VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

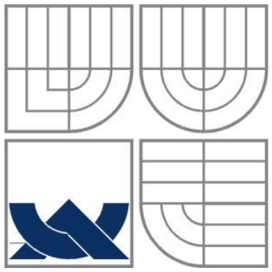
SÍŤOVÁ APLIKACE HRY KANASTA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

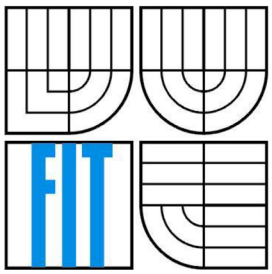
AUTOR PRÁCE
AUTHOR

PATRIK NĚMEČEK

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

SÍŤOVÁ APLIKACE HRY KANASTA

NETWORK APPLICATION OF THE GAME CANASTA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PATRIK NĚMEČEK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JAROSLAV STRUŽKA

BRNO 2011

Abstrakt

Tato práce popisuje analýzu, návrh, implementaci a testování síťové aplikace hry Kanasta. Jde o karetní hru pro 2 až 4 hráče, kde hráči spolu hrají po síti. V této práci je popsán a použit model klient/server, síťové protokoly TCP/IP a navržen vlastní komunikační protokol pro zasílání zpráv mezi klientskou a serverovou částí. Aplikace je implementována v programovacím jazyce C++ s využitím Qt frameworku. Výsledná aplikace je multiplatformní.

Abstract

This bachelor thesis focuses on the analysis, the design, the implementation and the testing of the network application of the game Canasta. It is a card game for 2 to 4 players playing over the network. In this Bachelor thesis you can find the description and the use of the model client/server, the network protocols TCP/IP and the self-proposed communication protocol for the sending of messages between client and server parts. The application is implemented in the programming language C++ with the use of Qt framework. The final application is multiplatform.

Klíčová slova

síťová aplikace, kanasta, klient/server, komunikační protokol, Qt framework

Keywords

network application, kanasta, client/server, communication protocol, Qt framework

Citace

Němeček Patrik: Síťová aplikace hry Kanasta, bakalářská práce, Brno, FIT VUT v Brně, 2011

Síťová aplikace hry Kanasta

Prohlášení

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

……
Patrik Němeček
18.5.2011

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce, panu Ing. Jaroslavu Stružkovi, za jeho názory, rady a postřehy při vytváření této práce.

Obsah

1 Úvod

Cílem práce bylo navrhnout a poté implementovat síťovou aplikaci karetní hry Kanasta. Vytvořená aplikace musí podporovat hru pro 2 až 4 hráče, zvláště hra ve 4 hráčích musí respektovat, že vždy dva hráči hrají spolu. Hra musí dodržovat a hlídat všechna pravidla hry a nesmí dovolit hráčům je porušovat nebo provádět neplatné herní operace. Hráčům je umožněno spolu hrát v lokální síti nebo přes internet.

Součástí práce bylo navrhnout komunikační protokol pro zasílání zpráv mezi klientem a serverem. Komunikační protokol je nutný z toho důvodu, aby byl klient schopen se dorozumět se serverem, aby mu poslal ty správné příkazy a naopak aby zprávy ze serveru dokázal správně interpretovat. Bez protokolu by komunikace nebyla možná.

Server musí udržovat spojení v rámci jedné hry se všemi klienty. Musí být schopen je obsluhovat všechny současně. Server představuje jádro celé hry. Veškeré činnosti spojené s přesouváním karet, dodržováním pravidel atd. se provádějí právě na serveru, který průběžně ukládá a pamatuje si stav hry. Poté veškeré informace o vývoji hry posílá klientům.

Klient naopak neprovádí žádný posun ve hře. Pouze dává příkazy serveru, jaké operace se mají provést, ale už se nestará o to, jak se to provede. Klient pouze zobrazuje výsledky, které spočítal server.

V této síťové aplikaci jsou pro předávání zpráv použity protokoly TCP/IP, které zajišťují bezproblémový přenos dat po síti bez jakýchkoli ztrát plynoucích z nespolehlivosti této sítě.

Práce je rozdělena do několika kapitol:

Druhá kapitola se věnuje nezbytnému teoretickému základu, který přibližuje síť, síťové modely, model klient/server, který je právě pro tuto aplikaci použit, a protokoly, především protokoly síťové a transportní vrstvy (TCP, UDP, IP).

Jeden z nejdůležitějších kroků při tvorbě aplikace je analýza a návrh, kterým se věnuje třetí kapitola. Je probráno, jak se při návrhu postupuje a jaké diagramy se používají k vyjádření těchto návrhů. Poté jsou navrženy a popsány diagramy odpovídající částem vytvářené aplikace.

Čtvrtá kapitola ukáže, jak je navržená aplikace implementována v konkrétním programovacím jazyce a v konkrétním integrovaném vývojovém prostředí. Je popsáno, jaké konstrukce jazyka byly použity, jak je co řešeno. Jsou nastíněny i problémy, které se při implementaci vyskytly. Nebude chybět ani ukázka navrženého komunikačního protokolu pro zasílání zpráv mezi klientem a serverem a ukázka částí kódu.

V předposlední kapitole je popsáno vše, co se týká testování. V jakých fázích vývoje testování probíhalo, na co se zaměřilo, jaké byly výstupy těchto testů, kde byly ty největší problémy po spuštění apod.

V závěrečné kapitole je shrnuta celá práce. Jsou zhodnoceny výsledky a přínos celé práce. Případně chyby, které aplikace má. Nakonec jsou navržena i možná rozšíření a vývoj, kterým by se mohla aplikace do budoucna ubírat.

2 Teoretický základ

Předtím než bude popsán samotný návrh a implementace aplikace, je potřeba projít a vysvětlit několik teoretických věcí a principů, na kterých je aplikace postavena. Jelikož se jedná o síťovou aplikaci, tak půjde o teorii síťových technologií a standardů, hlavně těch použitých v aplikaci. Na konci kapitoly jsou ještě zmíněny pravidla karetní hry kanasta.

Kapitola se zaměřuje na popis toho nezbytného teoretického minima, které je potřebné pro uvedení čtenáře do problematiky sítí, klient/server aplikací a síťových komunikačních protokolů.

2.1 Počítačová síť

Co je to vlastně ta počítačová síť? Existuje jistě mnoho definic, ale dalo by se říct, že se jedná o dvě nebo více navzájem propojených zařízení za účelem sdílení a výměny dat. Za taková zařízení můžeme považovat stolní počítače, servery, tiskárny, skenery, ale i mobilní telefony. Za účelem spojení těchto zařízení mezi sebou se používají síťové prvky. Mezi ně můžeme zařadit kabely, přepínače a směrovače. To, jak budou tyto síťově prvky propojeny a seřazeny, se nazývá topologie sítě. Poněvadž vytvářená aplikace běží na aplikační vrstvě (viz kapitola 2.2.1), principiálně nezáleží, na jaké topologii je síť postavena a jaké využívá síťové prvky, a proto se tomu už dále tato kapitola nebude věnovat.

2.2 Síťové modely

Aby komunikace mezi jednotlivými zařízeními v síti vůbec proběhla a aby si zařízení rozuměla, muselo dojít ke standardizaci. Za tímto účelem vznikly síťové modely, které rozdělují samotnou síť do několika vrstev. Pokud se síťové komponenty budou držet standardů v každé z vrstev, budou tyto komponenty navzájem kompatibilní.

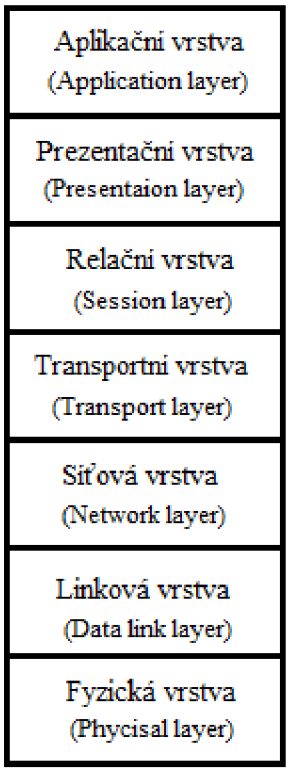
Architektura sítě je rozdělena do více vrstev hned z několika důvodů. Každá vrstva má za úkol jinou činnost, tzn., může se rychle a efektivně zaměřit pouze na jeden problém místo toho, aby těžkopádně a nespolehlivě prováděl všechny úkony spojené se síťovou komunikací. A navíc, pokud se na nějaké úrovni objeví problém, je velice jednoduché dohledat, která vrstva je za to zodpovědná a proč, a vykonat nápravu.

Modely také popisují, jak by měla probíhat síťová komunikace. Protokoly, pracující v různých vrstvách síťové architektury, mají samotnou komunikaci na starost. Více o protokolech v kapitole 2.3.

Informace o referenčních modelech ISO/OSI a TCP/IP jsou čerpány z [1],[2] a [3].

2.2.1 Referenční model ISO/OSI

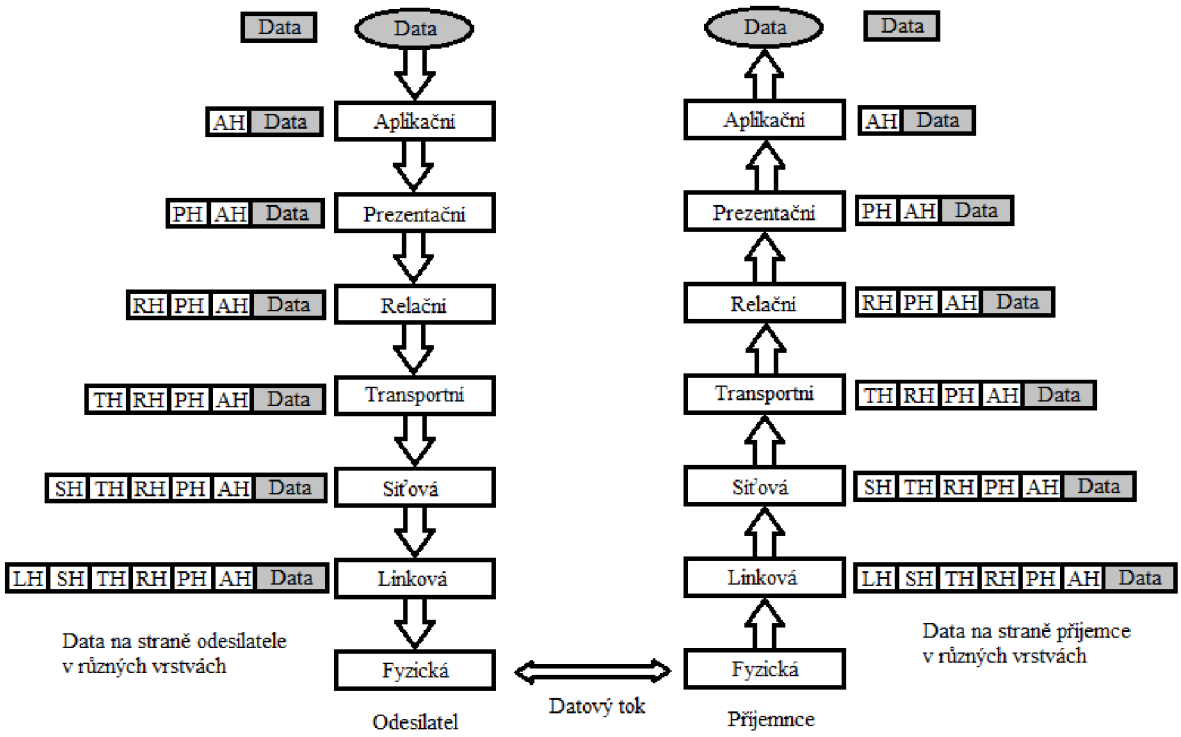
Model ISO/OSI (International Standards Organization/Open Systems InterConnection) je považován za nejčastější model používaný k popisu síťových technologií a zařízení. Byl vyvinut organizací ISO jako první krok směrem k mezinárodní standardizaci protokolů používaných v různých vrstvách. Rozděluje síťovou komunikaci do sedmi vrstev a říká, co by každá vrstva měla dělat. Naopak nespecifikuje přesně, jaké protokoly jsou v každé vrstvě. Model ISO/OSI je vidět na obrázku 2.1.



Obrázek 2.1 Referenční model ISO/OSI

Každá vrstva plní v síťovém komunikačním procesu určitou úlohu a data následně přenechává následující vrstvě (nahoru nebo dolů v závislosti na tom, zda probíhá příjem nebo odeslání dat). Komunikace začíná vždy v aplikační vrstvě odesílajícího systému. Data jsou v každé vrstvě opatřena hlavičkou. Tato hlavička se vloží před data. Jde o metadata obsahující dodatečné informace, které popisují obsah dat a jejich správné použití. Procesu přidání hlaviček se říká zapouzdření. Tímto se velikost odesílaných dat zvětšuje. Poté, co data projdou všemi vrstvami, je fyzická vrstva odešle. Na základě informací uložených v hlavičkách, dojde k doručení ke správnému cíli. Poté, co se data dostanou k příjemci, dochází k přesně opačnému postupu. Data jsou přepouštěna vrstvami nahoru od fyzické až k aplikační vrstvě a přitom jsou odebírány hlavičky v pořadí opačném, než jak byly přidávány. Když se datům podaří prorazit si cestu všemi vrstvami a jsou odstraněny všechny hlavičky, data se dostávají do své původní podoby a takové jsou prezentovány na přijímajícím počítači. Celý tento postup jde vidět na obrázku 2.2.

Činnosti jednotlivých vrstev modelu zde nebudou podrobně popisovány. Samotná aplikace Kanasta pracuje na aplikační vrstvě, ale není samotnou aplikační vrstvou. Jinak řečeno, aplikační vrstva zajišťuje interakci mezi aplikací a sítí. Na ostatní vrstvy se odkazuje v kapitole 2.3 Protokoly, kde je mimo jiné uvedeno, které protokoly patří do kterých vrstev.



Obrázek 2.2 Komunikace mezi vrstvami modelu ISO/OSI

V praxi je však velmi vzácné mít síť postavenou na architektuře přesně takto stanovených sedmi vrstev.

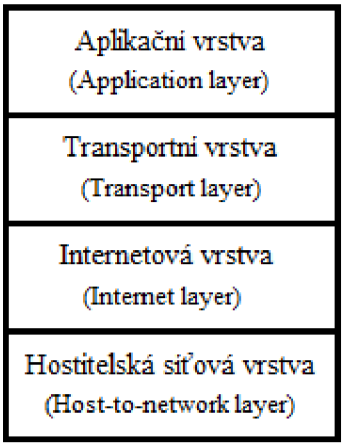
2.2.2 Referenční model TCP/IP

Referenční model TCP/IP je druhý nejznámější model. Většina moderních sítí je vybavena zařízeními, jejichž základem je právě referenční model TCP/IP. Tento model není však dostatečně flexibilní na popis jiných typů sítí, proto se neuvádí tak často jako model ISO/OSI.

Tento model je hodně podobný jako model ISO/OSI. Zachovává stejné principy, jde o stejná pravidla a důvody zavedení. Tím největším rozdílem je, že model TCP/IP je jednodušší, obsahuje pouze čtyři vrstvy. Síťovou vrstvu nahradila vrstva internetová; prezentační a relační vrstva úplně chybí a funkce linkové a fyzické vrstvy vykonává hostitelská síťová vrstva. Ostatně model TCP/IP je uveden na obrázku 2.3.

V tomto modelu jsou známé hlavně tři protokoly. Protokol TCP pro spojovanou komunikaci, protokol UDP pro nespojovanou komunikaci a protokol IP pro přenosy v síti. Všechny tyto protokoly budou podrobněji rozebrány v kapitole 2.3.

Jak už bylo řečeno, model ISO/OSI je vhodný pro teoretické vysvětlení podstaty věci, napomáhá porozumět síťové komunikaci, ale v praxi není vůbec rozšířen, kvůli své velké komplexnosti. Na druhou stranu je rozšířen model TCP/IP, kterému je naopak vytýkána nedostatečná obecnost. Také model sítě sítí, jak se lidově říká internetu, téměř identicky odpovídá modelu TCP/IP (liší se pouze fyzickou vrstvou navíc), navíc obsahuje shodné protokoly.



Obrázek 2.3 Referenční model TCP/IP

2.3 Protokoly

V předchozích kapitolách byl několikrát zmíněn termín protokol. Možná je intuitivně zřejmé, o co se jedná, nicméně je v této kapitole vysvětleno, co protokol je a jakou má v sítích funkci. Poté se tato kapitola zaměřuje už na vybrané konkrétní protokoly, hlavně na ty, které jsou v aplikaci Kanasta využívány. Jsou popsány protokoly TCP, UDP a IP.

Co je to tedy ten protokol? Dalo by se říci, že protokol je dohoda mezi komunikujícími stranami o tom, jak bude komunikace postupovat. Zkrátka jde o sadu určitých pravidel, podle kterých strany, v našem případě počítače, komunikují. Počítače komunikují formou zasílaní zpráv, které, jak je uvedeno v kapitole 2.2, postupují ve formě dat všemi vrstvami síťového modelu, a právě protokoly mají na starost rozlišit a poté určitým směrem zpracovat danou zprávu a její části. Jinak řečeno, protokoly definují formát zasílaných zpráv, pořadí zasílání zpráv a jejich příjem mezi dvěma nebo více prvky síťového prostředí a reakce na tyto přijaté a odeslané zprávy.

2.3.1 Sada protokolů TCP/IP

Zkratka vznikla z názvů dvou protokolů: TCP a IP. Sada protokolů TCP/IP je zdaleka nejrozšířenější. Jednak proto, že využívá flexibilní adresní schéma, které je dobře směrovatelné i v rozsáhlých sítích, a jednak proto, že TCP/IP je protokolem globálního Internetu, takže aby se systém mohl připojit do sítě Internet, musí na něm běžet.

Tato sada protokolů obsahuje i další protokoly a standardy pro zasílání dat na sítích. Tyto další protokoly pracují na všech vrstvách modelu TCP/IP.

Tato sada protokolů je využita i v této aplikaci Kanasta.

2.3.2 Protokol internetové vrstvy IP

Internetový protokol IP je hlavní složkou sady protokolů, jejichž funkcí je doručování paketů mezi dvěma koncovými body. Pakety jsou datové jednotky, na které je zpráva ještě před odesláním rozdělena. Jednotlivé pakety jsou poté prostřednictvím sítě odeslány. Na straně příjemce jsou pakety opět složeny do jednoho jediného celku – tím vznikne opět původní zpráva.

Existují dvě varianty IP: IP verze 4 (IPv4) a IP verze 6 (IPv6). IPv4 je mnohem rozšířenější než IPv6. V době psaní této práce organizace IANA již přiřadila veškeré adresy IPv4, a tudíž je celý adresní prostor IPv4 vyčerpán. Předpokládá se pozvolné rozšíření IPv6, nicméně IPv4 zůstane zachován. IPv4 je tvořen 32 bity a zapisuje se čtyřmi čísly desítkové soustavy v hodnotách od 0 do 255 oddělené tečkami, zatímco IPv6 je tvořen 128 bity a zapisuje se osmi skupinami oddělených dvojtečkami, kde každá skupina je tvořena čtveřicí šestnáctkových číslic 0 až F.

