VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

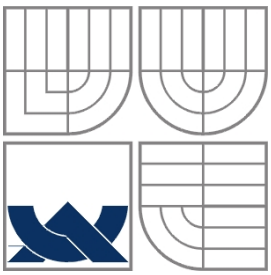
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

MULTIPLAYER KARETNÍ HRY ŽOLÍKY

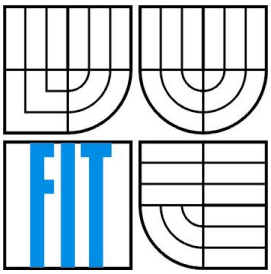
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE LUKÁŠ MAREK
AUTHOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

MULTIPLAYER KARETNÍ HRY ŽOLÍKY

MULTIPLAYER OF THE WHISP CARD GAME

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

LUKÁŠ MAREK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. DAVID KUBÁT

BRNO 2011

Abstrakt

Tato bakalářská práce popisuje návrh a implementaci karetní hry žolíky. Hra je navržena jako klient-server aplikace. Uživatelé se připojí na server a mají možnost přidat se k již vytvořené hře nebo si založit vlastní hru. Aplikace je implementována v programovacím jazyce C++ s použitím frameworku Qt.

Abstract

This bachelor's thesis describes the design and implementation of the whisp card game. The game is designed as a client-server application. Users connect to the server and have the opportunity to join an already established game or create your own game. The application is implemented in C++ using the Qt framework.

Klíčová slova

Karetní hra, žolík, síťová aplikace, Qt framework, multiplatformní.

Keywords

Card game, joker, network application, Qt framework, multiplatform.

Citace

Lukáš Marek: Multiplayer karetní hry žolíky, bakalářská práce, Brno, FIT VUT v Brně, 2011

Multiplayer karetní hry žolíky

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Davida Kubáta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Marek
12. 5. 2011

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce panu Ing. Davidu Kubátovi a panu Ing. Jaroslavu Stružkovi, který se mnou bakalářskou práci konzultoval.

Obsah

1 Úvod

Karetní hry byly vždycky stolní společenská hra, ale s érou počítačů a internetu se otevřely nové možnosti společenských her. Jako první karetní hra, u které bylo umožněno on-line hraní, byl poker, který je dnes již velmi rozšířen. Později se začali vytvářet i jiné, nejen karetní, společenské hry jako např. Člověče, nezlob se nebo Dostihy a sázky. Avšak počítačová verze karetní hry žolíky ještě není vytvořena tak kvalitně, aby bylo hraní jednoduché a zábavné. Některé vyzkoušené verze jsou buď složité na ovládaní nebo hra není funkčně správně vytvořená.

Proto bylo cílem naší práce vytvořit karetní hru žolíky jako počítačovou hru, která musí dodržovat pravidla této karetní hry. Aplikace musí být multiplatformní tzn. musí umožňovat spojení více hráčů po síti. Součástí aplikace má být i editor rubových a lícových karet, který umožní vytvořit si vlastní sadu karet. V následujících kapitolách je popsán návrh, implementace a zhodnocení této aplikace.

V druhé kapitole je popsán návrh aplikace. Popisuje oficiální pravidla karetní hry a možné modifikace pravidel. Dále pak síťovou komunikaci, vysvětlení použitých komunikačních protokolů, možné propojení klientů a navržení vlastních komunikačních zpráv. Kapitola také popisuje multiplatformní framework Qt, který je použit pro programovou implementaci aplikace i pro vytvoření grafické části. Poslední podkapitola popisuje objektový návrh aplikace, který je vytvořen pomocí grafického jazyka UML.

V třetí kapitole se zabýváme samotnou implementací aplikace. Je rozdělena na implementaci klientské části aplikace, serverové části aplikace a implementace komunikačního protokolu. V podkapitole pro implementaci klienta je ukázána tvorba GUI a implementace tříd. Dále je popsán vlastní průběh hry před připojením nebo vytvořením hry a průběh samotné hry. V části pro server je vysvětlena implementace konkurentního serveru, připojování klientů a řízení hry. V poslední části je ukázán navržený komunikační protokol.

Čtvrtá kapitola je věnována zhodnocení celé práce a jsou zde popsány dosažené výsledky testování a jsou navržena možná rozšíření aplikace.

2 Návrh aplikace

V této kapitole si popíšeme návrh aplikace. Ukážeme si úplná pravidla hry, popíšeme možnosti síťové komunikace a navržený síťový komunikační protokol. Dále bude vysvětlen objektový návrh serveru a klienta.

2.1 Pravidla hry

Pravidla karetní hry žolíky jsou mnohdy upravena podle potřeb hráčů a málokdo zná oficiální pravidla. Při návrhu hry jsme vycházeli z oficiálních pravidel [1], ale snažili jsme se zjistit nejčastěji používané změny v pravidlech a umožnili jsme změnu chování hry. Změnu nastavením pravidel jsme umožnily při vytváření nové hry. Všechny možné modifikace pravidel budou vysvětleny spolu s vysvětlením oficiálních pravidel.

2.1.1 Počet hráčů

Podle oficiálních pravidel jsou zapotřebí 2 až 4 hráči, není sice výjimkou, že hraje více hráčů, ale vzhledem k počtu karet se doporučuje maximálně 4. Další možností je použít dvou balíčků karet, ale vzhledem k možnosti 4 násobného počtu stejně nominované karty se zde ztrácí možnost taktizovat. Hra se stává nepřehledná, a proto se tento způsob nedoporučuje.

Žolíky 2 hráči – pokud hrají pouze 2 hráči tak je hra velmi pružná a snadno se dosahuje záporných bodů protihráče.

Žolíky 3 hráči – z hlediska zkušeností jsou tři hráči optimální počet. Ve třech hráčích se moc často nestává, že hráči drží stejné karty jako ostatní.

Žolíky 4 hráči – v tomto počtu se přechází do tzv. fáze taktiky a je potřeba co nejrychleji nasbírat potřebný počet bodů pro vyložení. Ostatní hráči mohou sbírat stejné karty a vy nemusíte docílit ani vyložení.

2.1.2 Hodnoty karet

Pro hru žolíky je potřeba kanastová sada karet [2]. Sada obsahuje 52 karet, rozdělených do čtyř barev (srdce, káry, kříže, piky), s hodnotami 2, 3, 4, 5, 6, 7, 8, 9, 10, J (kluk), Q (dáma), K (král), A (eso). Eso může nabývat hodnoty 1 nebo 10 podle způsobu použití. Každá z karet je v sadě dvakrát a dále jsou v sadě čtyři žolíci. Jedná se tedy dohromady o 108 karet.

Hodnota karet je pro karty 2-10 shodná s označením karet. Pro obrázkové karty (tzn. kluk, dáma, král) je dána hodnota 10. Žolík slouží jako kterákoli karta, jeho hodnota se tedy určuje podle karty, místo které je použit. Eso má hodnotu 10, pokud je ale použito v postupce jak je vidět na obrázku 2.1, má hodnotu 1.

Obrázek 2.1: Eso s hodnotou 1 [1]

2.1.3 Skládání karet

Skládat karty je možno dvěma způsoby. Jeden způsob je skládání karet se stejným označením k sobě tzv. *špinavá postupka* nebo *vedlejší postupka*.. Musí být minimálně tři, aby se započítaly jako možná kombinace k vyložení, ale nesmí se opakovat karty stejné barvy. Je samozřejmě možné místo jedné karty použít žolíka, který má pak stejnou hodnotu jako ostatní karty. Některé možné kombinace jsou vidět na obrázku 2.3.

Druhou možností jsou tzv. čisté postupky. Jde o karty stejné barvy jdoucí hned po sobě, nejméně však tří karet je potřeba k vyložení. Pokud je potřeba žolík, jeho hodnota odpovídá kartě místo které je použit. Nelze ale použít dva žolíky vedle sebe, příklady nepovolených kombinací jsou vidět na obrázku 2.2.

Obrázek 2.3: Povolené kombinace [1]

Obrázek 2.2: Nepřípustné kombinace [1]

2.1.4 Začátek hry a rozdávání

Rozdávající hráč musí balíček na začátku hry zamíchat. Pak následuje snímání, snímá hráč, který sedí v protisměru hodinových ručiček od rozdávajícího. Odebere část karet (nejméně však tři) a hledá žolíka do tří karet ze spodku snímnutých. Pokud nalezne žolíka, nechá si ho do hry. Následně vezme první kartu ze spodku balíčku (pokud byl první žolík, bere následující) a umístí ji viditelně pod balíček karet. Tato karta slouží pro zavírání z ruky, kdy hráč nemusí brát z balíčku a tato karta se mu hodí do postupky v ruce (pozn. některé modifikace pravidel s touto kartou nepočítají, proto bude při vytvářená hry možnost tuto kartu zakázat). Po snímání rozdávající začne rozdávat karty. Začíná u hráče, který sedí po směru hodinových ručiček, a rozdá mu první tři karty. Dalším hráčům rozdává po dvou kartách do počtu 14 karet (pozn. další pravidlo, které je možné pozměnit při vytváření hry, je ovšem závislé na zvoleném minimálním počtu bodů při prvním vykládání, viz. dále). První hráč má o kartu navíc a tudíž začíná.

2.1.5 Průběh hry a vykládání

První hráč odhodí na stůl kartu, kterou nepotřebuje. Další hráč v pořadí si odebírá jednu kartu z balíčku a taky vyhodí kartu, která se mu nehodí. Další hráči toto opakují až do kola, které je určené na začátku hry. V oficiálních plavidlech je to 4. kolo (pozn. toto pravidlo bude také možné změnit). Od tohoto kola mohou hráči vykládat karty na stůl (viz. dále). Dále se mohou rozhodnout jestli na začátku své hry vezmou kartu z balíčku nebo kartu, kterou odhodil hráč před nimi. Pokud vezmou odhozenou kartu, musí ji vyložit na stůl, buď samotnou nebo se svými kartami. Pořád platí pravidlo, že na konci svého tahu musí odhodit jednu kartu.

