VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

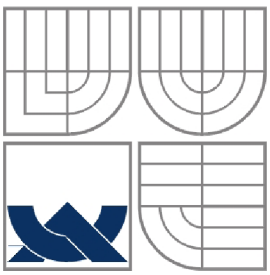
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

MODULÁRNÍ SYSTÉM KASINA

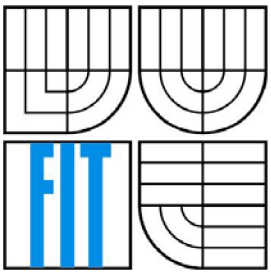
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE Petr Bartůněk
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

MODULÁRNÍ SYSTÉM KASINA

MODULAR SYSTEM OF CASINO

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — Petr Bartůněk
AUTHOR

VEDOUCÍ PRÁCE — Ing. Lukáš Grulich
SUPERVISOR

BRNO 2008

Abstrakt

Práce je zaměřena na modulární vývoj hazardního herního systému. V systému se nachází jeden hlavní modul obsluhující databázi hráčů a uživatelských účtů. Každá hra v tomto systému je dvojice modulů; herní server, který realizuje hru, a klientská aplikace, která nabízí uživatelské rozhraní této hře. Veškerá síťová komunikace probíhá pomocí mojí navržené multiplatformní síťové knihovny TCP_Network, pomocí navržených protokolů pro každý server.

Klíčová slova

modulární systém, kasino, ruleta, blackjack, kostky, válka, hazardní hry, TCP_Network

Abstract

This thesis is pointed to a modular development of hazard game system. There is one main module in this system, it services gamers database and user accounts. Every game in this system is module pair; first it is game server, it realizes the game, and second client application, it has user interface for this game. Every network communications are realized using my own multi platform network library TCP_Network, and using network protocol for every server.

Keywords

modular system, casino, roulette, blackjack, cubes, war, hazard games, TCP_Network

Citace

Petr Bartůněk: Modulární systém kasina, bakalářská práce, Brno, FIT VUT v Brně, 2008

Modulární systém kasina

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Lukáše Grulicha.
Odbornou pomoc při testování mi dobrovolně poskytli David Čeloud a Vlastimil Bartůněk a další dobrovolníci.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal. Uvedl jsem všechen software použitý při vytváření této práce.
Svou činností jsem neporušil žádná autorská práva.

……..
Petr Bartůněk
14.5.2008

Poděkování

Děkuji svému vedoucímu práce panu Ing. Lukášovi Grulichovi za odborný dohled nad mojí prací a za užitečné rady a připomínky k vylepšení a doladění projektu.
Další významné poděkování patří mému spolužákovi Davidovi Čeloudovi za odbornou pomoc při testování hlavní části mojí síťové knihovny TCP_Network.
Ještě bych rád poděkoval svému otci Vlastimilovi Bartůňkovi, který mi v roli testera pomohl odhalit a doladit poslední drobné chyby v klientských aplikacích všech her.

Obsah

Úvod

Každý má na kasino svůj vlastní názor. Víme, že jsou zde lákavé nabídky, bonusy a veliké výhry, které mají za úkol přilákat další hráče. Na druhou stranu je tu ale riziko veliké prohry a neúspěchu. Spousta lidí ani těmto systémům nevěří a domnívají se, že jde pouze o podvod. Já si ale myslím, že se v tomto případě jedná o statistiku a pravděpodobnost. Vše vězí v tom, že kasino má vždy pravděpodobnost na výhru o malý zlomek větší nežli hráč. Tato pravděpodobnost se tedy neuplatňuje na každého hráče jednotlivě, nýbrž na veliké skupiny hráčů a na spousty a spousty odehraných her. Při velkém počtu odehraných her se již toto miniaturní procento pravděpodobnosti proměňuje ve veliké peněžní částky a kasino tak velmi mnoho vydělává. Spousta hráčů vyhraje menší částky, méně hráčů vyhraje větší částku, ale spousta lidí na druhou stranu prohraje skutečně veliké peníze.

Tento herní systém ovšem není založen pouze na statistice a pravděpodobnosti. Dalším důležitým faktorem je zde psychologie. Lidé bývají nalákáni menšími výhrami, kterých lze poměrně snadno dosáhnout pomocí určitých systémů. Takto „ohromeni“ první výhrou, zkoušejí své štěstí dál a dál. Až nakonec systém selže a oni prohrají úplně vše. Např. sázení v ruletě na barvu může být hezká výzva, když si chce někdo přivydělat, stačí pouze při neúspěchu v příštím tahu svoji sázku zdvojnásobovat, a kdykoliv při výhře má hráč profit jeden žeton. Můj názor na všechny hazardní hry je ten, že hráč má opravdu **velikou** šanci vyhrát **málo** peněz, a **malou** šanci prohrát strašně **moc** peněz. Ale v průměru je ta šance na prohru o malinko větší než na výhru.

Já se ve svém projektu pokusím implementovat tento hazardní systém a budu se snažit nabídnout hráčům lákavé výherní nabídky. V návrhu se musím soustředit především na to, aby byl systém co nejvíce modulární, aby žádný projekt nebyl závislý na jiném dalším. Budu potřebovat, aby všechny hry v systému byly snadno udržovatelné, a aby bylo možno jednoduše přidávat nové hry do systému. Důležitý bude již celý návrh.

Dále bych chtěl, aby mohl systém fungovat v počítačové síti, měl by tedy existovat jeden centrální prvek, který všechny ostatní bude řídit. Bylo by dobré, aby byl projekt multiplatformní, proto se bude muset nějak vyřešit síťová komunikace v každém operačním systému, protože v Linuxu i ve Windows je trochu odlišný způsob komunikace pomocí síťových soketů. K tomuto je potřeba nastudovat síťové komunikace v různých operačních systémech. Nastíním zde vývoj vlastní multiplatformní síťové knihovny, které věnuji celou třetí kapitolu. Pro každý server navrhnu jeho komunikační protokol, aby bylo možno s ním komunikovat po síti. Tyto protokoly nemusí být složité, bude stačit, když si budou programy předávat příkazy k vykonání. Budu tedy používat vzdálené volání procedur (RPC).

Nedílnou součástí tohoto systému musí být i databázový systém. Je pochopitelně potřeba, aby v systému byly nějací uživatelé a jejich peněžní účty, a aby se ukládaly historie všech odehraných her. Ukážeme si, jak takovou databázi nainstalujeme a jak ji spravujeme.

Poté, co bude navržena struktura systému, začnu popisovat jednotlivé moduly kasina. Bude se jednat o jeden Hlavní server, který obsluhuje hlavní databázi se všemi uživatelskými účty, a dále sada jednotlivých her, kde každá hra bude mít také svoji databázi.

Servery nepotřebují mít nijak zvlášť hezké uživatelské rozhraní, bude stačit, když budou běžet v systémové konzoli. Zato klientské aplikace bude potřeba navrhnout tak, aby měly příjemné a intuitivní ovládání všech her. Klientské uživatelské rozhraní je v tomto ohledu velmi důležité, aby přilákalo spoustu hráčů. Když se hráčům nebude líbit herní rozhraní, tak by mohli přijít hrát do jiného kasina.

Pro každou hru navrhnu pravidla. Tato pravidla nemusí být nutně stejná jako v jiném kasinu, je tu možnost mít spoustu stejných her, pouze s jinou variantou pravidel. Takto by se také mohla přidat do systému nová hra pro hráče, a obecně platí, že čím má hráč více možností, více her, nebo řekněme více svobody ve výběru herní varianty, tak tím máme větší šanci, že hráč si vybere zrovna náš systém.

Systém je to opravdu veliký a náročný. Já se pokusím vysvětlit a předat touto formou všechny své nejhlavnější myšlenky, o těch méně důležitých se pouze okrajově zmíním.

1 Použité technologie

Projekt zasahuje do řady oblastí. Před samotným vytvářením návrhu je potřeba mít alespoň základní znalosti z některých oborů. Tyto jsou především Síťová komunikace a sítě, programování síťových aplikací, vzdálené volání procedur (RPC), pro rychlé a modulární programování je vhodné zvolit objektově orientované programování a objektově orientovaný programovací jazyk, a v neposlední řadě je potřeba znát databázové systémy, umět je spravovat a zabezpečit.

Tato kapitola je teoreticky zaměřená a má za úkol seznámit s prostředky použitými při návrhu i realizaci celého systému. Není možno uvést všechny detaily, proto se zaměřím zejména na nejpoužívanější technologie. Náměty pro tuto kapitolu a všechnu teorii čerpám z uvedené literatury.

1.1 Síťová komunikace a C++

Ve všech operačních systémech je to soket, který nám umožňuje posílání zpráv mezi programy pomocí síťových protokolů. Díky svým obecným vlastnostem jej lze používat na unixových operačních systémech a s menšími odchylkami v syntaxi i v systémech Windows. Není tedy úplně bezproblémové napsat síťovou aplikaci, která by měla být spustitelná pod více operačními systémy, aby byla multiplatformní. Já se zde zaměřím pouze na programování soketů v jazyce C++.

Komunikace mezi programy probíhá na síťové vrstvě, k tomuto se používají především dva hlavní protokoly TCP/IP nebo UDP/IP. Obecně řečeno, každý počítač potřebuje svoji unikátní IP adresu, aby mu ostatní počítače v síti mohly posílat zprávy. Komunikace probíhá předáváním tak zvaných IP paketů. IP paket obsahuje hlavičku a data, která přenáší. V hlavičce paketu se nachází především zdrojová a cílová IP adresa, aby bylo možno doručit tento paket správnému příjemci v celé síti. Ale abychom si nemuseli pamatovat adresy počítačů pouze jako nějaká čtyři čísla oddělena tečkou (např. 178.37.20.169), zavedly se doménová jména, která se na IP adresy teprve překládají pomocí DNS serverů (např. www.seznam.cz).

Pod pojmem soket si představíme ustavené spojení mezi dvěma počítači, nějaký virtuální kanál, ve kterém proudí data jedním nebo druhým směrem. Nejprve musíme vytvořit celý soket funkcí **socket()**. Poté musíme tento soket naplnit daty, aby se vědělo, kam bude připojený. A dále navázat spojení příkazem **connect()**. Máme-li takto ustanovené spojení, můžeme již do soketu zapsat nějaká data pro poslání, k tomu se použije funkce **send()**. Vzdálený počítač bude schopen poté tato data přijmout funkcí **recv()**. Jestliže už nebudeme posílat ani přijímat žádná data, můžeme toto spojení uzavřít příkazem **close()**(UNIX) nebo **closesocket()** (WINDOWS). Vzdálený počítač zjistí, že jsme ukončili spojení, a svůj soket také uzavře. Je zřejmé, že když navazujeme spojení, musí být na vzdáleném počítači otevřený soket, který bude přijímat připojení pro komunikaci. Tedy vzdálený počítač musí být TCP server, my jsme tím pádem TCP klient.

U soketů dále existují 2 základní režimy čtení, je to neblokovací a blokovací režim. Ve standardním nastavení jsou všechny nové sokety v blokovacím režimu. To je režim, kdy program se v daném úseku kódu zastaví a nepokračuje dál, dokud nepřečte nějaká další data. V praxi se toto nejvíce používá ve spojení s vlákny programu, kdy každé vlákno má jeden soket v blokovacím režimu, a pouze čeká, než přijdou nějaká data. Dále je to neblokovací režim, který se pokusí přečíst ze soketu data, program nebude zablokován, bude se pokračovat v provádění příkazů.