Internetová vrstva provádí úkoly směrování. IP umožňuje toto směrování používáním IP adres k identifikaci síťových prvků. Existuje více typů směrování. V této vytvářené aplikaci je využíváno jednosměrové směrování (unicast) – pakety nesou jedinou cílovou adresu. Protože každý počítač v síti má jedinečnou IP adresu, paket dorazí přesně tam, kam má. Tato adresa je spolu s dalšími potřebnými informacemi vložena do hlavičky internetové vrstvy a tato hlavička je připojena před data z vyšší vrstvy (transportní), jak je uvedeno v kapitole 2.2. Internetovou vrstvu nezajímá, jaké protokoly jsou použité v nižších vrstvách ani jaký je použit hardware.

Protože aplikace Kanasta je určena pro TCP/IP sítě, je využívání IP její nedílnou součástí.

V této kapitole se vycházelo z [1], [2] a [4].

2.3.3 Protokoly transportní vrstvy TCP a UDP

Protokoly TCP (Transmission Control Protocol) a UDP (User Datagram Protocol) operují na transportní vrstvě. Jak je uvedeno v kapitole 2.3.2, IP protokol zabezpečí spojení mezi vzdálenými počítači, takže protokoly TCP a UDP, které jsou nad ní, nevidí, co se děje v nižších vrstvách a slepě na ně spoléhají. Těmto protokolům je zcela jedno, zda stojí počítače vedle sebe nebo každý na jiném kontinentu. Transportní vrstva přepokládá, že je spojení mezi počítači zajištěno, a pouze se věnuje předávání dat mezi aplikacemi na vzdálených počítačích.

Na počítači může běžet více síťových aplikací současně. Z hlediska IP vrstvy se tyto aplikace nacházejí na počítači, který je zpravidla adresován stejnou IP adresou. Jak tedy počítač určí, které aplikaci má která data předávat ke zpracování? To je právě základním úkolem protokolů pracujících na transportní vrstvě. Pro adresaci aplikací zavádí tato vrstva tzv. porty, což jsou čísla, která jednoznačně identifikují aplikaci. Koncový bod je tedy definován dvěma parametry: IP adresou a

číslem portu. To, že jsou datové toky zasílány souběžně za pomoci několika paralelních procesů, se nazývá multiplexování.

Transportní vrstva tudíž řídí a spravuje datová spojení, která otevírá, používá a zavírá dle aktuální potřeby.

Protokol TCP

Protokol TCP je v současnosti nejpoužívanější transportní protokol v počítačových sítích. Jde o spojovanou službu, tzn., že nejdřív ustaví spojení před samotným přenosem dat a následně toto spojení udržuje. Ke vzniku TCP vedla snaha o spolehlivou komunikaci v nespolehlivé síti. IP pakety totiž můžou příjemci přijít porušené, opožděné, v jiném pořadí, nebo se můžou ztratit a nedorazit vůbec. Všechny tyto problémy řeší TCP. Aplikace vydá jediný povel pro zaslání dat a TCP se postará o všechny další okolnosti přenosu. Je zajištěno, že zpráva rozdělená do více paketů dojde ve správném pořadí, že zpráva dojde nepoškozena, i to, že jsou pakety doručeny v rozumně velkých úsecích (pokud je příjemce přetížen, odesílatel sníží tok dat). Tou hlavní nevýhodou TCP je jeho rychlost. Tím, že po každém odeslání paketu čeká TCP na potvrzení od přijímací strany, se objevuje zpoždění. Síť se zahlcuje pakety a potvrzeními. Používají se sice různé optimalizační techniky, ale v rychlosti se TCP nemůže UDP vyrovnat.

Spolehlivost se řeší vkládáním informací do hlavičky transportní vrstvy, která se připojuje před samotná posílaná data (jak je uvedeno v kapitole 2.2). Mezi tyto informace řadíme číslování paketů, kde každý paket je opatřen číslem, které říká, kolikátý paket to je v pořadí. Pokud dojde paket s jiným číslem, než je očekáváno, paket je zahozen a není odesláno potvrzení nazpět. Pokud nedojde odesílateli potvrzení, odešle stejný paket znovu. Příjemce duplikované pakety zahazuje. Další informací vkládanou do hlavičky je kontrolní součet, který ještě před odesláním odesílatel vypočítá nad daným paketem. Příjemce po obdržení paketu také vypočítá kontrolní součet a porovná ho s příchozím, pokud kontrolní součty sedí, je odesláno potvrzení.

Přenosová jednotka (TCP hlavička + aplikační data), se kterou pracuje TCP na transportní vrstvě, se nazývá TCP segment.

Informace o TCP byly získány z [1] a [4].

Protokol UDP

Druhým nejpoužívanějším transportním protokolem je protokol UDP. U UDP jde o nespojovanou službu. Tzn., že odesílatel neustavuje spojení před samotným odesláním dat. Jedná se o nespolehlivý přenos dat. Odesílatel tak nemá možnost se dozvědět, zda data přišly v pořádku, nebo ne. Přijímací strana odebírá pakety v tom pořadí, v jakém přišly. Nestará se o to, zda jsou doručeny ve správném pořadí nebo zda nějaký paket úplně chybí. Obsahuje ale alespoň kontrolní součet, aby si příjemce mohl ověřit, zda jsou data nedotčená. UDP tedy nemá možnost řídit tok, zahlcení ani přetížení.

Protože se nemusí zabývat režií pro zajištění spolehlivosti, je UDP poměrně rychlý. Využívá se proto tam, kde je kladen důraz na rychlost a ne už tak moc na spolehlivost dat. Příkladem může být streamování dat, videokonference, internetová telefonie apod., kde ztráta jednoho paketu nemá na subjektivní dojem ze streamu vliv, projeví se pouze šumem ve zvuku nebo spatně vykreslenými pixely ve videu.

U UDP bylo použito nesprávné označení – paket. Paket popisuje jednotku dat, na které je sekvenčně rozdělena zpráva na straně odesílatele a opět složena dohromady na straně příjemce. U nesekvencovaných datových jednotek, které jsou přenášeny UDP, se používá termín datagram.

Hlavička TCP má 24 bajtů, oproti ní má hlavička UDP pouze 8 bajtů.

Kapitola o UDP čerpá z [1].

Po zvážení pro a proti byl pro síťovou aplikaci Kanasta zvolen protokol TCP, protože je bezpodmínečně nutné, aby pakety přicházely mezi hráči v pořádku, a protože rychlost posílání zpráv je z důvodu pomalé hráčské interakce nízká (vzhledem k rychlostem dnešních síťových prvků). Mohl se použít i protokol UDP s tím, že by se musela řešit spolehlivost ve vlastní režii, ale to by se zbytečně zvýšila náročnost implementace a možnost vzniku dalších chyb.

2.4 Klient/server model

Model klient/server je v současnosti široce rozšířený a tvoří základ mnoha síťových použití. Jde o dvouvrstvou architekturu, ve které zpracování dat provádí serverový systém a výsledky poskytuje klientským systémům. Klient požaduje od serveru službu, server ji klientovi poskytuje. Aby se dva počítače daly označit za klienta a server, je potřeba, aby na straně serveru běžel serverový software, zatímco na straně klienta se očekává klientský software. Mezi klientem a serverem je nutná schopnost vzájemné komunikace prostřednictvím komunikačního protokolu. Ten se obvykle skládá ze dvou podprotokolů. Tím jedním klient žádá server o službu, tím druhým naopak server poskytuje data zpět klientovi. Jde o komunikační protokol dotaz/odpověď.

Celá komunikace obecně probíhá tak, že klient a server jsou každý spuštěn na jiném počítači, které jsou spolu propojeny sítí. Klient zahajuje spojení, vysílá požadavek na server a čeká na odpověď. Server naslouchá na určitém portu. Poté, co k němu dorazí požadavek od klienta, tento požadavek zpracuje a odesílá odpověď zpět klientovi. Vše je znázorněno na obrázku 2.4.

Server, aby mohl vykonávat svou funkci, musí být konkurentní. Musí být schopen obsloužit více klientů najednou. Dedikovaný server (tj. server, který je primárně určený k poskytování služeb a ke sdílení dat) je obvykle spuštěn na výkonném stroji, takže je schopný svou konkurentnost zajistit. Ať už paralelním zpracováním požadavků, nebo rychlým sekvenčním zpracováním, které budí dojem paralelismu.

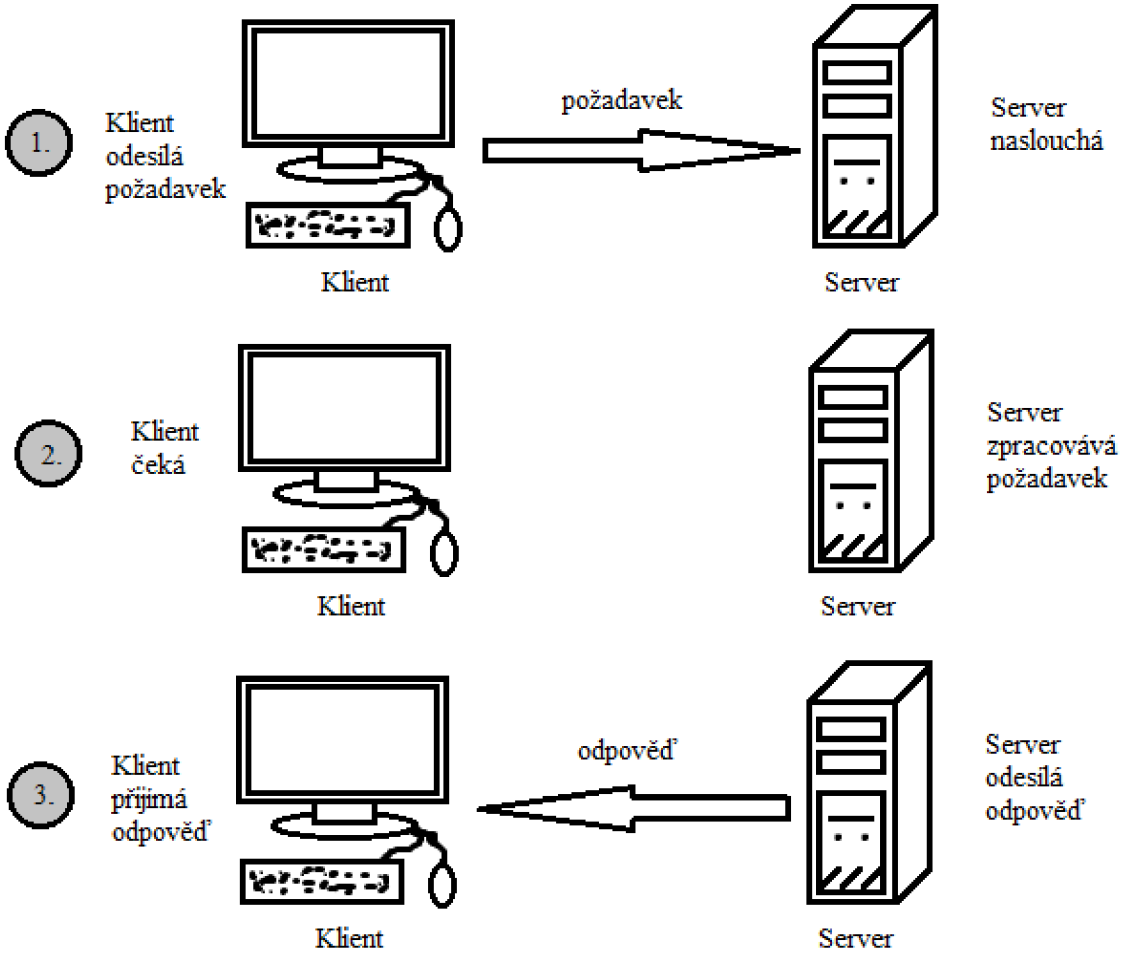
Příklady architektur založených na modelu klient/server jsou např. webový prohlížeč/webový server, emailový klient/emailový server, databázový klient/databázový server aj.

Výhodou tohoto centralizovaného přístupu je, že server provádí veškeré výpočty ohledně dané služby sám a minimalizuje tak samotné zatížení klientů. Naopak nevýhodou může být to, že pokud přijde v jeden okamžik na server velké množství požadavků od různých klientů, může server havarovat a stát se nedostupný. Na tomto principu jsou založeny DOS (Denial of Service) útoky, kdy je server zavalen obrovským množstvím požadavků od útočníků.

Pro úplnost ještě zbývá dodat, že další častá síťová architektura z pohledu návrhu je model peer-to-peer (P2P), kde jsou si všechny koncové body rovny a obstarávají jak funkci klienta, tak funkci serveru. Nicméně tento model není z pochopitelných důvodů v této aplikaci použit, a tak se mu nebude dále věnovat pozornost.

Pro tuto vytvářenou aplikaci, je model klient/server ideální. Hráči budou plnit roli klientů a samotná logika hry bude probíhat na serveru. Více o samotném návrhu aplikace je popsáno v kapitole 3.

Tato kapitola je zpracována podle [1] a [3].



Obrázek 2.4 Komunikace modelu klient/server

2.5 Karetní hra Kanasta

Kanasta je poměrně rozšířená karetní hra z rodiny Rummy, která vznikla v Uruguayském Montevideu. Existuje více variant této hry, které se liší většími či menšími detaily. Hra se hraje se dvěma balíčky francouzských karet – v každém balíčku jsou karty o hodnotách 2 – 10, J, Q, K, A ve čtyřech barvách (srdce, piky, káry, kříže) a dva žolíky. Celkem je tedy ve hře 108 hracích karet. Hra je určena pro 2 – 4 hráče. Nyní budou popsány pravidla té varianty hry, kterých se drží samotná implementace. Tato varianta vychází z [10].

2.5.1 Pravidla Kanasty

Nejčastější je varianta se 4 hráči, kde vždy 2 a 2 hráči sedící proti sobě hrají spolu. Když hrají 2 nebo 3 hráči, tak hraje každý sám za sebe. Rozdává se obvykle celkem 13 karet, postupně po jedné kartě po směru hodinových ručiček. Pokud hráč obdrží při rozdávání červenou trojku, okamžitě ji vyloží na stůl lícem nahoru a při nejbližší příležitosti si dokoupí chybějící karty do počtu 13. Po rozdání se dá jedna karta do středu, na tomto místě ve středu bude vznikat odkládací balík (dále jen balík). Zbylé karty se položí rubem nahoru vedle balíku, tato hromádka se označuje jako kmenový balík.

Každý hráč, když na něj přijde řada, provádí tři úkony hry: a) musí si koupit kartu z kmenového balíku (tu nikomu neukazuje) nebo (za určitých podmínek) z balíku, b) může vyložit nebo doložit karty, c) musí odhodit kartu na balík.

Cílem hry je dosáhnout určitého počtu bodů (nejčastěji se jedná o 10 000 bodů). Body získává strana součtem bodové hodnoty karet vyložených na stůl oběma spoluhráči a připočtením všech prémií. Bodové hodnoty karet a prémií jsou v tabulce 2.1, resp. tabulce 2.2.

žolík	50 bodů
dvojka	20 bodů
eso	20 bodů
K, Q, J, 10, 9, 8	10 bodů
7, 6, 5, 4, černé 3	5 bodů

Tabulka 2.1 Bodové hodnoty karet

divoká kanasta žolíková	2000 bodů
divoká kanasta dvojková	1000 bodů
čistá kanasta	600 bodů
smíšená kanasta	300 bodů
uzavření kola	200 bodů
červené trojky	100 bodů

Tabulka 2.2 Bodové hodnoty prémií

Pokud se straně podaří vyložit kanastu, na konci kola se jí sečtou body za vyložené karty a za prémie a odečtou body za karty, které hráčům zůstaly v ruce. Pokud se ovšem straně nepodaří vyložit alespoň jednu kanastu, jsou ji odečteny nejen body za karty v ruce, ale i body za karty vyložené její stranou i prémiové body za červené trojky. Pokud ovšem tým nemá vyloženou kanastu, tak nemůže uzavřít kolo. Po konci kola se karty opět zamíchají, rozdají a začne kolo další.

Kanasta je sedm vyložených karet stejné hodnoty. Pokud hráč vyloží všech sedm originálních karet, jde o kanastu čistou. Originální karty jsou všechny, kromě dvojek a žolíků. Dvojky a žolíky jsou karty divoké. Divoké karty můžou být při vykládání použity za jakoukoli originální kartu. Kanasta, která obsahuje alespoň jednu divokou kartu, se nazývá smíšená. Ovšem smíšená kanasta musí vždy obsahovat alespoň 4 karty originální. Pokud je kanasta tvořena ze samých divokých karet, jde o kanastu divokou, a to žolíkovou, pokud obsahuje všechny čtyři žolíky, jinak jde o kanastu dvojkovou.

Pravidla dovolují vykládat skupinu nejméně tří karet stejné hodnoty. Může přitom jít i o dvě karty originální a jednu divokou. Je připuštěno i vykládání sekvencí karet stejné barvy, vždy jen od nejnižší po nejvyšší. Vykládání sekvencí ale ztěžuje tvoření kanast. Doložit kartu k již vyloženým kartám svým nebo svého spoluhráče může hráč i jednotlivě, kdykoli bude na řadě. Doložit k sekvenci lze vždy jen vyšší kartu. Pokud strana vykládá v aktuálním kole poprvé, musí splnit dvě podmínky. Zaprvé hráč, který vykládá, musí vyložit alespoň jednu skupinu karet, která je složena minimálně ze tří karet originálních. Zadruhé musí hráč svým vyložením být schopný najednou dosáhnout určité hodnoty bodů. O jaké hodnoty se jedná, je uvedeno v tabulce 2.3.

0 až 1495 dosažených bodů	50 bodů
1500 až 2495 dosažených bodů	90 bodů
2500 až 4995 dosažených bodů	120 bodů
5000 a více dosažených bodů	150 bodů

Tabulka 2.3 Minimální bodové hodnoty pro první vyložení při dosaženém počtu bodů

Je tedy zřejmé, že minimálního počtu bodů může být dosaženo vyložením jedné nebo několika skupin, ale nejméně jedna skupina musí být tvořena alespoň třemi originálními kartami, zbylé můžou použít i karty divoké, ovšem s tou podmínkou, že nejméně dvě karty budou originálními.

Když hráč koupí červenou trojku z kmenového balíku, položí ji před sebe lícem nahoru a koupí si kartu novou. Pokud v kmenovém balíku nezbydou žádné karty, kolo je okamžitě ukončeno. Za určitých podmínek může hráč koupit i odkládací balík. Pokud tyto podmínky hráč nesplní, nemůže si balík koupit a musí si koupit kartu z kmenového balíku. Pokud ovšem hráč podmínky splní, může si vzít celý zbytek balíku do ruky. Děje se tak v situaci, kdy před hráčem na řadě hrající protihráč odložil kartu, která se hráčovi na řadě hodí.

Kartu z odkládacího balíku lze vzít až po třetím tahu.

Pokud jde o první vyložení, nebo je-li balík zmražen (viz dále), musí být hráč schopen k odebrané kartě z balíku přiložit dvě karty originální a ty pak vyložit. Samozřejmě při prvním vyložení musí hráč splnit i podmínku minimálního počtu bodů při prvním vyložení.

Pokud má strana už první vyložení za sebou a není-li balík zmražen, pak může odloženou kartu a celý balík hráč koupit i k jedné kartě originální z ruky a druhou nahradit kartou divokou; ty pak vyložit.

Pokud není balík zmrazen, může hráč koupit balík také, odloží-li jeho předchůdce kartu, která se hodí do skupiny jeho stranou již vyložených karet. V takovém případě hráč koupenou odloženou kartu přiloží k již vyložené skupině karet a balík si vezme.

S černými trojkami se normálně hraje, ale nelze je vykládat během hry. Lze je vyložit pouze, pokud tím hráč v aktuálním tahu zároveň uzavře kolo. Černé trojky mají svou zvláštní funkci. Pokud ji hráč odhodí na odkládací balík, následující hráč balík nesmí koupit. Jde tedy o obranou kartu.