Pro první vyložení karet na stůl je určen minimální součet hodnot karet, který je podle oficiálních pravidel 42 (pozn. minimální počet bodů bude také možné změnit), dále je nutné mít jednu čistou postupkou bez žolíka. Pokud má hráč poprvé vyloženo, může dokládat karty ke všem vyloženým kartám na stole. Dále může vyměnit svou kartu za žolíka ve vyložených kartách, ale musí ho zas použít na vyložení dalších svých karet. Pokud vyměněného žolíka nemůže použít, musí ho přenechat hráči, který je na řadě.

2.1.6 Konec hry

Ukončení nebo zavření hry je možné třemi způsoby:

1. Hráč vykládá a dokládá karty na stůl až mu zůstane jen jedna karta, kterou uzavře hru (položí ji na balíček odhozených karet rubem nahoru).
2. Hráč drží v ruce karty, dokud nemá všechny poskládány (tzn. všechny karty musí být součástí nějaké postupky, ať už špinavé nebo čisté) kromě jedné, kterou hru uzavře. Nazýváme to uzavření „z ruky“.

3. Stejný případ jako 2. bod, ale hráč použije kartu, která leží pod balíčkem karet a opět zavře kartou kterou má navíc.

Opět platí pravidla, že si hráč na začátku svého tahu vezme kartu (z balíčku nebo odhozenou), nutný minimální součet hodnot karet a jedna čistá postupka bez žolíka. V případě že hráč, který zavírá hru, má nepotřebného žolíka, zůstává mu do dalšího kola.

2.1.7 Počítání bodů

Pro každé ukončení hry (viz. kapitola 2.1.6 body 1, 2, 3) je bodování jiné. Každý hráč začíná s 0 body a prohraje první hráč, který dosáhne -1000 bodů.

1.

a) Pokud mají všichni hráči vyložené karty a jeden uzavře hru.
   Karty 2 až 10 se počítají stejně jako jejich hodnota. Pro karty kluk, dáma a král platí hodnota 10. Eso má v tomto případě hodnotu 20 a žolík 50. Hráč který hru uzavřel si muže přičíst 10 bodů. Ostatní hráči si spočítají hodnoty karet v ruce a tuto hodnotu si odečtou od celkového počtu svých bodů.
b) Pokud nemají všichni vyloženo.
   Pravidla sou podobná jako v případě 1a). Rozdíl je v hráčích, kteří nevyloží žádné karty na stůl. Ti si automaticky odečtou 100 bodů (žolíci se v tomhle případě nepočítají).

2. Stejné jako v bodech 1. a), b). S rozdílem, že všechny body se násobí dvakrát. Tzn. ten kdo uzavírá hru si přičte 20 bodů, kdo nevyloží karty tak si odečte 200 bodů a ostatní si odečtou hodnotu karet v ruce vynásobenou dvěma. Za každého žolíka je opět dvojnásobek tedy 100 bodů.
3. Stejné jako v bodě 2.

2.2 Síťová komunikace

V této kapitole si ukážeme rozdíl mezi používanými komunikačními protokoly, možné řešení připojování klientů, návrh našeho řešení a návrh vlastní síťové komunikace s komunikačním protokolem.

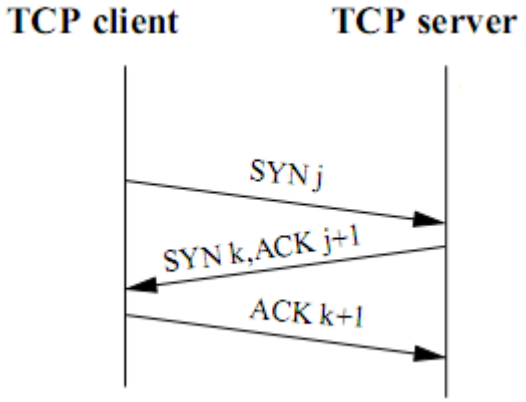
2.2.1 Komunikační protokoly

Existují základní dva komunikační protokoly TCP a UDP, které jsou vysvětlené v knize TCP/IP Network Administration [3]. Oba dva používají protokol IP pro posílání svých paketů. IP je základní protokol pro posílání dat v internetu. Protokol IP posílá data ve formě datagramů. Datagramy obsahují kromě vlastních dat např. informaci o IP adresách odesilatele a příjemce,

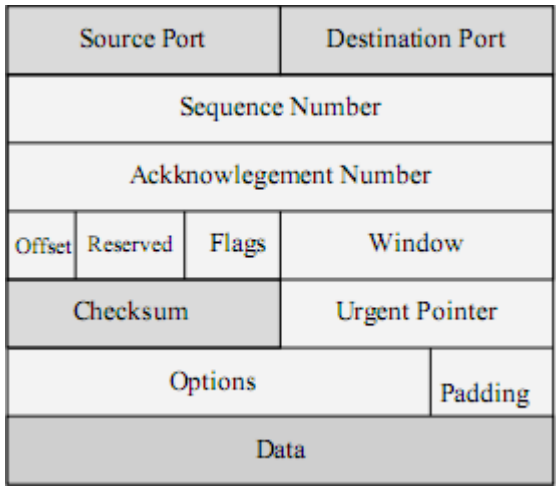
velikost datagramu, verzi protokolu atd. Protokol IP nijak nenavazuje spojení, neručí za doručení ani neurčuje na který port má datagram dojít. O to se starají protokoly TCP a UDP.

Protokol UDP je jednoduchý komunikační protokol, který nemá velkou režii. Svoje data posílá v paketech. V paketu jsou čísla portů pro identifikaci aplikačních protokolů. Protokol nezaručuje zda všechny pakety dorazí ani jestli dorazí ve správném pořadí. Při použití protokolu UDP nemáte jistotu, že zařízení na které data posíláte vůbec existuje. Protokol je vhodný pro nasazení v aplikacích, které pracují v reálném čase (např. VoIP, stream videa, online hry). Nebo v aplikacích, které komunikují stylem dotaz-odpověď, kde není problém odeslat dotaz opakovaně (např. DNS).

Protokol TCP je složitější. Pokud chceme navázat spojení přes TCP využívá se tzv. *trojcestný hand-shake*, který je zobrazen na obrázku 2.5. Pokud *trojcestný hand-shake* proběhne v pořádku, vytvoří se plně duplexní spojení (obousměrný přenos dat), které zanikne po ukončení komunikace nebo přerušení spojení. Dále je zaručeno, že data budou doručena beze ztráty a ve správném pořadí. TCP paket obsahuje (podobně jako UDP) čísla portů pro identifikaci aplikačních protokolů, ale oproti UDP obsahuje mnoho nastaveni (např. číslo paketu, potvrzovací číslo, kontrolní součet, atd.). Struktura TCP paketu je vidět na obrázku 2.4. Je tedy přenášeno větší množství dat a při posílání se potvrzuje, která data došla v pořádku, a která je nutno odeslat znovu. Přenos stejného množství dat trvá tedy déle než u UDP. Proto se TCP protokol používá u aplikací, u kterých je důležitá správnost dat a nevadí delší odezva při zpracování.



Obrázek 2.5: Trojcestný hand-shake [11]



Obrázek 2.4: TCP paket [12]

Existují knihovny založené na komunikaci protokolem UDP, které si sami řeší kontrolu dat a správné pořadí dat. Zaručují spolehlivý přenos s nižším množstvím přenášených dat. Jsou tedy vhodné do aplikací, kde potřebujeme rychlý, ale spolehlivý přenos.

Pro karetní hru využívající multiplayer není důležité, zda komunikace probíhá co nejrychleji, ale jestli se data přenášejí spolehlivě. Proto byl vybrán komunikační protokol TCP. Kniha UNIX Network Programming Volume 1 [4] vysvětluje příklady implementace síťové komunikace pro jazyk C++.

2.2.2 Propojení klientů

V této kapitole si ukážeme dvě základní architektury propojení více zařízení. První je tzv. *peep-to-peer*, jde o komunikaci ve které jsou si všechny zařízení rovny a komunikace probíhá mezi všemi zařízeními. Druhá architektura je tzv. *klient-server*, kdy jeden počítač je určen jako server a ostatní zařízení se k němu připojují.

Peer-to-peer znamená doslova rovný s rovným, protože každé zařízení pracuje jako klient i server současně. Použití peer-to-peer je v síťové karetní hře nevhodné, protože každý klient by si musel pamatovat všechny informace o ostatních klientech. Dále pak řízení a zjišťování stavu hry by bylo poněkud složité.

U *klient-server* architektury je oddělen server od klienta a je určen komunikační protokol mezi nimi. Server je většinou aplikace bez grafického uživatelské prostředí, je spuštěn na pozadí a pouze komunikuje s klienty. Klient většinou obsahuje uživatelské prostředí pro hru se všemi grafickými (popř. zvukovými) soubory. Klient posílá požadavky na server, který je zpracuje a odešle odpověď. Tato architektura je z pohledu síťových her výhodnější, protože klienti komunikují pouze se serverem, který přímo řídí průběh hry, pamatuje si veškerá nastavení popř. uchovává údaje o klientech. Ale při vyšším počtu klientů může docházet k přetěžováním sítě, nebo serveru.

2.2.3 Návrh síťové komunikace

Pro náš projekt jsme zvolili architekturu klient-server, protože je z pohledu síťových her výhodnější. Server bude obsluhovat připojování jednotlivých klientů, bude umožňovat vytvářet vlastní hry a připojovat klienty do již vytvořených her. Server je navržen jako konkurentní a neblokující, pro současnou obsluhu více klientů. Dále je navržen komunikační protokol mezi serverem a klientem, který je rozdělen do jednotlivých částí komunikace:

- Komunikace před zahájením hry
- Komunikace v průběhu hry
- Chat

Kompletní komunikační protokol viz. příloha 1.

2.3 Qt

Qt [5] je multiplatformní framework. Jeho vývoj začal roku 1991, když se Haavard Nord a Eirik Chambe-Eng rozhodli vytvořit objektově orientovaný grafický framework pro C++. V roce 1994 založily společnost Quasar Technologies, později byla přejmenována na Trolltech. V roce 2008 byl Trolltech zakoupen společností Nokia a přejmenován na QT software.

Roku 1995 byla pro veřejnost uvolněna první verze Qt 0.90. Už od první verze byl Qt vydáván jak pro komerční vývoj, tak pro open-source projekty a byl k dispozici na platformy Windows i Unix. O dva roky později r. 1997 byl vydán Qt 1.2 a Matthias Ettrich se rozhodl použít Qt pro vývoj KDE.

