třídy

Card

Instance této třídy reprezentují jednotlivé hrací karty. Skládají se z barev (**Suite**) a hodnot (**Rank**). Pro každou jednotlivou kartu existuje pouze jedna instance.

Move

Reprezentuje jeden tah. Obsahuje číslo hráče a kartu, kterou tímto tahem chce ten hráč vyjet.

Stack, TableStack, PlayerStack

Kolekce karet (libovolně velká kupka nebo vějíř karet).

K možnostem běžného seznamu (používá se **ArrayList**) tyto třídy přidávají funkce, které jsou pro hru užitečné (například lze zjistit, jestli obsahuje alespoň jednu kartu dané barvy).

Různé varianty obsahují metody smysluplné jen pro určitý druh kupek (např. PlayerStack umožňuje sčítání bodů).

4.2 Uživatelské rozhraní

Vykreslování je realizováno způsobem běžně používaným ve Swingu (Swing je součástí základních knihoven Javy a slouží k vytváření uživatelských rozhraní programu).
Obrázky karet jsou vloženy přímo do hlavního JAR souboru (jako resources).

GameBoard

Celou plochu appletu, který je zobrazen v uživatelově prohlížeči, pokrývá při hře jedna komponenta, což je instance třídy **GameBoard**. V této třídě je zpracováván vstup od uživatele, který je po kontrole pravidel předán přes síť hernímu serveru.

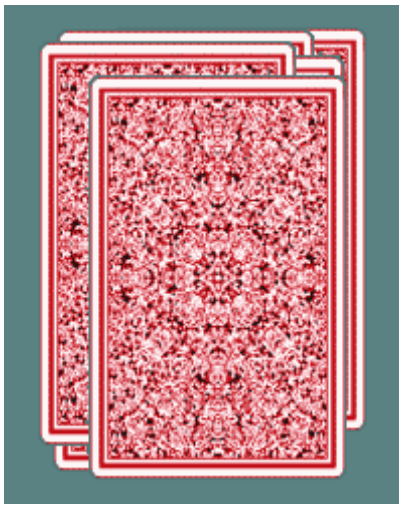
ImageManager

Je pomocná třída, která slouží k načítání obrázků. Ty jsou uloženy ve zvláštním balíčku (**bro130.marias.graphics.imgs**). Načítání realizuje knihovní třída **java.awt.Toolkit**. Třída **ImageManager** načtené obrázky ukládá do mezipaměti, takže je není nutné pokaždé znovu načítat.

Algoritmus výběru karet

Na některé karty na herní obrazovce lze v průběhu hry kliknout (například při výběru karty, kterou chce hráč vynést). Karty přitom můžou být vykreslené na různých místech na herní ploše, protože při vykreslování může být šířka appletu v závislosti na nastavení HTML stránky jiná. Abychom se vyhnuli složitým výpočtům, je to vyřešeno tak, že při vykreslování karet v ruce hráče se ukládá i obdélník, obsahující souřadnice, na které je karta vykreslena. Potom se při kliknutí na herní plochu projdou všechna uložená místa. Pokud některý obdélník obsahuje hledaný bod, je provedena odpovídající akce pro danou kartu.

Algoritmus vykreslení kupky karet



Obr. 9 - Náhodné rozmístění karet v kupce

Jak je možné si všimnout na obr. 9, karty, které jsou na kupce hráče, jsou vykresleny „rozházeně“. Tohoto je dosaženo náhodným posunem karet od výchozí pozice. Aby se však předešlo tomu, že se posunou pokaždé o jiný počet pixelů, je použit pro stejnou kupku vždy stejný seed (číslo, z něhož vychází generátor pseudonáhodných čísel). Tento postup je implementován takto:

```
private static final int RANDOM_SHIFT_MAX = 30;
private static final int RANDOM_SHIFT_LIMIT = 30;

/**
 * Vykreslí jednu kupku, na které jsou uloženy karty, které jeden
 * hráč sebral, přičemž karty budou vykresleny jako "rozházené".
 *
 * @param g         pomocí tohoto objektu se vykresluje
 * @param stack     kupka, která se má vykreslovat
 * @param x         x-ová souřadnice, na kterou se má kupka vykreslit
 * @param y         y-ová souřadnice, na kterou se má kupka vykreslit
 * @param seed      klíč pro generátor pseudonáhodných čísel - pokud
 *                  bude stejný, karty budou "rozházeny" stejně
 */
private void paintPlayerStack(Graphics g, PlayerStack stack, int x,
        int y, int seed) {
    int newX = x, newY = y;

    //generátor náhodných čísel vytvoříme se zadaným seed,
    //aby se při překreslování karty nepřeskupovaly
    Random random = new Random(seed);
    // rub karty
    Image reverse = ImageManager.getInstance().getCardReverseImage();

    for (int i = 0; i < stack.count(); i++) {
        Suite s = stack.getMarriageSuite(i);
        if (s == null) {
            //všechny karty vykreslit rubem nahoru (kromě hlášek)
            paintCard(g, reverse, newX, newY);
```

```
            } else {
                //pokud se jedná o hlášku, kartu vykreslíme lícem nahoru
                Card val = Card.valueOf(Rank.QUEEN, s);
                Image img = ImageManager.getInstance().getCardImage(val);
                paintCard(g, img, newX, newY);
            }

            //další karta nebude na stejné pozici jako tato karta, ale
            //o něco posunutá
            newX += RANDOM_SHIFT_MAX-random.nextInt(RANDOM_SHIFT_MAX*2);
            newY += RANDOM_SHIFT_MAX-random.nextInt(RANDOM_SHIFT_MAX*2);

            //ale zabránit tomu, aby se karta posunulo příliš mnoho
            if (Math.abs(x - newX) > RANDOM_SHIFT_LIMIT) {
                newX = x + RANDOM_SHIFT_MAX -
                           random.nextInt(RANDOM_SHIFT_MAX * 2);
            }
            if (Math.abs(y - newY) > RANDOM_SHIFT_LIMIT) {
                newY = y + RANDOM_SHIFT_MAX -
                           random.nextInt(RANDOM_SHIFT_MAX * 2);
            }
        }
    }
```

4.3 Síťová komunikace

Pro přenos zpráv mezi serverem a klienty se používá třída **MessageWithData.** Skládá se ze dvou částí – jedna hodnota výčtu **MessagesEnum** a řetězec data, který obsahuje případný parametr zprávy.

Seznam možných zpráv (ze serveru pro klienta):

Hodnota výčtového typu	Význam
START_GAME	Výzva k zahájení hry.
YOUR_HAND	Oznámení karet v ruce hráče.
YOUR_NUMBER	Oznámení čísla vybraného pro hráče.
COMMENT	Výzva k vyjádření se ke hře.
COLOR	Výzva k vyjádření se ke dobré barvě.
MAKE_MOVE	Výzva k tahu.
GAME_OVER	Oznámení konce jedné hry.
GAME_START	Oznámení začátku jedné hry a údajů o ní.
MOVE_MADE	Oznámení o provedení tahu (kterýmkoli hráčem).
TABLE_TAKEN	Oznámení o provedení třetího tahu v jednom štychu a určení vyjíždějícího hráče.
SELECT_GAMETYPE	Výzva k volbě druhu hry.
SELECT_BADGAMETYPE	Výzva k volbě druhu hry – pouze špatné barvy (talon je parametr).
START_DURCH	Výzva k zahájení hry Durch (talon je parametr).