V Unixu se pro změnu režimu soketů používá funkce **int fcntl(int fd, int cmd, long arg);** Ve Windows je to soketová funkce int **ioctlsocket(SOCKET s, long cmd, u_long* argp);**

Komunikace a předávání zpráv pomocí soketů není přesto tak jednoduchá, jak se může na první pohled zdát. Totiž uzavřené spojení mezi dvěma počítači funguje jako virtuální přenosový kanál, něco jako stream (proud) dat, takže soket sám nemůže zjistit, kde končí aktuální posílaná data, nebo kde začínají nová. Proto je nutné, aby obě dvě komunikující strany byly předem dohodnuty, jaká data si budou posílat. Může to být třeba nějaká stanovená délka jednoho příkazu, nebo někdy je lepší použít zarážku, která obsahuje znaky, které nikdy nebudeme posílat jako datové. Takovouto zarážku pochopitelně použijeme pouze v případě, když se nepřenáší žádná binární data, jako jsou soubory, ale např. pouze textové příkazy. Těmto pravidlům v komunikaci se říká **síťový protokol**.

1.1.1 Unix

V Unixu je soket reprezentován identifikátorem, který je celé číslo int. K vytvoření soketu slouží funkce **socket()**. Její deklarace se nalézá v hlavičkovém souboru sys/socket.h. Další makra, která se předávají této funkci jsou v hlavičkovém souboru sys/types.h. Hlavička funkce:

int socket(int domain, int type, int protocol);

Prvním parametrem je doména. Určuje způsob komunikace. Druhým parametrem je typ soketu. Určuje, zda budeme používat protokol TCP, předáme jako druhý parametr hodnotu makra SOCK_STREAM, nebo budeme používat UDP, tak použijeme hodnotu makra SOCK_DGRAM. Makro SOCK_STREAM se používá v případě, kdy chceme vytvořit dvoubodové spojení. Tato služba je spojová a zajišťuje potvrzování všech odeslaných dat a data určitě přijdou v pořadí, v jakém jsme je poslali. Naopak makro SOCK_DGRAM udává, že se bude jednat o nespojovou datagramovou službu. Třetím parametrem je identifikátor protokolu. Můžeme si zvolit jakýkoliv síťový protokol, který je podporován a nainstalován v našem operačním systému.

K připojení soketu slouží funkce **connect()**. Její hlavička je:

int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);

Prvním parametrem je identifikátor soketu (čili celé číslo, které nám vrátila funkce **socket()**). Druhým je adresa TCP serveru. Zde se používá instance struktury sockaddr_in a ukazatel na ni je přetypován na ukazatel na strukturu sockaddr. Třetím parametrem je velikost adresy.

V případě úspěšného navázání spojení funkce vrací 0, jinak -1.

Struktura **sockaddr_in**

Struktura, obsahující adresu serveru. Instance této struktury přesně charakterizuje místo, kam se budeme pomocí soketu připojovat. Obsahuje IP adresu a port vzdáleného počítače.

int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);

Prvním parametrem je identifikátor soketu. Druhým je adresa TCP serveru - vytvořená instance struktury sockaddr_in a ukazatel na ni se přetypuje na ukazatel na strukturu sockaddr. Třetím parametrem je velikost adresy, používá se zde **sizeof(socket).**

V případě úspěšného spojení funkce vrací 0, jinak -1.

int send(int s, const void *msg, size_t len, int flags);

Funkce odešle data. Prvním parametrem je identifikátor soketu, druhým parametrem je ukazatel na blok dat k odeslání. Třetím parametrem je velikost bloku dat v bytech. Posledním parametrem jsou příznaky zprávy, pomocí nich můžeme posílat standardní nebo urgentní zprávy.

int recv(int s, void *buf, size_t len, int flags); -

Funkce čte data ze soketu data. Prvním parametrem je identifikační číslo soketu. Druhým parametrem je ukazatel na blok paměti, do kterého se uloží příchozí data. Třetím parametrem je maximální počet dat, který je možno přijmout do vyrovnávací paměti. Čtvrtým parametrem jsou opět příznaky čtení standardních nebo urgentních zpráv.

Funkce vrací počet skutečně přijatých bytů, nebo -1 v případě chyby.

Následuje seznam všech potřebných hlavičkových souborů, které je nutno připojit k programu, abychom mohli využívat sokety v Linuxu: **<sys/types.h> <sys/socket.h> <netinet/in.h> <netdb.h> <arpa/inet.h>**

1.1.2 Windows

Soket je reprezentován svým identifikátorem, který je typu SOCKET. V Linuxu to bylo celé číslo int. Je to ale pouze takový formální rozdíl, protože typ SOCKET je definován jako u_int. K vytvoření soketu slouží funkce socket. Hlavička funkce:

SOCKET socket(int af, int type, int protocol);

Další funkce, které posílají a čtou data, jsou stejné jako v předcházející kapitole, pouze s tím rozdílem, že deskriptor soketu je zde typu SOCKET, namísto int.

int send(SOCKET s, char* buf, int len, int flags);

int recv(SOCKET s, char *buf, int len, int flags);

Jediným větším rozdílem ve Windows je ten, že pro uzavření soketu se zde musí použít funkce **closesocket(SOCKET)**. Kdybychom použili funkci close(int), jak je to v Unixu, tak by sice překladač nehlásil žádné chyby, ale soket by nebyl uzavřen.

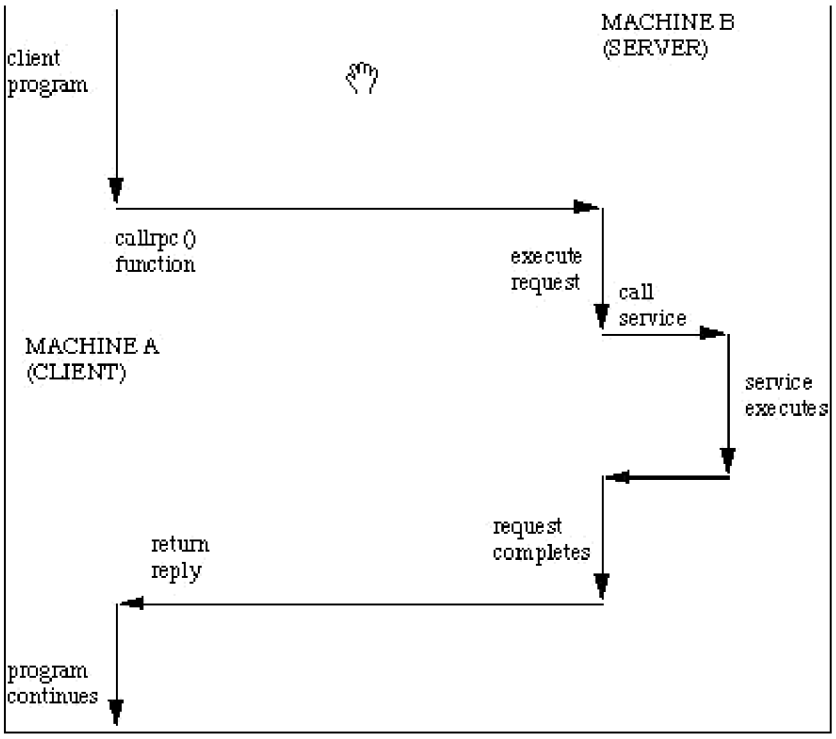
Ve Windows je pouze jediný hlavičkový soubor potřebný připojit k našemu projektu, je to **<windows.h>**, který už sám v sobě obsahuje veškeré definice a další hlavičkové soubory.

1.1.3 RPC – Vzdálené volání procedur

Jedná se o silnou technologii, která umožňuje programům, aby spouštěly funkce, které nejsou součástí jejich vlastního adresního prostoru v paměti, dokonce většinou nejsou uložené ani na počítači, odkud jsou volané. Tyto procedury nemusíme ani spravovat, pouze je můžeme využívat, to vše i bez znalosti jejich vlastní implementace. Tímto lze přehledně rozdělit zátěž jednoho systému na více počítačů a dosahovat mnohem větších výkonů při zpracovávání.

Klient, když volá vzdálenou proceduru, tak pošle toto volání i s parametry serveru a čeká na odpověď. Od této doby je klient zablokován, dokud nedostane nějakou odpověď, nebo dokud nevyprší časový limit spojení.

Když přijde na server takovýto požadavek, je spuštěna daná procedura, nebo služba, a její návratový kód se všemi výsledky jsou zaslány nazpět klientovi. Poté klientský program opět pokračuje v činnosti dále. Důležitým rozdílem mezi vzdáleným a lokálním voláním procedur je ten, že vzdáleně volané procedury nemusí být úspěšně provedeny a navrácen jejich návratový kód kvůli nepředpokládaným síťovým výpadkům. S touto možností je nutné počítat a zabezpečit klienta tak, aby nebyl zablokován, ale aby se po určitém časovém limitu sám vzpamatoval.



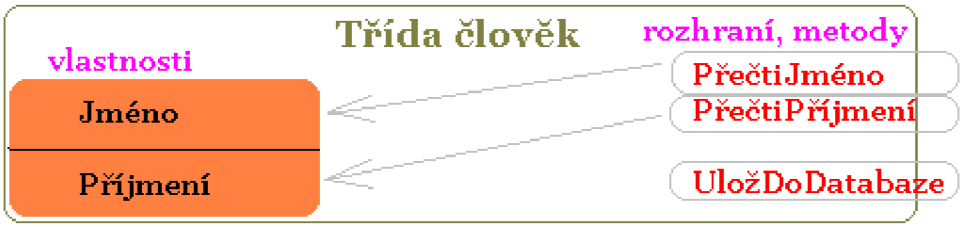
Každá tato procedura je jednoznačně identifikována trojicí: Číslo programu, číslo verze a číslo procedury. Číslo programu nám seskupí procedury do jednoho celku, kde každá procedura má své unikátní číslo. Program dále může být v jedné nebo více verzí, každá tato verze může obsahovat odlišné procedury, které mohou být volány vzdáleně. Na konec každá procedura má své vlastní identifikační číslo.

1.2 Objektově orientované programování

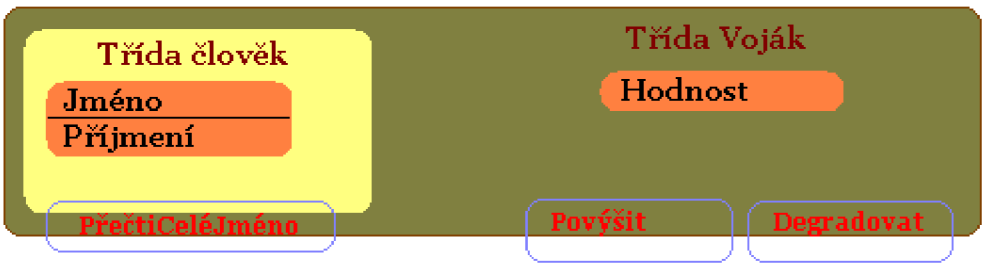
Objekty v programu jsou přirovnatelné k reálným objektům, se kterými pracujeme. Jsou to jednotlivé entity, které jsou schopny spolu navzájem komunikovat. Práce s těmito objekty je velmi intuitivní a nabízí prakticky stejné možnosti, jako v reálném světě, je tu určitá analogie mezi reálným světem a objekty Snadno se rozšiřují nebo upravují.. Objekty abstrahují z reálných skutečností pouze ty informace, které jsou důležité pro funkci programu, většinou ani není nutné, aby byly v objektech veškeré informace stejné jako v realitě. Proto je objektově orientované programování velmi oblíbeno a rozšířeno. Z pohledu programu je objekt instance jedné třídy, která implementuje svoje komunikační rozhraní, má svá data a metody a ty zapouzdřuje do jednoho logického celku. Objekty jsou základní jednotkou modularity i celé struktury programu.