Qt se poté stalo de facto standardem pro C++ GUI vývoj na Linuxu. Qt 3.0 byl vydán v roce 2001 a byl k dispozici pro Windows, Mac OS X, Unix a Linux. V roce 2005 byl vydán Qt 4.0 a kvůli velkému počtu tříd a funkcí byl rozdělen do více knihoven.

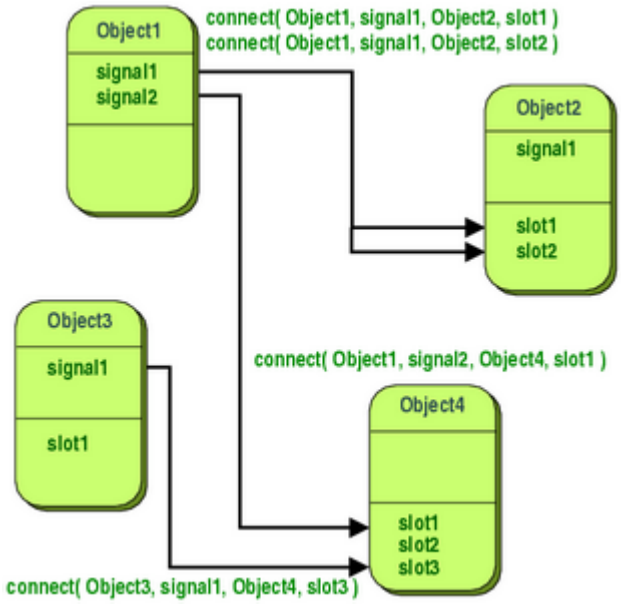
Dnes je Qt k dispozici nejen pro jazyk C++, ale také pro jiné hodně rozšířené programovací jazyky jako např Python, C# a Java. Navíc je k dispozici i pro jiné platformy jako jsou např. mobilní telefony s operačním systémem Symbian nebo vestavěné systémy.

2.3.1 Qt Creator

Qt Creator [6] je velmi schopné integrované vývojové prostředí (IDE) přímo určené pro vývoj programů v C++ s použitím frameworku Qt. Obsahuje velmi propracovaný editor C++ kódu s podporou doplňování, zvýrazňování syntaxe, průběžnou kontrolu syntaxe a mnoho dalšího. Dále obsahuje Qt Designer, což je nástroj na vytváření grafického uživatelského prostředí (GUI). Qt Creator ještě obsahuje kompletní dokumentaci Qt Assistant, která je propojena s editorem, kompilátor s debuggerem (GDB) a mnoho dalšího.

2.3.2 Signály a sloty

Signály a sloty jsou určeny pro komunikace mezi objekty. Jsou hlavním rysem Qt a pravděpodobně je to část, kterou se Qt liší od jiných podobných frameworků. Signály a sloty umožňují komunikace mezi libovolnými objekty, aniž by se navzájem znali. Všechny třídy, které dědí z objektu `QObject` nebo z jeho podtříd je mohou používat. Qt objekty mají svoje předdefinované signály a sloty, ale lze vytvářet libovolný počet vlastních.



Obrázek 2.6: Ukázka propojení signálů a slotů [5]

Signál je vytvořen při určité události, některé události vytvářejí signály automaticky např. signál `accepted()` je vytvořen po potvrzení dialogového okna. Slot je funkce, která je určená k propojením se signály. V praxi to znamená, že si vytvoříme slot, který chceme, aby se zavolal při nějaké události. Funkcí `connect()` ho propojíme se signálem, který se vytvoří, když nastane daná událost. Pokud potřebujeme ošetřit vlastní událost, jednoduše vytvoříme vlastní signál. Když nastane daná událost zavoláme signál funkcí `emit`. Dále je možné propojit signál se signálem. Pokud například chceme, aby se hlavní okno zavřelo spolu s dialogovým oknem, propojíme ukončovací signály obou oken. Příklad propojení slotů a signálů je vidět na obrázku 2.6.

2.3.3 Návrh GUI

Grafické uživatelské rozhraní se vytváří v Qt Designeru, jak už bylo řečeno dříve. Umožňuje vytváření grafických oken a umisťováním jednotlivých prvků (widgetů) pomocí přetahování (drag&drop) a snadno si tak můžete připravit obsah grafického okna. Jednotlivým widgetům lze jednoduše nastavit vlastnosti a následně je propojit jejich signály s požadovanými sloty. Je ovšem možné si vše rozmístit při implementaci, což nám umožňuje dynamické změny grafického vzhledu okna při běhu programu. Ale je jednoduší si celkovou strukturu vytvořit v Qt Designeru a v kódu pak dodělat zbylé části.

2.3.4 Vlákna

Pokud potřebujeme zpracovávat více dat současně (např. paralelní výpočty) nebo obsluhovat klienta, ale nadále chceme přijímat spojení od dalších klientů, potřebujeme vytvořit část programu, která bude oddělená od hlavního procesu. Dělá se pomocí tzv. vláken (anglicky thread). Použití vláken je vysvětleno v knize Modern multithreading [7]. Základní princip je takový, že se vytvoří funkce pro obsluhu vlákna a když je vytvořeno nové vlákno funkce se zavolá. Po vykonání funkce

```
class MyThread : public QThread
{
public:
    void run();
};

void MyThread::run()
 {
     QTcpSocket socket;
     // connect QTcpSocket's signals somewhere meaningful
     ...
     socket.connectToHost(hostName, portNumber);
     exec();
 }
```

Obrázek 2.7: Vytvoření vlastního vlákna

je vlákno ukončeno. Vlákna usnadňují jednoduchou vzájemnou komunikaci díky sdílené paměti, což přináší komplikace v podobě přístupu do sdílené paměti tzv. kritické sekce. Pro přístup do kritické sekce se používají zámky nebo semafory, které zajistí přístup pouze jednomu vláknu.

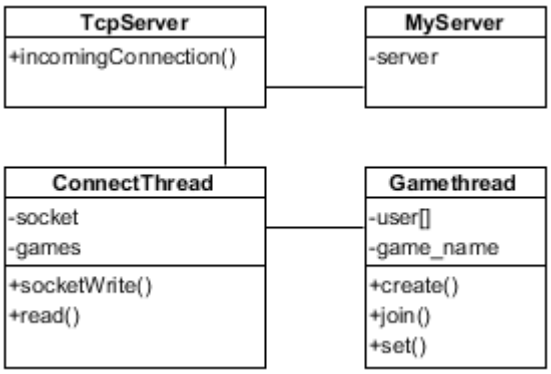
V knihovně Qt je možnost použít třídu `QThread`, která zjednoduše psaní aplikací využívajících vlákna. Dále je možno používat signály a sloty ke komunikaci mezi vlákny. Vlákno `QThread` se spouští funkcí `start()`, která zavolá funkci `run()`, pokud tedy chceme vytvořit vlastní vlákno, musíme vytvořit podtřídu třídy `QThread` a reimplementovat její funkci `run()`, příklad je vidět na obrázku 2.7.

2.4 Objektový návrh

V této kapitole si ukážeme teoretický návrh aplikace pomocí jazyka UML, který je popsán v knize UML a unifikovaný proces vývoje aplikací [8]. UML je grafický jazyk určený na modelování a návrh softwaru. Nejčastěji využíván v objektově orientovaných jazycích.

2.4.1 Server

Serverová část bude konzolová aplikace bežící na pozadí. Bude obsluhovat příchozí požadavky od klientů a zpracovávat je. Aby server mohl obsluhovat vice klientů současně, musí se pro každého klienta vytvořit vlákno (thread), ve kterém se zpracuje požadavek klienta. Pokud chce klient vytvořit novou hru, vytvoří se pro ni nové vlákno, pokud se chce jen připojit předá se informace příslušnému vláknu se spuštěnou hrou. Návrh serveru je vidět na zjednodušeném diagramu tříd na obrázku 2.8.



Obrázek 2.8: Diagram tříd pro server

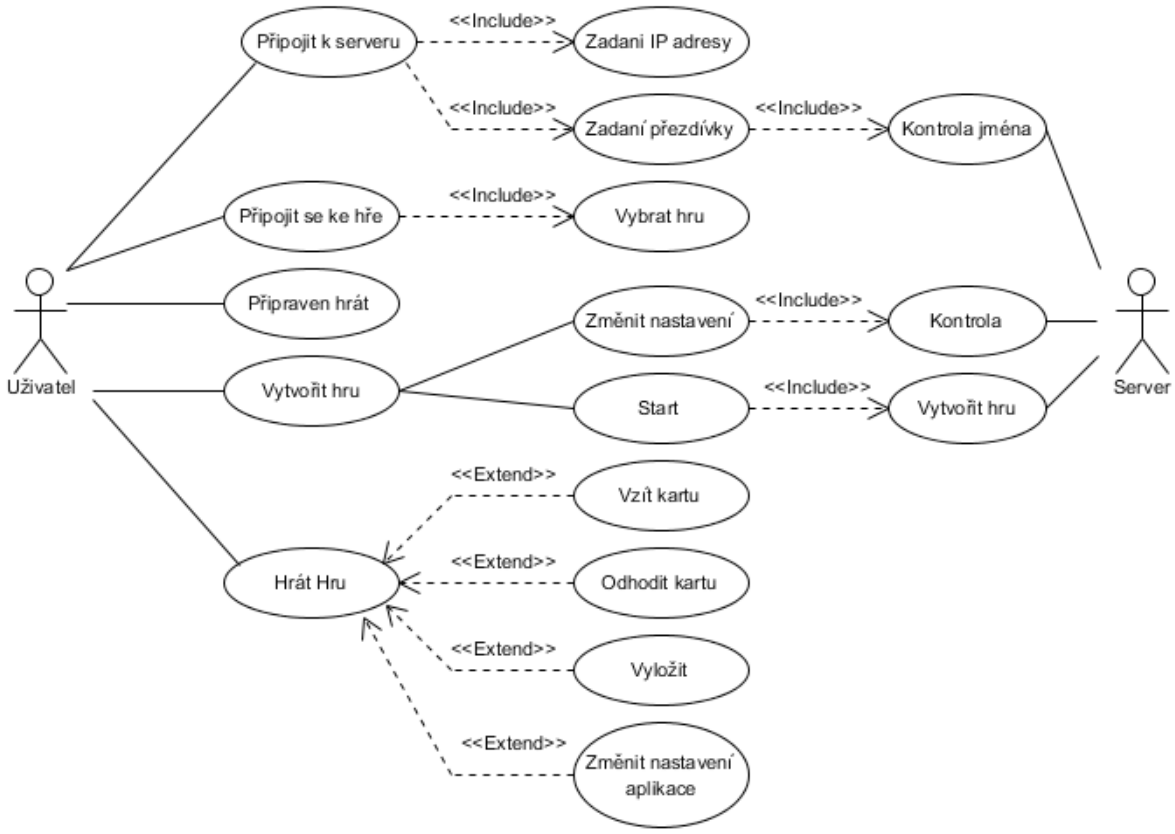
Kompletní Diagram tříd viz. příloha 2;

Dále bude vytvořena třída, která se bude starat o průběh hry a vlákno `Gamethread` bude tuto třídu používat při kontrole klientských požadavků. Tato třída bude také obsahovat objekt třídy `MyPack`, který se bude starat o balíček karet. Bude generovat karty na začátku hry a bude poskytovat karty z vytvořeného balíčku.