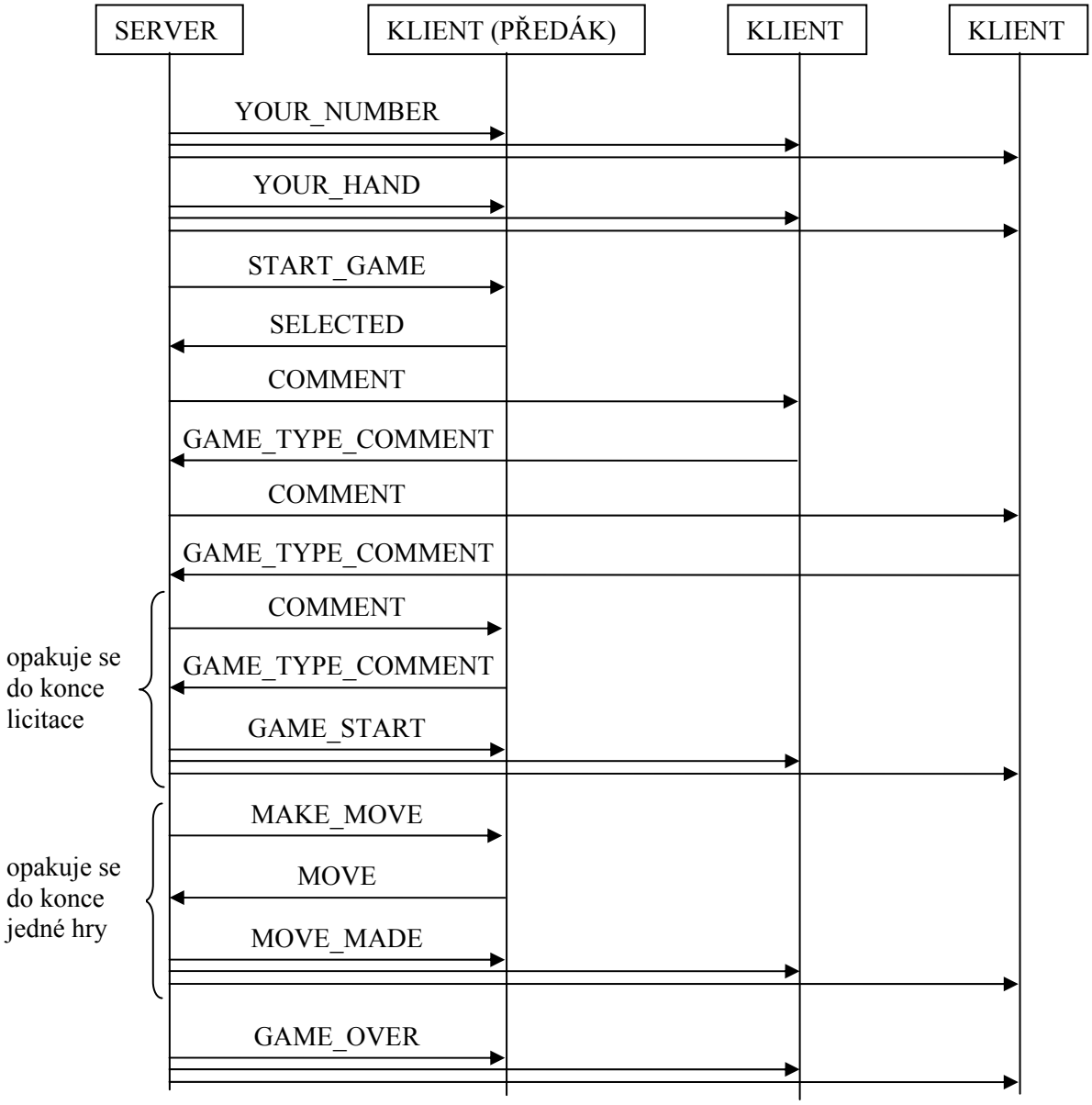
Tab. 3 - Seznam zpráv (ze serveru pro klienta)

Seznam možných zpráv (od klienta pro server):

COLOR_ACCEPTED	Přijetí dobré barvy.
COLOR_REJECTED	Hlášení špatné barvy.
GAME_TYPE_COMMENT	Vyjádření ke hře.
GAMETYPE	Volba druhu hry.
MOVE	Provedení jednoho tahu.
SELECTED	Volba druhu hry a talonu.

Tab. 4 - Seznam zpráv (od klienta pro server)

Možný průběh síťové komunikace při jedné hře (toto pořadí platí, pokud se hraje hra dobré barvy) je zobrazen v sekvenčním diagramu na obr. 10.

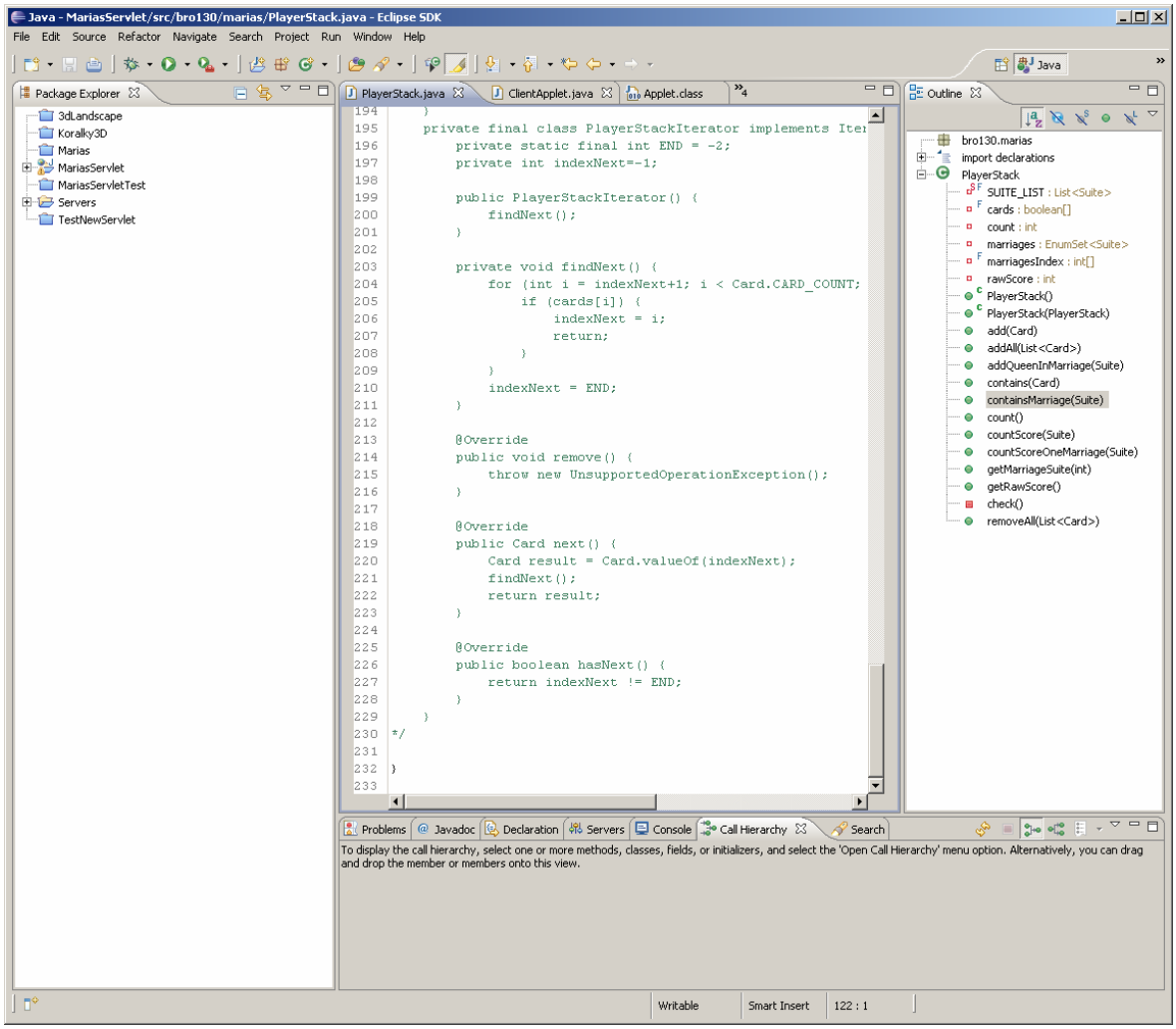


Obr. 10 - Průběh síťové komunikace

5 Použité programy

5.1 Eclipse

K vytváření kódu bylo použito Eclipse. Jedná se o populární open-source vývojovou platformu, která je známá především jako vývojové studio pro programování v Javě. Eclipse lze jednoduše rozšířit o další funkce pomocí zásuvných modulů.



Obr. 11 - Obrázek programu Eclipse

Snímek programu Eclipse je na obr. 11. Platformu Eclipse je možné stáhnout z [5].

5.2 Apache Tomcat

Apache Tomcat je implementace Java Servletu a JSP s otevřeným zdrojovým kódem. Je dostupný pod Apache licencí. Jedná se o jeden z nejznámějších serverů podporujících Java Servlet. Častokrát je integrován do vývojových prostředí, kde se používá pro spouštění vyvíjených projektů obsahujících Java Servlet. Do prostředí Eclipse je ho možné integrovat pomocí zásuvného modulu Web Developer Tools – vývojářské nástroje pro web).