Pokud hráč odhodí divokou kartu nebo červenou trojku, zmrazí balík do doby, než bude opět na řadě. Zmrazený balík nemůže koupit hráč následující a ztěžuje se koupě balíku i dalším hráčům. Ti můžou koupit balík jedině doplněním dvěma originálními kartami z ruky, jak bylo popsáno výše.

Kolo uzavře hráč, který vyložil všechny karty na stůl a ještě mu zbyla karta na odhození, touto kartou kolo uzavírá. Po konci kola se přepočítají body a strana, který dosáhne jako první 10 000 bodů, vyhrává. Pokud v témže kole obě strany překročí hranici 10 000 bodů, vyhrává ta strana, která bude mít těch bodů více.

3 Analýza a návrh

Každý softwarový projekt by měl při svém vývoji projít několika fázemi. Tyto fáze tvoří životní cyklus softwaru. Mezi typické fáze životního cyklu softwaru patří:

- analýza a specifikace požadavků
- návrh
- implementace
- testování
- provoz a údržba

Protože síťová aplikace hry Kanasta je bezpochyby softwarový projekt, měl by její vývoj projít právě těmito fázemi. Tím by se mělo dosáhnout úspěšné, kvalitní a snadno spravovatelné aplikace.[6]

3.1 Analýza požadavků

Požadavky na aplikaci byly pevně dány v zadání. Musí se jednat o síťovou aplikaci karetní hry Kanasta, která musí dodržovat její pravidla. Dále musí umožňovat hru pro 2 až 4 hráče. Mělo by se jednat o klient/server aplikaci, kde se hráči postupně připojí k hracímu serveru, který s nimi udržuje spojení po čas celé hry. Klientský program by měl sloužit jen jako rozhraní mezi hráčem a hracím serverem. Měl by umožňovat hráčům zadávat příkazy, které jsou posílány ve formě zpráv na server, a prezentovat příchozí odpovědi v požadované formě. Samotná logika aplikace, tedy dodržování pravidel a ukládání stavu hry, by měl obstarávat serverový program. Za účelem komunikace mezi klientem a serverem, je požadováno vytvoření komunikačního protokolu, což je celkem logické vzhledem k modelu klient/server. Kdykoli bude komunikovat hráč s hráčem, všechny zprávy půjdou přes server.

V zadání není přesně specifikováno, jak by měla aplikace na straně klienta a na straně serveru vypadat. Z povahy aplikace ale vyplývá, že by se mělo jednat, přinejmenším na straně klienta, o okenní aplikaci, pokud možno s intuitivním a zažitým vzhledem a ovládáním, jaké je pro podobné aplikace typické. Protože se jedná o karetní hru, kde je kladen důraz na přemýšlení, taktiku a zábavu, stačí, když půjde o obyčejnou 2D grafiku. Na tom, zda server bude navržen jako aplikace konzolová nebo okenní, tak moc nezáleží, protože server obvykle zůstane uživatelům skrytý po celou dobu hraní hry. Server si bude provádět své výpočty v tichosti u sebe, uživatel by neměl ani moc pocítit, že server vůbec existuje.

Co se týče samotných pravidel hry Kanasta, vychází se přesně z té varianty, která je popsána v kapitole 2.5.1.

3.2 Návrh aplikace

Návrh aplikace by měl navazovat na analýzu požadavků, sloužit k ujasnění koncepce celé aplikace a provést dekompozici na menší části. Měl by se soustředit na podrobnou specifikaci těchto částí, na specifikaci uživatelského rozhraní, na výběr algoritmů realizujících požadované akce a na stanovení logické a fyzické struktury funkcí a dat. Návrh bere také do úvahy platformu, na níž bude aplikace implementována, a programovací jazyk, který je pro realizaci využit.

Při vytváření aplikace je použit objektově orientovaný přístup. Tento přístup je obecně brán jako nejefektivnější a nejlepší při tvorbě rozsáhlých projektů. Tato aplikace nebude svým rozsahem extra obrovská, ale bude mít podle prvních odhadů přes 1500 řádků kódu, takže je vhodné zvolit objektově orientované modelování, návrh a implementační jazyk.

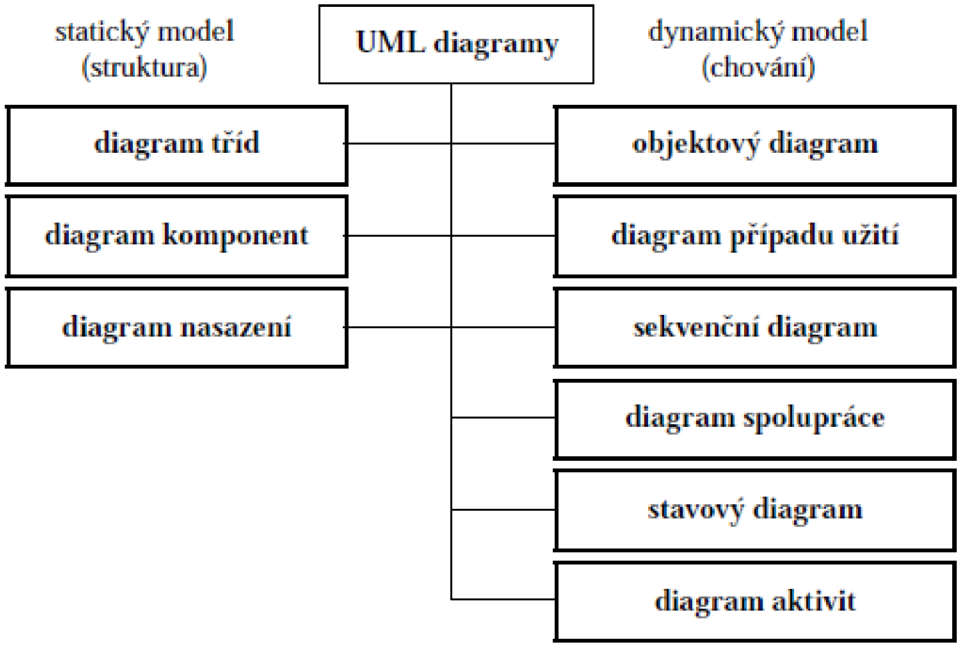
Objektově orientovaný návrh chápe aplikaci jako kolekci vzájemně komunikujících objektů, které zapouzdřují data a funkcionalitu. Mezi základní vlastnosti patří abstrakce, zapouzdření, polymorfismus a dědičnost. Výhodou objektově orientovaného přístupu k návrhu je stabilita navrhovaných prvků a jednoduchost jejich změn. Metodologie založené na objektech vykazují větší míru odolnosti. Modelovací jazyk, který je v současné době brán jako standardní modelovací jazyk pro objektově orientovaný přístup vývoje softwaru, je jazyk UML (Unified Modelling Language). Jde o vizuální modelovací jazyk doplněný textem. Syntaxe a sémantika symbolů použitých v modelech je dána definicí jazyka. Model, vytvořený v tomto jazyce, má podobu obrázků – diagramů. Mezi stavební bloky UML patří prvky, reprezentující prvky jazyka, vztahy, modelující vazby mezi prvky, a právě diagramy, které představují způsob vizualizace toho, co bude aplikace dělat, případně jak to bude dělat.

Jazyk UML nabízí několik základních diagramů, které můžeme rozdělit do dvou skupin: pro modelování statické struktury a pro modelování dynamické struktury aplikace. Přehled diagramů je zobrazen na obrázku 3.1. Některé diagramy jazyka UML jsou popsány trochu blíže, hlavně ty, které jsou považovány za nejdůležitější.

Předchozí odstavce vycházely z [5] a [6].

Protože je při návrhu aplikace Kanasta použitý objektově orientovaný přístup, je vytvořen vlastní návrh aplikace pomocí těchto několika diagramů jazyka UML. Níže uvedené diagramy jsou uvedeny proto, aby pomohly lépe se orientovat v samotném návrhu. Ukázky diagramů nezahrnují veškerou strukturu, prvky, vazby ani funkčnost celé aplikace. Diagramy přibližují jen ty nejdůležitější části nebo ukazují pro představu příklady fungování jen určitých procesů. První fáze návrhu diagramů se uskutečnila před samotnou implementací. Tento návrh vycházel jen ze samotné analýzy požadavků. Druhá fáze návrhu diagramů přišla na řadu až po úspěšné implementaci, kdy byly některé prvky a vazby předělány, jiné úplně vynechány a nahrazeny novými. Vycházelo se z poznatků, které dala samotná implementace problému. Tento postup není nijak výjimečný. Běžně se v určitých fázích implementace upravuje návrh za účelem odstranění problémů, které se mohly vyskytnout při

implementaci. Tím se potvrdila skutečnost, že tyto dvě fáze v životním cyklu softwaru se někdy více či méně prolínají.



Obrázek 3.1 UML diagramy (převzato z [6])

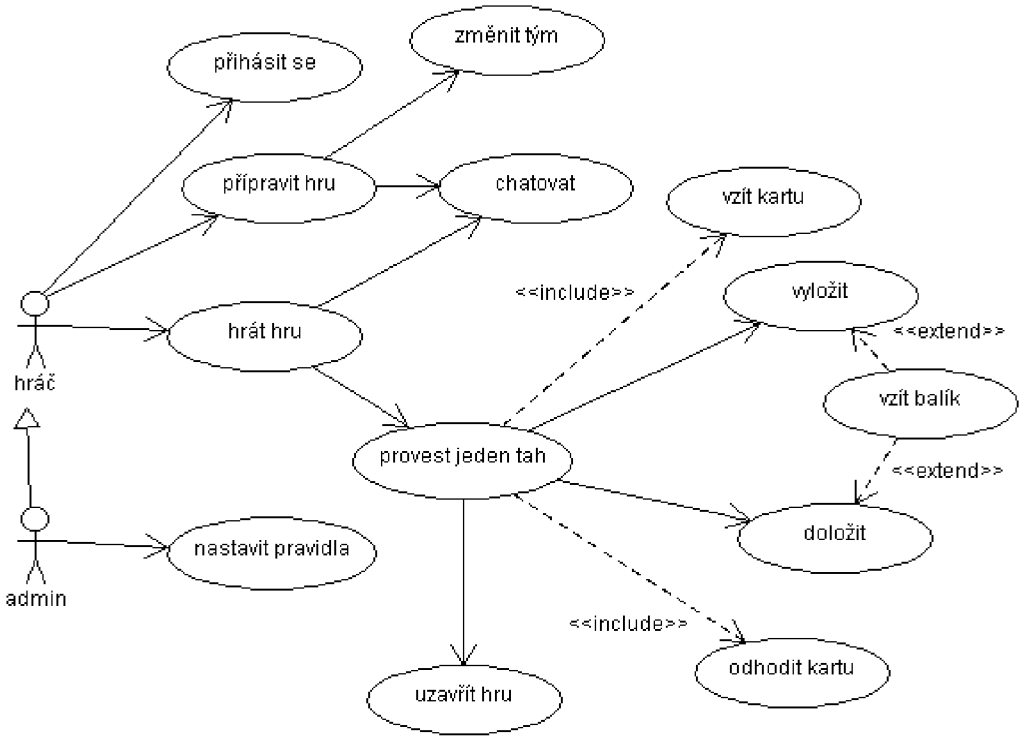
Teorie k diagramům, které jsou uvedeny v následujících kapitolách, staví na [5].

3.2.1 Diagram případu užití

Jedním z hlavních diagramů jazyka UML je diagram případu užití (Use Case Diagram). Jde o hlavní techniku modelování chování aplikace. Slouží pro nalezení případů užití aplikace (jednotlivých funkcí) a účastníků, kteří na případech participují. Zobrazuje případy užití, aktéry (účastníky) a interakce mezi aktéry a případy užití. Případ užití reprezentuje nějakou důležitou část celkové požadované funkčnosti vyvíjené aplikace. Aktér reprezentuje uživatele aplikace (osobu nebo jinou aplikaci) v nějaké roli, který přímo komunikuje s případem užití a čeká od něj nějakou zpětnou vazbu. Každý z těchto případů užití je zahájen samotným aktérem. Případ užití popisuje, co by měla aplikace dělat, ne jak by to měla dělat.

Na obrázku 3.2 se nachází diagram případů užití navržené aplikace Kanasta. Diagram není úplně detailní, jde jen o zevrubný pohled na případy užití; pro přehlednost je to dostačující. Vztah mezi aktérem *hráč* a aktérem *admin* se nazývá generalizace. Aktér *admin* dědí všechny vztahy k případům užití od aktéra *hráč* a ještě má navíc nové případy užití. Vztah *<<include>>* znamená, že dodatkový případ užití je proveden vždy, když je proveden jeho bázový případ užití. Vztah *<<extend>>* znamená, že rozšiřující případ užití lze provést jen za určitých okolností, když je proveden jeho bázový případ užití.

Z diagramu na obrázku 3.2 lze lehce vyčíst, jaké akce můžou hráči hry provádět.



Obrázek 3.2 Diagram případů užití aplikace Kanasta

Součástí diagramu případů užití obvykle bývá i textová specifikace případů užití. Právě tyto specifikace jsou důležitými zdroji informací. Jde o strukturovaný popis informací o případu užití. Mezi tyto informace patří mimo jiné i posloupnost kroků daného případu. V tabulce 3.1 je uveden příklad textové specifikace případu užití aplikace Kanasta s názvem *VyložKarty*. Jak název napovídá, jde o popis informací o případu, kdy chce hráč provést vyložení karet.

Případ použití: VyložKarty
ID: 5
Stručný popis: Hráč provede vyložení svých karet na stůl.
Primární aktéři: Hráč
Sekundární aktéři: Žádný
Předpoklady: Hráč je na řadě Hráč si vzal kartu z balíku
Hlavní tok: 1. Případ použití se spustí, když Hráč klikne do své vykládací plochy. 2. Server zkontroluje, zda jsou dodržena všechna pravidla. 3. Server odebere Hráčovi z ruky karty a vyloží je.
Následné podmínky: 1. Hráčovi se zobrazí vyložené karty na vykládací ploše.
Alternativní toky: NesouhlasíKombinaceKaret

Tabulka 3.1 Specifikace případu užití VyložKary aplikace Kanasta

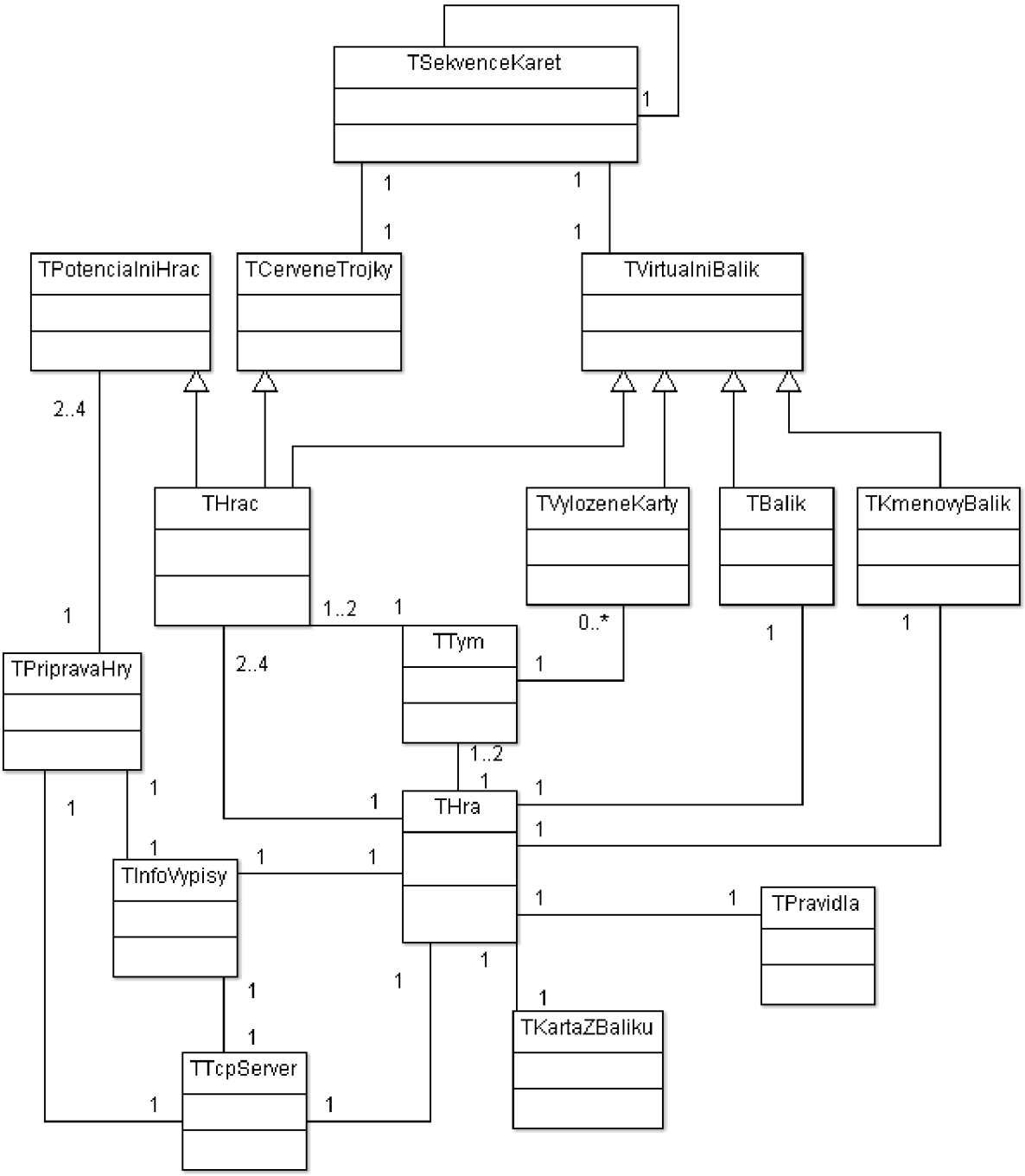
3.2.2 Diagram tříd

Další hodně důležitý diagram v objektově orientovaném modelování je diagram tříd (class diagram), který zobrazuje třídy, jejich interní strukturu a vztahy k ostatním třídám. Diagram tříd je logický pohled na statickou strukturu aplikace. Třída slouží jako předpis pro množinu objektů, které mají stejné atributy a metody. Třída samotná definuje tyto atributy a metody. Objekty jsou pak instancemi tříd. Objekt a třída jsou tedy základní pojmy objektově orientovaného programování.