2.4.2 Klient

Klient bude aplikace s grafickým uživatelským prostředím. Po jeho spuštěný bude uživatel vyzván k zadání přezdívky a adresy počítače, kde běží serverová část. V dalším kroku je vyzván buď k vybrání spuštěné hry nebo k vytvoření vlastní hry. Když uživatel zvolí vytvoření vlastní hry bude moct zadat název hry a změnit nastavení hry. Po vytvoření hry se ostatní uživatelé budou moct připojit ke hře. Po připojení ke konkrétní hře bude uživatel vidět nastavení hry, ostatní připojené klienty a bude moct zmáčknout tlačítko „Připraven“. Jakmile jsou všichni připojení uživatelé připraveni, zakladatel hry může spustit hru.

Pro klientskou část jsme také navrhli Diagram tříd. Kompletní diagram je v příloze 2. Dále jsme navrhli Diagram případů užití (viz. obrázek 2.9), který popisuje soubor případů užití, aktérů a jejich vzájemných vztahů. Aktér „Uživatel“ je fyzická osoba, která obsluhuje aplikaci a aktér „Server“ je aplikace, se kterou klientská aplikace komunikuje po síti.



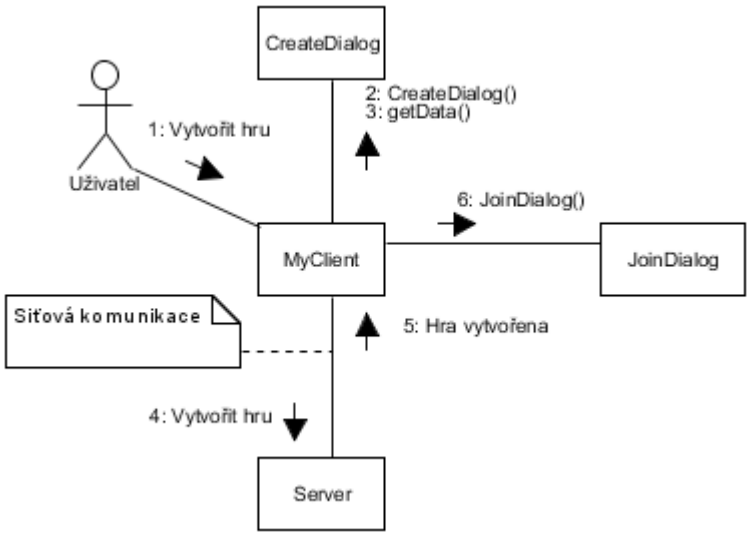
Obrázek 2.9: Diagram případů užití

Pro diagram případů užití byl vytvořen jeden případ užití, který je znázorněn v tabulce 1. Tento případ užití detailně popisuje situaci, kdy klient chce vytvořit novou hru.

Název případu užití	Vytvořit Hru
Identifikátor	ID4
Stručný popis	Uživatel vytvoří novou hru
Primární aktér	Uživatel
Sekundární aktér	Server
Vstupní podmínky	Aplikace je připojena k serveru
Základní scénář	1. Případ užití se spustí, když uživatel klikne na tlačítko „Vytvořit novou hru“. 2. Zadat název hry a upravit nastavení. 3. Vytvořit novou hru kliknutím na tlačítko „Vytvořit“ 3.1. Systém zkontroluje jméno a nastavení hry. Pokud jsou zadány špatné údaje, uživatel musí nastavení změnit. 4. Systém vytvoří novou hru.
Následné podmínky	Byla vytvořena nová hra
Alternativní toky	Uživatel zadal špatné nastavení

Tabulka 1: Případ užití pro vytvoření nové hry

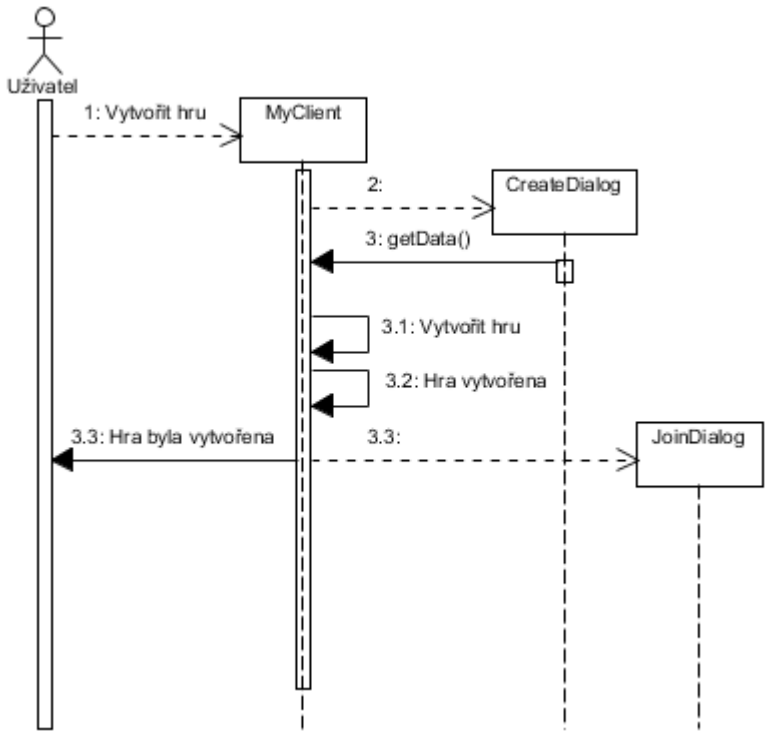
Dále byl pro tento případ užití vytvořen Diagram komunikace, který ukazuje účastníky interakce a datová spojení mezi nimi. Diagram je na obrázku 2.10.



Obrázek 2.10: Diagram komunikace

Diagram komunikace dovoluje volné umisťování účastníků, jejich propojování a číslování sekvence zpráv, kterými komunikují.

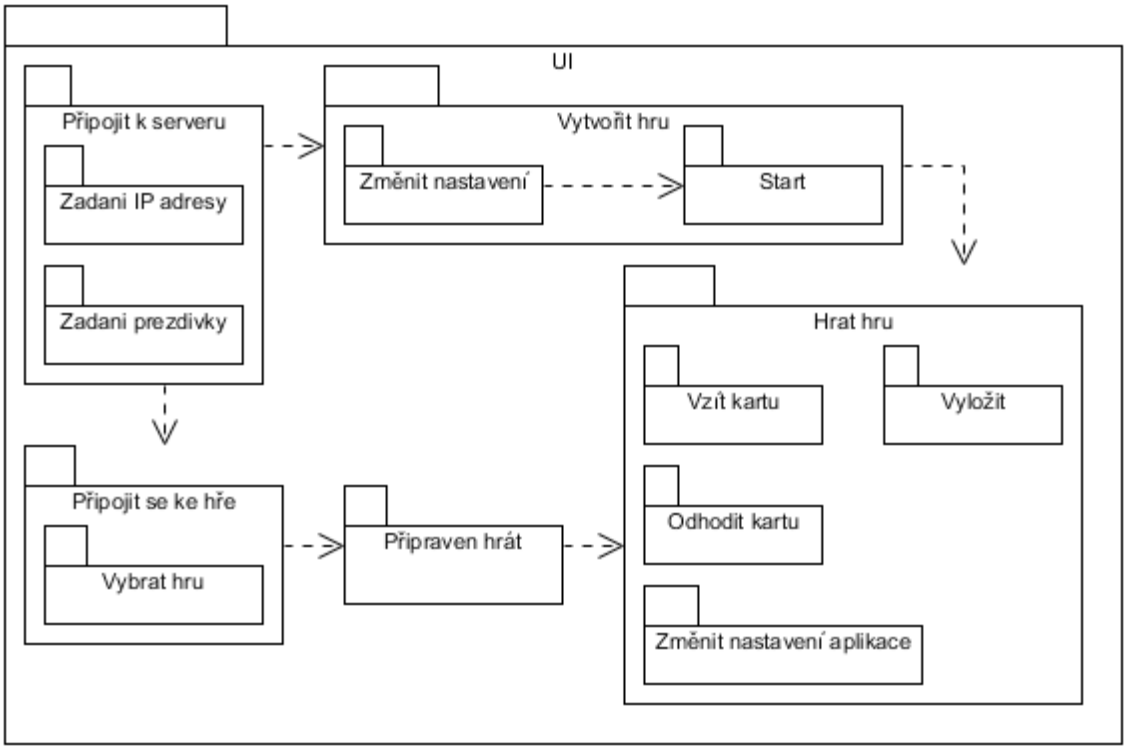
Na Diagram komunikace navazuje Sekvenční diagram, který je na obrázku 2.11. Graficky popisuje posloupnost zpráv, které si účastníci interakce posílají, uspořádané v čase. Do jisté míry popisuje to stejné co Diagram komunikace, ale Diagram komunikace je vhodnější pro zobrazení spojení a Sekvenční diagram je vhodnější pro zobrazení sekvencí zpráv.