5.3 Resin

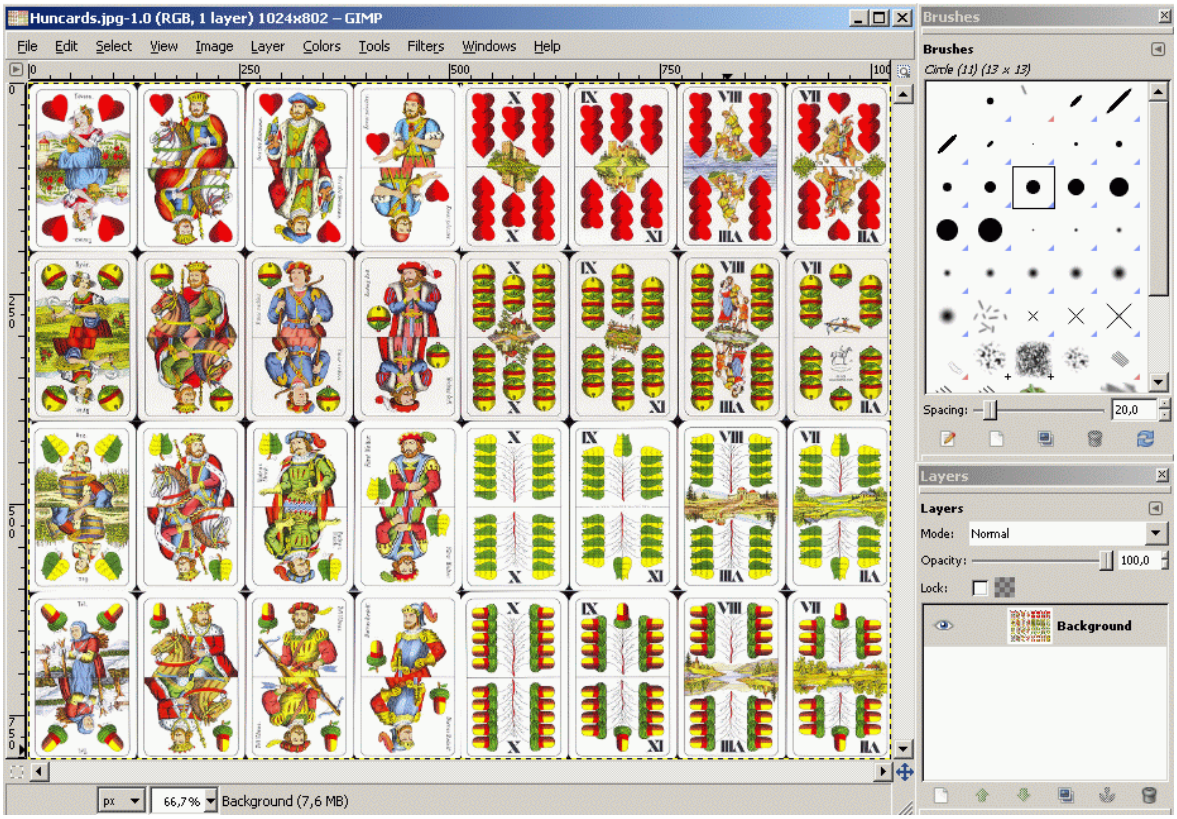
Protože jsem chtěla otestovat výsledný program i s použitím jiného serveru než Apache Tomcat, tak jsem použila také Resin, webový server s otevřeným zdrojovým kódem pod licencí GPL (přičemž tvůrce Resinu – společnost Caucho nabízí i komerční licenci).

Obr. 12 - Obrázek serveru Resin

Snímek z ovládacího okna je na obr. 12. Server Resin je dostupný na [6]. Lze ho spustit na běžných operačních systémech jako je Linux, Windows nebo Solaris.

5.4 GIMP

Obrázky karet použité ve hře pochází z [7], pomocí programu GIMP byly rozděleny na části a byla přidána průhlednost. Snímek programu GIMP je na obr. 13. Program GIMP je možné stáhnout z [8].



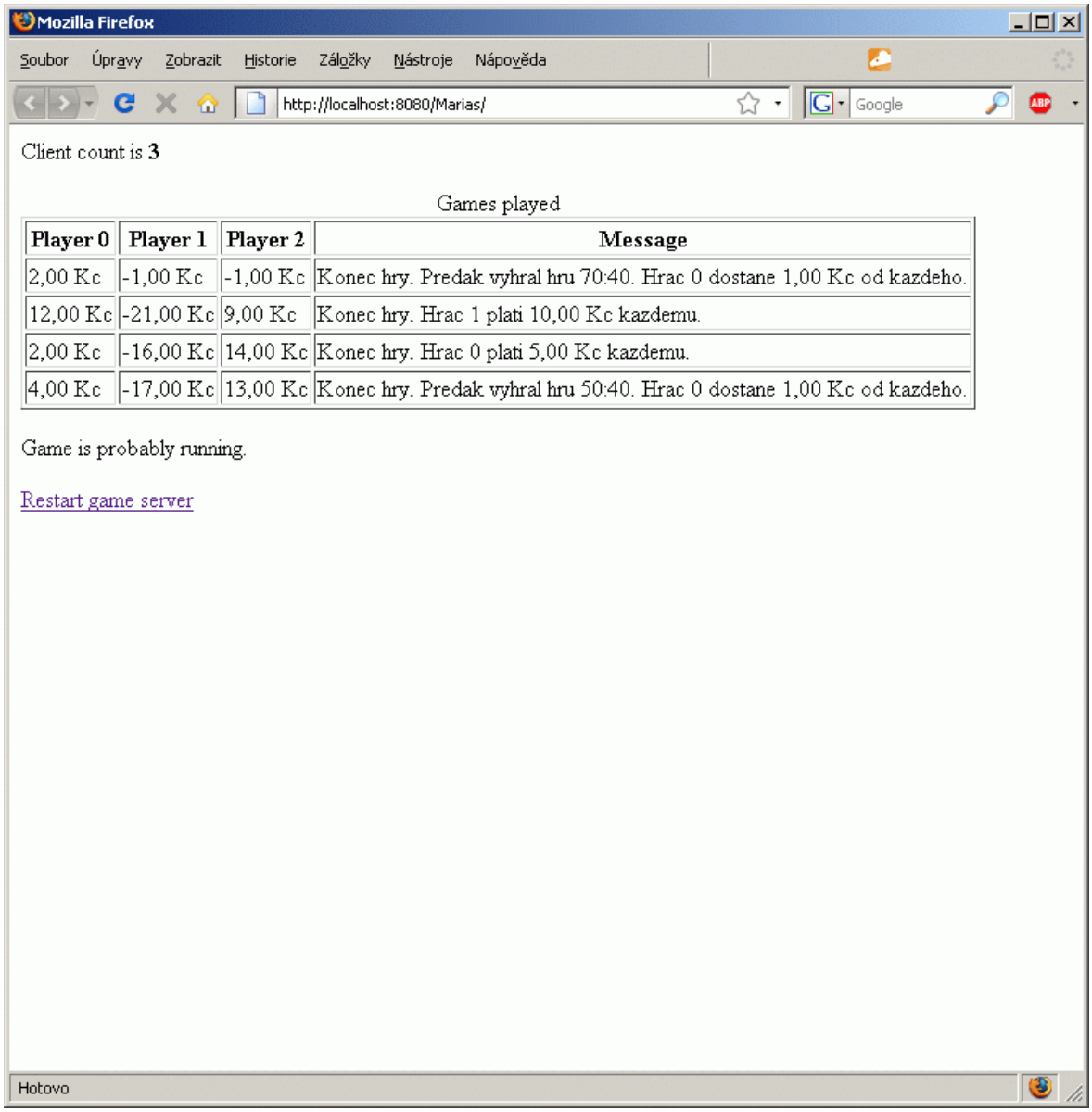
Obr. 13 - Obrázek z programu GIMP.

6 Uživatelská dokumentace

Detailní vysvětlení pravidel je v části Příloha A – Pravidla hry.