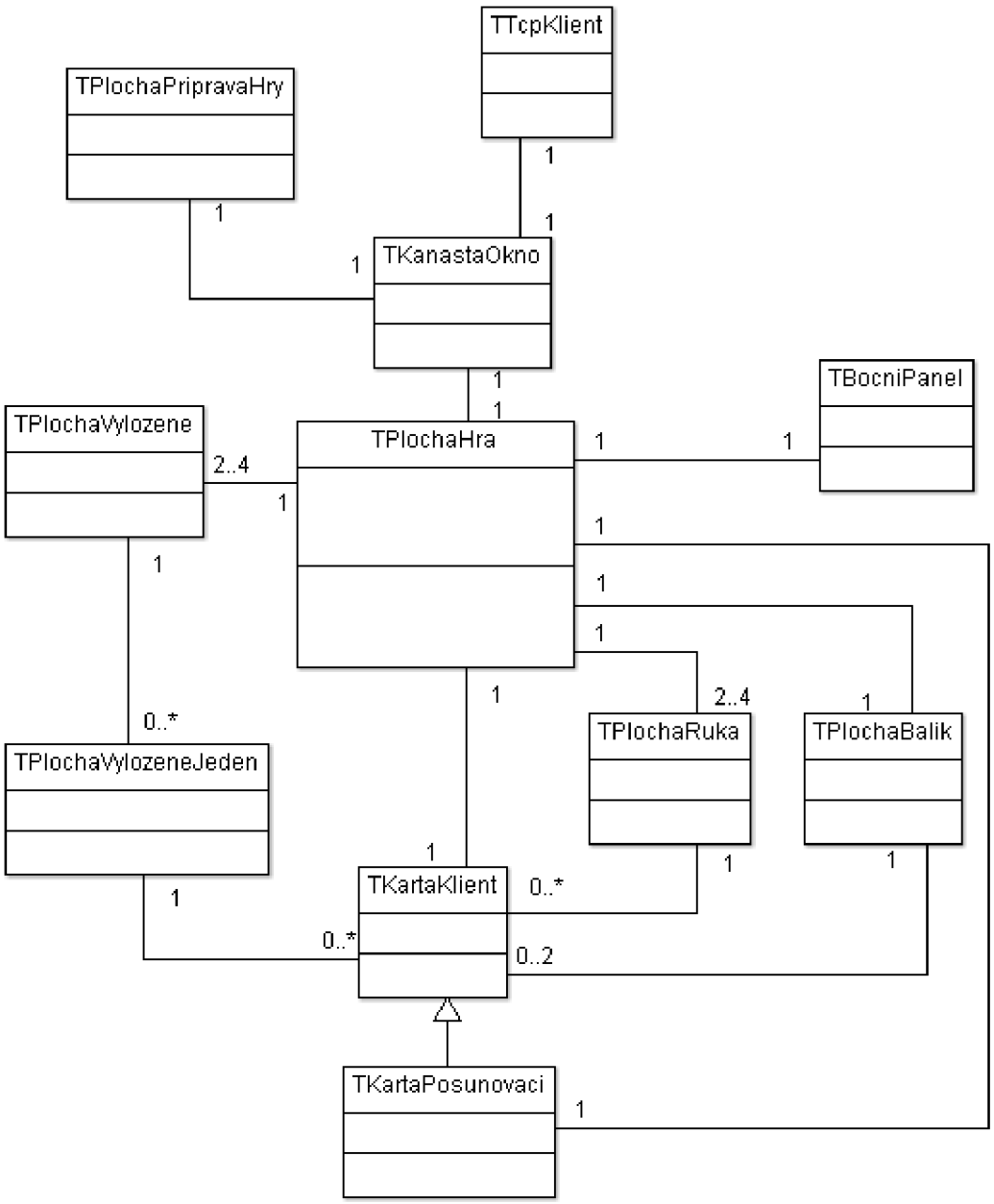
Spojení mezi třídami se nazývají vztahy. Existují čtyři typy vztahů: asociace, agregace, kompozice a generalizace. Asociace vyjadřuje ten nejobecnější vztah mezi objekty. Nejčastějším případem asociace je binární asociace, což je asociace mezi dvěma objekty zpravidla dvou různých tříd. Pokud jde o objekty stejné třídy, nazývá se asociace reflexivní. Agregace a kompozice jsou speciální typy asociace. V diagramu tříd vztah generalizace znamená, že třída specifičtější dědí od té obecnější její metody, tedy získává její rozhraní. Součástí asociace někdy bývá i její jméno a údaj o násobnosti. Násobnost určuje, kolik instancí jedné třídy může být ve vztahu reprezentovaném danou asociací s jednou instancí druhé třídy.

Koncepční diagram tříd je taková ochuzená varianta kompletního diagramu tříd. Koncepční diagram tříd neobsahuje atributy ani metody daných tříd. Jde v něm především o zobrazení existujících tříd a vazeb, které se vyskytují mezi objekty jednotlivých tříd, ne o to, jaké má třída rozhraní.

Na obrázcích 3.3 a 3.4 se nachází diagramy tříd aplikace Kanasta, serverové části, resp. klientské části. Jedná se o koncepční diagramy tříd, protože každá třída obsahuje poměrně velké množství atributů a metod a uvedení těchto atributů a metod by mělo negativní vliv na přehlednost těchto diagramů. Atributy a metody jednotlivých tříd jsou k vidění ve zdrojových kódech aplikace. V těchto diagramech jsou jen dva typy vztahů – asociace (obyčejná čára) a generalizace (čára s šipkou na konci). Diagramy jsou tudíž dost obecné, bez žádných detailních podrobností.



Obrázek 3.3 Diagram tříd aplikace Kanasta - část server

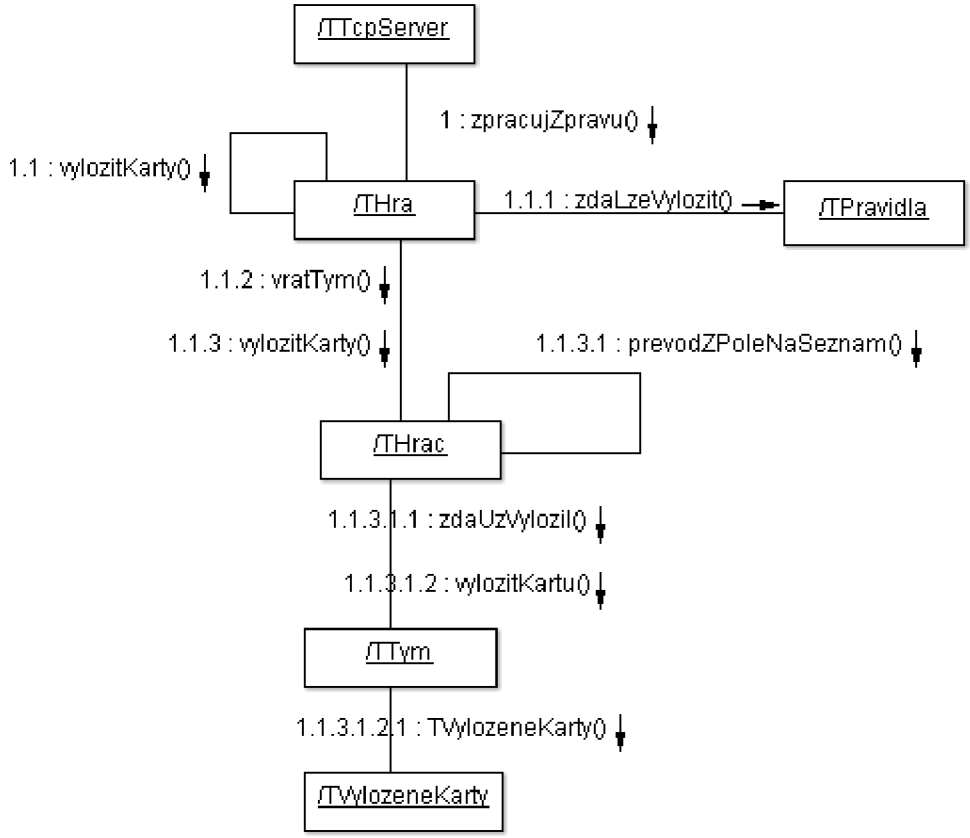


Obrázek 3.4 Diagram tříd aplikace Kanasta - klientská část

3.2.3 Diagram komunikace

Dalším z diagramů jazyka UML, ale už ne tak významným, je diagram komunikace (commucation diagram). Diagram komunikace zdůrazňuje statickou strukturu (propojení tříd), která se využije při interakci k dosažení požadovaného chování. Takže úkolem návrhu tohoto diagramu je určit třídy a navrhnout komunikaci, která zajistí toto požadované chování. Je zde podstatné hierarchické číslování zpráv, které definuje pořadí a zanoření. Diagramy komunikace jsou velmi vhodné k prvotnímu rychlému načrtnutí komunikace mezi objekty jednotlivých tříd.

Na obrázku 3.5 se nachází diagram komunikace případu užití vyložení karet aplikace Kanasta. V diagramu je vyznačeno, jaké metody jakých tříd jsou volány za účelem tohoto konkrétního případu užití. Podrobnější popis tohoto případu užití se nachází v následující kapitole.



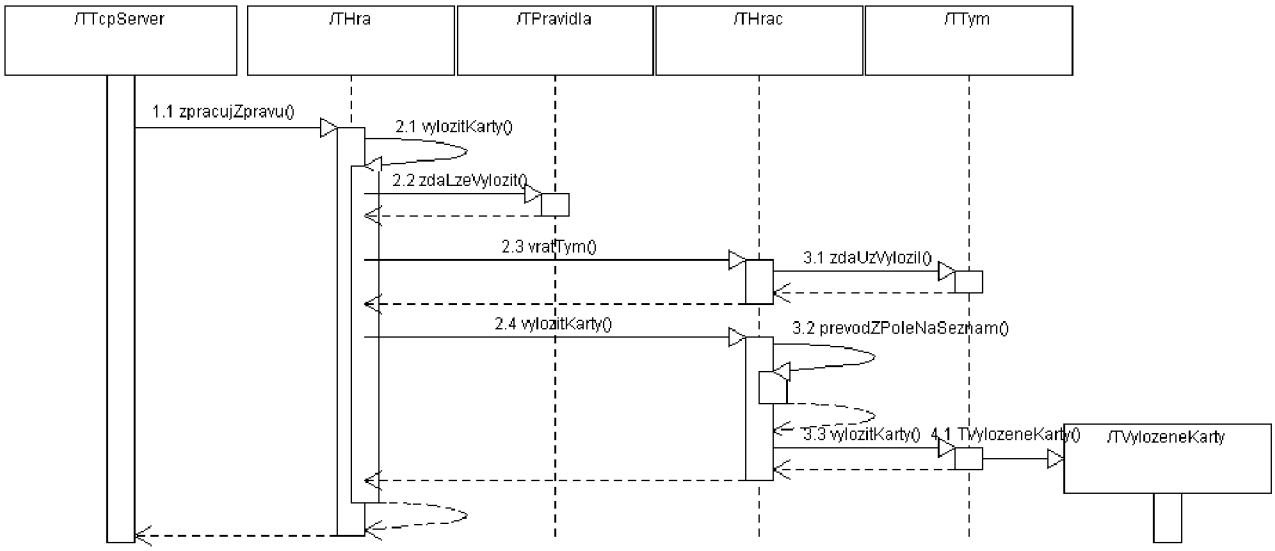
Obrázek 3.5 Diagram komunikace popisující případ užití vyložení karet aplikace Kanasta

3.2.4 Diagram sekvence

Sekvenční diagram (sequence diagram), který se taktéž označuje jako diagram sekvence, vizualizuje posloupnost zpráv zasílaných mezi účastníky interakce. Je to dvou dimenzionální diagram. První dimenze (zleva doprava) je dimenze účastníků interakce. Druhá dimenze (shora dolů) je čas. Zprávy reprezentují komunikaci mezi dvěma účastníky interakce, která může znamenat volání metody, vytvoření nebo zrušení instance nebo zaslání signálu.

Na obrázku 3.6 je k vidění sekvenční diagram popisující stejný případ užití, jaký je popsán v diagramu komunikace v kapitole 3.2.3. Z diagramu lze vyčíst, že objekt třídy `TTcpServer` nejdříve, poté, co dostane zprávu od klienta, volá metodu `zpracujZpravu()` objektu třídy `THra`. V této metodě je pak volána metoda téže objektu s názvem `vylozitKarty()`. V ní probíhá samotné vyložení. To je rozděleno do několika částí. V první části se zjišťuje, zda chce hráč vyložit takovou kombinaci karet, jaká je podle pravidel přípustná. V další části se získává informace, jestli hráč už vyložil. V tomto konkrétním případu užití se nejedná o první vyložení, proto se vynechává

kontrola počtu bodů za tyto karty. Tento případ užití pokračuje dále, až končí vytvořením instance třídy `TVylozeneKarty`.



Obrázek 3.6 Sekvenční diagram popisující případ užití vyložení karet aplikace Kanasta

3.2.5 Jak to celé funguje

Samotná hra je navržena jako klient/server aplikace, která je dostatečně popsána v kapitole 2.4. Pro to, aby hráči mohli spolu začít hrát hru, je potřeba nejdříve spustit serverovou aplikaci. Buď na počítači jednoho z hráčů, nebo úplně na jiném stroji. V rámci spuštění serveru je potřeba zadat číslo portu, na kterém server bude naslouchat. Každý hráč si poté spustí samotnou klientskou aplikaci, zadá své jméno, IP adresu počítače, na kterém server běží, a číslo portu, na kterém server naslouchá. Pokud hráč zadá správné údaje a pokud nemá problémy s připojením, ať už do lokální sítě nebo na Internet, mělo by se mu podařit se úspěšně připojit. Po úspěšném připojení se objeví okno s nastavením samotné hry. Hráči se v této části pomocí chatovacího okna domluví, kdo bude jejich spoluhráč a kdo protihráči, a patřičně si to nastaví. Samotné pravidla hry může upravovat pouze správce (admin), kterým se stane hráč, který se k serveru připojil jako první. Pravomocí správce je kromě nastavení pravidel i nastavení počtu hráčů ve hře. Pokud se hráč, který zastává funkci správce, odpojí, stává se správce hráč s nejnižším číslem. Poté, co je hráč připraven vstoupit do hry, nastaví, že je připraven. Pokud jsou všichni hráči připraveni, spustí se samotná hra.

Hráči se následně střídají ve hře ve směru hodinových ručiček, a posílají na server zprávy prostřednictvím svých příkazů. Příkazy jsou generovány na základě ovládání hry hráči. Hráči ovládají hru pomocí myši. Server tyto příkazy zpracuje a výsledek odešle všem připojeným klientům, kteří jej zobrazí. Pokud hráč není na řadě, je mu pouze dovoleno chatovat si s ostatními hráči prostřednictvím chatovacího okna, které je opět součástí okna celé aplikace. Hráči je vymezena doba 60 sekund na to, aby provedl celý svůj tah, tj. vzal kartu, případně vyložil nebo doložil karty a odhodil kartu. Jde o záměrný cíl zamezit hráčům zdržovat hru. Pokud vyprší hráčovi jeho časový limit, je za předpokladu,

že si kartu ještě nevzal, odebrána karta z kmenového balíku. A dále je odhozena na odkládací balík právě ta karta, která je v jeho ruce co nejvíc vlevo. Poté je na řadě standardně opět hráč po jeho levici. Pokud se hráč odpojí ze hry, celá hra je okamžitě ukončena, protože nemá smysl pokračovat uprostřed rozehrané hry s jiným počtem hráčů. Po takto ukončené hře jsou ostatní hráči nedobrovolně odpojeni.

Po každém kole se objeví tabulka se souhrnem kola, kde jsou uvedeny informace o získaných a ztracených bodech za hodnoty vyložených karet a karet v ruce a za prémie. Po konci hry, jsou zveřejněni vítězové a jejich dosažený počet bodů.

3.2.6 Návrh serveru

Tou základní třídou serverové části je třída `TTcpServer`. Ta má za úkol zajišťovat samotnou komunikaci s klientem (přijímání a odesílání zpráv). Další dvě neméně důležité třídy jsou `TPripravaHry` a `THra`. Instance třídy `TPripravaHry` se vytvoří vždy, když server čeká na připojení hráčů, tedy když je ve fázi přípravy hry. Tato třída umožňuje hráčům se domlouvat přes chat, nastavit pravidla hry, nastavit počet hráčů a určit, kteří hráči budou spoluhráči v dané hře (pokud se jedná o hru pro čtyři hráče). Poté, co jsou všichni hráči připraveni ke hře, instance třídy `TPripravaHry` zaniká a vytváří se instance třídy `THra`. V ní dochází k inicializaci hodnot důležitých pro samotnou hru a k přeskupení hráčů tak, aby spoluhráči hráli naproti sobě. Vytvoří se jen právě takový počet týmů a hráčů, jaký je požadován. V čase, kdy probíhá samotná hra, jsou zprávy zpracovávány právě metodami třídy `THra`. Po skončení celé hry zaniká instance třídy `THra` a vytváří se opět instance třídy `TPripravaHry`, která je opět připravena zpracovávat zprávy týkající se přípravy další hry.

Pro všechny důležité objekty reálné karetní hry Kanasta jsou navrženy odpovídající třídy. Mezi tyto reálné objekty patří například karta, hráč, balík nebo tým. Kupříkladu objekt z reálného světa hráč je reprezentován třídou `THrac`. Každá z těchto tříd obsahuje metody, které odrážejí skutečné operace, jenž může reálný objekt provádět. Již zmíněný hráč může například vzít kartu, může odhodit kartu nebo vyložit karty. Každé z těchto operací je navržena odpovídající metoda.

Každá posloupnost po sobě jdoucích karet, které jsou obsaženy v balíku, u hráče v ruce nebo v řadě vyložených karet, je reprezentována třídou `TVirtuálníBalík`. Od ní pak třídy reprezentující balík (`TBalik`), kmenový balík (`TKmenovyBalik`), hráče (`THrac`) a vyložené karty (`TVylozeneKarty`) dědí.

U operací, u kterých je to vyžadováno, jako například vyložení nebo doložení karet, se kontroluje dodržování pravidel. Za tímto účelem byla navržena třída `TPravidla` a její metody pro kontrolu hodnot karet, pro kontrolu toho, zda je za karty dostatečný počet bodů, nebo pro kontrolu, zda jsou všechny karty originální, atd.

Pro odeslání zpráv jsou navrženy dvě metody. Jedna pro odeslání zprávy jednomu konkrétnímu hráči. Tato metoda se využívá hlavně, když server posílá zprávu s informací, která sama o sobě nemá vliv na posun ve stavu hry. Jde o varovné zprávy, které informují hráče, když se pokouší o něco, co pravidla nepovolují. Například když chce hráč vzít balík před třetím tahem nebo když chce uzavřít kolo a přitom nemá kanastu. Druhá metoda pak klasicky odesílá zprávu všem hráčům. Jde o situace, kdy se samotná hra vyvíjí. Pak už je na klientovi, aby z informací získaných ze zprávy, správně interpretoval herní akce. Například pokud si hráč vezme kartu z balíku, server odešle všem klientům stejnou zprávu. Na klientovi je, aby ze zprávy vyčetl, o kterého hráče se jedná, a podle toho se zachoval (zobrazí kartu lícem nebo rubem vzhůru).

Po vizuální stránce samotné okno pouze vypisuje informace, v jaký čas a co provedli připojení hráči za akce. Lze si tedy pak zpětně prohlédnout, jakou posloupnost kroků hráči vykonávali.

3.2.7 Návrh klienta

Druhá část, ta, s kterou se obvykle hráč dostane do kontaktu, je klientská aplikace. U ní je kladen důraz především na uživatelské rozhraní. Byla snaha o intuitivní ovládání, které mají uživatelé zažité. Hlavní okno aplikace zobrazuje dvě plochy. Tou první je plocha obsahující položky pro nastavení hry, tou druhou je samotná hrací plocha.

Třídy pro reprezentaci obou ploch mají název `TPlochaPripravaHry` a `TPlochaHra`. Obě třídy obsahují prvky uživatelského rozhraní. Plocha pro samotnou hru je rozdělena do několika logických částí (viz obrázek 3.7). Každá z těchto částí je instance třídy, která byla pro tento účel navržena. Mezi tyto třídy patří:

- třída `TPlochaRuka`, která představuje tu část plochy, kde má hráč rozprostřené své karty a vyložené červené trojky,
- třída `TPlochaVylozene`, která představuje tu oblast plochy, jenž obsahuje hráčovy vyložené karty,
- třída `TPlochaBalik` reprezentující plochu, kam hráči odkládají karty,
- třída `TKartaKlient` představující plochu, která symbolizuje kmenový balík a konečně
- třída `TBocniPanel`, což je plocha zobrazující chatovací okno a informace o počtech bodů jednotlivých týmů v různých fázích hry, informace o aktuálních kolech nebo o času, který hráčovi zbývá v aktuálním tahu.