Obrázek 2.11: Sekvenční diagram

Uživatel chce vytvořit novou hru, klientský program vytvoří dialogové okno (třída `CreateDialog`), ve kterém uživatel zadá nastavení hry. Tyto nastavení se pak metodou `getData()` vrátí do klienta, který je zpracuje a odešle požadavek na server. Jakmile od serveru dojde odpověď, že je daná hra vytvořena, je vytvořeno nové dialogové okno (třída `JoinDialog`). V tomto okně jsou informace o vytvořené hře a ostatních připojených hráčích.

Dalším diagramem je diagram balíčků (Package diagram), který ukazuje seskupení jednotlivých případů užití do balíčků. Diagram znázorňuje závislosti mezi jednotlivými balíčky a jejich hierarchickou strukturu. Další výhodou diagramu balíčků je snížení závislosti mezi jednotlivými elementy, proto je vhodný na uspořádání rozsáhlých systémů. Diagram balíčků pro uživatelské prostředí klientské aplikace je znázorněn na obrázku 2.12.



Obrázek 2.12: Diagram balíčků

3 Implementace

V této kapitole se budeme zabývat tvorbou Grafického uživatelského prostředí pro klienta a implementací jeho programové části, budeme také popisovat implementaci pravidel hry v klientské aplikaci. Dále bude vysvětlena implementace serverové aplikace, zpracování z pohledu konkurentního serveru a popis objektů kontrolující samotnou hru. Bude také rozebrána síťová implementace komunikačního protokolu. Programování za pomocí Qt frameworku popisuje kniha C++ GUI Programming with Qt 4 [9].

3.1 Klient

3.1.1 Tvorba GUI

Uživatelské prostředí jsme vytvářeli pomocí grafického nástroje Qt Designer, který byl popsán v kapitole 2.3.3. Pomocí tohoto nástroje jsme vytvořily grafickou podobu jednotlivých dialogových oken a hlavního herního okna. Pomocí Qt Designeru jsme umístily do grafických oken všechny textové pole tlačítka a widgety, jejich obsah se bude měnit za běhu programu. Příklad widgetu je grafická plocha pro hru, ve které jsou umístěny uživatelovy karty, karty protihráčů, balíček karet a odhozené karty.

Pro hlavní okno aplikace jsme zvolili formulář `QMainWindow`, který je pro aplikace nejvhodnější. Obsahuje už menu (`QMenuBar`), hlavní zobrazovací widget (`QWidget`) a stavový řádek (`QStatusBar`). Stačí tedy vytvořit strukturu menu a umístit základní grafické prvky do hlavního widgetu. Pro zobrazení herního pole jsme použily `QGraphicsView`, je to widget který umožňuje zobrazit `QGraphicsScene`. `QGraphicsScene` je kontejner umožňující kreslení a zobrazování 2D objektů, jako např. obrázky a geometrické obrazce.

Pro ostatní okna jsme zvolili jednoduší `QDialog`, jehož funkčnost stačí pro zobrazení oken např. pro vytvoření či vybrání hry.

3.1.2 Objekty aplikace

Jak již bylo uvedeno o kapitolu výše, jako hlavní formulář byl použit `QMainWindow`. Vytvořili jsme tedy třídu `MyClient`, která z třídy `QMainWindow` dědí a pro vzhled jsme použily formulář vytvořený v Qt Designeru. Tato třída se stará o chod celého programu, zajišťuje přidávání, úpravu a mazání grafických prvků okna, komunikaci mezi ostatními objekty, vytváření a zobrazování dialogů, zpracování dat z dialogů, zpracování síťového protokolu atd.

Pro ostatní dialogy byla vytvořena třída `Dialog` pro sjednocení používaných dialogových oken. Tato třída dědí z třídy `QDialog`. Každá třída pro dialogové okno, která dědí z třídy `Dialog`, si importuje vlastní grafický formulář a implementuje funkci `getData()` pro získání dat od uživatele. Definice třídy dialogového okna pro přihlášení je na obrázku 3.1.

```
class ConnectDialog : public Dialog
{
  Q_OBJECT
public:
  ConnectDialog(QWidget *parent = 0);
  ~ConnectDialog();
  QStringList getData();

private:
  Ui::ConnectDialog *ui;
};
```

Obrázek 3.1: Třída pro dialogové okno přihlášení

Pro průběh hry byla vytvořena třída `MyGame`, která se stará o samotnou hru. Kontroluje pravidla hry a správnost vykládaných karet, uchovává informace o kartách uživatele, umožňuje karty poskládat podle jejich barev a hodnot, počítá hodnotu karet pro vyložení a kontroluje první vyložení. Dále uchovává informace o ostatních hráčích a celkových bodech. Byla navržena i třída, která by se starala o karty. Umožňovala by karty skládat, počítala by jejich hodnoty a možnost vyložení. Tato třída nakonec nebyla použita, protože veškeré její funkce byly implementovány ve třídě `MyGame`.

Pro zobrazování grafických prvků se používá třída `QGraphicsScene`, jak již bylo popsáno výše. Ale kvůli jednoduššímu odchytávání událostí jako je kliknutí myši, jsme si vytvořily vlastní třídu `myGraphicsScene`, která dědí od `QGraphicsScene`. `QGraphicsScene` zavolá vlastní metodu `mousePressEvent()` pokud je pozice kliknutí myši uvnitř scény. Tuto metodu jsme reimplementovaly tak, aby se vytvořil signál, který jsme propojily se slotem v hlavní třídě. Tato funkce (slot) obsluhuje kliknutí na uživatelovy karty, např. při odhazování karty nebo při výběru karet k vyložení. Pro kontrolu kliknutí na ostatní grafické objekty jsme si vytvořily třídu `MyGraphicsItem`, která dědí z `QGraphicsItem`. Do nové třídy jsme přidali atribut `id` a opět jsme reimplementovali metodu `mousePressEvent()`, která nám vyvolá signál `click(id)`. Tento signál jsme propojily se slotem `item_click(id)`. Pokud tedy vytváříme objekty z třídy `MyGraphicsItem` a nastavíme jim id, tak při kliknutí víme id objektu na který se kliklo. Můžeme tedy vytvářet skupiny objektů, pro které můžeme vytvořit stejné chování, aniž bychom museli zjišťovat, na který konkrétní grafický prvek se kliklo.

Pro komunikaci se serverem jsme zvolili komunikaci nad protokolem TCP, jak bylo popsáno v kapitole 2.2.1. Pro síťovou komunikaci existuje mnoho knihoven pro jazyk C++, příklad knihovny může být `sys/socket.h`. Její použití je vysvětleno v knize UNIX Network Programming Volume 1 [4]. My jsme pro tuto aplikaci vybrali třídu `QTcpSocket`, která používá knihovnu obsaženou přímo v Qt frameworku. Pracuje na podobném principu jako `sys/socket.h`, ale je implementovaná objektově. Další výhoda je používání signálů a socketů. Při navázání spojení je

vytvořen signál `connected()`. Pokud signál není vytvořen, nebylo navázáno spojení se serverem. Pokud je spojení ukončeno je vytvořen signál `disconnected()`, tímto signálem se dá jednoduše ošetřit přerušení spojení se serverem. Poslední signál `readyRead()`, který dává aplikaci vědět, že došla data od serveru. To je velká výhoda, protože u většiny ostatních knihoven bylo potřeba neustále kontrolovat, jestli nedošla nějaká data. U `QTcpSocketu` stačí propojit tyto signály se sloty.

Pro vytvoření spojení se serverem je vytvořen objekt třídy `QTcpSocket` a zavolána jeho metoda `connectToHost()`, jako parametry potřebuje adresu a port serveru. Adresa se předává formou třídy `QHostAddress` a port je typu `int`. Příklad připojení k serveru a propojení signálů se sloty pro obsluhu je vidět na obrázku 3.2.

```
QString addr_string = "127.0.0.1";
int port = 2258;
//novy socket
socket = new QTcpSocket(this);
//propojeni signalu se sloty
connect(socket, SIGNAL(connected()), SLOT(gotConnected()));
connect(socket, SIGNAL(disconnected()), SLOT(gotDisconnected()));
connect(socket, SIGNAL(readyRead()), SLOT(read()));
QHostAddress addr;
//zkontrolujeme platnost IP adresy
if(!addr.setAddress(addr_string))
  return;//neplatna adresa
//pripojeni k serveru
socket->connectToHost(addr, port);
```

Obrázek 3.2: Připojení k serveru

3.1.3 Výběr hry

První návrh spočíval v otevření hlavního okna a změnu obsahu tohoto okna při změně stavu aplikace (např. aplikace načte grafické prvky a nastavení, až když uživatel spustí hru). To bylo zamítnuto a byl použit návrh, ve kterém je hlavní okno spuštěno na pozadí a uživatelské vstupy budou probíhat formou dialogových oken.

Po spuštění aplikace je vytvořen objekt třídy `MyClient`, který představuje hlavní okno. Protože klient bez serveru neumožňuje hrát, tak je hned je zobrazeno dialogové okno (`ConnectDialog`), do kterého uživatel zadá informace potřebné pro připojení k serveru a svou přezdívku.