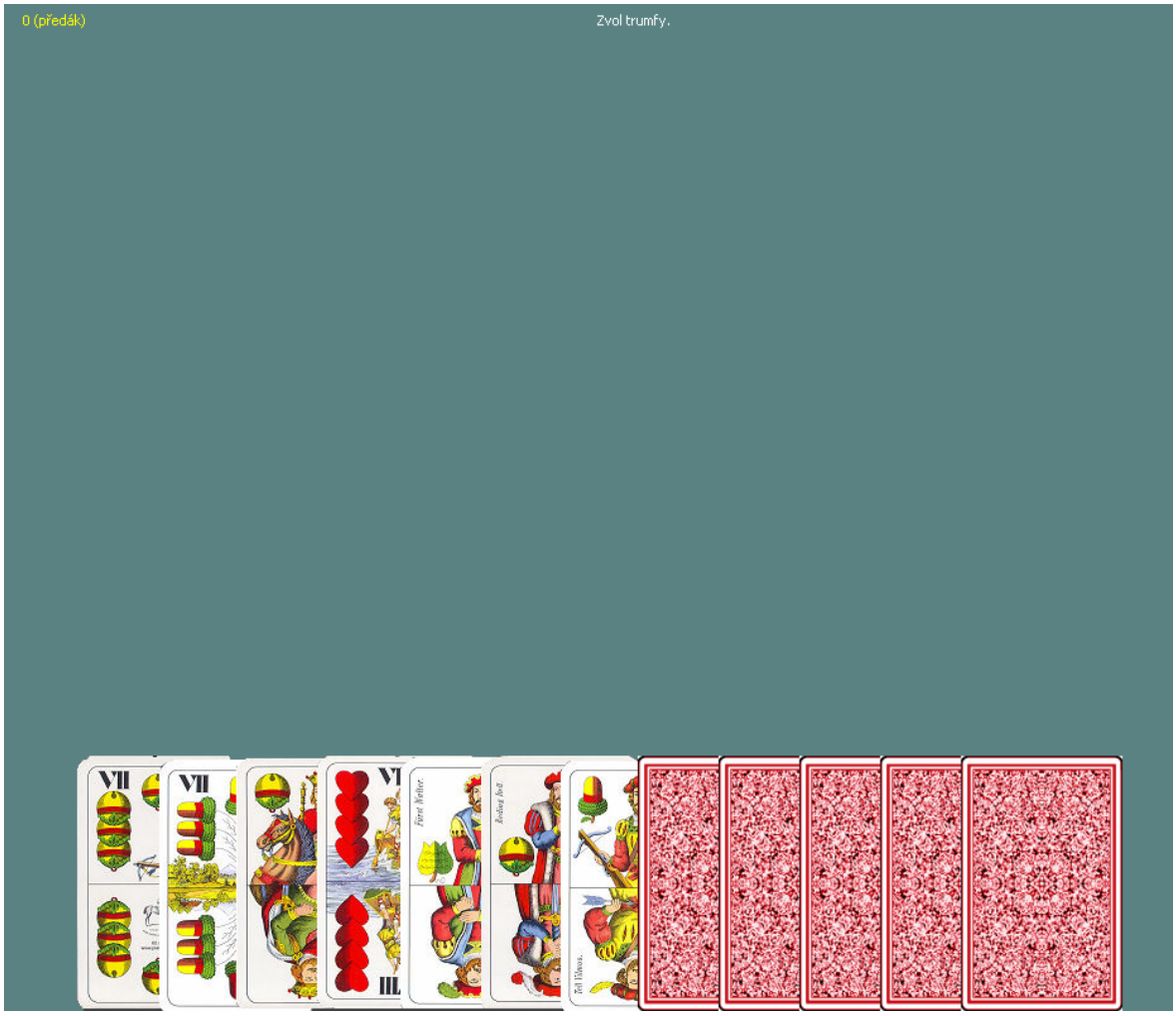
Při otevření stránky obsahující applet je možné připojit se k serveru kliknutím na tlačítko Připojit. V textovém poli je možné změnit URL pro přístup k servletu.

Na stejném URL, kde běží servlet, je možné sledovat statistiky hry a celkový stav serveru pomocí jakéhokoli webového prohlížeče (viz obr. 14).



Obr. 14 - Sledování statistik hry

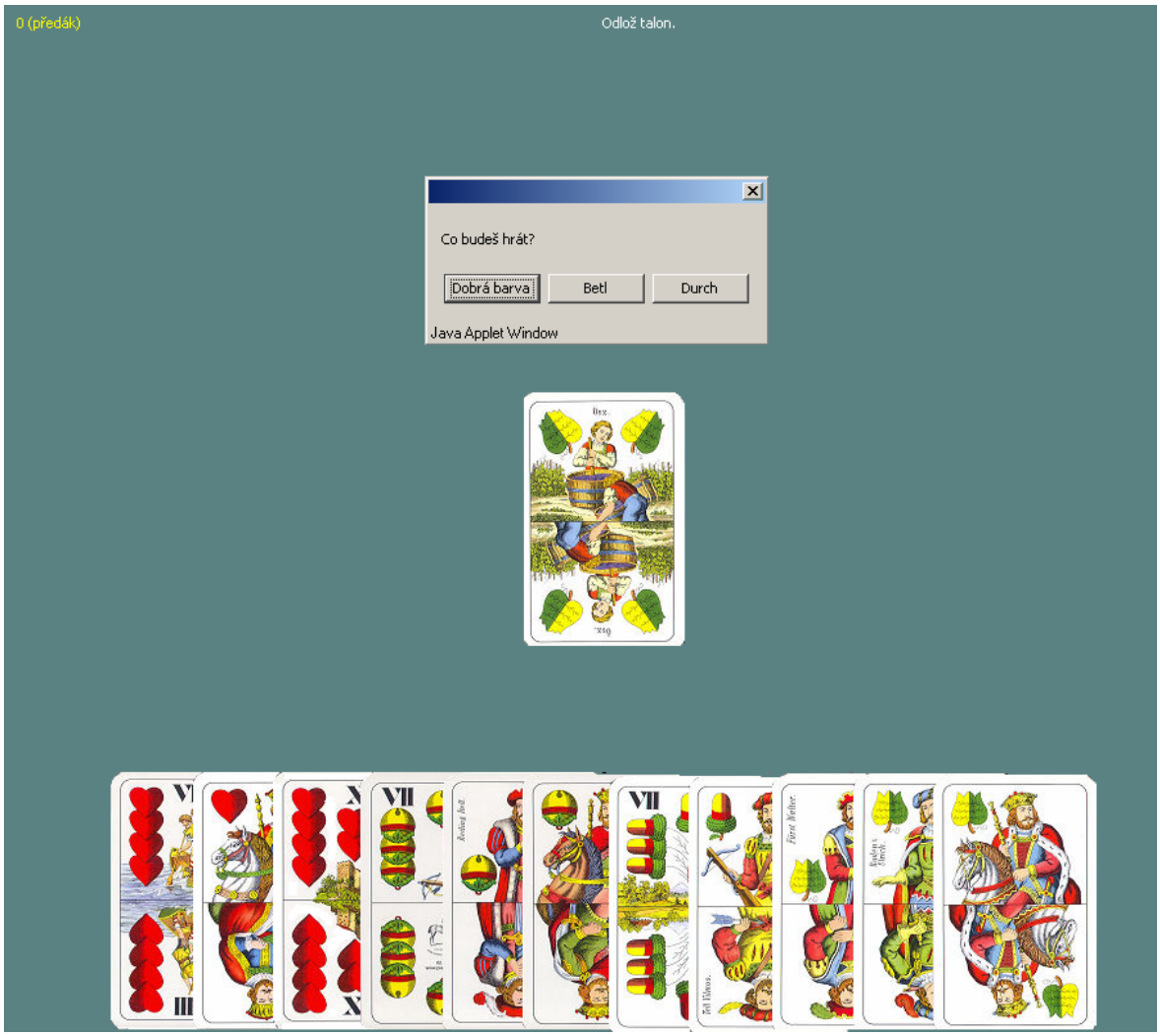
Po připojení všech hráčů je zahájena hra. Předák dostal 12 karet, z nichž je 7 zakryto (viz obr. 15).



Obr. 15 - Volba trumfů

Kliknutím na libovolnou kartu zvolí trumfy.

Poté jsou odkryty ostatní karty. Předák se vyjádří, jakou barvu hry chce hrát (viz obr. 16).