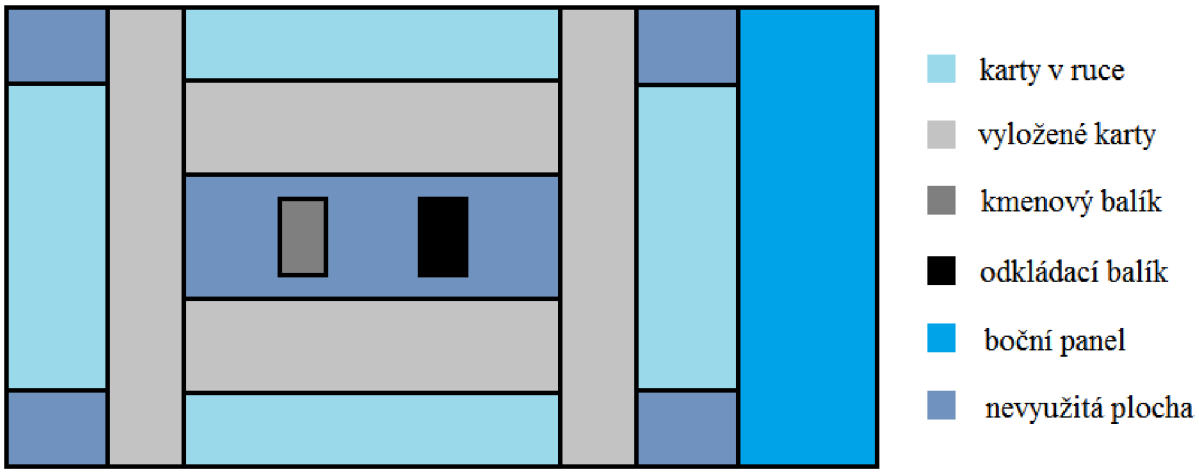
Zobrazené karty jsou objekty třídy `TKartaKlient`. S těmito objekty poté pracují jednotlivé části hrací plochy. Nefunguje to tak, že by instance všech karet byly vytvořeny najednou na začátku hry a poté by se jen přesouvaly, nýbrž jsou jednotlivé instance vytvářeny a rušeny právě v čase, kdy jsou aktuálně potřeba. Kdykoli si například hráč vezme kartu z balíku, je vytvořena nová instance

třídy `TKartaKlient`, a kdykoli se hráč karty zbaví, je odpovídající instance z objektu třídy `TPlochaRuka` odstraněna a naopak vytvořena v objektu třídy `TPlochaBalik`.

Posílání a přijímání zpráv na a ze serveru zajišťuje instance třídy `TTcpKlient`. O samotné zpracování a generování zpráv se starají už dříve zmíněné dvě hlavní instance tříd `TPlochaPripravaHry` a `TPlochaHra` (která z nich záleží na tom, zda jde o fázi přípravy nebo hraní).

Samotnou logiku hry sice obstarává server, ale proto, aby se příliš nezatěžovala síť, jsou některé triviální věci hlídány a kontrolovány na straně klienta. Jde především o snahu dosáhnout toho, aby hráč neprováděl akce, když není na řadě, nebo si nevzal více karet z balíku. Bylo by zbytečné tímto zatěžovat server, když na samotnou hru to nemá vliv.

Časovač je spuštěn na straně klienta, server o něm nemá vůbec tušení. Spouští se vždy po rozdání karet před začátkem kola a restartuje vždy po odhození karty jednotlivými hráči.



Obrázek 3.7 Rozdělení hrací plochy na logické části v klientské části aplikace Kanasta

4 Implementace

Po fázi návrhu přichází na řadu fáze implementace. Jedná se o programovou realizaci, o převedení navržených částí do určitého programovacího jazyka. Aplikace Kanasta je implementována v programovacím jazyce C++ s využitím vývojového prostředí Qt.

Jazyk C++ je jeden z nejrozšířenějších programovacích jazyků. Jde o výkonný, ucelený, rychlý a přenositelný jazyk. Vychází ze strukturálního jazyka C, který rozšiřuje o objektově orientované programování. Jazyk vyvinul Bjarne Stroustrup v Bell Labs na začátku 80. let. Detaily ohledně jazyka C++ byly při implementaci čerpány z [7] a [8].

Qt je multiplatformní framework, který mimo jiné podporuje tvorbu uživatelského rozhraní. Hlavním programovacím jazykem tohoto frameworku je C++, nicméně existuje podpora i pro jiné jazyky (Java, C#, Python, Ruby). Qt byl vyvíjen společností Trolltech, než byl prodán v roce 2008 Nokii. Framework je uvolněn pod licencí LGPL.

Nástroj pro psaní zdrojového kódu se nazývá Qt Creator. Prvky uživatelského rozhraní lze vytvořit ručně nebo pomocí nástroje Qt Designer. Tento nástroj umožňuje jednoduché přetahování ovládacích prvků do okna metodou drag&drop. Qt Designer ukládá informace o ovládacích prvcích a jejich rozloženích v okně do XML souboru. Při vývoji aplikace nebyl nástroj Qt Designer použit.

Qt navíc používá speciální typ komunikace mezi objekty – signály a sloty. Aby bylo možné tento mechanismus komunikace použít, je nutné do definice třídy vložit makro `Q_OBJECT`. Signál slouží k upozornění ostatních objektů, že došlo k nějaké události. Slot je metoda, která se provede po vyvolání signálu. Ke svázání signálu se slotem slouží statická funkce `connect`. Obsahuje 4 parametry: objekt vysílající signál, název signálu, objekt přijímající signál a slot reagující na signál.

Všechny Qt třídy začínají na písmeno Q. Informace o jednotlivých Qt třídách byly čepány z [9].

4.1 Server

Protože kód samotného serveru obsahuje přes 3000 řádků, nelze popsat úplně vše. V této kapitole jsou proto popsány jen některé stěžejní části. Zbytek je k prozkoumání ve zdrojových kódech.

4.1.1 Reprezentace karet a míchání karet

Hrací karty jsou instance třídy `TKarta`. Je vytvořeno statické pole s názvem `noveKarty` o 108 prvcích této třídy `TKarta`. Každá karta má tedy jedinečný index v tomto poli. Každá karta má atributy s informací o hodnotě, barvě a počtu bodů za kartu. Jak už bylo popsáno v kapitole 3, posloupnost karet je reprezentována třídou `TVirtualniBalik`, ze které jiné třídy dědí. V této třídě

`TVirtualniBalik` jsou karty reprezentovány dynamickým seznamem. Uzly tohoto dynamického seznamu jsou instance třídy `TSekvenceKaret`, jejíž atributy jsou index do statického pole `noveKarty`, který definuje, o jakou kartu se jedná, a ukazatel na další uzel. Třída `TVirtualniBalik` pak obsahuje ukazatel na první prvek této sekvence karet. Poslední karta tohoto dynamického seznamu neukazuje nikam (ukazatel ukazuje na hodnotu `NULL`). Díky reprezentaci posloupnosti karet dynamickým seznamem, lze velice snadno přidávat a ubírat další karty. Pouze se jinak provážou ukazatele. Provádění operací nad seznamem je rychlejší než provádění těchto operací nad polem. Není nutná neustálá alokace a dealokace adresních míst nebo neustálé přesouvání prvků v rámci jednoho pole.

Na začátku celé hry se vytvoří uzly všech karet a do nich jsou náhodně pomocí funkce `rand()` vloženy indexy odpovídající indexům statického pole `noveKarty`. Po skončení hry jsou všechny uzly vymazány. Takto si objekty pracující s kartami pouze předávají ukazatele. Alokace a dealokaci proběhne pouze jednou v rámci jedné hry.

Seznam 108 uzlů (v náhodném pořadí) je tedy na začátku hry vložen do objektu `kmenovyBalik` třídy `TKmenovyBalik`. Aby míchání karet po každém kole odpovídalo skutečnému míchání, neprovádí se stejný mechanismus jako na začátku hry, kdy jsou indexy náhodně vloženy do uzlů seznamu. Míchání funguje tak, že jednotlivé posloupnosti karet (vyložené karty, karty v ruce) jsou po ukončení kola vloženy za sebe, tím utvoří opět celistvou hromádku, kde jsou karty za sebou tak, jak byly použity ve hře. Poté přichází na řadu samotné míchání. Jedním zamícháním je myšleno to, že se odebírá určitý počet karet z vrchu balíku a vkládá se do spodu balíku do té doby, než jsou takto přemístěny všechny karty. Nejprve se určí, kolik karet lze maximálně odebrat jako skupinu v rámci jednoho zamíchání. Jde o hodnotu *max* z intervalu <10, 25). Poté je postupně odebráno náhodně 1 až *max* karet, dokud nejsou všechny karty přemístěny. Takto je náhodně provedeno 1 až 10 zamíchání.

4.1.2 Reprezentace hráčů

Hráč ve fázi příprava hry je reprezentován třídou `TPotencialniHrac`. Při startu fáze přípravy hry se vytvoří pole 4 prvků pro uchování ukazatelů na objekty této třídy. Tato třída uchovává informace o socketu, přes který je hráč připojen, hráčovo dočasné číslo a číslo týmu, za který chce hráč hrát. Instance této třídy je vytvořena, kdykoli se hráč připojí, a zrušena, kdykoli se odpojí. Tito hráči jsou označováni jako potenciální, protože se můžou odpojit a vůbec se tak nezapojit do hry. Pokud jsou připojeni 4 hráči, server se uzavře a přestane naslouchat požadavkům od jiných klientů o připojení.

Při spuštění fáze hraní hry se vytvoří objekty třídy `THrac` (která dědí z `TPotencialniHrac`) reprezentující hráče ve hře, a objekty třídy `TTym`, reprezentující jednotlivé týmy, do kterých budou jednotliví hráči patřit. Tyto objekty se vytvoří všechny najednou. Postupuje se následovně: Prochází se jednotliví potenciální hráči, dokud se nenarazí na prvního, který chce hrát

za tým číslo 0. Poté, co na něj server narazí, je vytvořen objekt prvního hráče s jeho příslušnými hodnotami (socketem, jménem), ukazatelem na jeho nový tým (tým číslo 0) a navíc jeho novým číslem 0, které je od tohoto okamžiku jeho fixní číslo ve hře. Poté je objekt třídy `TPotencialniHrac` vymazán. Celé to pokračuje znovu s tím, že se tentokrát hledá hráč, který chce hrát za tým číslo 1, a jemu je pak přiděleno nové číslo 1, až jsou tímto postupem, který se opakuje, vytvořeni všichni hráči a zrušeni všichni potenciální hráči.

4.1.3 Vyložení karet

Pravidla hry kontrolují metody třídy `TPravidla`. Protože se jedná o statické metody, nemusí se vytvářet instance této třídy. Kontrola dodržování pravidel je jedna z nejdůležitějších věcí celé aplikace. Kontrolu vyložení karet má na starost metoda `TPravidla::zdaLzeVylozit(const TPole & kartyNaVylozeni, THrac *hrac, QString & duvod, bool budeBratBalik)`. V ní se nejprve zkontroluje, zda hráčovi svým vyložením zůstaly nějaké karty v ruce. Pokud ne, není mu vyložení povoleno, protože by neměl jak uzavřít hru. Následně se zjistí, zda jde alespoň o 3 karty. Poté se prozkoumá, zda jde o povolenou kombinaci karet. Postupně se prochází pole s indexy karet `kartyNaVylozeni` a hledá se první originální karta. Pokud jsou všechny karty divoké, lze vyložit bez problémů. Pokud se najde alespoň jedna originální karta, tzn., že hráč nechce vyložit samé divoké karty, uloží se její hodnota a barva jako výchozí. Pokračuje se v prohledávání pole, dokud se nenajde druhá originální karta. Po jejím nalezení se porovnají hodnoty obou originálních karet, pokud sedí, předpokládá se, že jde o vyložení skupiny karet, kde mají všechny stejnou hodnotu (v aplikaci se tato posloupnost karet nazývá *Skupina*). Pokud nesedí, porovnají se barvy obou karet, pokud souhlasí, předpokládá se, že jde o sekvenci karet, tedy o posloupnost po sobě jdoucích karet stejné barvy (v aplikaci se tato posloupnost karet nazývá *Sekvence*). Pokud ani barvy nesedí, není povoleno vyložit karty. Pole indexů karet se projde až do konce a přitom se zkoumá hodnota u *Skupin* a hodnota s barvou u *Sekvencí*. Aby se u *Sekvencí* dobře kontrolovalo, že hodnoty karet jdou skutečně po sobě, bylo nutné pole nejdříve uspořádat. Uspořádání se provedlo ještě před voláním metody `zdaLzeVylozit(...)`. Po celou dobu procházení pole se počítal počet originálních karet, zda jich není méně než 2. Ke konci metody se ještě kontroluje pár skutečností. Jednak to, zda si hráči v přípravě hry nezadali, že nechtějí povolit vykládání sekvencí po sobě jdoucích hodnot karet. Pokud si zvolili nepovolování vykládání sekvencí, vyložit se opět nepodaří. Poté si aplikace hlídá i to, zda hráč chce vyložit černé trojky. Pokud ano, musí mít v ruce pak už jen právě jednu kartu akorát na uzavření kola. Nakonec, pokud jsou všechny předchozí kontroly úspěšné, metoda zjistí, zda by hráči po vyložení zůstala akorát jedna karta na uzavření kola. Pokud ano, tak se metoda podívá hráčovi do týmu, zda už získal kanastu. Pokud kanastu tým ještě nevyložil a nestane se tak ani při aktuálním vyložení a zároveň pokud nevykládá kartu odebranou z odkládacího balíku (tudíž mu zůstane právě ta jedna karta v ruce), není mu opět dovoleno vyložit.

Další metoda, která souvisí s dodržováním pravidel ohledně vyložení, se jmenuje `TPravidla::triKartyOriginalni(const TPole & kartyNaVylozeni)`. Jak už název napovídá, tato metoda projde celé pole `kartyNaVylozeni` a spočítá počet originálních karet.

`TPravidla::jeDostBodu(TPole *vicePoli[], int pocet, THrac *hrac, QString & duvod)` má zase na starost spočítat celkovou hodnotu bodů karet. První parametr obsahuje pole polí indexů karet.

Pokud se jedná o první vyložení, většinou podmínka o počtu bodů není splněna hned první vyloženou skupinou karet. V takovém případě se pole indexů karet uloží do pole ukazatelů na takováto jednotlivá pole (první parametr metody `jeDostBodu(...)`) a jsou v něm uloženy do doby, než se provede vyložení všech těchto polí, nebo než hráč vyložení zruší.

Jak už bylo popsáno dříve, každá posloupnost karet je reprezentována dynamickým seznamem. To platí i u vyložených karet. Každý tým pak obsahuje pole těchto seznamů s vyloženými kartami. Toto pole je realizováno pomocí šablony `vector`.

Po všech kontrolách se provede samotné vyložení. Nejprve se pole indexů karet převede na dynamický seznam. Poté se vytvoří instance třídy `TVylozeneKarty`, které se vloží ukazatel na právě vytvořený seznam karet.

4.1.4 Síťová komunikace

Pro přijímání žádostí o připojení a reprezentaci samotných funkcí serveru se používá Qt třída `QTcpServer`. Vytvořením objektu této třídy, se zároveň vytvoří socket, na který budou klienti posílat žádosti o připojení. Socket je název pro místo, kterým putují zprávy mezi aplikacemi. V samotné aplikaci se objekt třídy `QTcpServer` jmenuje `server`. Metodou `setMaxPendingConnections(int)` se určí, kolik maximálně připojení může čekat ve frontě na zpracování. Metoda `listen(QHostAddress, int)` spustí naslouchání serveru na dané adrese a portu. Signál `newConnection()` se spojí s odpovídajícím slotem. Tento signál je vyvolán kdykoli přijde žádost o připojení od klienta.

```
server = new QTcpServer(this);
server->setProxy(QNetworkProxy::NoProxy);
server->setMaxPendingConnections(MAXHRACU);
if(!server->listen(QHostAddress::Any, port)){
… //tento úsek kódu se provede, pokud server není schopen naslouchat
}
connect(server, SIGNAL(newConnection()), this, SLOT(pripojitHrace()));
```

Kdykoli přijde žádost o připojení ze strany klienta, je tato žádost zařazena do fronty a poté je zavolána metoda `pripojitHrace()`. V této metodě je volána metoda třídy `QTcpServer` s názvem `nextPendingConnection()`. Ta vrací číslo socketu (jde o objekt třídy `QTcpSocket`), přes který bude server komunikovat s daným klientem žádajícím o připojení. Signály těchto socketů

se opět provážou s odpovídajícími sloty. Signál `readyRead()` je vyvolán vždy, pokud je na socketu nová příchozí zpráva, a signál `disconnected()` je vyvolán, pokud je vazba mezi tímto socketem na straně serveru a socketem na straně klienta přerušena.

```
QTcpSocket *socket = server->nextPendingConnection();
connect(socket, SIGNAL(readyRead()), this,
            SLOT(zpracovaniPozadavkuVPripraveHry()));
connect(socket, SIGNAL(disconnected()), this, SLOT(odpojeniJednoho()));
```

Server má postupně několik socketů. Ten, na který přichází požadavky na připojení (dokud není uzavřen z důvodu připojení maximálního počtu hráčů), a ty, pomocí nichž komunikuje s klienty. Aby zjistil, z kterého socketu dorazila zpráva, použije příkaz:

```
QTcpSocket *socketSDaty = qobject_cast<QTcpSocket *>(this->sender());
```

Poté použije metodu `readAll()`, která přečte všechna příchozí data. Ty jsou následně uloženy do řetězce, s kterým pak pracují metody na zpracování příchozí zprávy.

Třídy Qt pro síťovou komunikaci, řeší konkurentnost ve vlastní režii, není tudíž nutné se tím zabývat (žádná dodatečná implementace vlastních neblokujících socketů, funkcí zajišťujících multiplexing, vláken a signálů).

Zprávy server odesílá tak, že prochází všechny sockety, které jsou propojené s klienty, a volá jejich metodu `write(const QByteArray)`.

4.1.5 Odebrání balíku

Pokud se hráč pokusí si vzít kartu z odkládacího balíku, je zavolána statická metoda `TPravidla::zdaLzeVzitBalik(TBalik *balik, int tah, QString & duvod)`. V ní se kontroluje, zda jsou dodržena pravidla spojená s odebráním karty z balíku. Takováto pravidla jsou dvě: První se podívá do parametru `tah`, který říká, o kolikátý tah kola se jedná, protože se karta z balíku nemůže vzít první tři tahy. Druhé pravidlo pojednává o tom, že nelze vzít kartu z balíku, pokud jde o trojku nebo divokou kartu. Podívá se tedy na index první karty objektu `balik`, který je prvním parametrem této metody.

Pokud jsou dodržena pravidla, uloží se do objektu `kartaZBaliku` třídy `TKartaZBaliku` index této odebrané karty a nastaví se příznak tohoto objektu, že karta z balíku byla odebrána. Kdykoli přijde žádost o vyložení karet a je nastaven příznak odebrání karty z balíku, kontroluje se, zda, pokud je balík zmrazený, hráč vykládá alespoň tři originální karty. Pokud je vše v pořádku, provede se vyložení a zároveň odebrání všech karet z balíku. Je zavolána metoda `vzitBalik(TBalik *balik)` třídy `THrac`. Tím se seznam karet v balíku připojí k seznamu hráčových karet a vrátí se pole indexů těchto karet, které jsou následně převedeny do zprávy, jenž bude odeslána klientovi. Příznak, že byla karta z balíku odebrána, se nastaví na `false`.