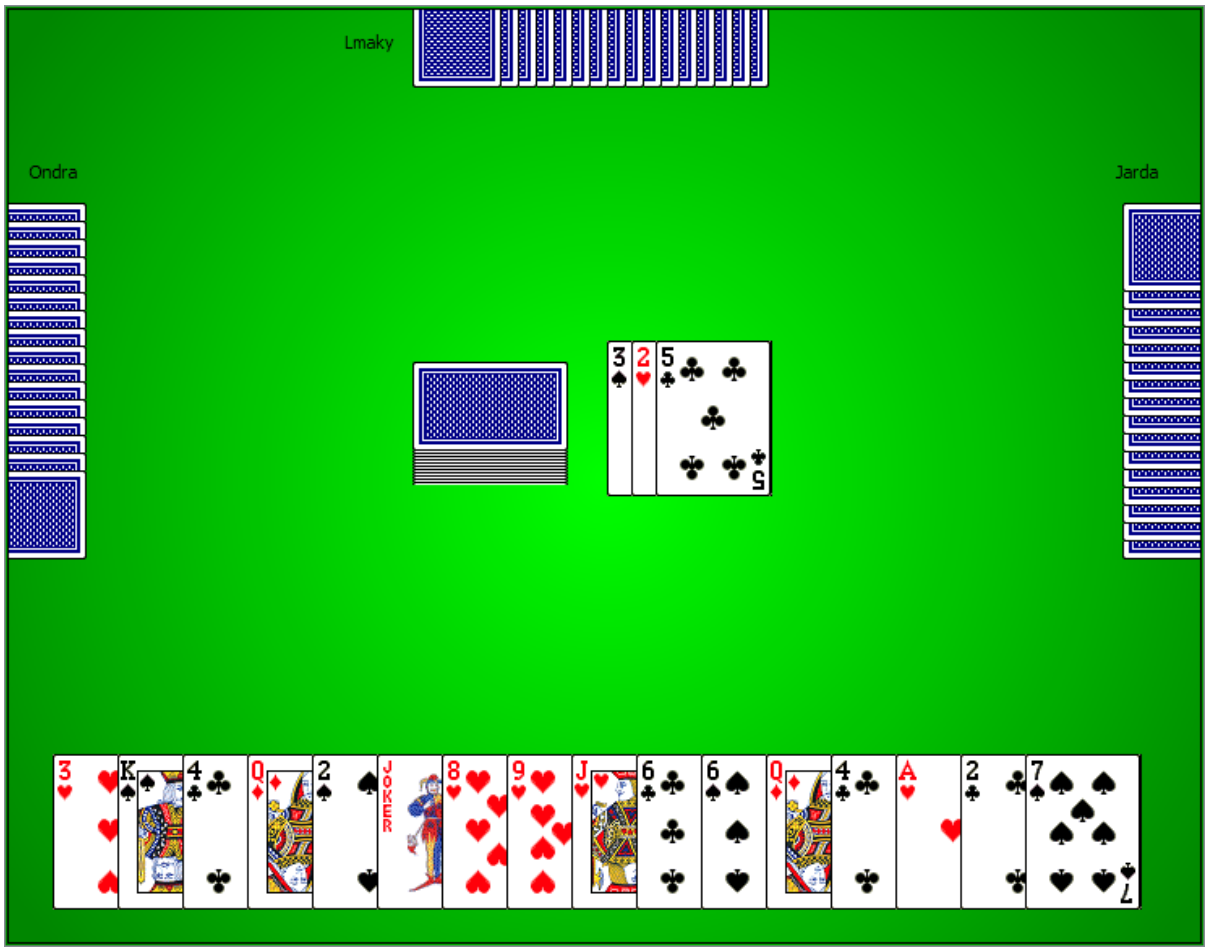
Po připojení na server jsou načteny všechny spuštěné hry a zobrazeny v dialogovém okně `GamesDialog`. Spuštěné hry jsou načteny do grafického objektu `QListWidget`, který má nastavenu možnost výběru jednoho prvku. Uživatel si tak může vybrat jednu hru z tohoto widgetu a připojit se k ní nebo si vytvořit vlastní hru kliknutím na tlačítko „Vytvořit vlastní hru“.

Po připojení ke hře je vytvořeno dialogové okno `JoinDialog`, ve kterém uživatel vidí všechna nastavení hry, ostatní připojené hráče s jejich statusem a chat, který umožňuje komunikaci mezi hráči. Když je uživatel připraven hrát, klikne na tlačítko „Připraven“, změní se mu status v seznamu hráčů. Jakmile jsou všichni hráči připravení, může zakladatel hry spustit hru.

Vytvoření hry probíhá v dialogovém okně `CreateDialog`. Uživateli je umožněno zadat název hry a změnit výchozí nastavení. Po vytvoření hry je vytvořeno dialogové okno `JoinDialog` stejně jako v případě připojení k již existující hře, zakladatel má možnost hru spustit.

3.1.4 Spuštění hry

Jakmile je hra spuštěna, klient zobrazí rozdané karty a hráč, kterého určil server, může začít hru vyhozením první karty. Karty jsou načteny do objektu `QPixmap` jako jeden obrázek, jednotlivé karty jsou pak z tohoto obrázku zkopírovány a načteny do objektu `QGraphicPixmapItem`. Tyto objekty jsou umístěny na hrací ploše v kontejneru `myGraphicsScene`. Hra se ovládá pomocí myši, uživatel může vždy provést pouze akce, které mu status hry umožní. Status hry určuje zda hráč čeká na dokončení hry ostatních hráčů nebo má hrát sám. Pokud hráč čeká, má možnost pouze používat chat. Jakmile je na řadě musí si zvolit, odkud si vezme kartu. Následuje vykládání (popř. dokládání)



Obrázek 3.3: Vzhled grafické plochy při hře

karet, povinnost vykládání je určeno tím odkud si hráč vzal kartu. A nakonec musí vybrat kartu kterou odhodí. Všechny akce musí dodržovat pravidla hry viz. kapitola 2.1.

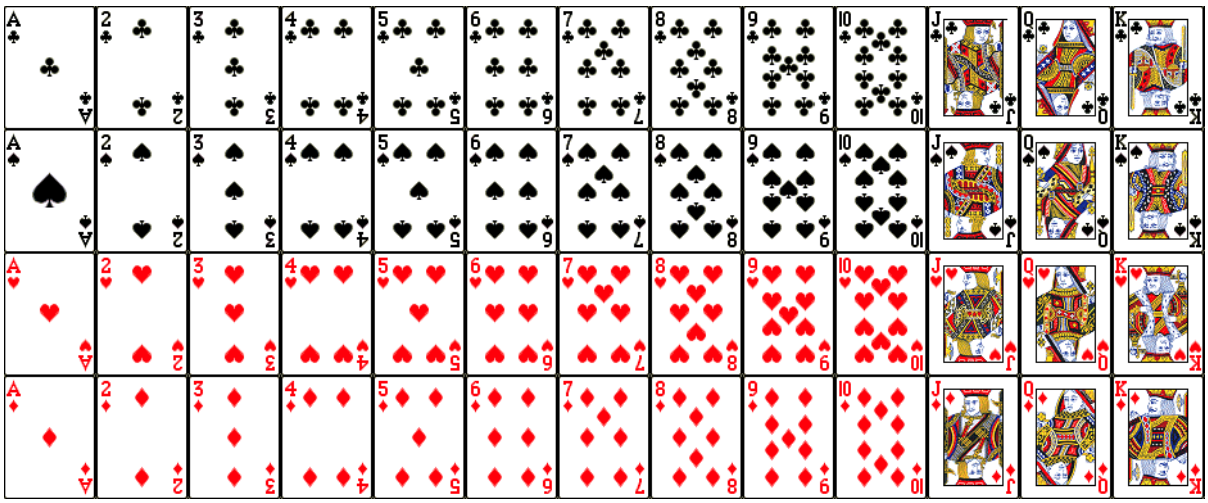
Vzít kartu na začátku tahu může uživatel kliknutím na balíček karet nebo na poslední odhozenou kartu. Pro vyložení postupky musí uživatel kliknout na tlačítko „Vyložit postupku“, dále pak musí vybrat všechny karty, které chce do postupky použít a kliknout na tlačítko „Vyložit“. Aplikace zkontroluje správnost vykládaných karet a karty vyloží na stůl. Vyložené karty se u protihráčů zobrazí, až uživatel ukončí svůj tah odhozením karty. Pro odhození karty uživatel vybere jednu kartu a klikne na tlačítko „Odhodit“. Pokud se jedná o první vykládání, tak aplikace zkontroluje minimální počet bodů pro vyložení. Vzhled načtené grafické hrací plochy okna ve hře, ke které jsou připojeni 4 hráči, je vidět na obrázku 3.3.

3.1.5 Nastavení aplikace

V menu je možné pozměnit chování aplikace a vzhled karet. Do nastavení se dostanete před menu Soubor->Nastavení. Lze změnit např. řazení karet, zobrazování informačních textů, způsob zobrazování bodování nebo vzhled karet. Vzhled karet se dá změnit buď na některý předdefinovaný v aplikaci, nebo lze nahrát vlastní vzhled karet.

3.1.6 Vlastní vzhled karet

Obrázky vzhledu musí dodržovat určitou strukturu. Každá karta musí mít rozměr 72x96 a vzhled základní sady karet se vkládá jako jeden obrázek složený ze všech karet v sadě. Karty v sadě musí být poskládány přesně jako na obrázku základní sady, která je v aplikaci standardně nahraná. Základní sada je vidět na obrázku 3.4 a má rozměr 936x384. Obrázek pro žolíka a pro rub karet se vkládá zvlášť. Druhá ukázková sada karet se žolíkem a obrázkem pro rub je vložena ke zdrojovým kódům viz. příloha 3.



Obrázek 3.4: Základní sada karet [13]

3.2 Server

3.2.1 Vytvoření serveru

Server je implementován jako konzolová aplikace bez grafického uživatelského prostředí. Jako vzor pro vytvoření serveru byl článek Konzolové programy v Qt 4 [9]. Po spuštění server přijímá požadavky na spojení od klientů, kteří jsou obsluhováni paralelně. Takový server, který dokáže zpracovávat více klientů zároveň se nazývá konkurentní. Příjímání síťových požadavků je implementováno pomocí třídy `TcpServer`, která dědí od `QTcpServer`. Třída `QTcpServer` používá pro síťovou komunikaci knihovnu obsaženou přímo Qt frameworku, stejně jako třída `QTcpSocket`, která je použita v klientské části.

Třída `QTcpServer` zavolá metodu `incomingConnection()`, proto jsme ve třídě `TcpServer` reimplementovali tuto metodu, aby po připojení nového klienta vytvořila nové vlákno (viz. kapitola 3.3.2) `ConnectThread`, které klienta obslouží. Reimplementovanoá metoda `incomingConnection()` je vidět na obrázku 3.5.

```
void TcpServer::incomingConnection(int socketDescriptor)
{
  //vytvoreni vlakna pro obsluhu kliena a predani descriptoru
  ConnectThread *th = new ConnectThread(socketDescriptor,this);
  //spusteni noveho vlakna
  th->start();
}
```

Obrázek 3.5: Připojení nového klienta

3.2.2 Připojení klienta

Aby server mohl obsluhovat více klientů zároveň, musí být každý klient zpracováván v paralelním procesu nebo vlákně. Vlákna jsou v Qt frameworku plně podporována, proto jsme je vybrali pro náš projekt. QT framework umožňuje vlákna obsluhovat za použití třídy `QThread`. Vytvořili jsme tedy třídu `ConnectThread`, která dědí z třídy `QThread`. Při vytvoření objektu třídy `ConnectThread` se vytvoří nový `QTcpSocket`, kterému se nastaví předaný `socketDescriptor`, aby se vytvořilo spojení s klientem. Dále se propojí signály `readyRead()` a `disconnected()` se sloty `read()` a `gotDisconnected()`, které obsluhují dané události nového socketu. Slot `read()` získá data od klienta a zpracuje je.