Tato data zapouzdřena uvnitř se nazývají **atributy**, jsou reprezentována identifikačním názvem jedinečným pro celou třídu. Objekty spolu mohou vzájemně komunikovat pomocí zasílaných zpráv. Nemyslí se tím ale fyzické zaslání zprávy, ale jsou to **metody** tříd, pomocí kterých se dá daný objekt ovládat. Pod každou metodou je ukryta vnitřní funkce, která může mít jakékoliv parametry, případně i nějaký návratový kód. Pouze tyto funkce by měly provádět přímou práci v objektu, změnu jeho vlastností a atributů. Hlavními vlastnostmi objektů jsou **zapouzdření**, **dědičnost**, a **polymorfismus**.

Zapouzdření zde bylo malinko naznačeno, jedná se o to, že programátor, který využívá služeb nějaké třídy, nemůže přímo přistupovat do vnitřní struktury tohoto objektu, nemůže měnit jeho vlastnosti jiným způsobem, než použitím rozhraní třídy. S objekty je povoleno komunikovat pomocí metod v tomto rozhraní. Takže rozhraní se dá chápat jako určitý protokol, který přesně stanovuje, jak spolu mohou objekty komunikovat.



Pomocí **dědičnosti** lze implementovat sdílené chování. Je to možnost, rozšířit jednu třídu o nové vlastnosti a funkce bez nutnosti znovu implementovat již vytvořené funkce. Takto vytvořené nové třídy **specializují** existující chování objektů.



Na obrázku je bázová třída člověk, která má vnitřní vlastnosti Jméno a Příjmení, dále nabízí rozhraní s jednou metodou PřečtiCeléJméno(), která vrací celé jméno člověka (Jméno a Příjmení). Třída voják zdědí všechny tyto atributy ze třídy člověk, a tohoto člověka blíže specifikuje jako vojáka, přidá mu další vlastnost Hodnost, a rozhraní ještě navíc rozšíří o dvě metody, Povýšit() a Degradovat().

Polymorfismus dovoluje implementovat stejné zprávy v každém objektu jiným specifickým způsobem. Můžeme mít více objektů, které budou přijímat stejné zprávy, ale reakce na ně bude odlišná. Takto můžeme třeba v každém kontextu získat jiné chování objektů. Mějme např. třídy auto a letadlo, obě mohou obsahovat metodu Přemístit(), ale již je zřejmé, že letadlo bude asi implementovat funkce pro odlet vzduchem, zatímco auto implementuje odjezd po silnici.

1.2.1 Třídy v jazyce C++

V jazyce C++ se vytváří nové třídy pomocí klíčového slova **class**, za ním následuje jméno této třídy a do složených závorek je potom napsán celý kód třídy, na konci těchto závorek musí být středník. Na praktické ukázce si vysvětlíme přesněji, jak se nové třídy vytvářejí.

```
class Clovek {
  private: /**< Sem patří veškeré vlastnosti a funkce, které používá pouze tato třída sama pro
  sebe. Všechny tyto prvky jsou skryté pro všechny okolní objekty. Ani nová třída, která zdědí
  tuto třídu nebude mít přístup k těmto informacím */
  std::string Jmeno; // Jméno člověka
  std::string Prijmeni; // Příjmení člověka
  protected: /**< Zde budou funkce a vlastnosti, které budou také pro všechny okolní objekty
  skryté, ale pokud jiná třída bude dědit z této, tak pro ni budou tyto položky přístupné. */
  public: /**< Rozhraní třídy pro okolní objekty. Jsou zde metody, pomocí kterých se
  komunikuje s touto třídou. Není obvyklé, aby zde byly nějaké proměnné, ty by měly být
  zapouzdřeny v private nebo v protected sekci, a zde by měly být metody, které budou
  takovéto proměnné obsluhovat, nastavovat nebo vracet jejich hodnotu. */
  std::string PrectiJmeno() { return Jmeno; }; /** Vrátí jméno člověka */
  std::string PrectiPrijmeni() { return Prijmeni; }; /** Vrátí příjmení člověka */
  void UlozDoDatabaze();
};
```

Funkce je možno definovat přímo vevnitř třídy, jako jsou zde právě funkce PrectiJmeno() a PrectiPrijmeni(), ale pro přehledné psaní tříd se spíše využívá druhá možnost, to je napsat zde pouze deklaraci, neboli hlavičku funkce, a tuto funkci naimplementovat později v kódu, třeba i v jiném souboru. Podle mě je toto asi nejpřehlednější možnost psaní tříd, v jednom hlavičkovém souboru mít všechny deklarace tříd i funkcí, a samotné definice zakódovat až v jiném souboru.

Pokud nějaké třída má dědit druhou, zapíše se za název nové třídy ještě dvojtečka, za níž následuje klíčové slovo **private**, **protected** nebo **public**, kterým definujeme, kam bude spadat rozhraní děděné třídy (public sekce) v nové třídě. Jestliže tedy chceme zachovat již existující rozhraní, jednoduše napíšeme **public**. Za klíčovým slovem dále následuje název děděné třídy:

```
class Vojak : public Clovek { /** Nová třída, která už v sobě nese vlastnosti třídy Clovek */
  private:
  int Hodnost; /** Nová vlastnost člověka, který je voják */
  public:
  Vojak()
  { //* Této funkci se říká konstruktor. Je to inicializační funkce každé nové instance této třídy,
  je zavolána vždy, když se vytváří objekt této třídy. V tomto příkladu každému nově
  vytvořenému vojákovi inicializujeme jeho hodnost na jedničku. */
        Hodnost = 1;
  };
  /** Rozhraní třídy člověk bude rozšířeno o dvě metody navíc... */
  void Povysit(int rate = 1); /** Povýšit vojáka, standardně o jednu hodnost výše */
  void Degradovat(int rate = 1); /** Degradovat vojáka, standardně o jednu hodnost níže */
};
```

Zde si ukážeme, jak se implementují dvě metody Povysit() a Degradovat() mimo kód třídy:

```
void Vojak::Povysit(int rate) { // Povýšíme ho pouze v případě, nemá-li již nejvyšší hodnost
  if (Hodnost + rate <= NEJVYSSI_HODNOST) Hodnost += rate;
} // Za takovéto definice metod se již nesmí psát středník.
void Vojak::Degradovat(int rate) { // Degradujeme ho v případě, nemá-li již nejnižší hodnost
  if (Hodnost - rate >= NEJNIZSI_HODNOST) Hodnost -= rate;
} // Za takovéto definice metod se již nesmí psát středník.
```

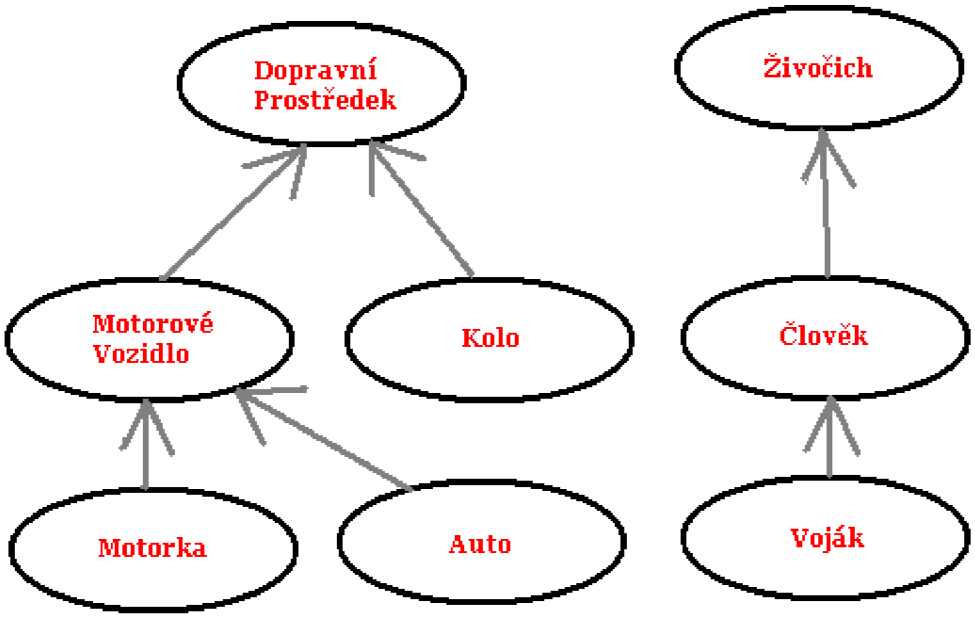
1.2.2 UML Diagramy

Zobrazují třídy systému s vyznačením závislostí. Celý systém nemusí nutně být reprezentován pouze jedním UML diagramem, pomocí balíčků je možno sdružovat několik diagramů do jednoho celku. Existuje velké množství různých typů diagramů, například **diagram případů použití**, **diagram tříd**, **diagram objektů**, **diagram komponent**, **diagram nasazení**, **diagram aktivit**, **stavový diagram**, **diagram spolupráce** a **sekvenční diagram**. Každý z výše vyjmenovaných typů diagramů obsahuje poněkud odlišný repertoár dostupných grafických značek (předmětů a relací). Pro tento projekt budou nejvíce zajímavé diagramy tříd a stavové diagramy.

Diagramy tříd jsou nejčastěji používané nástroje UML a tvoří základ všech prostředků objektové analýzy a designu . Tyto diagramy by měly být vytvořeny již ve fázi analýzy jako pojmové modely, jejichž úkolem by mělo být formálněji definovat jednotlivé termíny používané ve studované problémové oblasti. Takové modely budeme nejvíce využívat v okamžiku, kdy je nutno zpětně vstřebat terminologii oboru. Když se životní cyklus softwaru přibližuje do fáze vývoje jsou diagramy tříd poměrně zásadně přehodnocovány, v ideálním případě by měly být zpracovány až do podoby grafického modelu, který bude ekvivalentní se zdrojovým kódem.

Diagramy tříd zobrazují především vztahy mezi třídami. UML explicitně rozlišuje několik druhů tříd, např. parametrizovatelné třídy, stejně jako rozličné množství vztahů, které jednotlivé třídy navzájem pojí (asociace, agregace, kompozice, specializace/generalizace, ...).

Jeden diagram tříd by měl obsahovat jen asi 7 až 9 tříd navzájem propojených. Je to kvůli přehlednosti, nicméně na druhou stranu vyvstává problém vhodného rozdělení systému na dílčí oblasti (balíčky). Dědičnost je v tomto diagramu naznačována šipkami vedoucími od specializované třídy ke třídě bázové, neboli kořenové. Třídy, které už nejsou dále specializovány se nazývají listové.



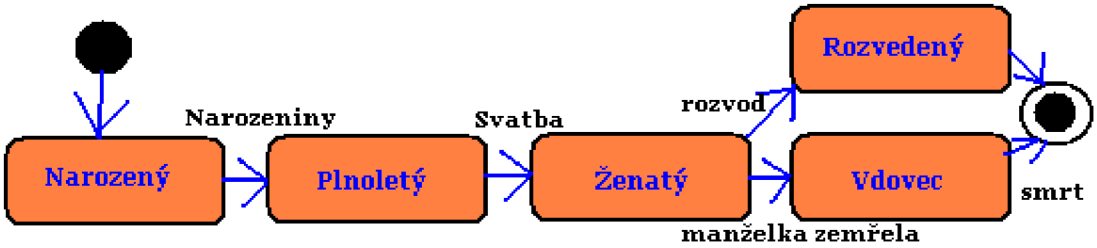
Příklad diagramu tříd.

Stavový diagram patří mezi klasické a osvědčené nástroje objektového modelování. V UML je tento prostředek přítomen v lehce modifikované podobě Harelových diagramů. Diagram stavů a přechodů, jak je někdy tento prostředek nazýván, slouží pro modelování životního cyklu části systému svým rozsahem pouze na jeden objekt. Přechod mezi jednotlivými stavy bývá vyvolán podnětem z vnějšího okolí, nejčastěji ve formě zprávy zaslané příslušnému objektu nebo jinou externí

událostí. Validní stav objektu je zjednodušeně řečeno definován přípustnými hodnotami jeho atributů, přechod mezi stavy je pak vlastně vyjádřením změny hodnot těchto atributů.