4.1.6 Počítání bodů

Každá instance třídy `TKarta` má atribut `body`, který říká kolik je za danou kartu bodů. Body jsou počítány v průběhu hry, aby tým věděl, jak na tom průběžně je. Kdykoli tým vyloží karty, vytvoří se nová instance třídy `TVylozeneKarty`. V ní je volána metoda, která spočítá body vyložených karet. Hodnota bodů se uloží do právě vzniklé instance. Tým má taky atribut `body`, reprezentující počet bodů za všechny skupiny vyložených karet. K tomuto atributu `body` se přičtou body uložené v nově vzniklé instanci vyložených karet. Nepočítají se tudíž body úplně všech vyložených karet znovu. Celkový počet bodů za vyložené karty je odeslán klientům.

Při doložení se nejprve od atributu `body` objektu reprezentující daný tým odečtou body té instance karet, ke které se dokládají další karty, aby se zjistil počet bodů týmu bez této skupiny karet. Poté proběhne doložení karet. Přepočítají se body této instance vyložených karet a opět se přičtou k atributu `body`. Tentokráte skupinka obsahuje už i doložené karty. Body žádné karty tudíž nebudou připočteny dvakrát.

Body za kanastu jsou taky počítány průběžně. Pokud při vyložení nebo doložení utvoří některá skupinka karet kanastu, je do atributu `bodyZaKanasty` objektu reprezentující tým vložen odpovídající počet bodů. Děje se tak prostřednictvím metody `jeKanasta()` třídy `TVylozeneKarty`. Tato metoda vrací počet bodů za případnou kanastu, ale také říká, zda je kanasta vytvořena. Protože za neexistenci kanasty vrací 0 bodů, tedy `false`, jinak vrací `true`.

Všechny ostatní body jsou vypočítány až na konci kola.

4.1.7 Řazení karet

Řadit karty je nutné na straně serveru z toho důvodu, že metody kontrolující dodržování pravidel, jsou navrženy tak, že už počítají se seřazeným polem, tedy hlavně část kontrolující sekvence karet s po sobě jdoucími hodnotami a stejnou barvou. Řazení karet probíhá tak, že se seřadí pole indexů. Nejnižší karty (dvojky) mají nejnižší index a nejvyšší karty (žolíky) mají nejvyšší index. Řazení provádí metoda třídy `TPole` s názvem `usporadat()` a má dvě fáze.

První fází je takový trošku upravený bubble sort. Úprava spočívá v tom, že se prvky vedle sebe vymění nejen, když je prvek vlevo větší než prvek vpravo, ale i pokud je vlevo divoká karta. Celé řazení skončí tak, že jsou karty seřazeny, jak mají, s výjimkou divokých karet, ty jsou všechny v poli úplně vpravo.

Druhá fáze řazení je z důvodu sekvencí. Jde o to, že divoká karta může nahradit libovolnou kartu v sekvenci. Například může být sekvence 7-8-2-10, kde dvojka plní roli devítky. Po první fázi řazení je dvojka nejvíc vpravo. Druhá fáze právě posouvá divoké karty směrem doleva na místo, kde by mohli teoreticky být. Druhá fáze je implementována následovně:

```
for(int i = 0; i < pocet - 1; i++){
```

```
        if(noveKarty[pole[i+1]].jeDivoka())
            break;

        j = pocet - 1;    //proměnná j reprezentuje možnou divokou kartu

        //pokud dvě sousedící karty se liší hodnotou o více než jedna,
        //tak dojde k přesunutí divoké karty
       if(pole[i+1]/8-(noveKarty[pole[i]].jeDivoka()?zastupneID:pole[i])/8
        >= 2){

            //pokud není žádná divoká karta, metoda se ukončí, protože
            //nelze žádnou přesouvat
            if(!noveKarty[pole[j]].jeDivoka())
                return;

            //posunutí divoké karty na odpovídající místo
            while(j != i+1){
                pom = pole[j]; pole[j] = pole[j-1]; pole[j-1] = pom;
                j--;
            }

            //divoká karta reprezentuje ID karty, kterou zastupuje
            zastupneID = pole[i]+8;
       }
    }
```

Stejný algoritmus řazení, jako je použit v první fázi, tedy upravený bubble sort, je implementován i na straně klienta k uspořádání hráčových karet v ruce.

4.1.8 Červené trojky

Červené trojky mají ve hře Kanasta zvláštní postavení. Hráč tuto kartu vždy odloží před sebe lícem nahoru. Může tuto kartu použít pro zmrazení balíku nebo si ji ponechat do konce kola a získat za ni prémiové body. Pro reprezentaci červených trojek byla navržena třída `TCerveneTrojky`. Od této třídy pak dědí třída `THrac`, která reprezentuje samotného hráče.

Posloupnost červených trojek je opět tvořena dynamickým seznamem. Právě třída `TCerveneTrojky` obsahuje ukazatel na tento seznam. Dále obsahuje údaj o tom, kolik si hráč musí při nejbližším tažením karty z kmenového balíku vzít náhradních karet právě za červené trojky.

Kdykoli si hráč vezme kartu z kmenového balíku, kontroluje se, zda je to červená trojka. Tato informace je poslána klientovi. Pokud se jedná o červenou trojku, inkrementuje se atribut s počtem náhrad. Po rozdání karet je klientovi zaslán údaj, kolik hráč potřebuje náhrad za červené trojky. Klient to zjistí a podle toho se zařídí. Pokud je nějaká náhradní karta potřeba, opět pošle žádost na odebrání karty z kmenového balíku na server.

4.2 Klient

Základní třídou pro tvorbu uživatelského rozhraní, od které všechny ostatní třídy reprezentující ovládací prvky a samotné prvky uživatelského rozhraní dědí je v Qt frameworku třída `QWidget`.

V této aplikaci třídy pro reprezentaci částí hrací plochy dědí právě ze třídy `QWidget`. Všechny ovládací prvky uživatelského rozhraní se obecně nazývají widgety. Na straně klienta jsou využívány vlastnosti widgetů. Widgetům lze nastavit rodiče metodou `setParent(QWidget*)`. Rodičem může být jakýkoliv jiný widget, na kterém se potom jeho potomci zobrazují. Lze nastavit souřadnice metodou `move(int, int)`, na kterých se widget vyskytne (jde o relativní pozici vůči svému rodičovi). Dále aby bylo možné nastavit barvu nebo obrázek na pozadí, je potřeba nastavit na `true` metodu `setAutoFillBackground(bool)`. A aby widgety generovaly události pohybu myší, i když není stisknuté žádné tlačítko, je potřeba nastavit na `true` metodu `setMouse-Tracking(bool)`.

Jsou vytvořeny dva hlavní widgety, které se vkládají do hlavního okna aplikace: Widget `widgetPriprava`, což je instance třídy `TPlochaPripravaHry`, a widget `widgetHra`, což je instance třídy `TPlochaHra` (naznačeno už v kapitole 3.2.7). Oba dva widgety jsou vytvořeny po úspěšném připojení se k serveru. Každý z nich obsahuje další widgety (prvky uživatelského rozhraní), kterým je rodičem. Jak názvy napovídají, widget `widgetPriprava` je do hlavního okna vložen ve fázi přípravy hry, kde si hráči nastavují parametry hry, a widget `widgetHra` jej v hlavním okně pak nahradí po startu samotné hry.

Dále v této kapitole jsou popsány stěžejní věci části klient.

4.2.1 Karty

Každá karta má odpovídající vzhled. Bylo vyrobeno 54 obrázků, které reprezentují možný vzhled karty. Pro reprezentaci karty byla navržena třída `TKartaKlient`, která dědí z třídy `QWidget`. Bylo vytvořeno, podobně jako na straně serveru, pole `vzhled` v počtu 109 prvků (108 karet + 1 karta reprezentující obrázek rubu karet). Jde o pole řetězců, kde každý řetězec je název souboru, ve kterém je uložen obrázek dané karty. Každý obrázek karty z tohoto pole odpovídá totožné kartě na straně serveru. Když tedy dojde zpráva s indexy karet, budou indexy odpovídat stejným kartám, jak na straně serveru, tak i na straně klienta.

Při vytváření instance třídy `TKartaKlient` se použije ten název obrázku z pole `vzhled` odpovídající té určité kartě. Obrázek má stejnou velikost jako samotný widget a je nastaven jako jeho pozadí. Když je objekt reprezentující samotnou kartu zmenšen, zmenší se nejen widget, ale přenastaví se i rozměry samotného obrázku. Změna rozměrů obrázku se provádí jen v samotné aplikaci a na samotný soubor s obrázkem nemá žádný vliv.

Pro změnu rozměrů slouží metoda třídy `TKartaKlient zmenitRozmery(int x, int y, int uhel)`. Ta právě nastaví velikost jak widgetu, tak i obrázku. Navíc, pokud je zadán i parametr `uhel`, je obrázek i pootočen. Toho se využívá při zobrazení karet postranních hráčů.

Třída `TKartaKlient` obsahuje i položku `umisteni`, která říká, kde je karta přesně na ploše umístěna. Na základě této hodnoty má karta pokaždé jiné chování. Existuje devět možných

umístění karet z hlediska chování karet. Může být aktivní karta v ruce, aktivní červená trojka a aktivní vyložená karta. Tyto všechny karty jsou samotného hráče a může s nimi manipulovat (aktivní vyložená karta je i ta, kterou vyložil spoluhráč). Dále existuje pasivní karta v ruce, pasivní červená trojka a pasivní vyložená karta. Tyto karty jsou karty ostatních hráčů. Poslední tři typy karet jsou karta v kmenovém balíku, karta v balíku a karta z balíku v ruce.

Každý druh umístění karty reaguje na akce (kliknutí a najetí myši) uživatele jinak.

4.2.2 Posun karty

Když hráč vezme nebo odhodí kartu, lze vidět kartu pohybující se od balíku k hráčovi nebo naopak. K posouvání karty po hrací ploše je použit časovač. Po určitém časovém intervalu se karta posune ve svém směru o pár pixelů. Tím vznikne dojem plynulého pohybu karty. Časovač je implementován jako objekt třídy `QTimer`.

Důležité jsou tři parametry: O jaký balík se jedná, o jakého hráče se jedná a zda jde o posun z balíku k hráči, nebo naopak. Všechny tyto údaje jsou uloženy do objektu třídy `TKartaPosunovaci`, která dědí ze třídy `TKartaKlient`. Na základě těchto informací je objekt třídy `TKartaPosunovaci` vytvořen, posunut na startovní pozici a je mu určena cílová pozice. Spustí se časovač metodou `start()` a pokaždé, když uběhne 20 ms, je vyvolán signál `timeout()`, jenž spustí metodu, která zkontroluje, zda je karta v cíli. Pokud ano, je časovač zastaven metodou `stop()` a objekt reprezentující posunovací kartu je zrušen. Pokud ne, časovač je opět automaticky restartován.

Akce vzít a odhodit kartu mají dvě fáze: Začátek akce a konec akce. Každá z těchto dvou fází má svou vlastní metodu. Například při odhození karty dojde k těmto akcím:

1. Provede se metoda na odebrání karty hráčovi z ruky (fáze začátek akce) – odstraní se tedy objekt třídy `TKartaKlient`, reprezentující tu konkrétní kartu, z objektu třídy `TPlochaRuka`, který představuje hráčovy karty v ruce.
2. Vytvoří se objekt třídy `TKartaPosunovaci`, spustí se časovač a první metoda je opuštěna.
3. Po dokončení posunu se objekt třídy `TKartaPosunovaci` zruší a spustí se metoda (fáze konec akce), která dokončí odhození karty – tedy vytvoří objekt třídy `TKartaKlient` v objektu reprezentujícím odkládací balík.

Řeší se to dvěma fázemi proto, protože jinak by karta byla v odkládacím balíku ještě dřív, než by tam sama doputovala.

4.2.3 Události widgetů

Aby widgety reagovaly na události uživatelů, je nutné předefinovat některé speciální chráněné metody třídy `QWidget` ve vlastních třídách. Jsou to metody `mouseMoveEvent()`, která se

provede vždy, když je nad daným widgetem pohnuto myší, a `mouseReleaseEvent()`, která se provede, kdykoli je uvolněno některé z tlačítek myši.

Co za akci se provede u widgetu `TKartaKlient`, závisí na tom, kde je karta na hrací ploše umístěná (viz kapitola 4.2.1). Například po najetí myší na kartu vyloženou, se karta zvětší, naopak po najetí myší na kartu, kterou má hráč v ruce, se karta dá do popředí, aby byla vidět celá. Navrácení karet do původní pozice, tzn. zmenšit kartu nebo zařadit kartu zpět na svou pozici, aby šla vidět jen levá část karty, se vyvolá opět pomocí výše zmíněných metod.

U příkladu vyložených karet: Pokud se najede na vyložené karty myší, celý widget reprezentující tu danou skupinku vyložených karet se zvětší. Pokud se vzápětí najede myší na plochu obsahující vyložené karty (tzn., pokud se myší sjede z té skupinky vyložených karet), ty widgety, které byly zvětšeny, se zmenší; takže se zmenší právě ta jedna zvětšená skupinka vyložených karet.

U příkladu karet v ruce: Pokud se najede myší na kartu, zavolá se metoda `raise()`, která daný widget posune nad všechny ostatní widgety, tím se karta zobrazí celá. Pokud se myší sjede z dané karty na jinou kartu nebo na plochu obsahující hráčovy karty v ruce, projde se zleva seznam postupně všech karet v ruce a u každé se zavolá metoda `raise()`, takže se widgety postupně naskládají na sebe a předtím zvýrazněná karta se opět zařadí.

4.2.4 Vyložené karty

Oblast na hrací ploše určená pro vyložené karty je tvořena objekty třídy `TPlochaVylozene`. Tato třída opět dědí z `QWidget`. Tento widget pro vyložené karty má neměnnou pozici na hrací ploše.

Kdykoli hráč provede vyložení, vytvoří se instance třídy `TPlochaVylozeneJeden`. V této instanci jsou teprve vytvořeny objekty třídy `TKartaKlient` (reprezentující jednotlivé karty) a vloženy do objektu `seznamKaret` Qt třídy `QList`. `QList` je šablona, reprezentující seznam hodnot určitého datového typu. V tomto případě je to seznam ukazatelů na objekty třídy `TKartaKlient`.

Stejně jako třída `TPlochaVylozeneJeden` má svůj `QList` hodnot typu `TKartaKlient`, má i třída `TPlochaVylozene` svůj `QList` hodnot typu `TPlochaVylozeneJeden`. Poté co je instance třídy `TPlochaVylozeneJeden` naplněna všemi kartami, které hráč vyložil, je tato instance vložena do objektu `seznamVylozene` k ostatním vyloženým skupinkám karet.

Pokaždé, když je takto přidána nová skupinka vyložených karet, se přepočítají vzdálenosti mezi jednotlivými instancemi třídy `TPlochaVylozeneJeden` uložených v objektu `seznamVylozene`. Tím jsou zajištěny pořád stejné rozestupy.

Widget třídy `TPlochaVylozeneJeden` má přesně právě takovou velikost, aby se do něj vlezly všechny widgety třídy `TKartaKlient`.

Po dosažení kanasty se rozestupy mezi widgety třídy `TKartaKlient` zmenší, což vyvolá dojem, že jsou karty shrnuty na sebe.

Jednotlivé widgety jsou na sobě (resp. v sobě) naskládány. Úplně nejníže je widget reprezentující hrací plochu – tudíž instance třídy `TPlochaHra`, na něm pak widgety třídy `TPlochaVylozene`, na nich widgety třídy `TPlochaVylozeneJeden` a na nich, resp. v nich, jsou widgety třídy `TKartaKlient`. Po najetí myši na vyloženou kartu, tj. na widget třídy `TKartaKlient`, který má nastavené umístění, že jde o aktivní nebo pasivní vyloženou kartu, jsou postupně zvětšeny ty widgety, u kterých je to potřeba, tzn., u kterých by zvětšení karet mělo za následek, že by nebyly vidět dceřiné widgety celé. Vše je toho docíleno, tento kaskádový postup, pomocí signálů.

4.2.5 Zobrazení hráčů

Před startem samotné hry jsou informace o všech hráčích odeslány ze serveru na klienty. Jde o jméno hráče a o jeho staré číslo, tj. číslo, pod kterým vystupoval ve fázi přípravy hry. Hráči jsou v příchozí zprávě seřazeni podle svých nových čísel. Všechny jména si klient uloží, aby je pak mohl zobrazit na hrací ploše. Své staré číslo si klient pamatuje, takže ho porovná s příchozími čísly, aby zjistil, jaké je jeho nové číslo. Nové číslo si hráč uloží. Inicializuje se plocha pro karty v ruce a pro vyložené karty umístěná dole.

K číslu hráče se postupně přičítá číslo jedna, zbytek po dělení počtem hráčů pak řekne, jaké číslo má hráč postupně vlevo, nahoře a vpravo (pokud jde o hru čtyř hráčů). V těchto místech se inicializují příslušné plochy.

4.2.6 Síťová komunikace

Socket na straně klienta je reprezentován instancí třídy `QTcpSocket`. Pro připojení se k serveru se používá její metoda `connectToHost(const QHostAddress &address, int port)`, Jejími parametry jsou adresa a port serveru. Metoda `waitForConnected(int ms)` určí dobu, po kterou se bude klient snažit připojit k serveru.

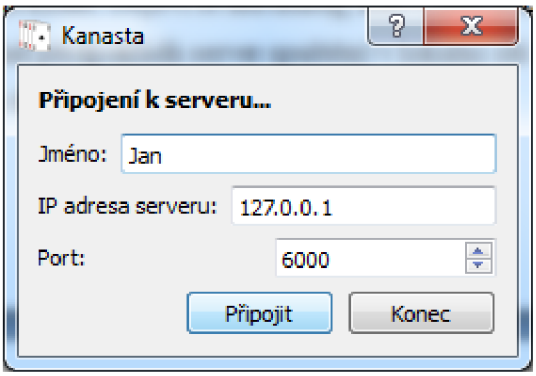
Pokud se podaří úspěšně připojit, je propojen pomocí funkce `connect()` signál socketu `readyRead()` se slotem, který obsahuje metodu `readAll()` pro přečtení, uložení a zpracování příchozích dat.

Pro odeslání dat na server slouží, podobně jako pro odesílání dat ze serveru na klienta, metoda třídy `QTcpServer write(const QByteArray)`.

Pro odpojení se ze serveru, je využita metoda téže třídy `disconnectFromHost()`. Po ní je socket uzavřen.