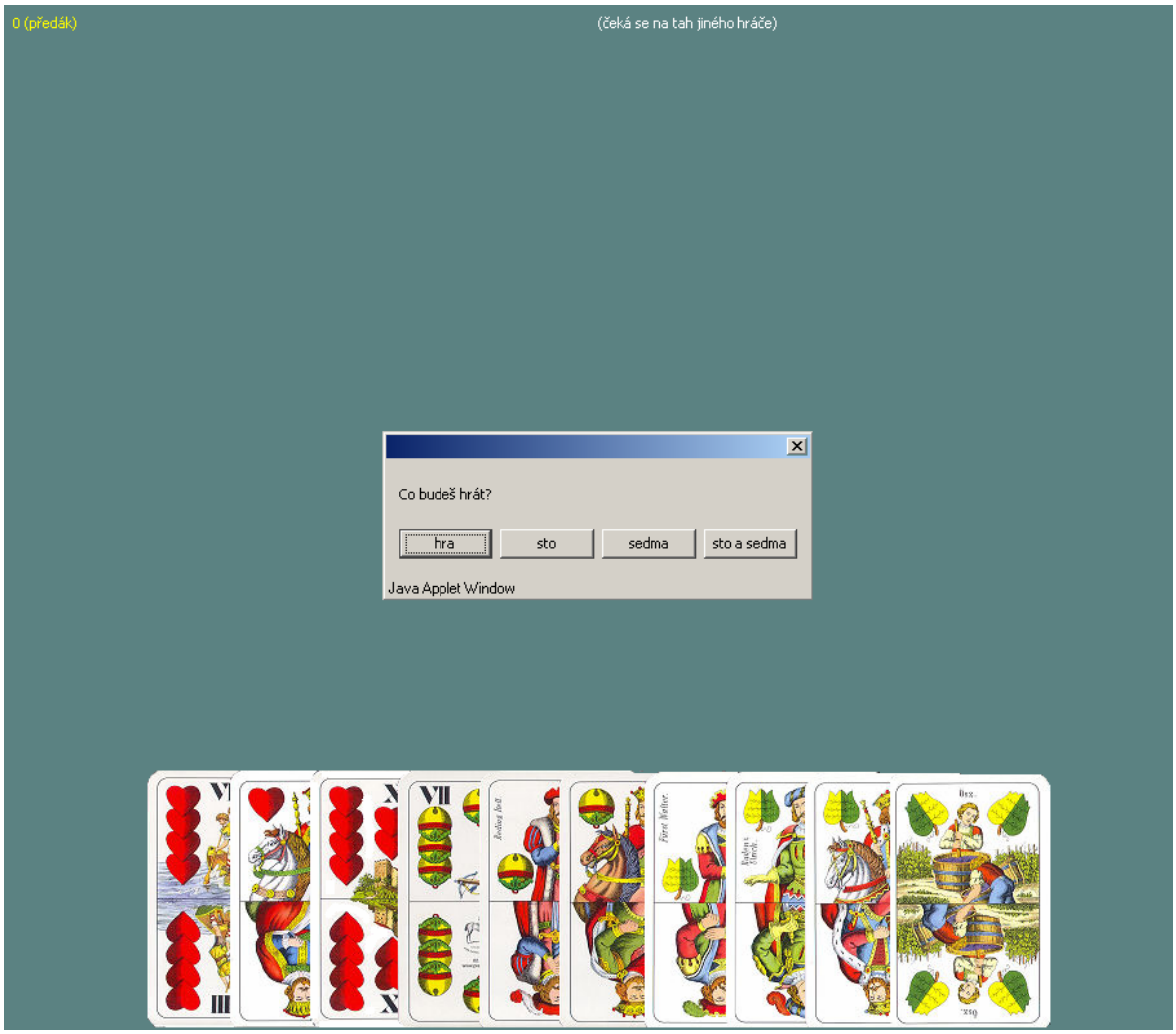
Obr. 16 - Volba barvy hry

Pak vybírá dvě karty – talon. Pokud zvolil dobrou barvu, nelze do talonu dát žádné eso ani desítku. Jinak může odhodit libovolnou kartu.

Po volbě talonu ostatní dva hráči jsou dotázáni na barvu hry.

Jestli zvolí špatnou barvu, dostanou karty z talonu a vyberou vlastní dvě. Jinak se iniciativa vrací k předákovi.

Potom v každém případě následuje licitace (viz obr. 17).



Obr. 17 - Licitace

Hráči jsou postupně tázáni na vyjádření ke hře. Po skončení licitace je předák vyzván k prvnímu tahu.

Hra končí buď poté, co hráči vynesou všechny karty, nebo pokud někdo neuhraje hru Betl nebo Durch.

Po ukončení jedné hry je hráč nalevo od hráče, který minule volil trumfy (po směru hodinových ručiček) předákem a zahajuje novou hru. Ostatní hráči čekají na dotaz na barvu.

7 Závěr

Zpracovávání tohoto projektu mi dalo příležitost procvičit si práci s applety a servlety ve vývojovém prostředí Eclipse. Naučila jsem se tvorbě webových aplikací a práci se sítí v Javě. Nastudovala jsem problematiku vytváření appletů. Obeznámila jsem se s postupem návrhu uživatelského rozhraní a zpracování vstupu od uživatele. Vyzkoušela jsem taky práci s obrázky pomocí knihovny Swing. Navrhovala jsem vhodnou objektovou strukturu pro realizaci programu.

Herní server je možné spustit v kterémkoli prostředí obsahující webový server s podporou spouštění Java Servletů. Uživatel se může ke hře připojit z libovolné platformy, která umožňuje spouštění Java Appletu.

Rozlišení obrazovky není zásadní pro běh programu, protože rozměry appletu jde jednoduše změnit v HTML kódu. Pozice obrázků uvnitř programu se vždy vypočítávají podle zadané velikosti (to znamená, že například karty na stole se vždy zobrazí uprostřed i při různých rozměrech).

Velikost appletu je přednastavena na 950x750, což znamená, že pokud má uživatel nastaveno menší rozlišení než 1024x768 a nenastaví jinou velikost appletu, tak se pravděpodobně applet nevejde celý najednou na obrazovku.

Tento projekt by bylo možné rozšířit například o jiné varianty hry, kterých existuje velké množství. Další možnost rozšíření spočívá v přidání hráče řízeného umělou inteligencí.

8 Literatura

[1] Netcraft. April 2009 Web Server Survey [online]. 2009 [cit. 2009-04-19]. Dostupný z WWW: <http://news.netcraft.com/archives/2009/04/06/april_2009_web_server_survey.html>.

[2] *HTML 4 Specification : Objects, Images, and Applets* [online]. 1999 [cit. 2009-04-22]. Dostupný z WWW: <http://www.w3.org/TR/html4/struct/objects.html#edef-APPLET>.

[3] *The Java Tutorials : Deploying Applets* [online]. c1995-2008 , Last Updated 2/14/2008 [cit. 2009-04-23]. Dostupný z WWW: <http://java.sun.com/docs/books/tutorial/deployment/applet/deployindex.html>.

[4] *Flash Player Penetration : Flash content reaches 99.0% of Internet viewers* [online]. 2008 [cit. 2009-04-20]. Dostupný z WWW: <http://www.adobe.com/products/player_census/flashplayer/>.

[5] *Eclipse Downloads* [online]. Dostupný z WWW: <http://www.eclipse.org/downloads/> [cit. 2009-04-20]

[6] *Resin Download* [online]. Dostupný z WWW: <http://www.caucho.com/download/> [cit. 2009-04-20]

[7] *File:Huncards.jpg*. [online]. Dostupný z WWW: <http://commons.wikimedia.org/wiki/Image:Huncards.jpg?uselang=cs> [cit. 2009-04-20]

[8] *GIMP - The GNU Image Manipulation Program* [online]. Dostupný z WWW: <http://www.gimp.org/> [cit. 2009-04-20]

[9] *PHP: Hypertext Preprocessor* [online]. Dostupný z WWW: <http://www.php.net/> [cit. 2009-04-20]

[10] NCSA HTTPd Development Team. *Common Gateway Interface* [online]. [1998] , last modified 1-21-98 [cit. 2009-04-15]. Dostupný z WWW: <http://hoohoo.ncsa.uiuc.edu/cgi/intro.html>.

[11] SNÍŽEK, Martin. *AJAX – kde jsou hranice?* [online]. 13.9.2005 [cit. 2009-04-18]. Dostupný z WWW: <http://www.snizekweb.cz/clanky/ajax-kde-jsou-hranice/>. ISSN 1802-2103.

[12] MAIXNER, David. *Java servlety, JSP a webové servery s jejich podporou*. [s.l.], 2002. 41 s. Masarykova univerzita, Fakulta informatiky. Vedoucí bakalářské práce Mgr. Lubomír Markovič. Dostupný z WWW: <http://www.slunecnice.cz/sw/java-servlety/>.

[13] OMASTA, Vojtěch, RAVIK, Slavomír. *Karty, hráči, karetní hry*. : Levné knihy KMa, 2005. 582 s. ISBN 80-7309-206-9.

[14] Refsnes Data. *Full Web Building Tutorials : HTML applet tag* [online]. c1999-2009 [cit. 2009-09-19]. Dostupný z WWW: <http://w3schools.com/tags/tag_applet.asp>.

[15] VisualBuilder. *JSP Tutorial* [online]. c2009 [cit. 2009-04-21]. Dostupný z WWW: <http://www.visualbuilder.com/jsp/tutorial/>.

9 Seznam obrázků

10 Seznam tabulek

11 Příloha A – Pravidla hry

Mariáš se hraje se 32 kartami ve čtyřech barvách – srdce, kule, listy a žaludy. Hodnoty karet jsou 7, 8, 9, desítka, spodek, svršek, král a eso. Použitá pravidla jsou vypracována podle [13].

Hry se účastní tři hráči. První hráč je „předák“. Druzí dva hráči hrají spolu proti němu. Role předáka se po každé hře posune o jednoho hráče po směru hodinových ručiček.