Jak je asi každému zřejmé, životní cyklus objektu nějak začíná a zpravidla i nějak končí (v případě, že se nejedná o objekt perzistentní, který je například uchováván v objektové databázi). Ovšem nemusí být již zcela samozřejmé, že tzv. startovní stav by měl být v diagramu vždy pouze jeden, zatímco konečných stavů, tj. stavů ukončujících životní cyklus sledovaného objektu může být více, pochopitelně většinou alespoň jeden. Z implementačního hlediska je startovní stav ekvivalentní alokaci paměti, resp. zavolání konstruktoru třídy, naopak konečný stav je realizován vykonáním destruktoru, v lepším případě prostředky automatické správy paměti (garbage collector).

V neposlední řadě je možné specifikovat podmínky, které musí platit při přechodu do stavu a při odchodu z něho. Dále již následuje příklad jednoduchého stavového diagramu.



1.3 Databáze

Databáze je strukturovaná kolekce záznamů nebo dat. Ve většině případů se dnes využívají relační modely databáze. Relace je matematický pojem, a vyjadřuje podmnožinu kartézského součinu všech dat v databázi. Hlavní jednotkou v této databázi je tabulka, která má přesně danou hlavičku, a tudíž všechny datové typy ve stejném sloupci jsou stejné. Tabulka tedy obsahuje sloupce a všechny svoje záznamy, to jsou řádky. Relační databáze obsahuje více tabulek, každá uchovává jiná data. Dalo by se říci, že každá tabulka uchovává kolekci objektů stejného typu. Takto může vypadat příklad jedné tabulky, která uchovává základní informace o lidech.

Tabulka Člověk			
Jméno	**Příjmení**	**RokNarození**	**Adresa**
Jan	Novák	1978	Ulice Nového, Nová Paka
Jana	Nováková	1977	Ulice Nového, Nová Paka

V tabulce mohou platit i některé omezení hodnot v příslušných řádcích. Nejzákladnějším a nejčastějším omezením je tzv. **primární klíč**. V každé tabulce může být označen vždy max. jeden sloupec jako primární klíč, a znamená to, že data v tomto sloupci se nesmí nikdy opakovat. Používá se většinou pro identifikační číslo nějakého objektu, kdy je potřeba, aby každý objekt měl unikátní číslo. Vedle primárního klíče to může být dále **kandidátní klíč**, který má stejné omezení pro data v jednom sloupci. Následující tabulka rozšiřuje předchozí tabulku Člověk o jeho identifikační číslo:

Tabulka Člověk				
ID	**Jméno**	**Příjmení**	**RokNarození**	**Adresa**
1000001	Jan	Novák	1978	Ulice Nového, Nová Paka
1000002	Jana	Nováková	1977	Ulice Nového, Nová Paka

Jedním z dalších nejpoužívanějších omezení je **cizí klíč**. Tento klíč se vztahuje na nějakou druhou tabulku z databáze, na sloupec s primárním klíčem, jakoby se první tabulka pomocí cizího klíče připojila ke druhé tabulce, a čerpala z ní informace. Každá položka ve sloupci s cizím klíčem se musí shodovat s jednou položkou v druhé tabulky ze sloupce, do kterého se tento cizí klíč odkazuje. Názornější bude asi obrázek se dvěma tabulkami. Tabulka vpravo nese informace o pracovních odděleních, obsahuje pouze sloupce název pracovního oddělení a identifikační číslo. V levé tabulce jsme opět rozšířili tabulku Člověk o cizí klíč, pomocí kterého každého člověka můžeme přiřadit do jednoho pracovního oddělení. Všimněme si zde, že žádný člověk nemusí být z Oddělení veřejnosti.

Tabulka Člověk					
ID	**Jméno**	**Příjmení**	**RokNarození**	**Adresa**	**Oddělení**
1000001	Jan	Novák	1978	Ulice Nového, Nová Paka	2
1000002	Jana	Nováková	1977	Ulice Nového, Nová Paka	3
1000003	Petr	Vánek	1973	Pražská ulice, Praha	2

Tabulka Oddělení	
Číslo	**Název**
1	Veřejnost
2	Prodej
3	Marketing

1.3.1 Jazyk SQL

Pomocí SQL dotazů se provádí požadavky v databázích. Může jít o čtení, zápis do tabulek, vytváření nových tabulek, rušení celé databáze, vytváření uživatelských účtů a pod. Omezím se jen na pár hlavních dotazů, které jsou stěžejní pro můj projekt. Máme zde hlavní dotazy pro práci s databází:

CREATE – Vytvoření objektu do databáze, celou databázi, tabulku, pohled....

DROP – Zrušení objektu z databáze

ALTER – Změna parametrů nějakého objektu v databázi

Vytvoření nové tabulky se provádí takto:

```
CREATE TABLE JménoTabulky (
  ID INT,
  Jméno VARCHAR(10),
  Příjmení VARCHAR(20),
Oddělení INT,
PRIMARY KEY(ID));
```

Vytvoření celé nové databáze:

```
CREATE DATABASE JménoDatabáze
```

Zrušení tabulky a databáze:

DROP DATABASE JménoDatabáze;

DROP TABLE JménoTabulky;

Omezení typu primární klíč se zde provádí pomocí klíčových slov **PRIMARY KEY**, a v závorce je jméno sloupce. Omezení typu cizí klíč na sloupec Oddělení by se přidalo do tabulky pomocí příkazů **FOREIGN KEY**(Oddělení) **REFERENCES** DruháTabulka(Odkazovaný sloupec).

Dále jsou to manipulační příkazy, které manipulují s daty v tabulkách:

SELECT, INSERT, UPDATE, DELETE

Jejich operandem jsou bázové tabulky, výsledkem těchto dotazů je vždy tabulka!

Příkaz **SELECT**:

SELECT [ALL|DISTINCT] Položka [[AS] AliasSloupce], …

FROM TabulkovýVýraz [[AS] [AliasTabulky]], …

[**WHERE** podmínka]

[**GROUP BY** JménoSloupceFROM|číslo, …]

[**HAVING** podmínka]

[**ORDER BY** JménoSloupceSELECT|číslo [ASC|DESC]], …

Klauzule **WHERE** určuje podmínku pro výběr řádků, např. WHERE Zůstatek > 100000, budou vybrány ty řádky tabulky, u kterých je ve sloupci Zůstatek větší částka než 100000.

Pomocí klauzule **ORDER BY** lze výslednou tabulku uspořádat podle nějakého sloupce, vzestupně nebo sestupně.

Klauzule **HAVING** uspořádá tabulku, ale aplikovaná na celou skupinu, používá se ve spojení s **GROUP BY**, kdy se stejné řádky v tabulce seskupují do jednoho.

Příkaz **INSERT**:

INSERT INTO JménoTabulky [(sloupec1),...] **VALUES** (hodnota1,...)

Vloží se do existující tabulky nové záznamy, pokud jsou splněna všechna omezení.

Příkaz **DELETE**:

DELETE FROM JménoTabulky

[**WHERE** podmínka]

Vymaže záznamy z tabulky, které odpovídají podmínce v klauzuli WHERE. Pozor, pokud nezadáme žádnou klauzuli WHERE, budou vymazána všechna data z tabulky!

Příkaz **UPDATE**:

UPDATE JménoTabulky

SET Sloupec = Výraz|NULL|DEFAULT, …

[**WHERE** podmínka]

Změní hodnoty specifikovaných sloupců v řádcích splňujících podmínku WHERE.

1.3.2 MySQL

Nejpopulárnější databázový systém, který je open-source, a všude velice dobře dostupný a podporovaný je MySQL. Je ke stažení zdarma na svých oficiálních stránkách http://www.mysql.com Má velice široké využití ve všech různých programovacích jazycích. Jsou k němu dostupné potřebné konektory, pomocí nichž se můžeme k této databázi připojovat využitím programovacích jazyků, jako jsou C, C++, Java, Perl, Python, Ruby PHP, .NET a další. Vedle konektorů jsou také spolu s databázovým systémem nabízeny další nástroje pro snazší správu. Tento databázový systém je podporovaný pro většinu používaných operačních systémů, tím si získal velkou oblibu u menších firem nebo vývojářů. Pro můj projekt bude tento databázový systém vyhovovat.

1.3.2.1 Instalace serveru

Nejdříve si vybereme verzi systému, na výběr máme ze starších verzí 4.1, 5.0, 5.1, nebo dnes aktuální verzi 6.0. Dále si zvolíme operační systém, na kterém budeme mít databázi spuštěnou. V závislosti na našem výběru si poté stáhneme verzi produktu, kterou chceme. Já jsem používal verzi 5.1 pod systémem Windows 2000 Professional, a verzi 6.0 v unixovém systému CentOS 5.1. Asi nejjednodušším způsobem instalace je stáhnout si již připravené a předkompilované soubory ve formě instalátoru. Ve Windows se instalují i konfigurují pomocí standardního systémového instalačního průvodce, v Unixu lze použít připravený RPM balíček MySQL, který nainstalujeme do systému pomocí Update manageru. Verze v systému Windows již v sobě obsahuje standardní konektory pro připojení použitím C nebo C++, v Unixu je potřeba tyto konektory zvlášť nainstalovat do systému. Jedná se o C API konektor, který bude potřeba při překládání mého systému.

Při překladu je nutné uvést další dva přídavné parametry překladači, aby se projekt přeložil spolu s knihovnou C API. Jsou to tyto dva parametry:

-lmysqlclient -lz

Pokud nemáme knihovnu nainstalovanou v systému, můžeme uvést cestu k ní pomocí parametru **-L**. např. pokud máme knihovnu uloženou v /usr/local/mysql/lib, napíšeme do parametrů:

-L/usr/local/mysql/lib -lmysqlclient –lz

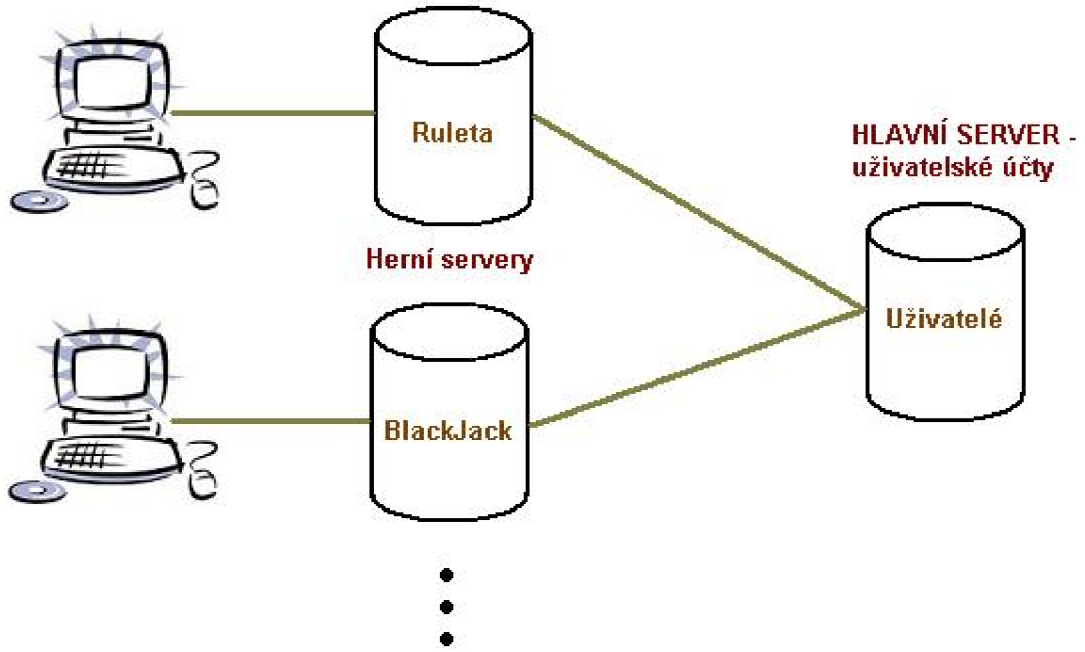
Pro ty případy, že bychom používali hlavičkové soubory někde v externím adresáři, bylo by nutné za parametr -I uvést cestu k nim, aby mohl překladač najít tyto soubory. Např.:

-I/usr/local/mysql/include.