První možností je, že se klient přípojil k serveru. Server zkontroluje přezdívku klienta jestli už není použitá. Pokud přezdívku nelze použít server informuje klienta, který musí zadat novou přezdívku. Pokud lze přezdívku použít server odešle klientovy informace o spuštěných hrách.

Druhou možností je, že klient chce vytvořit novou hru. Server zkontroluje název hry, pokud je správné tak vytvoří nové vlákno pro hru (třída `GameThread`) a spustí ho. Do vlákna pro hru předá přezdívku uživatele, jméno hry a nastavení hry. Dále do tohoto vlákna přesměruje komunikaci s klientem. Ukázka vytvoření vlákna pro hru je na obrázku 3.6.V ukázkovém kódu jsou použity statické proměnné (např. `ConnectThread::games`), jsou použity na ukládání spuštěných her. Statická proměnná vytvořená ve třídě `ConnectThread` je společná pro všechny objekty dané třídy. Toto je ukázkový kód, v aplikaci musí ještě proběhnout kontrola, jestli k těmto proměnným nepřistupuje jiný objekt. To se dělá pomocí třídy `QMutex`. Pokaždé, když chceme přistupovat ke společné proměnné, musíme `QMutex` zamknout metodou `lock()`. Pokud je `QMutex` zamčený, tak počká, až bude jiným objektem odemknut.

```
//vytvorime nove vlakno
GameThread* thread = new GameThread(this);
//ulozime si informace o novem vlaknu
ConnectThread::games << thread;
ConnectThread::names << game_name;
//spustime vlakno a predame mu nastaveni hry
thread->start();
thread->create(this, game_name, player_name, setting);
```

Obrázek 3.6: Vytvoření vlákna pro hru

Poslední možností je, že si klient vybral hru ke které se chce připojit a odeslal serveru požadavek na připojení k ní. Server klienta přesměruje do vlákna příslušné hry metodou `join(parent, player_name)`.

3.2.3 Vytvořená hra

Každá vytvořená hra má svoje vlákno definované třídou `GameThread`. Při vytvoření hry nebo při připojení nového hráče je veškerá komunikace s tímto hráčem přesměrována do tohoto vlákna. Také je všem ostatním hráčům odeslána zpráva, která je informuje o nově připojeném. Většina takovýchto herních zpráv je odesílána všem hráčům, např. který hráč je na řadě se odešle všem, aby ostatní hráči byli informováni, kdo je na řadě.

Jakmile všichni hráči odešlou zprávu, že jsou připravení, a zakladatel hry odešle požadavek na spuštění hry, vytvoří se objekt game třídy `MyGame`. Tahle třída řídí stav hry a kontroluje logiku hry. Když je objekt game vytvořen, vytvoří se objekt třídy `MyPack`, který vygeneruje balíček karet a zamíchá ho. Odebírání karet z balíčku je umožněno metodou `takeCard()`. Dále třída `MyGame`

rozdá karty a určí kdo hru začíná. Karty jsou následně poslány hráčům a je odeslána informace o začínajícím hráči.

V Původním návrhu třídy `MyGame` byla kontrola správnosti vykládaných karet, což jsme ale nakonec nechali na klientské aplikaci. Odpadá tak zbytečné dotazování se serveru na kontrolu karet. Navíc při větším počtu spuštěných her by mohl algoritmus kontroly karet zpomalit funkčnost serveru. Hraní hry teď tedy z pohledu serveru spočívá v přeposílání stavu hry všem hráčům (např. hráč odhodil kartu nebo hráč vyložil postupku), přeposílání společného chatu a uchovávání celkového počtu bodů. Hráč který ukončí hru (nezbudou mu žádné karty), odešle na server zprávu o zavření hry. Server následně odešle všem dosažené body a pošle dotaz na zakladatele jestli se bude pokračovat.

3.2.4 Komunikace s klienty

Třída `GameThread` potřebuje komunikovat s klienty a každý klient má své vlákno `ConnectThread`. Bylo tedy nutné, odeslat zprávu do konkrétního vlákna a odtud zprávu odeslat klientovy. K řešení tohohle problému byla vytvořena třída `MyUser`, ve které jsme vytvořily signál `_send(data)`. Do objektu této třídy načteme jako rodiče vlákno `ConnectThread` a funkcí `connect()` propojíme signál `_send(data)` se slotem ve třídě rodiče. V objektu třídy `GameThread` je vytvořeno pole těchto objektů, které představuje připojené klienty. Odesílání zpráv se provádí zavoláním metody `send()` u konkrétního objektu třídy `MyUser`, která vyvolá signál `_send(data)`.

3.3 Síťová komunikace

Veškerá síťová komunikace, která mezi serverem a klienty probíhá, používá navržený komunikační protokol. Komunikace probíhá formou dotaz-odpověď. Server čeká na zprávu od klienta, zpracuje ji a odešle odpověď. Odpověď je většinou rozesílána jako informační zpráva všem, pokud se ale jedná o chráněnou informaci (např. karty) je odeslána jen konkrétnímu klientovy. Zpráva, kterou klient posílá požadavek na server může mít dva tvary.

První je odesílaní požadavku od konkrétního klienta (např. odhození karty nebo chat). Tvar zprávy je „PŘÍKAZ;NASTAVENI;ID“, příkaz je identifikátor požadavku a nastavení jsou volby daného příkazu. Část zprávy pro nastavení je ovšem volitelná, všechny příkazy nepotřebují odesílat volby (např. příkaz „READY“, který značí, že je uživatel připraven hrát). ID je identifikátor uživatele, jedná se o unikátní přezdívku na serveru.

Druhá zpráva je informace pro celou hru (např. „START“ pro Start hry), její tvar je „PŘÍKAZ;NASTAVENI“. Část zprávy definující nastavení je opět volitelná.

Zprávy, které posílá server klientům jsou složitější, protože je nutno odesílat více dat. Zpráva je tedy rozdělena na dvě části oddělené řetězcem dvou středníků „ ;; “. První část zprávy identifikuje druh zprávy (např. pro hru je to „PLAY“). Druhá část už je podobná zprávám, které odesílá klient

na server tzn. „PŘÍKAZ;NASTAVENÍ;ID“. Příklad zprávy, která určuje který hráč hru začne, je „PLAY;;FIRST;Lmaky“. Pokud je odesíláno více zpráv po sobě, je nutné zajistit, aby se data z více zpráv nesmíchaly. Proto se na konec zpráv přidává řetězec tří středníků „ ;;; “, který představuje konec jedné zprávy.

Kompletní komunikační protokol viz. příloha 1.

4 Závěr

Cílem této práce bylo vytvořit počítačovou hru vycházející z karetní hry žolíky. Hra měla umožňovat multiplayer tzn. hraní více hráčů proti sobě. Dále by uživatelé hry měli mít možnost změnit si vzhled karet.

Prvním krokem k vytvoření aplikace byl návrh, který je obsahem kapitoly 2. Zde jsme se museli rozhodnout, které technologie použijeme. Jako framework byl vybrán Qt, který je vysvětlen v podkapitole 2.3. Je velice jednoduchý na instalaci, je multiplatformní a dodávané integrované vývojové prostředí Qt Creator je pro naši aplikaci ideální. Výběrem síťové technologie se zabývá podkapitola 2.2. Byla vybrána síťová technologie client-server, protože server může snadněji spravovat spuštěné hry a při odpojení některého klienta může hra pokračovat dál, i když se jedná o zakladatele hry. Jako další byl navržen síťový komunikační protokol. Když byla vybrána síťová technologie a vývojové prostředí, přišel na řadu samotný návrh aplikace pomocí UML, který je obsahem podkapitoly 2.4. Tento návrh pomohl při vytváření základních objektů a struktury aplikací, při řešení zpráv, které se posílají mezi objekty, a při rozdělování úloh jednotlivých objektů. V neposlední řadě museli být zpracovány pravidla hry, které jsou popsány v podkapitole 2.1. Muselo se rozhodnout, které části aplikace budou kontrolovat jednotlivá pravidla.

Po návrhu následovala samotná implementace, jenž je popsána v kapitole 3. Museli jsme navrhnout grafické prostředí a způsob získávání dat od uživatelů. Dále pak byli implementovány jednotlivé objekty podle návrhu v jazyce UML. Tento návrh při implementaci velice pomohl a usnadnil některé kroky implementace. U klientské části jsme museli propojit vytvořené objekty s grafickým prostředím, začít vytvářet jednotlivé kroky získávání dat od uživatelů, implementovat zpracovávání komunikačního protokolu a nakonec implementovat samotnou hry. U serverové části jsme vytvořily třídu která obsluhuje klienta a třídu která řídí hru. Při implementaci propojení jednotlivých objektů nám hodně pomohli signály a sloty z Qt frameworku, protože je to nejlepší metoda na posílání zpráv mezi objekty.

Aplikace byla otestována na operačních systémech Windows 7 a Windows XP, kde aplikace běžela v pořádku (pozn. díky frameworku Qt, by nebyl problém s používáním aplikace i na ostatních podporovaných systémech). Dále byla otestována síťová komunikace. Přenos dat byl za použití síťové knihovny obsažené v Qt bez problému. Avšak připojení v serveru se v 5% pokusů nepovedlo. Zjistili jsme, že důvodem bylo nevytvoření příslušného signálu, který se má vytvořit po spojení. Tento problém se vyskytl, i když byl server a klient na stejném počítači, tzn. nejednalo se o chybu síťové komunikace.

Práce pro nás byla velkým přínosem, naučili jsme se pracovat s Qt frameworkem a jeho integrovaným vývojovým prostředím. Vyzkoušeli jsme si vytváření grafické aplikace a práci s Qt Designerem. Osvojili jsme si vytváření konkurentního serveru. Naučili jsme se pracovat s vlákny `QThread`, se kterými se velmi snadno vytváří aplikace, ve kterých je potřeba paralelní běh funkcí a pomocí signálů a slotů se velmi jednoduše vlákna synchronizují. Dále jsme se naučili používat UML jazyk v praxi při návrhu skutečné aplikace.