Předák dostane v první části sedm karet, ostatní hráči po pěti kartách. V druhém kole rozdávání všichni tři dostanou po pěti kartách, předák však svých pět karet nesmí vzít, dokud nezvolí trumfovou barvu – to udělá vybráním jedné karty (může vybrat i ze zakrytých karet – tzv. „z lidu“).

Předák ze všech dvanácti karet vybere dvě karty, které se mu nehodí – tzv. „talon“. Ty odloží rubem nahoru na stůl.

Do talonu se nesmí hodit žádné esa ani desítky (výjimkou je, pokud hraje tzv. špatnou barvu – viz později).

Předák má dvě možnosti: Může hned zahlásit špatnou barvu, potom bude hrát buď variantu Betl, nebo Durch (viz dále). Nebo může hlásit hru (dobrou barvu) – k té se vyjadřují ostatní hráči tak, že buď přijmou dobrou barvu, anebo hlásí barvu špatnou – v tom případě daný hráč dostane do ruky karty z talonu, přičemž vzápětí dvě (stejné jako dostal nebo jiné z ruky) do talonu položí.

Pokud někdo hraje hru Betl (ať už hráč, který na začátku volil trumfy nebo nějaký jiný), tak jiný hráč může převýšit a hrát Durch (opět dostane talon a určí svůj).

Pokud se hraje dobrá barva, tak hráč, který je předákem, ukáže zvolenou trumfovou kartu a řekne zvolenou variantu hry (hra, sedma, sto, sto a sedma).

Ostatní hráči se vyjadřují – mohou dát flek na hru, a jestli byla hlášena sedma, tak i flek na sedmu. Rovněž mají možnost hlásit sedmu proti nebo sto proti. Fleky mají vliv na konečné vyúčtování – každý flek zdvojnásobuje cenu (viz dále).

Pokud někdo z hráčů hlásil sedmu, zavazuje se, že vezme poslední štych trumfovou sedmou. Jestliže někdo hlásil stovku, zavazuje se, že počet získaných bodů bude alespoň sto.

Předák (hráč, který volil), vynáší první kartu. Pro vynášení karet platí následující pravidla:

- povinnost „ctít barvu“ – každý hráč musí táhnout kartou stejné barvy, jako byla první vynesená
- povinnost přebíjet – každý hráč musí hodit kartu vyšší hodnoty
    - pořadí hodnot (od nejnižší) je 7,8,9, spodek, svršek, král, desítka, eso
- nemůže-li přebít, musí dát kartu stejné barvy
- nemá-li žádnou kartu dané barvy, musí hodit trumfovou kartu (trumfové karty přebíjejí karty jiných barev)
- nemá-li ani trumfovou kartu, může hodit kartu libovolnou (ta nikdy nepřebíjí, nezávisle na hodnotě)

Štych (trojici karet) si bere hráč, který hodil nejvyšší kartu. Karty položí na svoji kupku. První vynáší vždy hráč, který bral poslední štych.

V mariáši se hraje o počet dosažených bodů. Každá desítka a eso na kupce se počítá za deset bodů, ten, kdo zvedl poslední štych, si přičte dalších deset.

Dva hráči, kteří hrají spolu, si body sčítají.

Navíc se počítají „hlášky“ – jestliže někdo dostal dvojici svršek a král v jedné barvě, přičte si 20 bodů. Pokud to byla barva trumfová, přičte si místo toho 40 bodů. Při vynášení svrška z hlášky se karta pokládá lícem nahoru na vlastní kupku (o hlášku nelze přijít – i když je přebita, tak se počítá hráči, který ji vynesl).

Při hře ve špatné barvě se nesčítají žádné body, hlášky tedy nemají žádný zvláštní význam. Desítka je narozdíl od hry v dobré barvě ve stupnici hodnot mezi devítkou a spodkem. Trumfy rovněž nehrají žádnou roli.
Existují dvě varianty hry ve špatné barvě:
Betl (neboli žebrák) je hra, kdy hráč nesmí vzít ani jediný štych.
Durch je hra, kdy hráč musí vzít úplně všechny štychy.

Pokud předák vyhraje, dostává částku od každého z hráčů. Stejně tak, jestliže předák prohraje, platí částku každému z hráčů. (Dá se tedy říci, že předák hraje o dvojnásobek.)

Obyčejná hra je hodnocena jako jedenkrát základ. Každý flek na hru cenu zdvojnásobuje (takže například při základu hry 0,50 Kč by po prohrané hře se třemi fleky předák platil každému 4 Kč).

Pokud někdo poslední štych vzal trumfovou sedmou, vyhrává jedenkrát základ. Jestli byla sedma předem hlášená, tak se cena zdvojnásobuje. Cenu sedmy rovněž zdvojnásobuje každý flek. Jestliže však hráč trumfovou sedmu do posledního štychu hodí, ale někdo ji přebije vyšším trumfem, sedma je „zabitá“ a hráč tu částku prohrává (a to i kdyby mu sedmu „zabil“ jeho spoluhráč).

Za každých deset bodů nad devadesát se cena hry zdvojnásobuje (tzv. „stovka“). Pokud stovka byla předem hlášena, počítá se další dvojnásobek.

Když byly zvoleny srdce jako trumfy, tak se celkové sumy zdvojnásobují.

12 Příloha B – Popis metod a proměnných hlavních tříd

Seznam hlavních metod třídy Status

public void newGame()	Rozdá nový balíček karet a zahájí hru.
public void dealCards(Stack deck)	Rozdá karty.
private void reset()	Vynuluje všechny proměnné.
public boolean isMoveLegal(Move move)	Zjistí, jestli je podle pravidel, aby hráč, který je na tahu, táhnul danou kartou.
private void takeTable()	Volá se po provedení třetího tahu v jednom štychu.
private void gameFinished(int takingPlayer, boolean seven, boolean taken)	Volá se po skončení jedné hry.
public void move(Card card)	Provede jeden tah.
public void selection(Card trumpCard, Card first, Card second, GameType game)	Volá se po vybrání typu hry, talonu a případně trumfové karty.
public void color(boolean accept)	Volá každý hráč po volbě barvy.
public void gameTypeSelected(GameType newGameType)	Volá se po upřesnění typu hry (obyčejná, stovka, sedma, stovka a sedma).
public void gameTypeComment(EnumSet<Comment> comment)	Vrátí všechny možné vyjádření ke hře v dané situaci.

Tab. 5 - Seznam metod třídy Status

Seznam datových proměnných třídy Status

int	commentingPlayer	Který hráč se právě vyjadřuje ke hře při licitaci.
Controller[]	controllers	Controllery pro komunikaci s hráči.
int	flek7Count	Počet fleků na sedmu.
int	flekCount	Počet fleků na hru / na Betl / na Durch.
boolean	g100	Jestli se hraje stovka proti.
boolean	g7	Jestli se hraje sedma proti.
ArrayList< GameResultListener>	gameResultListeners	Objekty, které mají být informovány o konci hry.
boolean	gameStarted	Jestli už začala hra (nebo jestli se teprve volí trumfy / typ hry / flekuje).
Status.GameType	gameType	Zvolený typ hry.
ArrayList< Move>	moves	Historie všech tahů.
Stack[]	playerHand	Karty, které mají hráči v rukou.
PlayerStack[]	playerStack	Karty, které mají hráči na kupkách.
int	round	Flekovací kolo.

Stack	talon	Karty, které jsou v talonu.
Card.Suite	trump	Trumfová barva (nebo null při špatné barvě).
Card	trumpCard	Karta, která byla zvolena jako trumfová (nebo null při špatné barvě hry).

Tab. 6 - Seznam datových proměnných třídy Status

Seznam hlavních metod třídy Controller

void makeMove()	Je zavoláno, když se čeká na provedení tahu.
void notifyMoveMade(Move move)	Je zavoláno, když byl proveden tah.
void notifyGameOver(String message)	Je zavoláno, když hra byla dohrána.
void startGame()	Je zavoláno, když se čeká na zvolení trumfové barvy a odhození talonu (tento hráč je předák).
void selectForemanGameType()	Je zavoláno, když se čeká na zvolení typu hry (hra, 100, 7, 107, Betl, Durch).
void gameColor()	Je zavoláno, když se čeká na potvrzení dobré barvy hry.
void startBadColorGame(Card firstTalon, Card secondTalon)	Je zavoláno, když se čeká na odhození talonu (po převzetí iniciativy). Musí hrát špatnou barvu hry.
void startDurchGame(Card firstTalon, Card secondTalon)	Je zavoláno, když se čeká na odhození talonu (po převzetí iniciativy). Musí hrát durch.
void commentGameType(GameType gameType, Card trumpCard, int flekCount, int flek7count, boolean g7, boolean g100, int round)	Čeká na vyjádření ke hře.
void notifyGameStart(Card trumpCard, GameType gameType, boolean g7, int foreman)	Je zavoláno těsně před prvním tahem.
void setNumber(int number)	Nastaví číslo tohoto controlleru (0 až 2).
void setHand(Stack hand)	Nastaví karty v ruce tohoto hráče.