Ve Windows budu projekty překládat pomocí vývojového prostředí Dev-C++, do něhož potřebuji nainstalovat hlavičkové soubory pro C API. Provede se to v menu „Nástroje“ pod tlačítkem „Zjistit Updaty“. Zde si mohu vybrat a nainstalovat spousty balíčků pro toto vývojové prostředí, mému projektu postačí nainstalovaný balíček **libmysql** ze skupiny Databáze.

2 Návrh řešení – Modulární návrh

Aby byl systém modulární, je potřeba, aby jednotlivé části na sobě nebyly nijak závislé, aby se daly lehce změnit, upravit nebo doladit. Následující obrázek ilustruje obecný návrh celého systému.



Bude tedy potřeba vytvořit několik aplikací, které spolu budou schopny komunikovat a předávat si informace potřebné pro správný chod systému. Dále je potřeba pro každý server navrhnout jeho komunikační protokol. Důležitou částí je tedy síťová komunikace mezi programy. Jelikož síťová komunikace pomocí soketů se v každém operačním systému liší, bude potřeba navrhnout síťovou knihovnu, nazvu ji TCP_Network, která bude multiplatformní, tedy přeložitelná jak pod operačním systémem Windows, tak i pod systémem linuxového charakteru. Této síťové komponentě bude věnována celá 4. kapitola, jelikož vytvoření takovéto síťové knihovny je pro mě veliká výzva a veliký úspěch.

Každá jedna hazardní hra bude dvojice programů: Herní server a Klientská aplikace.

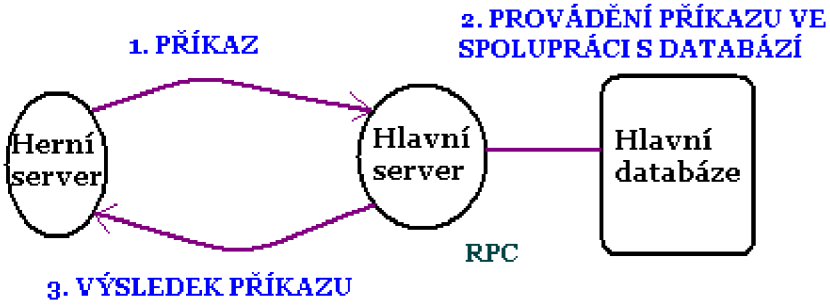
Klientská aplikace nebude moci přistupovat do Hlavní databáze, ani k rozehrané hře přímo. Klient pouze posílá Hernímu serveru své prováděné akce. Pouze Herní servery budou zpracovávat klientské požadavky a přistupovat k rozehrané hře, a následně budou klienta pouze informovat o tom, co se děje ve hře, nebo jak hra skončila. Toto je důležité opatření k tomu, aby nemohl hráč nijak podvádět nebo předpovídat další vytaženou kartu a pod. Každá klientská aplikace bude navíc umožňovat cvičnou hru bez připojení k serveru. Ale tam ať si uživatel podvádí, jak chce. :-)

2.1 Hlavní server

Na obrázku vpravo je naznačena hlavní databáze, je to nejdůležitější databáze v celém systému. Zde budou uloženy informace o uživatelích, o uživatelských účtech a budou se zde zaznamenávat všechny operace nad účty, všechny transakce, které byly provedeny. Dále je zde tabulka blokovaných hráčů nebo IP adres, ta by měla sloužit k tomu, aby byl zamezen přístup do systému některým nepovoleným hráčům nebo osobám ze zakázaného seznamu. Hráč nebo IP adresa se mohou stát nebezpečnými např. při pokusech neoprávněným způsobem vniknout do systému nebo jakkoliv napadnout systém. Těmto hráčům nebude umožněno se ani přihlásit. Hlavní databáze musí být zpracovávána nějakým programem, který se k této databázi připojí, já jej nazývám Hlavní server.

Hlavní server kontroluje hlavní databázi a provádí nad ní operace, které požadují Herní servery. Hlavní server tedy pracuje jako server vzdáleného volání procedur. Herní servery tedy komunikují s Hlavním serverem pomocí síťového protokolu, který jim umožňuje vzdáleně spouštět funkce obsluhující uživatelské účty. Všechny funkce jsou provedeny na straně Hlavního serveru a zpět pomocí síťového protokolu jsou herním serverům zaslána požadovaná data jako jsou např. zůstatek na uživatelském účtu nebo pouze informace o úspěšnosti nebo neúspěšnosti provedené akce.

Protokol, jímž komunikuje Hlavní server s dalšími aplikacemi je velice jednoduchý. Funguje na principu dotaz – odpověď. Herní servery tedy posílají příkazy a hned v zápětí čekají na jejich provedení a následnou odpověď a informaci o stavu provedeného příkazu. Následující obrázek celou komunikaci demonstruje.



Kdybych měl popsat komunikaci z pohledu síťové komunikace, tak se jedná o:

1. Herní server pošle v blokovacím režimu příkaz k provedení Hlavnímu serveru, a hned nato Herní server v blokovacím režimu čeká na odpověď.
2. Hlavní server v neblokovacím režimu čeká na příkazy od klientů. Přijde mu příkaz, přečte ho ze soketu. Podle povahy příkazu vykoná určitou činnost, většinou se jedná i o zásah do databáze. A nakonec pošle zpět Hernímu serveru návratovou hodnotu provedené operace.
3. Herní server nakonec příjme výsledek příkazu a vrátí se zpět do neblokujícího režimu.

Zde je výpis příkazů, které jsou v aktuální verzi serveru podporovány:

- **GAME** - Název herního serveru
- **PING** - Zkouška spojeni (ECHO PING)
- **EXIT** - Vypne celý mainSERVER
- **LOGIN** - Přihlášení hráče
- **MONEY** - Stav uživatelova účtu
- **TRANSACTION** – Zadání peněžní transakce k jednomu účtu
- **BAN** - Zablokuje nějakého uživatele nebo IP adresu
- **UNBAN** - Odblokuje nějakého uživatele nebo IP adresu
- **REGISTER** - Zaregistruje nového uživatele
- **CHPASS** – Změna uživatelova hesla
- **USERINFO** - Zadost o informace o uživateli
- **CHUSERINFO** – Změna uživatelských údajů

Podrobný popis všech příkazů i s jejich správnou syntaxí bude v kapitole 5.

Herní servery, nebo i jakékoliv jiné aplikace, které by se připojily k Hlavnímu serveru by mohly tedy jednoduchou formou ovládat Hlavní server i s celou Hlavní databází. Je tedy nutné, aby tento server i s databází byly v zabezpečené privátní síti, do které by měly přístup pouze Herní servery, aby nemohlo dojít ke zneužití tohoto jednoduchého protokolu. Hlavní server obsahuje opravdu velmi důležitá a citlivá data uživatelů, proto je nanejvíce nutné zabezpečit jej i firewallem.

Ve svém projektu nebudu prozatím řešit zálohování nebo obnovování databáze, ale pro budoucí vývoj systému je toto velmi důležitá oblast, takže v další verzi počítám i se zálohováním všech databází celého systému.

2.2 Herní servery

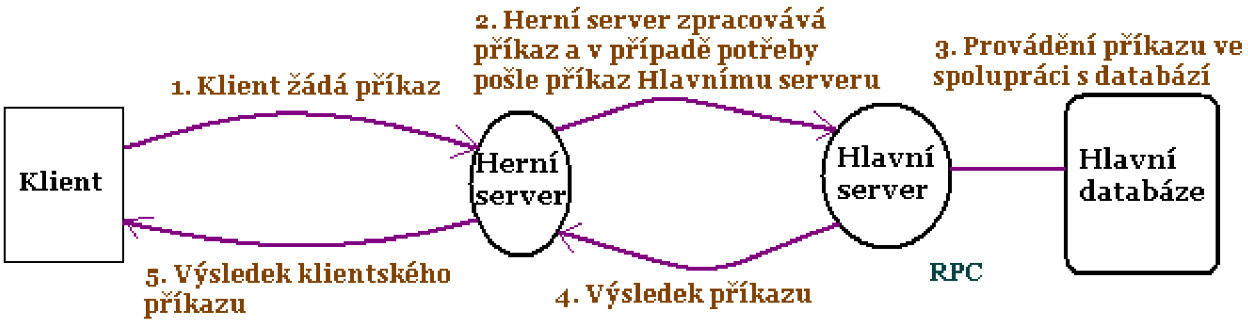
Vedle hlavního serveru, který zpracovává požadavky týkající se uživatelských účtů je tu již zmiňovaný herní server, nebo sada herních serverů. Každý tento server by měl běžet jako samostatný program, aby byla zachována modularita systému, a aby byl systém velmi lehce spravovatelný. To znamená, že když budeme upravovat jeden server, tak nepotřebujeme nijak zasahovat ani do ostatních Herních serverů, a už vůbec ne do Hlavního serveru. To samé bude platit při přidávání nové hry, kdy stačí znát pouze síťový protokol, kterým budeme komunikovat s Hlavním serverem.

Herní server je tedy modul, který realizuje jednu konkrétní hazardní hru. Všechny hry jsou svou činností více či méně podobné, proto je také výhodné vytvořit jednu bázovou třídu, nazvu ji **GAME**, ze které budou všechny ostatní hry dědit společné vlastnosti všech her, jako jsou např. funkce **Bet()** nebo **Play()** (**Bet** pro vsazení určité částky na určitou kombinaci, **Play** na spuštění hry),

nebo také delegáty či callback funkce (zaregistrované funkce, které se mají provádět při nějaké události) **PlayerBets** nebo **Result**. V podstatě se jedná o ukazatele na funkce, které jsou spouštěny při nějaké události ve hře (**PlayerBets** se spustí, když ve hře uživatel provedl sázku, **Result** se spustí na konci každé hry), jsou velmi důležité, aby i hlavní program dostal informaci o tom, co se děje ve hře, a mohl zapsat sázky nebo výhry do databáze.

Herní server komunikuje jak s Hlavním serverem (předává mu příkazy ke spouštění), tak i s klientskými aplikacemi, které uživateli hráči poskytují grafické rozhraní pro jednotlivé hry. Z toho vyplívá, že vytvoření celé jedné nové hry bude spočívat ve dvou nebo třech fázích (záleží, z jakého pohledu se díváme na věc). Je tedy nejdříve potřeba naprogramovat novou třídu, která zdědí již zmiňovanou bázovou třídu GAME, a která bude mít kompletní rozhraní podle každé konkrétní hry, a dále potom musíme tuto třídu zavést do Herního serveru, a také do klientské aplikace. Tady pozor, v klientské aplikaci bude potřeba ještě tuto novou třídu navíc rozšířit o další funkce, jako je především síťová komunikace.

Herní server používá ke komunikaci s klienty podobný protokol jako Hlavní server. Také se jedná o druh komunikace typu dotaz – odpověď. Následující obrázek již ukazuje kompletní komunikaci, jak by mohla probíhat v celém systému. Není to však podmínkou, protože některé příkazy nemusí vůbec potřebovat práci s Hlavním serverem a jeho databází.