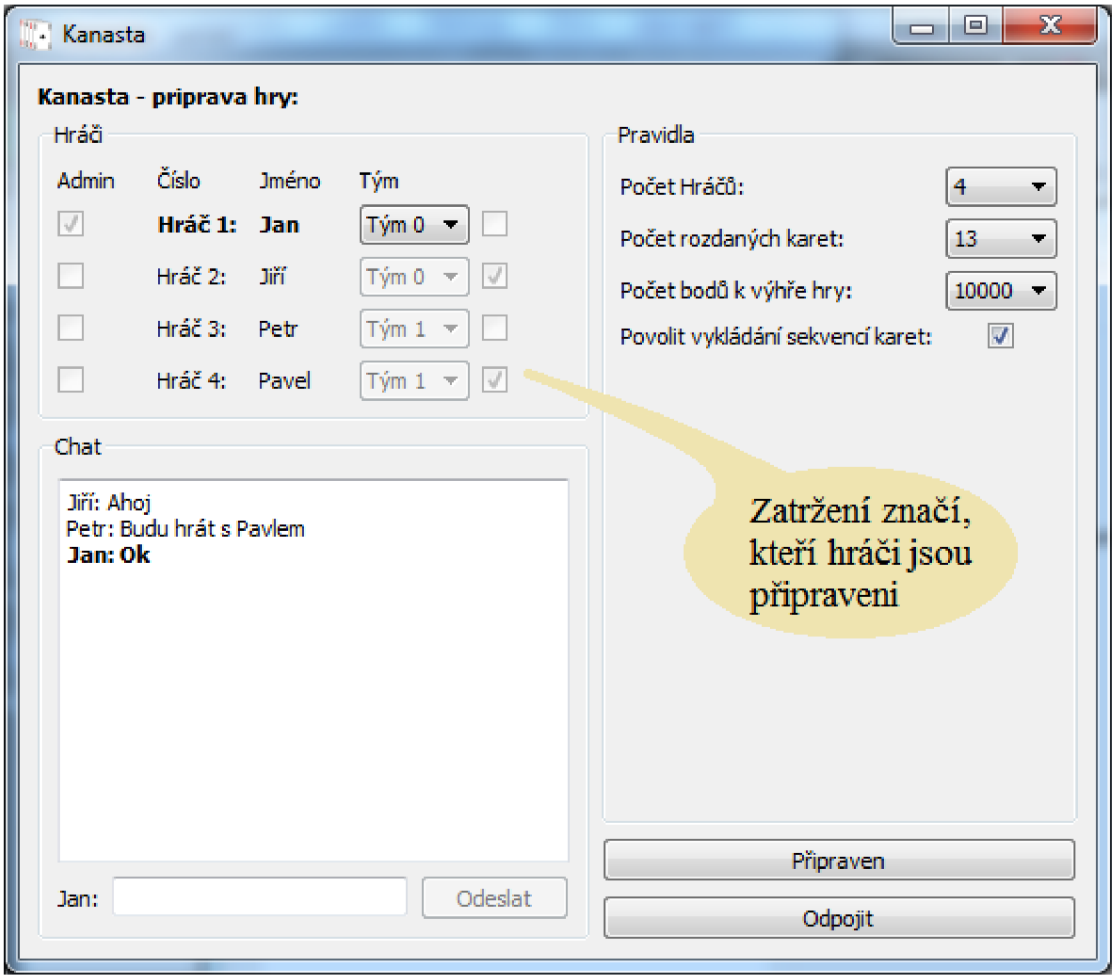
4.2.7 Ovládání hry

Hráč spustí aplikaci Kanasta-klient. Objeví se mu dialog, kde vyplní požadované údaje, jak lze vidět na obrázku 4.1. Tento příklad předpokládá server spuštěný v lokální síti (adresa 127.0.0.1) na portu 6000. Hráč poté klikne na tlačítko „Připojit“.



Obrázek 4.1 Okno pro připojení hráče k serveru

Objeví se okno s nastavením parametrů hry. Hráč si zvolí číslo týmu. Pokud je hráč zároveň správce (admin), nastaví pravidla hry. Až je připraven, klikne na tlačítko „Připraven“. Příklad se nachází na obrázku 4.2.



Obrázek 4.2 Okno s nastavením parametrů hry

Pokud jsou všichni hráči připraveni, spustí se samotná hra. Sám hráč je zobrazen dole, kde má odkryté i své karty. V ní pokud hráč na řadě, který si ještě nevzal kartu, klikne na kmenový balík (ta karta uprostřed hrací plochy s rubem vzhůru), vezme si kartu. Klikne-li na balík (karta uprostřed hrací plochy lícem vzhůru), vezme si kartu z balíku. Pokud hráč klikne na balík poté, co si už jednu kartu v aktuálním kole vzal, odhodí označenou kartu v ruce a kolo pro něj končí.

Kartu hráč označí kliknutím na kartu v ruce. Znovukliknutím ji odznačí. Označená karta se pozná tak, že je vysunuta výš než neoznačené karty. Označených karet může být více naráz.

Pokud klikne do své plochy pro vyložené karty, provede se pokus o vyložení. Pokud jde o první vyložení, zobrazí se plocha, do které se provádí potenciální vyložení, jak je vidět na obrázku 4.3. Po kliknutí na tlačítko „Zrušit“ se žádné vyložení neuskuteční.

Obrázek 4.3 Plocha zobrazená pro potenciálně vyložené karty při prvním vyložení týmu

Klikne-li hráč na skupinku svých nebo spoluhráčových (ten hráč nahoře) vyložených karet, provede se pokus o doložení označených karet v ruce k právě té skupince, na kterou hráč klikl.

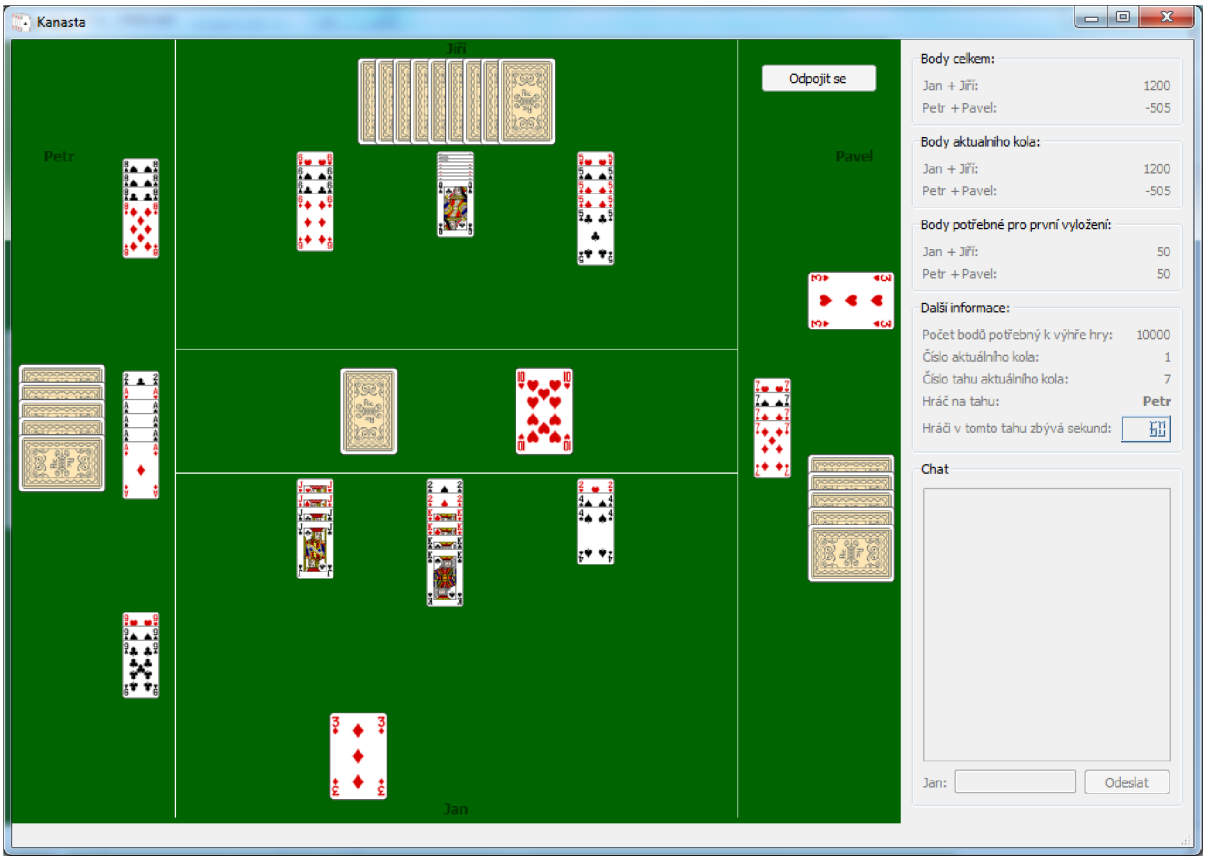
Pro zvětšení malých vyložených karet, musí hráč na tyto karty najet myší. Pro zviditelnění celé karty v ruce, musí najet myší zase na tuto kartu. Naopak po najetí na kmenový nebo odkládací balík, se zobrazí titulek, kolik karet ještě v balíku zbývá.

Kartu odebranou z odkládacího balíku hráč pozná tak, že je v jeho ruce pořád označená (povysunutá) a když na ni klikne, vrátí se zpět do balíku.

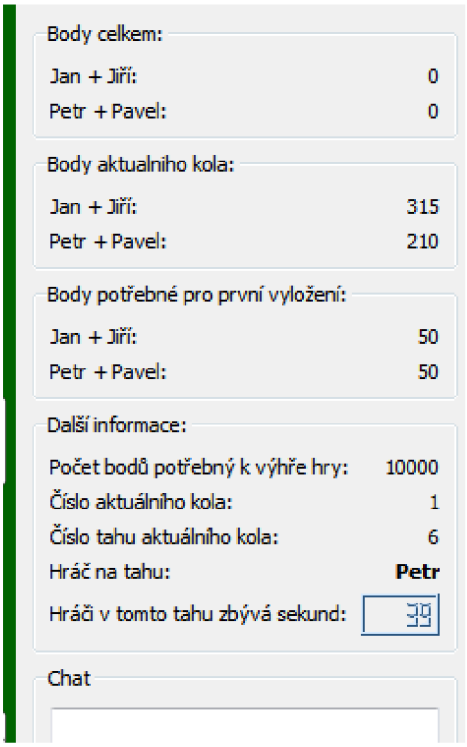
Informace o tom, proč není nějaká akce povolena, se hráčovi zobrazují ve stavovém řádku (dole v liště).

Na obrázku 4.4 lze vidět aplikaci uprostřed rozehrané hry a na obrázku 4.5 je zobrazena ta samá hra po konci kola z pohledu jiného hráče (toho, kdo hru uzavřel). Na obrázku 4.6 se vyskytuje ukázka bočního panelu. A konečně na obrázku 4.7 se nachází tabulka s informacemi, která se zobrazí na konci každého kola.

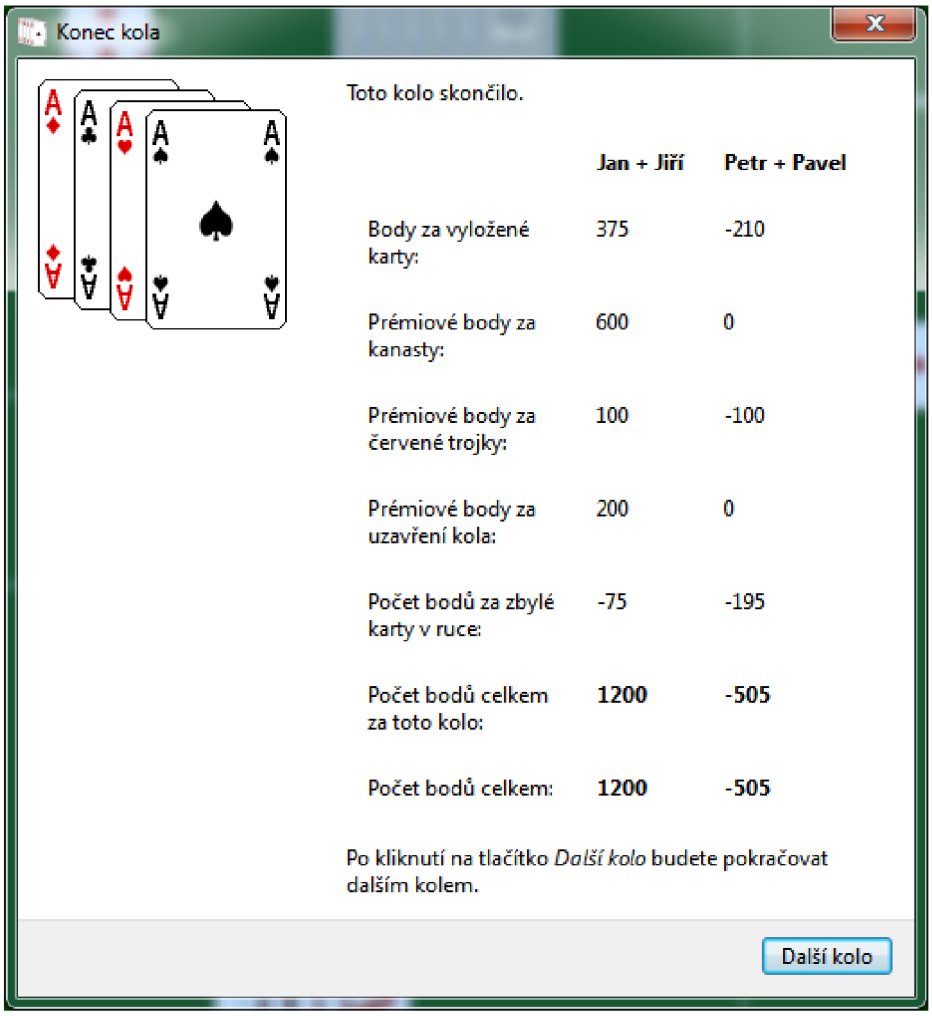
Obrázek 4.4 Rozehraná hra



Obrázek 4.5 Hra na konci kola z pohledu hráče, který zavíral kolo



Obrázek 4.6 Ukázka bočního panelu s aktuálními informacemi



Obrázek 4.7 Dialog se souhrnem kola, který se zobrazuje na konci každého kola

4.3 Komunikační protokol a zprávy

Klient a server mezi sebou posílají zprávy. Zpráva je textový řetězec, který je na straně odesílatele vytvořen a na straně příjemce zpracován. Řetězec je tvořen informacemi oddělenými od sebe středníkem (;). Každá zpráva obsahuje různý počet těchto informací. Závisí na konkrétní zprávě, jaké všechny údaje musí druhé straně poslat, aby zajistila správnou funkčnost. První informace ve všech zprávách říká, o jakou jde událost. Za událostí jsou pak parametry dané události. Počet a typ parametrů se v závislosti na události liší. Většinou se však jedná o počet karet a jejich indexy.

Při posílání zpráv nastal jeden problém. Kdykoli server poslal více zpráv relativně rychle za sebou, klient tyto zprávy přijal jako jeden celek, jako jednu zprávu. Tento problém byl vyřešen tak, že za každou odeslanou zprávu byl navíc ještě připojen znak mřížka (#). Na straně klienta je pak celá zpráva rozsekána na více zpráv, pokud jich více opravdu obsahuje. V opačném směru tento problém nenastává, protože uživatel je oproti počítači mnohonásobně pomalejší. Uživatel tedy nestihne odeslat více zpráv naráz.

Za účelem rozdělení zprávy na jednotlivé části byla vytvořena třída `TZprava`. Tato třída obsahuje metodu `vyjmiPrvek(char c = ';')`, která vyjímá ze začátku řetězce znaky, než narazí na první znak určen parametrem c. Implicitně je tento parametr nastaven na středník, ale tímto znakem bývá i mřížka pro vyjmutí celé jedné zprávy, když je jich více. Tato metoda tedy vrací vyjmutý řetězec. Další používaná metoda je `vyjmiCislo()`, ta děla to samé jako předchozí metoda, jen navíc řetězec už rovnou převede na číslo.

Další metodou třídy `TZprava` je `vytvoritPoleID()`, která provádí vytvoření pole, tj. objektu třídy `TPole`. `TPole` je třída uchovávající velikost pole a hodnoty jednotlivých položek pole. Tyto položky pole reprezentují indexy karet. Zmíněná metoda prochází zprávu, kde vyjme první číslo. Toto číslo udává, kolik čísel bude ještě následovat, tedy velikost pole. Po alokaci paměti pro pole, je toto pole naplněno zbylými hodnotami ze zprávy.

Přesně opačnou operaci provádí metoda `vytvoritZpravuZPoleID(TPole *poleID)`. Z objektu třídy `TPole` jsou vyjímány velikost pole a jednotlivé prvky. Vše je uloženo do textového řetězce, jednotlivá čísla jsou opět oddělena středníky. Tyto funkce, které převádí řetězec na pole a naopak, se využívají ve funkcích, kde jde o posloupnosti karet – vyložení nebo doložení karet a odebrání balíku.

Struktura a význam všech posílaných zpráv z klienta na server a naopak je vypsána v příloze A.

5 Testování

Cílem testování je odhalit chyby během vývoje aplikace. Testy, které neodhalí nesprávné chování aplikace, jsou neúspěšné. Je obecně složité pokrýt celou množinu testovacích vstupů, těch bývá velké množství. I po sebelepším testování hrozí, že aplikace bude obsahovat chyby. Testování se snaží minimalizovat počet těchto chyb.[6]

Nejprve byla navržena a implementována část server. Tudíž první testování bylo provedeno právě na třídách serveru. Byly napsány pomocné metody. Například pro vypsání hodnot jednotlivých uzlů dynamických seznamů reprezentující posloupnosti karet, aby se zjistilo, zda skutečně dochází k přemísťování karet mezi jednotlivými objekty.

Následovalo vytvoření velice jednoduchého serveru a klienta, kde se testovalo, zda síťová komunikace vůbec bude fungovat, tak jak byla navržena. Zda jsou odeslány všechny znaky řetězce, zda lze připojit více klientů, zkrátka zda chování bude takové, jaké se očekává.

Po implementaci klienta se testovaly jednotlivé reakce na uživatelské akce.

Samotná komunikace už byla časově náročnější na testování. Neustále se musely uměle vyvolávat různé situace ve hře, aby se otestovala každá možnost, každý případ užití. Do programu byly zadávány i chybné vstupy, aby se zjistilo, zda je program odolný vůči úmyslně či neúmyslně špatně zadaným údajům od uživatele. To zahrnovalo zadávání špatných údajů pro připojení, po nichž vyskočí chybová hláška, zadání dlouhého jména, které je v aplikaci ořezáno na 15 znaků, a pak samotné akce spojené s hraním. Jako například odebrání více než jedné karty z kmenového balíku, odhození více než jedné karty do odkládacího balíku, snahu vzít kartu, když není hráč na řadě, snahu doložit kartu na soupeřovy vyložené karty nebo pokus provést odhození karty uprostřed prvního vyložení.

Používaly se textové výpisy, které říkaly, jakou funkci program prováděl a jakou větev podmíněného příkazu navštívil.

Nejvíce věcí se otestovalo samotným hraním Kanasty. Spuštěno někdy bylo až 5 aplikací (1 server a 4 klienti) najednou. Pro testování pozdějších fází hry se musel upravovat zdrojový kód. Například pro otestování, zda správně funguje konec hry, se snížil počet bodů potřebný k výhře hry na 100 bodů.

Při hraní hry často docházelo k pádu buď serveru, nebo klienta. Toto násilné ukončení aplikace operačním systémem měl na svědomí neoprávněný přístup do paměti. V implementaci se vyskytuje velké množství dynamicky alokované paměti, a proto zákonitě musely přijít chyby, kde se přistupuje do pamětí, která nebyla alokována, nebo u které už proběhla její dealokace.

Takové problémy se řešily pomocí debuggeru. To je program, pomocí něhož lze spuštěnou aplikaci krokovat po jednotlivých příkazech. Lze vidět zanoření jednotlivých funkcí, hodnoty jednotlivých objektů a proměnných v jednotlivých fázích provádění aplikace a hlavně lze nalézt

místo, kde operační systém vyšle signál pro ukončení aplikace. Celé se to nazývá ladění programu. Lehce lze nalézt chybu, kterou by programátor jen obtížně hledal, zvláště u rozsáhlejších programů.