Jeden z možných návrhů na rozšíření by mohla být umělá inteligence, aby bylo možné hrát proti počítači. Další rozšíření by mohlo být ukládání informací o uživatelích na serveru. Ukládala by se zde např. statistika vítězství, doba strávená hraním a přátelé.

Literatura

[1] Joker Club [online]. 2009 [cit. 2011-04-04]. Pravidla hry žolíky. Dostupné z WWW: <http://www.clubjoker.cz/pravidlajoker.htm>.

[2] Hrací karta. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2006, last modified on 2011 [cit. 2011-05-09]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Hrací_karta>.

[3] HUNT, Craig. *TCP/IP Network Administration*. 3rd Edition. : O'Reilly Media, 2002. 752 s. ISBN 978-0-596-00297-8.

[4] STEVENS, W. Richard; FENNER, Bill ; RUDOFF, Andrew M. *UNIX Network Programming : Volume 1*. Third Edition: The Sockets Networking. : Addison-Wesley Professiona, 2003. 1024 s. ISBN 978-0-13-141155-5.

[5] *Qt - A cross-platform application and UI framework* [online]. 2008 [cit. 2011-05-09]. Dostupné z WWW: <http://qt.nokia.com/products/>.

[6] *Qt Creator IDE and tools* [online]. 2008 [cit. 2011-05-09]. Qt framework. Dostupné z WWW: <http://qt.nokia.com/products/developer-tools>.

[7] CARVER, Richard H.; TAI, Kuo-Chung. *Modern multithreading : implementing, testing, and debugging multithreaded Java and C++/Pthreads/Win32 programs*. : John Wiley and Sons, 2006. 465 s. ISBN 978-0-47-172504-6.

[8] ARLOW, Jim; NEUSTADT , Ila . *UML a unifikovaný proces vývoje aplikací* . [překl.] Bogdan Kiszka . Brno : Computer press, 2003. 387 s. ISBN 80-7226-947-X.

[9] BLANCHETTE, Jasmin; SUMMERFIELD, Mark . *C++ GUI Programming with Qt 4*. Second Edition. [s.l.] : Prentice Hall, 2008. 752 s. ISBN 978-0-13-235416-5.

[10] WATZKE, David. Konzolové programy v Qt 4 : TCP server. *Abclinuxu* [online]. 16. 9. 2009, 3, [cit. 2011-05-09]. Dostupný z WWW: <http://www.abclinuxu.cz/clanky/programovani/konzolove-programy-v-qt-4-3-tcp-server>.

[11] ŠVÉDA , Miroslav. *Síťové aplikace a správa sítí* [online]. 2009 [cit. 2011-05-11]. Studijní opora. Dostupné z WWW: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ISA-IT/texts/kapitola1.pdf>.

[12] ŠVÉDA , Miroslav. *Síťové aplikace a správa sítí* [online]. 2009 [cit. 2011-05-11]. Studijní opora. Dostupné z WWW: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ISA-IT/texts/kapitola2.pdf>.

[13] *Jfitz* [online]. 2003 [cit. 2011-05-11]. Playing Cards. Dostupné z WWW: <http://www.jfitz.com/cards/>.

Seznam příloh

Příloha 1 : Komunikační protokol

Komunikace probíhá formou dotaz-odpověď, komunikační protokol je tedy rozdělen na požadavek klienta a možné odpovědi od serveru.

Klient — Server

Komunikace před zahájením hry

Klient se připojí a odešle svou přezdívku. Server odpoví výpisem založených her nebo zprávou, že jméno existuje.

Klient	Server
JOINED;NAME	EXIST
	GAMES;;HRA01:2;HRA02:1

Klient se chce připojit ke hře GAME_NAME. Server odešle všem info o novém uživateli a novému uživateli odešle info o už připojených hráčích. Nebo klientovy pošle, že je hra plná.

Klient	Server
TOGAME;GAME_NAME	JOIN;;NEWNAME
	CREATE;;NAME1;NAME2
	FULL

Klient chce vytvořit novou hru. Server hru vytvoří nebo odešle zprávu, že jméno existuje.

Klient	Server
CREATE;GAME_NAME;SETTING	CREATE;;
	GAMEEXIST

Klient je připraven hrát. Server odešle všem zprávu, že uživatel je připraven.

Klient	Server
READY	READY;;NAME

Komunikace v průběhu hry

Zakladatel hry odešle zprávu o startu hry. Server všem uživatelům odešle zprávu o startu s jejich kartami a pak odešle všem zprávu kdo začíná.

Klient	Server
START	START;;CARDS
	PLAY;;FIRST;NAME

Uživatel odhodí kartu. Server odešle všem info a zprávu o hráči, který je na řadě.

Klient	Server
OUT;CARD	PLAY;;OUT;CARD
	PLAY;;GO;NAME

Uživatel žádá kartu z balíčku. Server mu ji posílá.

Klient	Server
TAKE;PACKCARD;NAME	PLAY;;TAKE;PACK;CARD

Uživatel žádá odhozenou kartu. Server mu ji posílá.

Klient	Server
TAKE;OUTCARD;NAME	PLAY;;TAKE;OUT;CARD

Uživatel vykládá karty, zároveň pošle i odhozenou kartu. Server informuje všechny uživatele.
Sekce CARDS může obsahovat vyložené i doložené karty. Vyložené karty následují za řetězcem TABLE, postupky jsou odděleny středníkem a jednotlivé karty v postupkách dvojtečkou. Doložené karty následují za řetězcem ADD a mají tvar WHERE:CARD;. WHERE je id dříve vyložené postupky.

TABLECARD;CARDS;OUT;CARD;NAME; PLAY;;TABLECARDS;CARDS;NAME

Uživatelovy nezůstaly žádné karty a uzavírá tak hru, sekce DATA obsahuje poslední vyložené karty. Server odesílá všem vyložené karty viz. zpráva výše a informuje o ukončení hry.

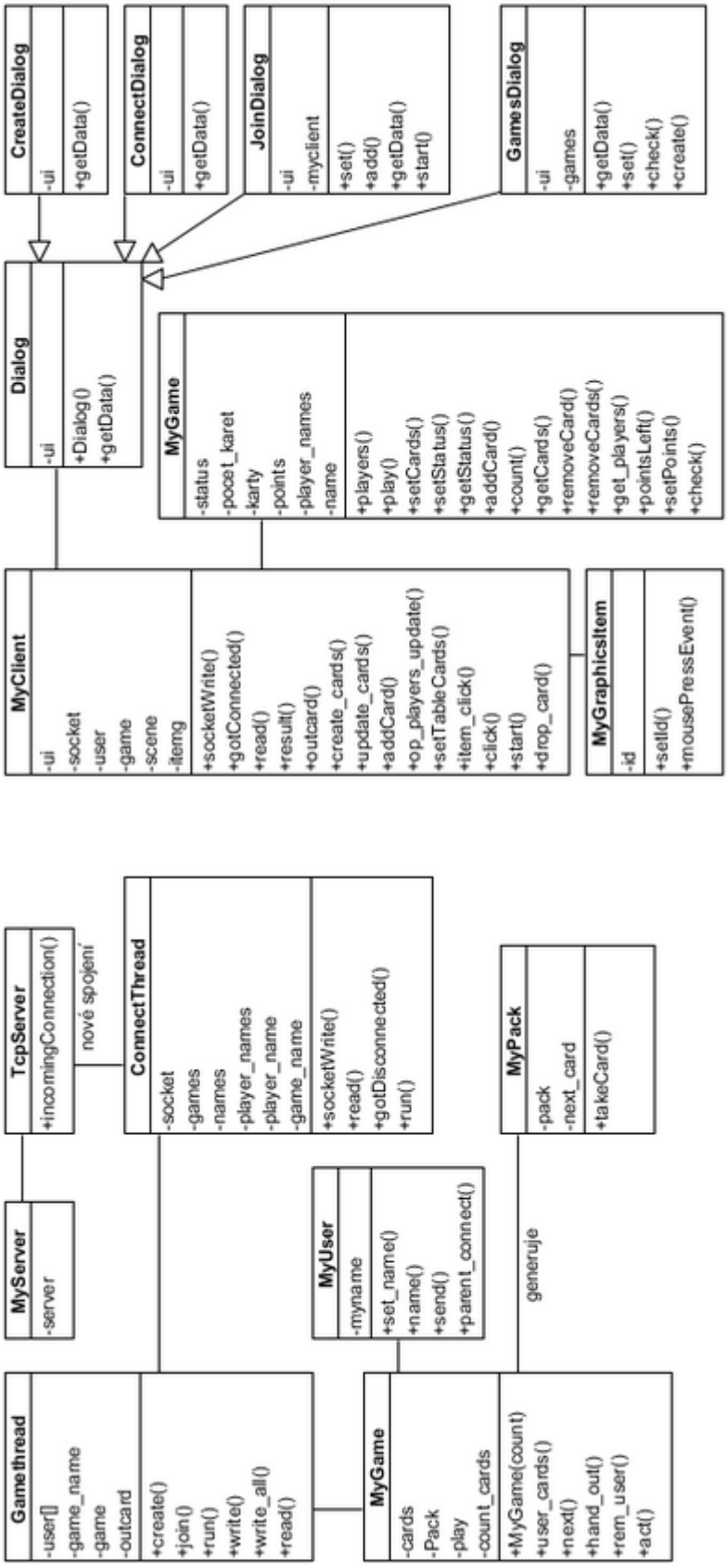
END;CARDS;NAME PLAY;;TABLECARDS...
PLAY;;END;NAME

Chat

Klient odešle zprávu do chatu. Server ji odešle všem.

CHAT;MESSAGE;NAME CHAT;;MESSAGE;NAME

Příloha 2 : Diagram tříd



Příloha 3 : DVD

/SRC – Zdrojové kódy programu
/DOC – Dokumentace a manuál k programu
/BP – Bakalářská práce ve formátu ODT a PDF
/BIN – Přeložené binární soubory pro operační systém Windows 7 32b
README.txt – všeobecné informace