Síťová komunikace je zde podobná jako u Hlavního serveru:

1. Klient pošle v blokovacím režimu příkaz k provedení Hernímu serveru, a hned nato čeká v blokovacím režimu na odpověď.
2. Herní server v neblokovacím režimu čeká na příkazy od klientů. Přijde mu příkaz, přečte ho ze soketu. Podle povahy příkazu vykoná určitou činnost, většina těchto příkazů bude potřebovat i komunikaci s Hlavním serverem (jeden nebo více příkazů).
3. Hlavní server v neblokovacím režimu čeká na příkazy od klientů. Přijde mu příkaz, přečte ho ze soketu. Podle povahy příkazu vykoná určitou činnost, většinou se jedná i o zásah do databáze. A nakonec pošle zpět Hernímu serveru návratovou hodnotu provedené operace.
4. Herní server nakonec příjme výsledek příkazu a vrátí se zpět do neblokujícího režimu.
5. Nakonec klient příjme výsledek příkazu.

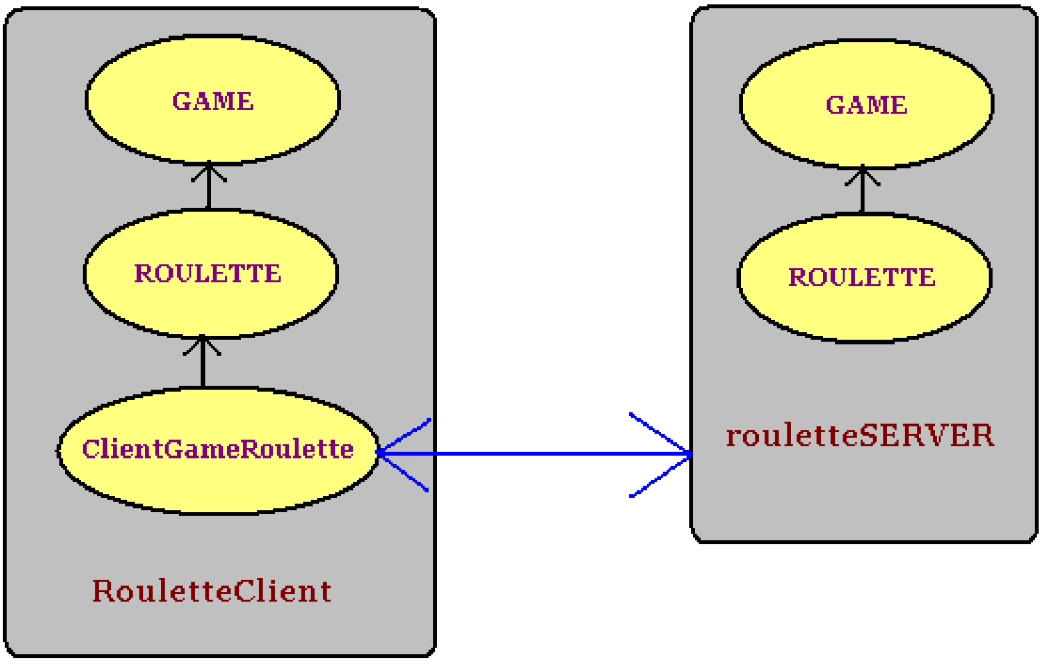
V každém Herním serveru bude implementována jedna herní třída, a však spousta vlastností všech Herních serverů bude stejná, navrhnul jsem několik společných příkazů:

- **PING** - Test spojení
- **LOGOUT** – Odhlásit se
- **MONEY** - Stav uživatelova účtu
- **REGISTER** - Zaregistruje nového uživatele
- **CHPASS** – Změna uživatelova hesla
- **USERINFO** – Žádost o informace o uživateli
- **CHUSERINFO** – Změna uživatelských údajů

Kromě prvních dvou příkazů Herní server neudělá nic jiného, než že přepošle celý příkaz Hlavnímu serveru ke zpracování, počká na návratový kód a přepošle jej zase klientovi. Dalo by se tedy říct, že pro tyto příkazy funguje Herní server pouze jako přemostěné připojení z klienta na Hlavní server. Klient je tedy schopen komunikovat přímo s hlavním serverem, proto je nutné, aby tyto příkazy byly maximálně zabezpečeny. Tyto příkazy jsou kontrolovány Herním serverem, který přepošle Hlavnímu serveru příkaz, pouze když je uživatel přihlášen.

2.3 Klientské aplikace

Skoro vše potřebné ke klientským aplikacím již bylo napsáno výše. Díky tomu, že i klientská aplikace bude v sobě obsahovat herní modul, bude uživateli umožněna cvičná hra, kdy nemusí být ani připojen k žádnému serveru, může si jen pro zábavu nebo na zkoušku zahrát třeba na cestách ve vlaku, když nemá připojení k Herním serverům.



Obrázek výše znázorňuje příklad návrhu ruletní klientské aplikace spolu s Herním serverem. Zde je dobře vidět, že klient nemá přímý přístup ke hře spuštěné na serveru, pouze komunikuje se serverem. Třída ClientGameRoulette tedy rozšiřuje funkčnost třídy ROULETTE o síťovou komunikaci a o možnost připojení se k hernímu serveru. Pokud je tato třída připojena k serveru, tak zajišťuje komunikaci pomocí síťového protokolu, a hráč může hrát o skutečné peníze přes vytvořenou hru na serveru. V případě, že třída ClientGameRoulette není připojená k serveru, nabízí uživateli standardní lokální hru. Hráč si bude moci vyzkoušet hru nanečisto.

Co se týče grafického provedení klientských částí, speciální grafické efekty budou vynechány, nejsou hlavní náplní mého modulárního ani objektového návrhu. Ale díky modularitě bude možno v budoucnu přidat nebo vylepšit jakékoliv grafické prvky všech klientských aplikací, v budoucnu hodlám v projektu použít grafické prvky OpenGL, aby byly hry pro uživatele ještě více zajímavější. Při testování klientských aplikací uživateli by měly být brány v potaz další návrhy nejen na funkčnost jednotlivých her, ale právě tak i na grafické provedení programu, protože hráč je tím, kdo bude tyto aplikace používat, tak by měla být pro něj práce s nimi co nejpříjemnější a intuitivní.

Dále zatím hodlám mít všechny klientské části spustitelné pouze pod operačním systémem Windows, použiji pro jejich vývoj programovací prostředí Borland C++ Builder 2006, které neumožňuje psát programy pod jinými operačními systémy. Budoucí vývoj těchto aplikací by měl směřovat ke ztenčování klientů, aby byly spustitelné i pod linuxem, nejideálnější by byly webové aplikace společně s informačním systémem, který by uživatelům umožnil přehlednou správu účtu i hraní her a administrátorům by usnadňoval správu všech serverů i her. Rád bych pokračoval tímto směrem při studiu magisterského navazujícího programu.

3 Knihovna TCP_Network

Nejdůležitějším modulem celé hry je síťová komunikace, zajištění, aby spolu mohly komunikovat všechny programy a předávat si příkazy. Síťová komunikace a práce se sokety se liší v závislosti na použitém operačním systému. Jelikož má být celý systém všech serverů multiplatformní, musel jsem vytvořit tuto knihovnu, která bude přeložitelná na jakémkoliv operačním systému a která bude umět posílat síťové zprávy a příkazy. Při této komunikaci se nevytváří žádné podprocesy, vše by mělo být řízeno neblokujícím přístupem ze strany hlavního programu.

3.1 Popis a funkce

V podstatě šlo o to, prostudovat odlišnosti v celé komunikaci v různých operačních systémech a tyto odlišnosti poté spojit a nahradit je vlastními definovanými příkazy. K tomuto jsem použil direktivy překladače „#ifdef“ a „#else“, jimiž jsem nadefinoval jednotné síťové příkazy, a ty dále používám pro síťovou komunikaci. Pro lepší představu zde uvádím kus kódu ze souboru TCP_Network.h, který definuje pro každý operační systém své systémové hlavičkové soubory, a dále je zde příklad vytvoření jednoho společného příkazu TCP_CLOSE:

```
#if ((defined WIN32) || (defined WIN64) || (defined MSWINDOWS)) // WINDOWS
  #include <winsock.h>
  ...
  #define TCP_CLOSE(socket) closesocket(socket)
  ...
#else  // LINUX (NO WIN)
  #include <sys/types.h>
  #include <sys/socket.h>
  #include <netinet/in.h>
  #include <netdb.h>
  #include <arpa/inet.h>
  #include <unistd.h>
  #include <fcntl.h>
  #include <errno.h>
  ..
  #define TCP_CLOSE(socket) close(socket)
  ..
#endif
```

Na tomto obrázku je dobře vidět, že ve Windows se uzavírá soket příkazem closesocket() a v Linuxu je to příkaz close(). Také seznam připojených hlavičkových souborů se výrazně liší v každém operačním systému. Díky takovéto definici pomocí direktiv „#ifdef“ a „#else“ jsem si pro použití těchto odlišných příkazu nadefinoval vlastní příkaz TCP_CLOSE, ve kterém bude vždy právě ten správný příkaz, podle operačního systému, na kterém budu program překládat.

V síťové komunikaci bylo dále potřeba vyřešit, aby všechny příkazy i odpovědi přišly vcelku, aby např. při zatížení linky nemohlo dojít k rozpůlení jednoho příkazu. K tomu jsem si zvolil zarážku, kterou posílám na konec každého příkazu, abych přesně věděl, kde příkaz skončil, a kde začíná další. To si můžu dovolit, protože pomocí této knihovny budu posílat výhradně textové příkazy a čekat na textové odpovědi, proto jsem si jako zarážku zvolil znak dalšího řádku, aby bylo možné programy jednoduše testovat i pomocí jakéhokoliv programu, který používá textovou komunikaci pro posílání příkazů, např. program TELNET.

Aktuální verze této knihovny nepodporuje zatím šifrovaný přenos dat. Rozšíření této knihovny o šifrování pomocí SSL bude dalším krokem k vylepšení systému.

Tato knihovna tedy obsahuje dvě hlavní síťové třídy, jsou to TCP_Server a TCP_Client. Obě jsou navíc pro přehlednost umístěny do jmenného prostoru TCP_Network.

3.2 TCP_Server

Zde je seznam a použití funkcí, které nabízí rozhraní třídy TCP_Server:

Start() – Inicializace serveru.
Listen(int q) – Otevření portu a vytvoření fronty pro klienty.
Connecting() – Neblokující připojování klientů. Pokud byl připojen další nový klient vyvolá se callback funkce OnConnected(t_client).
Reading() – Neblokující čtení od všech připojených klientů. Pokud nějaký z klientů už poslal celý příkaz ukončený zarážkou, vyvolá se callback funkce OnDataReceived(t_client).
Send(t_client client, std::string text) – Poslání odpovědi jednomu klientovi v blokujícím režimu – Čeká se, až se pošle celý příkaz.
SendToAll(std::string text) – Poslání dat všem připojeným klientům.
Disconnect(t_client client) – Odpojení jednoho klienta od serveru.
Close() – Vypne server, přestane naslouchat na žádném portu.

Procedury událostí (CALLBACK)
void (*OnConnected)(t_client) – Akce, když se klient připojí.
void (*OnDisconnected)(t_client) – Akce, když se klient odpojí.
void (*OnDataReceived)(t_client) – Akce, když přijde další příkaz od klienta.

3.3 TCP_Client

Tato třída nabízí připojení se k serveru na zadaný port a dále posílání příkazů a čekání na odpovědi.

Zde je seznam a použití funkcí, které nabízí rozhraní třídy TCP_Server:

Connect(std::string server, int _port) – Připojení se k serveru na určité port.
Reading() – Neblokující čtení ze serveru.
Read(std::string &text) – Blokující čtení ze serveru, běh programu je blokován, dokud nepřijde další příkaz nebo odpověď od serveru.
Send(std::string text) – Poslání příkazu serveru v neblokujícím režimu.
Disconnect() – Odpojení se od serveru.