Tab. 7 - Seznam metod třídy Controller

Seznam hlavních metod třídy Card

public static Stack newDeck()	Vrátí nový balíček.
public static Card valueOf(Rank rank, Suite suite)	Vrátí instanci karty na základě barvy a hodnoty.
public static Card valueOf(int hash)	Vrátí instanci karty na základě barvy a hodnoty.
public boolean ranksAbove(Card card, boolean isGameGood, Card.Suite trump)	Jestli daná karta přebíjí tuto kartu v zadané situaci.
public static int getHighest(Card first, Card second, Card third, boolean isGameGood, Suite trump)	Zjistí, která karta je nejvyšší v zadané situaci.

Tab. 8 - Seznam metod třídy Card

Seznam datových proměnných třídy Card

private final Rank rank	Hodnota této karty.
private final Suite suite	Barva této karty.

Tab. 9 – Seznam datových proměnných třídy Card

Seznam datových proměnných třídy Move

private final int player	Číslo hráče provádějícího tah.
private final Card card	Karta, kterou hráč táhne.
private boolean isMarriage	Jestli je tento tah královna z hlášky.

Tab. 10 - Seznam datových proměnných třídy Move

Seznam hlavních metod třídy Stack a jejích podtříd

public boolean containsAny(Suite suite)	Vrátí true, jestli tato kupka obsahuje alespoň jednu kartu dané barvy.
public int containsCount(Suite suite)	Vrátí, kolik tato kupka obsahuje karet zadané barvy.
public boolean containsMariage(Suite suite)	Jestli tento balíček obsahuje dámu a krále v zadané barvě.
public boolean hasHigher(Card card, boolean game)	Vrátí true, jestli tato kupka obsahuje alespoň jednu kartu, která je vyšší než zadaná karta a zároveň je stejné barvy.
public void shuffle()	Zamíchá tento balíček.
public void sort(final boolean isGameGood, final Suite trumpSuite)	Setřídí karty v ruce.

Tab. 11 - Seznam metod třídy Stack

pouze TableStack:

public void addQueenInMarriage(Suite queenOfSuite)	Volá se při vynesení královny z hlášky. Karta se na stole nezobrazuje, ale počítá se s ní jako by tam byla (např. při rozhodování, který hráč bere štych).
public List<Card> getWithoutMarriages()	Vrátí karty na stole bez hlášek (tzn. vrátí karty, které se na stole mají zobrazovat).
public boolean isMarriage(int index)	Vrátí true, jestli karta na dané pozici je královna z hlášky.
public int getHighestIndex(boolean isGoodGame, Suite trump)	Vrátí index nejvyšší karty.

Tab. 12 - Seznam metod třídy TableStack

pouze PlayerStack:

public void addQueenInMarriage(Suite marriageSuite)	Přidá hlášku dané barvy.
public int getRawScore()	Spočítá body v tomto balíčku (esa a desítky) bez hlášek.
public int countScore(Suite trumpSuite)	Spočítá body v tomto balíčku (esa a desítky), včetně všech hlášek.
public int countScoreOneMarriage(Suite trump)	Spočítá body v tomto balíčku (esa a desítky), ale včetně jenom jedné hlášky (pokud je v balíčku trumfová, tak trumfové).
public boolean containsMarriage(Suite suite)	Vrátí true, jestli balíček obsahuje hlášku dané barvy.

Tab. 13 - Seznam metod třídy PlayerStack

Seznam hlavních metod třídy GameBoard

private void paintCard(final Graphics g, Image cardImage, int x, int y)	Vykreslí jednu kartu na stůl.
private void paintHand(Graphics g, Stack stack, int x, int y, boolean centerX)	Vykreslí karty, které má hráč v ruce.
private boolean canTalon(Card card)	Zjistí, jestli zadaná karta může jít do talonu.
private void cardSelected(Card card)	Je zavoláno, když je kliknuto na nějakou kartu.
public void paint(Graphics g)	Hlavní metoda vykreslování této komponenty. Swing ji volá vždy, když je potřeba překreslit okno.
private void paintPlayerNumber(Graphics g)	Vykreslí číslo hráče do levého horního rohu.
private void paintTrumpCard(Graphics g)	Vykreslí kartu, která právě byla zvolena jako trumfová.
private void paintMessage(Graphics g)	Vykreslí zprávu, která se vysvětluje současnou fázi hry.
private void paintTable(Graphics g)	Vykreslí případné karty, které jsou položeny na stole.
private void paintPlayerStack(Graphics g, PlayerStack stack, int x, int y, int seed)	Vykreslí jednu kupku, na které jsou uloženy karty, které jeden hráč sebral, přičemž karty budou vykresleny jako "rozházené".
private void paintPlayerStacks(Graphics g)	Vykreslí kupky všech tří hráčů.
private void paintBackground(Graphics g)	Vymaže pozadí.

private void refreshPlayerStacks()	Načte reálné kupky hráčů ze stavu hry do vykreslovaných kupek (nemůže se brát přímo při vykreslování, protože může být ještě neodklepnutý štych, který by se tak zobrazoval dvakrát (jednou na stole a jednou na kupce).
private void askTalon()	Zeptá se, na co bude hráč odhazovat.
private void askBad()	Zeptá se, jakou variantu hry při špatné barvě si hráč přeje.
private synchronized void showGameResult()	Zobrazí výsledek jedné hry.
public void selectForemanGameType()	Zeptá se, jakou variantu hry při dobré barvě si hráč přeje.
public synchronized void commentGameType(GameType newGameType, Card trumpCard, int flekCount, int flek7coun	Zeptá se na vyjádření se ke hře daného hráče.

Tab. 14 - Seznam metod třídy GameBoard

Seznam datových proměnných třídy GameBoard

private final TableStack currentTable	Obsah stolu.
private TableStack lastTable	Obsah stolu (před odklepnutím štychu).
private int currentStartingPlayer	Hráč, který vyjíždí.
private int lastStartingPlayer	Hráč, který vyjížděl (před odklepnutím štychu).
private PlayerStack[] currentPlayerStacks	Kupky hráčů.
private final PlayerStack[] lastPlayerStacks	Kupky hráčů (před odklepnutím štychu).
private final Stack playerHand	Karty v hráčově ruce.
private int number	Číslo tohoto hráče.
private GameType gameType	Typ probíhající hry.
private Suite trumpSuite	Která barva je trumfová.
private boolean g7	Jestli se hraje sedma proti.
private int foreman	Číslo hráče, který je předák.
private String gameResult	Zpráva zobrazená po dohrání jedné hry.

Tab. 15 - Seznam datových proměnných třídy GameBoard

Seznam hlavních metod třídy ImageManager

public Image getCardImage(Card card)	Vrátí obrázek zadané karty.
public Image getCardReverseImage()	Vrátí obrázek rubu karty.
public int getCardHeight()	Vrátí výšku karty.
public int getCardWidth()	Vrátí šířku karty.

Tab. 16 - Seznam metod třídy ImageManager

Seznam datových proměnných třídy ImageManager

private final HashMap<Card, Image> cardsMap	Hashovací tabulka, pro každou kartu obsahuje její obrázek.

Tab. 17 - Seznam datových proměnných třídy ImageManager

13 Příloha C – Část zdrojového kódu

Tato část obsahuje ukázku zdrojového kódu.