Zprvu aplikace padaly celkem často, ale postupným testováním se podařilo příčiny chyb lokalizovat a poté i úspěšně odstranit.

Výsledkem tohoto testování není bezchybná aplikace. Je možné, že se v aplikaci objeví chyba, která unikla pozornosti tohoto testování. Nicméně testování bylo prováděno do doby, než vše fungovalo tak, jak má, i po několikerém spuštění.

6 Závěr

V této práci se podařilo úspěšně navrhnout a implementovat síťovou aplikaci karetní hry Kanasta. Cílem celé hry je dosáhnout určitého počtu bodů během několika kol. Dodržovány jsou pravidla hry včetně toho, že při hře ve čtyřech hráčích hrají vždy dva a dva hráči spolu. Jsou tedy vytvořeny týmy hráčů.

Celá aplikace je tvořena dvěma částmi – klient a server. Server obstarává samotnou logiku aplikace, zajišťuje dodržování pravidel hry a informuje všechny připojené klienty o stavu hry a jejím vývoji. Klient představuje rozhraní mezi uživatelem a serverem. Uživatel posílá požadavky na server prostřednictvím svých akcí, server tyto požadavky zpracuje a posílá odpovědi klientům nazpět.

Za tím účelem byl navržen komunikační protokol, který umožňuje bezproblémovou komunikaci mezi klientem a serverem. O samotné zasílání zpráv se stará rodina protokolů TCP/IP.

Server je implementován jako konkurentní, tzn., že dokáže zpracovat více požadavků od různých klientů najednou. Dokáže se bez problémů vypořádat s požadavky na připojení a odpojení v jakékoli fázi hry a informuje o těchto událostech ostatní klienty.

Navržená logika hry spočívá v hlídání hráčů, aby jim nebylo umožněno porušovat zásady hry (např. odebrání více karet z kmenového balíku naráz). Díky tomuto hlídání hráči nemají prostor k podvádění. Stěžejní částí logiky hry je kontrola pravidel, hlavně vykládání karet obsahuje množství podmínek a výjimek.

Uživatelské rozhraní na straně klienta je vytvořeno v souladu se zažitým vzhledem podobně zaměřených aplikací. Karty mají tradiční vzhled, objekty jsou na hrací ploše rozvrženy inteligentně a ovládání hry je intuitivní.

Aplikace byla implementována v programovacím jazyce C++ s využitím vývojového prostředí Qt framework. Aplikace byla vyvíjena na operačním systému Windows 7, jelikož je ale Qt multiplatformní, lze aplikaci spustit i na systému Linux (testováno na distribuci Ubuntu 10.10). Na systému Windows byl k získání binárního exe souboru využit statický překlad, aby výsledný soubor ke spuštění nepotřeboval externí knihovny. Vše je tedy zkompilováno do jednoho (byť většího, než při dynamickém překladu) výstupního souboru. Není potřeba žádná instalace. Bylo vytvořeno na dvě desítky tříd, které tvoří jádro a uživatelské rozhraní celé aplikace. Celkem celá aplikace obsahuje přes 6000 řádků kódu.

Další vývoj aplikace by se mohl ubírat směrem rozšíření o další varianty této karetní hry. Mezi různými variantami by si uživatel vybíral. Dále by bylo možné rozšířit hru pro až šest hráčů. Rozšířit by se mohla i činnost serveru v tom směru, že by server mohl v jeden čas spravovat více her. V neposlední řadě by bylo možné rozšířit hru na hru pro jednoho hráče. Za ostatní hráče by hrál počítač. Šlo by tedy o implementaci umělé inteligence.

Literatura

[1] SOSINSKY, Barrie. *Mistrovství : počítačové sítě*. vydání první. Brno : Computer Press, 2010. 840 s. ISBN 978-80-251-3363-7

[2] SHINDER, Debra Littlejohn. *Počítačové sítě*. 1. vydání. Praha : SoftPress, 2003. 752 s. ISBN 80-86497-55-0

[3] TANENBAUM, Andrew S. *Computer Networks*. 4th ed. New Jersey : Pearson Education, 2003. 891 s. ISBN 0-13-066102-3

[4] KABELOVÁ, Alena; DOSTÁLEK, Libor. *Velký průvodce protokoly TCP/IP a systémem DNS*. 5. aktualizované vydání. Brno : Computer Press, 2008. 488 s. ISBN 978-80-251-2236-5

[5] ZENDULKA, Jaroslav. *Analýza a návrh informačních systémů* [online]. Brno : VUT, 2006. 178 s. Studijní opora. VUT, FIT

[6] KŘENA, Bohuslav. *Úvod do softwarového inženýrství* [online]. Brno : VUT, 2006. 99 s. Studijní opora. VUT, FIT

[7] PRATA, Stephen. *Mistrovství v C++*. 3. aktualizované vydání. Brno : Computer Press, 2007. 1119 s. ISBN 978-80-251-1749-1

[8] *The C++ Resources Network* [online]. 2011 [cit. 2011-04-20]. Dostupné z WWW: <http://www.cplusplus.com/>

[9] *Qt 4.7: All Classes* [online]. Finland : Nokia Corporation, 2010 [cit. 2011-05-01]. Dostupné z WWW: <http://doc.qt.nokia.com/4.7/classes.html>

[10] *Canasta : Pravidla pro hru*. Kolín : Obchodní tiskárny, [199?]. 16 s.

Seznam příloh

Příloha A: Struktura a význam jednotlivých zpráv komunikačního protokolu

Zprávy posílané ve směru od klienta k serveru

VZIT_KARTU;balík balík ∈ {KMENOVY_BALIK, BALIK} - druh balíku, z kterého hráč kartu bere - když hráč klikne na kmenový nebo odkládací balík, pokud si ještě nevzal kartu
ODHODIT_KARTU;n;ID1;ID2;…;IDn n ∈ <0, 108> - počet odhozených karet ID1,ID2,…,IDn ∈ <0, 108) - jednotlivé indexy odhozených karet - když hráč klikne na odkládací balík, pokud už si kartu vzal
VYLOZIT_KARTY; n;ID1;ID2;…;IDn n ∈ <0, 108> - počet vyložených karet ID1,ID2,…,IDn ∈ <0, 108) - jednotlivé indexy vyložených karet - když hráč klikne do své oblasti pro vyložené karty
DOLOZIT_KARTY; pozice;n;ID1;ID2;…;IDn pozice ≥ 0 - pozice, kam doložit karty n ∈ <0, 108> - počet doložených karet ID1,ID2,…,IDn ∈ <0, 108) - jednotlivé indexy doložených karet - když hráč klikne na vyložené karty svého týmu
VRATIT_VYLOZENI; - když hráč u prvního vyložení klikne na tlačítko „zrušit“
VRATIT_KARTU_Z_BALIKU; - když hráč klikne v ruce na tu kartu, kterou vzal z odkládacího balíku
ZACATEK_NOVEHO_KOLA; - když hráč po ukončení kola klikne na tlačítko „další kolo“, které se nachází pod souhrnem kola
CHAT;text text – jakýkoli řetězec - samotná zpráva zaslaná ostatním - když hráč klikne na tlačítko „Odeslat“ nebo stiskne „enter“, když je kurzor v poli pro psaní zprávy
ZMENA_TYMU;tým tým ∈ {0, 1, 2} - číslo týmu - když si hráč změní tým
ZMENA_POCET_BODU_K_VYHRE_HRY;body body ∈ {5000, 6000, 7000, 8000, 9000, 10000} - počet bodů potřebný k výhře hry - když admin v pravidlech změní počet bodů potřebný k výhře hry
ZMENA_POCET_HRACU;počet počet ∈ {2, 3, 4} - počet hráčů - když admin v pravidlech změní počet hráčů
ZMENA_POCET_ROZDANYCH_KARET;počet počet ∈ {11, 13, 15} - počet karet, kolik se bude před kolem rozdávat

- když admin v pravidlech změní počet karet, kolik se bude před každým kolem rozdávat
ZMENA_POVOLIT_SEKVENCE;povoleno povoleno ∈ {0, 1} - 0 znamená nepovoleno, 1 povoleno - když admin v pravidlech změní, zda povolit vykládaní sekvencí po sobě jdoucích hodnot karet stejné barvy
PRIPRAVEN_KE_HRE; - když hráč klikne na tlačítko „Připraven“
NEPRIPRAVEN_KE_HRE; - když hráč klikne na tlačítko „Nepřipraven“

Zprávy posílané ve směru od serveru ke klientovi

VZAL_KARTU;balík;hráč;ID;trojka;počet	
balík ∈ {KMENOVY_BALIK, BALIK}	- druh balíku, z kterého hráč kartu bere
hráč ∈ {0, 1, 2, 3}	- číslo hráče, který akci provedl
ID ∈ <0, 108)	- index odebrané karty
trojka ∈ {0, 1}	- zda je odebraná karta červená trojka (pokud se bere karta z odkl. balíku, tento parametr chybí)
počet ∈ <0, 108)	- počet zbývajících karet v balíku
- pokud bylo hráčovi dovoleno si vzít kartu	
ODHODIL_KARTU;hráč;ID;trojka;počet;tah;jméno	
hráč ∈ {0, 1, 2, 3}	- číslo hráče, který akci provedl
ID ∈ <0, 108)	- index odhozené karty
trojka ∈ {0, 1}	- zda je odhozená karta červená trojka
počet ∈ <0, 108)	- počet karet v odkládacím balíku
tah ≥ 0	- údaj o kolikátý tah se v kole jedná
jméno – jakýkoli řetězec	- jméno hráče, který je na řadě
- pokud bylo hráčovi dovoleno odhodit kartu	
VYLOZIL_KARTY;hráč;tým;body;n;ID1;ID2;…;IDn	
hráč ∈ {0, 1, 2, 3}	- číslo hráče, který akci provedl
tým ∈ {0, 1, 2}	- číslo týmu, který provedl vyložení
body ∈ Z	- počet bodů týmu včetně tohoto vyložení
n ∈ <0, 108>	- počet vyložených karet
ID1,ID2,…,IDn ∈ <0, 108)	- jednotlivé indexy vyložených karet
- pokud je hráčovi povoleno vyložit karty	
DOLOZIL_KARTY;hráč;pozice;tým;body; n;ID1;ID2;…;IDn	
hráč ∈ {0, 1, 2, 3}	- číslo hráče, který akci provedl
pozice ≥ 0	- pozice, kam doložit karty
tým ∈ {0, 1, 2}	- číslo týmu, který provedl doložení
body ∈ Z	- počet bodů týmu včetně tohoto doložení
n ∈ <0, 108>	- počet doložených karet
ID1,ID2,…,IDn ∈ <0, 108)	- jednotlivé indexy doložených karet
- pokud je hráčovi povoleno vyložit karty	
KARTA_NA_STRED;ID;náhrady 0;…náhrady n;počet	
ID ∈ <0, 108)	- index karty vložené na střed
náhrady 0,…,náhrady n ∈ {0, 1, 2, 3}	- počet náhrad za červené trojky u jednotlivých hráčů, kde 0 až n jsou čísla jednotlivých hráčů

počet ∈ <0, 108)	- počet karet v odkládacím balíku
- po rozdání karet se vždy dá karta na střed (na odkládací balík)	
CHAT;text	
text – jakýkoli řetězec	- samotná zpráva zaslaná ostatním
- když hráč klikne na tlačítko „Odeslat“ nebo stiskne „enter“, když je kurzor v poli pro psaní zprávy	
NOVE_ROZDANI;hráč;kolo;tah;jméno	
hráč ∈ {0, 1, 2, 3}	- číslo hráče, který je na řadě
kolo ≥ 0	- číslo nového kola
tah ≥ 0	- číslo nového tahu
jméno – jakýkoli řetězec	- jméno hráče, který je na řadě
- když je konec kola	
KONEC_HRY;body celkem 0;…;body celkem n;body kolo 0;…;body kolo n;body vylož 0;…;body vylož n;info;tým;počet;jméno 0;jméno 1;body	
body celkem 0,…,body celkem n ∈ Z	- počet bodů týmu celkem, kde 0 až n jsou čísla týmu
body kolo 0,…,body kolo n ∈ Z	- počet bodů týmu za skončené kolo, kde 0 až n jsou čísla týmu
body vylož 0,…,body vylož n ∈ Z	- počet bodů týmu potřebných k prvnímu vyložení, kde 0 až n jsou čísla týmu
info – řetězec	- informace o právě skončeném kole
tým ∈ {0, 1, 2}	- číslo týmu, který vyhrál
počet ∈ {1, 2}	- počet hráčů ve vítězném týmu
jméno 0, jméno 1 – řetězec	- jméno vítězného hráče
body ∈ Z	- počet bodů vítězného týmu
- pokud je po konci kola dosaženo potřebný počet bodů k výhře hry	
KONEC_KOLA; body celkem 0;…;body celkem n;body kolo 0;…;body kolo n;body vylož 0;…;body vylož n;info	
body celkem 0,…,body celkem n ∈ Z	- počet bodů týmu celkem, kde 0 až n jsou čísla týmu
body kolo 0,…,body kolo n ∈ Z	- počet bodů týmu za skončené kolo, kde 0 až n jsou čísla týmu
body vylož 0,…,body vylož n ∈ Z	- počet bodů týmu potřebných k prvnímu vyložení, kde 0 až n jsou čísla týmu
info – řetězec	- informace o právě skončeném kole
- pokud skončí kolo (některý z hráčů uzavře hru nebo nezbudou karty v kmenovém balíku)	
ODPOJIL_SE;hráč;jméno	
hráč ∈ {0, 1, 2, 3}	- číslo hráče, který se odpojil
jméno – řetězec	- jméno hráče, který se odpojil
- pokud se hráč odpojí ze serveru	
VZAL_BALIK;hráč;n;ID1;ID2;…;IDn	
hráč ∈ {0, 1, 2, 3}	- číslo hráče, který vzal balík
n ∈ <0, 108>	- počet karet odebraného balíku
ID1,ID2,…,IDn ∈ <0, 108)	- jednotlivé indexy karet odebraného balíku
- pokud hráč vyložil nebo doložil kartu odebranou z odkládacího balíku	
VRATIL_VYLOZENI;z balíku;n;ID1;ID2;…;IDn	
z balíku ∈ <-1, 108)	- index karty, která byla odebrána z balíku; pokud je -1, tak karta z balíku odebrána nebyla

n ∈ <0, 108>	- počet karet, které se vrací hráčovi do ruky
ID1,ID2,…,IDn ∈ <0, 108)	- jednotlivé indexy vrácených karet
- pokud hráč klikne na tlačítko „zrušit“ v průběhu prvního vyložení	
VRATIL_KARTU_Z_BALIKU;hráč;ID;počet	
hráč ∈ {0, 1, 2, 3}	- číslo hráče, který vrátil odebranou kartu z balíku zpět
ID ∈ <0, 108)	- index vrácené karty
počet ∈ <0, 108)	- počet karet v odkládacím balíku
- pokud si hráč vzal kartu z balíku a pak na ni v ruce klikl	
CASTECNE_VYLOZENI; n;ID1;ID2;…;IDn	
n ∈ <0, 108>	- počet potenciálně vyložených karet
ID1,ID2,…,IDn ∈ <0, 108)	- indexy jednotlivých vyložených karet
- pokud hráč vykládá za tým poprvé a ještě nesplnil podmínky prvního vyložení	
ZMENIL_TYM;hráč;tým	
hráč ∈ {0, 1, 2, 3}	- číslo hráče, který si změnil tým
tým ∈ {0, 1, 2}	- číslo týmu
- když si hráč změnil tým	
ZMENIL_POCET_BODU_K_VYHRE_HRY;počet	
počet ∈ {5000, 6000, 7000, 8000, 9000, 10000}	- počet bodů potřebný k výhře hry
- když admin v pravidlech změnil počet bodů potřebný k výhře hry	
ZMENIL_POCET_HRACU;počet	
počet ∈ {2, 3, 4}	- počet hráčů
- když admin v pravidlech změnil počet hráčů	
ZMENIL_POCET_ROZDANYCH_KARET;počet	
počet ∈ {11, 13, 15}	- počet karet, kolik se bude před kolem rozdávat
- když admin v pravidlech změnil počet karet, kolik se bude před každým kolem rozdávat	
ZMENIL_POVOLIT_SEKVENCE;povoleno	
povoleno ∈ {0, 1}	- 0 znamená nepovoleno, 1 povoleno
- když admin v pravidlech změnil, zda povolit vykládaní sekvencí po sobě jdoucích hodnot karet stejné barvy	
PRIPRAVEN_KE_HRE;hráč	
hráč ∈ {0, 1, 2, 3}	- číslo hráče, který je připraven ke hře
- když hráč kliknul na tlačítko „Připraven“	
NEPRIPRAVEN_KE_HRE;hráč	
hráč ∈ {0, 1, 2, 3}	- číslo hráče, který je nepřipraven kehře
- když hráč kliknul na tlačítko „Nepřipraven“	
KONEC_SERVERU;	
- pokud je serverová část aplikace uzavřena	
VAROVANI;text	
text – jakýkoli řetězec	- zpráva, která bude hráčovi vypsána ve spodní liště
- pokud hráč provedl akci, která nevedla k úspěšnému konci	
PRIPOJIL_SE;jméno;číslo;admin;počet;rozdané;body;sekvence;hráč 0;…;hráč n;tým 0;…;tým n;připraven 0;…;připraven n	
jméno – řetězec	- jméno hráče, který se připojil
číslo ∈ {0, 1, 2, 3}	- číslo připojeného hráče
admin ∈ {0, 1, 2, 3}	- číslo hráče, který zastává roli správce

počet ∈ {2, 3, 4}	- nastavený počet hráčů
rozdané ∈ {11, 13, 15}	- nastavený počet rozdávaných karet v kole
body ∈ {5000, 6000, 7000, 8000, 9000, 10000}	- nastavený počet bodů potřebný k výhře hry
sekvence ∈ {0, 1}	- zda je povoleno vykládat sekvence karet
hráč 0,…,hráč n – řetězec	- jména všech dosud připojených hráčů, kde 0 až n pozice (čísla) připojených hráčů
tým 0,…,tým n ∈ {0, 1, 2}	- čísla týmů dosud připojených hráčů, kde 0 až n jsou pozice (čísla) připojených hráčů
připraven 0,…;připraven n ∈ {0, 1}	- informace, zda jsou dosud připojení hráči připraveni ke hře, kde 0 až n jsou pozice (čísla) připojených hráčů
! pokud na nějaké pozici není žádný hráč připojený, zasílá se místo atributů hráč, tým a připraven jediný atribut (speciální sekvence znaků) symbolizující prázdného hráče !	
- když se hráč připojí na server	
START_HRY;počet;jméno 0;…;jméno n;číslo 0;…;číslo n;hráč;kolo;tah;body	
počet ∈ {2, 3, 4}	- počet hráčů spouštějící se hry
jméno 0,…,jméno n – řetězec	- jméno hráče, kde 0 až n je nové číslo hráče
číslo 0,…,číslo n ∈ {0, 1, 2, 3}	- staré číslo hráče, kde 0 až n je nové číslo hráče
hráč ∈ {0, 1, 2, 3}	- číslo hráče, který je na řadě
kolo ≥ 0	- číslo začínajícího kola
tah ≥ 0	- číslo začínajícího tahu
body ∈ {5000, 6000, 7000, 8000, 9000, 10000}	- počet bodů potřebný k výhře hry
- pokud všichni hráči nastavili, že jsou připraveni	