Procedury událostí (CALLBACK)
void (*OnConnected)(t_client) – Akce, při připojení k serveru.
void (*OnDisconnected)(t_client) – Akce při odpojení od serveru.
void (*OnDataReceived)(t_client) – Akce, když přijde další příkaz od serveru.

3.4 Testování funkčnosti

Funkčnost této knihovny je stěžejní pro můj projekt, proto jsem jejímu testování věnoval i dva programy navíc. Jedná se o client.cpp a server.cpp. Je to takový malý příklad programů pro posílání zpráv mezi klienty a serverem. Server přeposílá Všechny zprávy, které mu přijdou všem ostatním klientům, dále je server schopen přijímat i nějaké příkazy, např. /LOGIN, kde si zaznamená jméno uživatele a to dále připisuje k jeho komunikaci.

Při testování šlo především o to, zda je server schopen přijímat příkaz jako celek, nikoliv jako „roztržený“ příkaz na více kusů.

Jelikož tyto programy nejsou hlavní součástí projektu, nepřikládám k nim Makefile a dále se již o nich nebudu zmiňovat. Pro vyzkoušení jsou ale přeložitelné standardními C++ překladači.

4 Hlavní server

Toto je nejdůležitější server v systému, bez něj není možné ani spustit Herní servery. Obsluhuje Hlavní databázi všech uživatelských účtů.

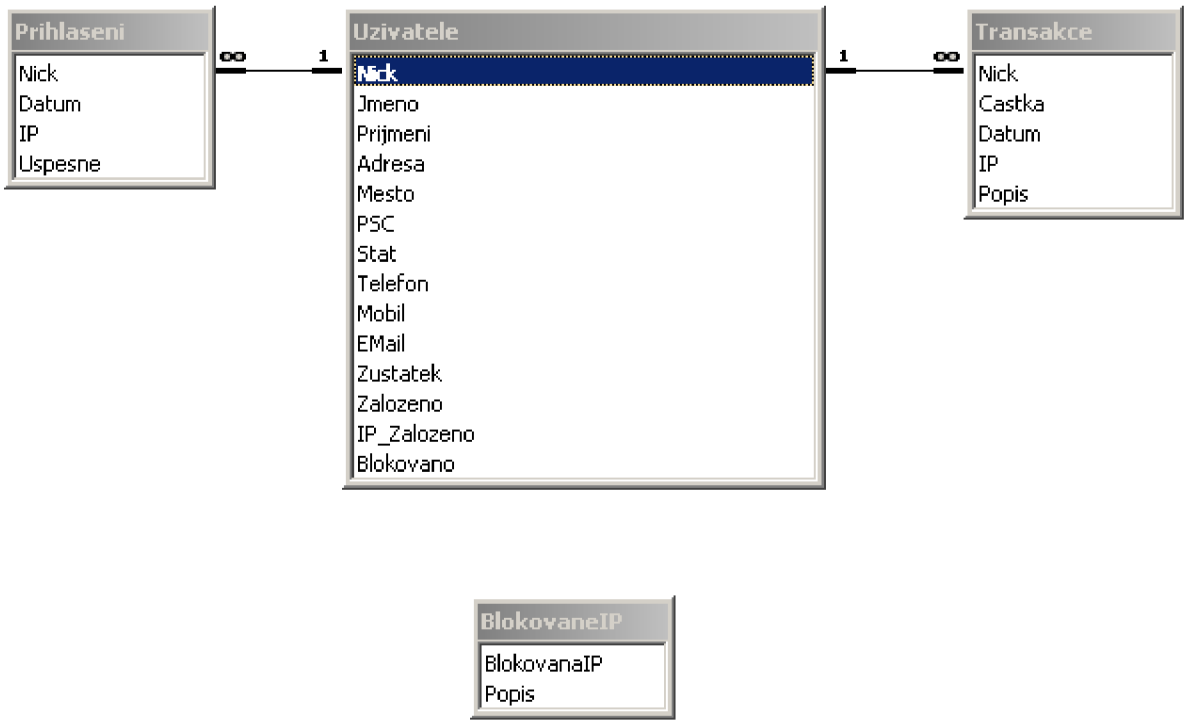
Hlavní server naslouchá na určitém portu a čeká na připojení Herních serverů. Jelikož zatím moje síťová knihovna nepodporuje šifrovaný přenos, tak je prakticky možné, že by se k němu mohl připojit kdokoliv také pomocí příkazu TELNET, a spouštět jeho příkazy, nebo by bylo možné odposlouchávat síťovou komunikaci, proto je nutné, aby byl server prozatím oddělen od veřejné sítě, a aby k němu neměly přístup žádné počítače z vnější sítě.

Bude nutné vytvořit celou Hlavní databázi a naplnit ji minimálně jedním administrátorským účtem a dalšími daty určenými pro testování.

Hlavní server tedy běží na jednom počítači a přijímá klienty. Ještě dále popíši protokol, pomocí kterého se dá komunikovat s tímto serverem, pomocí něhož se serveru předávají příkazy k vykonání.

4.1 Databáze serveru

Následuje ER diagram, který znázorňuje celou strukturu databáze:



Do této databáze se ukládají uživatelské účty a transakce na účtech, dále pak čas přihlášení jednotlivých uživatelů do systému a nakonec je zde jedna tabulka blokovaných hráčů.

Nainstalovanou a vytvořenou prázdnou databázi je možno naplnit a inicializovat testovacími daty pomocí skriptu přiloženého na CD (mainINITIATE.sql).

4.2 Síťový protokol

Jedná se o server vzdáleného volání procedur (RPC). Klienti posílají textové zprávy, které jsou vyhodnocovány jako příkazy. Těmto klientům je zpět vrácen návratový kód procedury nebo nějaká data, o která žádají. Zde je seznam a podrobný popis všech příkazů (včetně správné syntaxe), které jsou v aktuální verzi Hlavního serveru podporovány:

Příkaz	Parametry
GAME	NazevHry
PING	
EXIT	
LOGIN	Nick*Heslo
MONEY	Nick
TRANSACTION	Nick*Castka*Popis
BAN	Nick*IP*Popis
UNBAN	Nick*IP
REGISTER	Nick*Heslo*Jmeno*Prijmeni*Adresa*Mesto*PSC*Stat*Telefon*Mobil*Email*IP_Zalozeno
CHPASS	Nick*OldPass*NewPass
USERINFO	Nick
CHUSERINFO	Nick*Jmeno*Prijmeni*Adresa*Mesto*PSC*Stat*Telefon*Mobil*Email

GAME:

Tento příkaz zatím nemá speciálnější využití než to, že Hlavní server vypíše do logu IP adresu a název serveru, který je připojen z této IP adresy.

PING:

Test spojení. Odpověď na tento příkaz je "SUCCESS".

EXIT:

Pozor na tento příkaz. Jím lze vzdáleně ukončit a vypnout celý server. Nevím, zda bude tento příkaz v budoucnu nějak užitečný nebo využívaný, nicméně je tu a je funkční.

LOGIN:

Zkontroluje se v databázi uživatelské jméno s heslem. V případě úspěšného přihlášení je odpověď "SUCCESS", v případě neúspěšné operace, je odpověď na tento příkaz "BAD". Nicméně pokus o přihlášení je zaznamenán do databáze ať už byl úspěšný nebo neúspěšný. Pomocí tabulky Přihlašování by mohla být v budoucnu přidána další funkce serveru, která by zablokovala daného uživatele v případě, že by třikrát po sobě zadal špatné heslo.

MONEY:

Přečte se z databáze zůstatek peněz na účtu uživatele podle parametru Nick.

TRANSACTION:

Provede se daná transakce s uživatelským účtem. Celá transakce je zaznamenána do databáze, ale je proveditelná pouze v případě kladného konečného zůstatku na účtu.

BAN:

Zablokuje se daný uživatel a/nebo IP adresa zadané jako parametry.

UNBAN:

Odblokuje se daný uživatel a/nebo IP adresa zadané jako parametry.

REGISTER:

Zaregistrování nového hráče a inicializace jeho konta na 0.0Kč. Při úspěšné registraci je vráceno "SUCCESS", v opačném případě "BAD". Podmínkou pro vytvoření nového účtu je, že daný účet ještě v databázi neexistuje.

CHPASS:

Změní uživateli jeho heslo z OldPass na NewPass.

USERINFO:

Podá informaci o daném uživateli ve tvaru: Nick * Jmeno * Prijmeni * Adresa * Mesto * PSC * Stat * Telefon * Mobil * Email * Zustatek * Zalozeno * IP_Zalozeno.

CHUSERINFO:

Již existujícímu uživateli lze tímto příkazem změnit následující informace: Jméno, Příjmení, Adresu, město, PSČ, Stát, Telefonní číslo, Mobilní číslo, Email.

4.3 Postup instalace a spuštění

4.3.1 Prerekvizity

Spouštění a funkce Hlavního serveru předpokládá nainstalovaný MySQL server s vytvořenou databází mainDB podle ER diagramu z kapitoly 5.1. Druhým předpokladem je, mít v této databázi nainstalovaný konektor MySQL C API spolu s předkompilovanými knihovnami, tím jsme schopni přeložit zdrojový kód, který má k této databázi přistupovat. Žádná jiná speciální síťová knihovna než TCP_Network není potřeba. Je úplně jedno, pod kterým operačním systémem budeme překládat tento server, stačí pouze, aby systém splňoval tyto dva předpoklady (MySQL a C API konektor).

Před kompilací je potřeba, nastavit správně konstanty v programu. Jsou to 4 databázové konstanty, pomocí nichž se bude spuštěný program připojovat do databáze (mainSERVER.cpp).

89. **DB_HOST** - DATABÁZOVÁ KONSTANTA Jméno počítače
90. **DB_NAME** - DATABÁZOVÁ KONSTANTA Jméno databáze
91. **DB_USER** - DATABÁZOVÁ KONSTANTA Jméno uživatele
92. **DB_PASS** - DATABÁZOVÁ KONSTANTA Heslo uživatele

Pokud tedy dojde ke změně těchto konstant je nutné znovu překompilovat celý program. Toto pochopitelně není nejvhodnější způsob uchování konstant přímo napsaných a zkompilovaných do programu, ale toto bylo pouze prozatímní testovací řešení, které bude v dalších verzích nahrazeno konstantami zapsanými do souboru na disku nebo na jiné médium.

4.3.2 Kompilace zdrojových souborů

4.3.2.1 Linux

Pro kompilaci pouze Hlavního serveru můžeme použít soubor Makefile.mainSERVER, takže napíšeme do terminálu příkaz: **make –f Makefile.mainSERVER**

4.3.2.2 Windows

Ve Windows se projekt přeloží ve vývojovém prostředí Dev-C++, celý projekt je uložený a připravený v souboru mainSERVER.dev. Ještě zkontrolujeme, zda máme nainstalovanou knihovnu libmysql (**Nástroje->Package Manager**). Pokud se v seznamu tato knihovna nenachází, je potřeba si ji před kompilací stáhnout a nainstalovat, stačí spustit **Nástroje -> Zjistit updaty**, zde si zvolíme nějaký funkční server a z něj si stáhneme a nainstalujeme balíček **libmysql** v aktuální verzi.

Nyní již přistoupíme konečně ke kompilaci. V menu **Spustit** zmáčkneme **Zkompilovat**, a projekt se přeloží do spustitelného EXE souboru.