Herní pravidla

Tato metoda ověřuje, jestli je tah nějakou kartou podle pravidel. Patří do třídy **Status**. Odpovídá aktivitnímu diagramu na obr. 8.

```
    /**
     * Zjistí, jestli je podle pravidel táhnout danou kartou v zadané
situaci.
     * @param card          karta, která se má ověřit
     * @param hand          ruka hrajícího hráče
     * @param gameType      aktuální typ hry
     * @param table         případné karty na stole
     * @param trump         barva, která je trumfová
     * @param g7            jestli se hraje sedma proti
     * @return              <code>true</code>, jestli je tah podle
pravidel
     */
    public static boolean isMoveLegal(Card card, Stack hand, GameType
gameType, TableStack table, Suite trump, boolean g7) {
        assert table.size() < 3;

        // nelze vynést trumfovou sedmu, pokud se hraje sedma nebo sedma
proti a
        // není to poslední trumf v ruce hráče
        if ((gameType.is7() || g7) && card.getSuite() == trump &&
card.getRank() == Rank.SEVEN
                && hand.containsCount(trump) > 1) {
            return false;
        }

        // pokud má hráč hlášku, musí vynést dámu dřív než krále
        if (gameType.isGood() && card.getRank() == Rank.KING &&
hand.containsMariage(card.getSuite())) {
            return false;
        }

        if (table.size() > 0) { // pokud nevyjíždí

            // pomocné proměnné

            Card startCard = table.get(0); // první karta
            Card.Suite startSuite = startCard.getSuite();

            // zjisit, jestli ctí barvu
            if (card.getSuite() != startSuite) {
                if (hand.containsAny(startSuite)) {
                    // nectí barvu, a přitom má - nelze
                    return false;
                }
```

```
                // nemá
                if (hand.containsAny(trump)) {
                    if (card.getSuite() != trump) {
                        // má trumfy a přitom jimi netáhne - nelze
                        return false;
                    }

                    // má trumfy a táhne jimi, pokračujeme dál
                } else {
                    // nemá ani barvu, ani trumfy - může hodit cokoli
                    return true;
                }
            }

            Card higher;

            boolean game = gameType.isGood();

            assert card.getSuite() == trump || card.getSuite() ==
startSuite;

            if (table.size() == 2) {
                Card secondCard = table.get(1);

                if (secondCard.ranksAbove(startCard, game, trump)) {
                    higher = secondCard;
                } else {
                    higher = startCard;
                }
            } else
                higher = startCard;

            if (card.getSuite() == trump) {
                // hráč vyjíždí trumfy, musí přebít, jestli může
                if (!card.ranksAbove(higher, game, trump)) {
                    if (hand.hasHigher(higher, game)) {
                        // má vyšší - nelze
                        return false;
                    }
                }
            } else {
                // hráč nevyjíždí trumfy, ale cti barvu

                assert card.getSuite() != trump;
                assert startCard.getSuite() != trump; // to by nectil
                // barvu, viz nahoře

                // druhá karta byla trumfova, vyjizdejici nemusi prebijet
                if (higher.getSuite() == trump)
                    return true;

                if (!card.ranksAbove(higher, game, trump)) {
                    if (hand.hasHigher(higher, game)) {
                        // má vyšší - nelze
                        return false;
                    }
                }
```

```
            }

        }
        return true;
    }
```

Vykreslování

Tyto metody se starají o zobrazování grafiky. Patří do třídy **GameBoard**. Vykreslování je pro přehlednost rozděleno do několika metod, přičemž každá vykresluje jednu část herní plochy.

Metoda **paint** je vždy volána, když je potřeba překreslit okno, volá ostatní metody, které kreslí jednotlivé části.

```
    /**
     * Hlavní metoda vykreslování této komponenty. Swing ji volá vždy,
když je potřeba překreslit okno.
     * @param g     pomocí tohoto objektu se vykresluje, je předáván
Swingem
     */
    @Override
    public void paint(Graphics g) {
        paintBackground(g);


        paintHand(g, playerHand, getWidth() / 2, getHeight() -
ImageManager.getInstance().getCardHeight(), true);
        paintPlayerStacks(g);
        paintTable(g);
        paintPlayerNumber(g);
        paintMessage(g);
        paintTrumpCard(g);
    }
```

Metoda **paintPlayerNumber** vykresluje číslo hráče do rohu okna.

```
    /**
     * Vykreslí číslo hráče do levého horního rohu.
     * @param g     pomocí tohoto objektu se vykresluje, je předáván
Swingem
     */
    private void paintPlayerNumber(Graphics g) {
        if(number<0) return;

        String message=String.valueOf(number);
        if(number==foreman) {
            message+=" (předák)";
        }
```

```
        g.setColor(Color.yellow);
        g.drawString(message, 20, 20);
    }
```

Metoda **paintTrumpCard** vykreslí kartu, která je zvolena jako trumfová, stranou od ostatních karet. Toto platí pouze během odpovídající části licitace, jindy nedělá nic.

```
    /**
     * Vykreslí kartu, která právě byla zvolena jako trumfová.
     * @param g     pomocí tohoto objektu se vykresluje
     */
    private void paintTrumpCard(Graphics g) {
        if (selectedTrumpCard != null
                && (currentStage == Stage.TALON || currentStage ==
Stage.TALON_SECOND || currentStage == Stage.COMMENTING)) {
            Image img =
ImageManager.getInstance().getCardImage(selectedTrumpCard);
            // Image img =
ImageManager.getInstance().getCardReverseImage();
            paintCard(g, img, (getWidth() - img.getWidth(null)) / 2,
(getHeight() - img.getHeight(null)) / 2);
        }

    }
```

Metoda **paintMessage** vykresluje zprávu pro hráče, která mu oznamuje, co má dělat.

```
    /**
     * Vykreslí zprávu, která se vysvětluje současnou fázi hry.
     * @param g     pomocí tohoto objektu se vykresluje
     */
    private void paintMessage(Graphics g) {
        g.setColor(Color.white);
        String message;
        if (lastTable != null) {
            message = "Klikni myší nebo stiskni libovolnou klávesu.";
        } else {
            if (currentStage == null)
                message="(čeká se na ostatní hráče)";
            else
                message = currentStage.getMessage();
        }
        g.drawString(message, getWidth() / 2, 20);
    }
```

Metoda **paintTable** vykreslí karty, které jsou právě položené na stole.

```
    /**
     * Vykreslí případné karty, které jsou položeny na stole.
     * @param g     pomocí tohoto objektu se vykresluje
     */
    private void paintTable(Graphics g) {
        TableStack table;
        int sp;

        if (lastTable != null) {
            table = lastTable;
            sp = lastStartingPlayer;
        } else {
            table = currentTable;
            if (table == null || table.size() == 0)
                return;

            sp = currentStartingPlayer;
        }

        int num=number!=-1 ? number : 0;

        //kolikata karta na stole se ma zobrazit vpravo / dole / vlevo
        int indexRight = (2 - sp + num) % 3;
        int indexBottom = (3 - sp + num) % 3;
        int indexLeft = (4 - sp + num) % 3;

        Card bottom = null, left = null, right = null;

        if (indexBottom < table.size() && !table.isMarriage(indexBottom))
            bottom = table.get(indexBottom);
        if (indexLeft < table.size() && !table.isMarriage(indexLeft))
            left = table.get(indexLeft);
        if (indexRight < table.size() && !table.isMarriage(indexRight))
            right = table.get(indexRight);

        int x, y;

        int cardHeight = ImageManager.getInstance().getCardHeight();
        int cardWidth = ImageManager.getInstance().getCardWidth();

        int startX = (getWidth() - cardWidth) / 2;
        int startY = (getHeight() - cardHeight) / 2;

        if (bottom != null) {
            x = startX;
            y = startY + cardHeight / 2 + SPACE_TABLE;

            paintCard(g, ImageManager.getInstance().getCardImage(bottom),
x, y);
        }

        if (right != null) {
            x = startX + cardWidth / 2 + SPACE_TABLE;
```

```
            y = startY - cardHeight / 2 - SPACE_TABLE;
            paintCard(g, ImageManager.getInstance().getCardImage(right),
x, y);
        }

        if (left != null) {
            x = startX - cardWidth / 2 - SPACE_TABLE;
            y = startY - cardHeight / 2 - SPACE_TABLE;
            paintCard(g, ImageManager.getInstance().getCardImage(left), x,
y);
        }
    }
```

Metoda **paintPlayerStacks** vykresluje kupku karet, které hráč vzal.

```
    /**
     * Vykreslí kupky všech tří hráčů.
     * @param g     pomocí tohoto objektu se vykresluje
     */
    private void paintPlayerStacks(Graphics g) {
        assert lastPlayerStacks != null;

        int cardWidth = ImageManager.getInstance().getCardWidth();
        int cardHeight = ImageManager.getInstance().getCardHeight();

        int num = number!=-1 ? number : 0;

        PlayerStack bottom=lastPlayerStacks[num];
        PlayerStack left=lastPlayerStacks[(num+1) %PLAYER_COUNT];
        PlayerStack right=lastPlayerStacks[(num+2) %PLAYER_COUNT];

        if (bottom != null)
            paintPlayerStack(g, bottom, getWidth() - cardWidth,
getHeight() - cardHeight * 2, 123);
        if (left != null)
            paintPlayerStack(g, left, 0, cardHeight / 4, 2341);
        if (right != null)
            paintPlayerStack(g, right, getWidth() - cardWidth, 0, 234);
    }
```

Metoda **paintBackground** vyprázdní celou plochu, aby nezůstávaly vidět obrázky z předchozího kreslení

```
    /**
     * Vymaže pozadí.
     */
    private void paintBackground(Graphics g) {
        // smazat pozadi
        g.setColor(Color.DARK_GRAY);
        g.fillRect(0, 0, getWidth(), getHeight());
        }
```