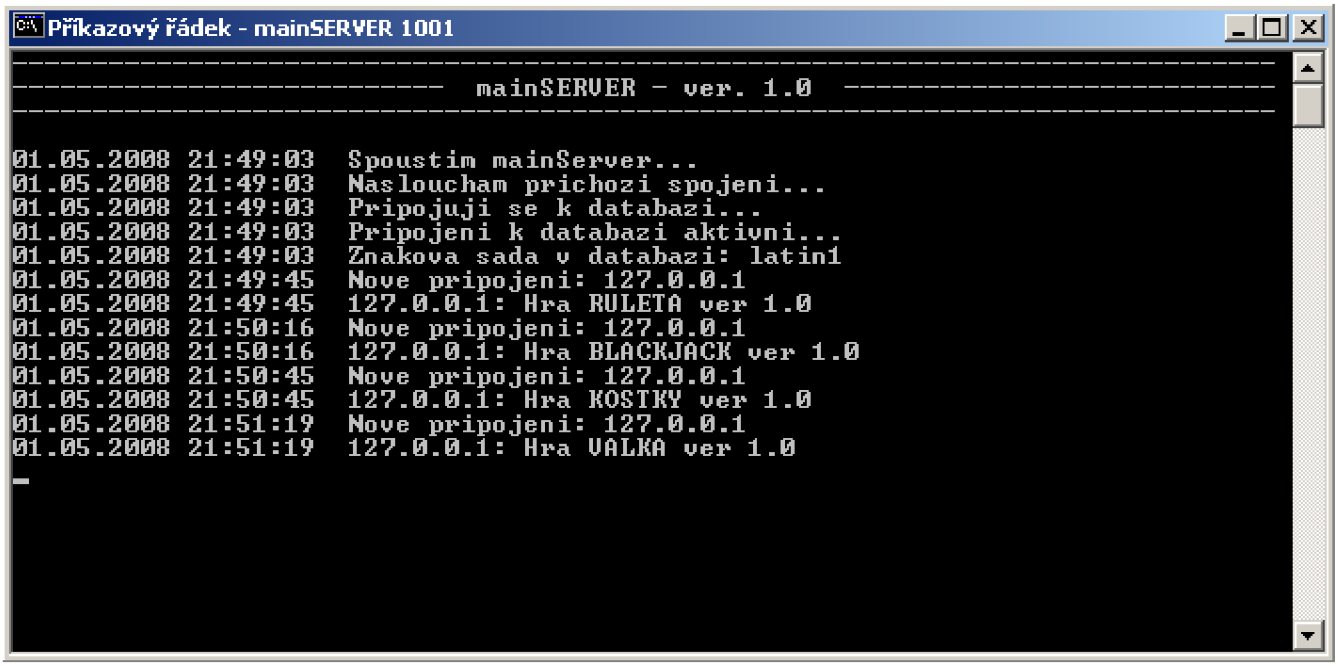
4.3.3 Spouštění serveru

Syntaxe pro spuštění Hlavního serveru z příkazové řádky:

mainSERVER *port*, kde *port* je číslo portu, na kterém server naslouchá příchozí připojení.

V aktuální verzi není možno za běhu programu psát příkazy přímo do konzoly serveru. Prozatím se mi nepodařilo nijak zjistit, jak odchytávat stlačené klávesy. Ale při dalším vývoji tento nedostatek určitě napravím.

A takto konečně vypadá spuštěný server nasazený v provozu:



5 Přidávání nových her do systému

Tato kapitola bude demonstrovat jednoduchost modulárního návrhu. Vytvořil jsem bázovou třídu pojmenovanou GAME, ze které dědí všechny ostatní hry. Tato třída obsahuje Společné vlastnosti a funkce pro všechny hry. Vytvořením této třídy jsem si ulehčil návrh všech ostatních her, jelikož společné vlastnosti všech her jsou definované na jednotném místě, nemusí se tedy programovat pro každou hru zvlášť.

Následuje rozhraní této bázové třídy a popis jednotlivých funkcí:

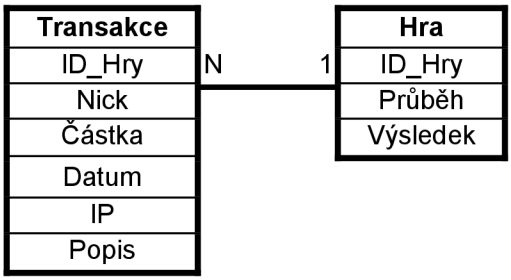
Bets() – Vrátí tabulku všech aktuálních sázek
Bet(std::string bet) – Vsadí se určitá částka peněz na nějakou kombinaci. Sázka se zde bere jako řetězec znaků, přičemž se jedná o regulární výraz Sázka@Peníze[*Sázka@Peníze]*, to znamená, že sázka na nějaké pole se od částky oddělí zavináčem, a více sázek je od sebe oddělených hvězdičkou.
Play() – Virtuální funkce pro začátek hry. Funkce je virtuální, protože každá hra si bude začátek hry implementovat podle svých požadavků.

Procedury událostí (CALLBACK)
void (*Result)(std::string Nick, std::string result, double money) – Když hra skončila. Vnější funkci předáme výsledky hry včetně konečné výhry, které hráč dosáhl. Tato funkce by měla zapsat uživateli vyhranou částku do databáze, a informovat ho o ukončení hry.
bool (*PlayerBets)(std::string Nick, double money) – Hráč se pokusil vsadit nějaké peníze. Tuto skutečnost předáme nějaké vnější funkci, aby se zjistilo, zda uživatel má na účtu dostatek financí a pro provedení transakce. Pokud funkce vrátí true, víme, že uživateli se peníze odečetly z databáze a můžeme proto potvrdit jeho sázky.

Rozhodl jsem se vytvořit prozatím 4 celé hry. Ruletu, BlackJack, Kostky a Válku. Ještě před návrhem herních serverů je potřeba navrhnout pravidla těchto her a poté vytvořit herní třídy, které budou tato pravidla uskutečňovat.

Nyní se tedy zaměřím na vytvoření každé hry zvlášť. Pro každou hru navrhnu také další příkazy (aplikační síťový protokol), které bude umět zpracovávat. Seznam příkazů společných pro všechny hry jsem již uvedl při návrhu systému, nyní je už pouze potřeba navrhnout specifické příkazy pro každý Herní server. Jedna konkrétní hra není určena pro více hráčů, pokaždé, když hráč začne hrát s krupiérem novou hru, vytvoří se jen pro něj instance této hry proti krupiérovi. Hráči tedy nemohou hrát proti sobě, hrají pouze proti krupiérovi, vlastně každé kolo začíná jakoby celá hra znovu od začátku. Toto je také jedno z opatření, aby hráč nemohl např. počítat karty, a potom by věděl, které zbývají v balíku. Po každé hře jsou karty znovu rozmíchány, tím tento problém eliminuji.

Dále navrhnu databázi pro každý herní server. Databáze pro všechny herní servery budou prozatím stejné a budou obsahovat pouze informace o ukončených hrách. V budoucnu by měly být rozšířeny o zaznamenávání přihlašování hráčů, a o další různé tabulky pro práci s informačním systémem. Následuje ER diagram navržené databáze:



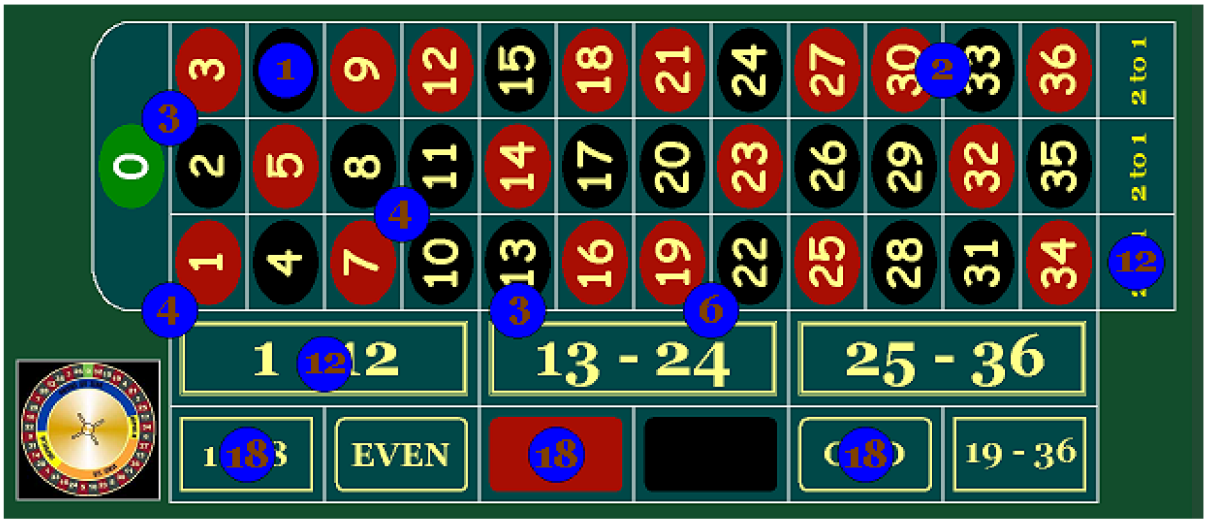
Inicializační SQL soubor je pro všechny servery stejný, je uložený v souboru gameINITIATE.sql, avšak nesmíme zapomenout, že tyto databáze musíme vytvořit čtyři, pro každou hru zvlášť. Ve svém databázovém systému jsem si herní databáze nazval rouletteDB, blackjackDB, cubesDB a warDB. Je pochopitelně možno zvolit si vlastní názvy databází, ale bude potom nutné změnit tyto názvy i v konstantách programů.

5.1 Ruleta

5.1.1 Pravidla této varianty

Cílem hry je, předpovědět, které další číslo bude vytočeno na ruletě (0 - 36).

Na plátno hráč pokládá svoje sázky a v případě úspěchu je mu vyplacena výhra. Zde je obrázek plátna, na němž jsou příklady jednotlivých možných sázek.



Číslo uvnitř žetonu nám zde říká, na kolik čísel se daná sázka vztahuje. Následující tabulka zahrnuje informace o všech povolených sázkách a o vyplácených výhrách:

Vnitřní sázky		
1	Sázka pouze na jediné číslo.	Při výhře se vyplácí 35:1
2	Sázka na dvě čísla vedle sebe. Žeton se položí na čáru mezi těmito čísly.	Při výhře se vyplácí 17:1
3	Sázka na tři po sobě jdoucí čísla je možná Umístěním žetonu na čáru nejbližšího z nich. A nebo je možná sázka na trojici včetně nuly umístěním žetonu na spojnici těchto tří čísel.	Při výhře se vyplácí 11:1
4	Sázka na čtveřici se provede umístěním žetonu na spojnici těchto čtyř čísel. Dále je možná sázka na čtveřici včetně nuly, jak ilustruje obrázek.	Při výhře se vyplácí 8:1
6	Sázka na šestici po sobě jdoucích čísel nebo-li na dvě řady vedle sebe. Žeton se umístí na začátek spojnice mezi těmito řadami	Při výhře se vyplácí 5:1
Vnější sázky		
12	Sázka na 12 čísel může být provedena na celý sloupec čísel. Dále je možná sázka na čtyři řady čísel. Umístěním žetonu na dané pole.	Při výhře se vyplácí 2:1
18	Ještě můžeme sázet na SUDÁ/LICHÁ nebo MALÁ/VELKÁ nebo ČERVENÁ/ČERNÁ čísla umístěním žetonu na příslušné pole.	Při výhře se vyplácí 1:1

Sázky jsou rozděleny na Vnitřní a vnější sázky, u každých platí jiný limit maximální možné vsazené částky. Prozatím je limit vnitřních sázek nastaven na 500Kč, u vnějších sázek je to 15000Kč.

5.1.2 Třída pro hru ruleta

Dalším krokem při vytváření nové hry do systému je napsání třídy ROULETTE, která zdědí vlastnosti třídy GAME, a bude implementovat přesná pravidla hry, které byly stanoveny výše. Tato třída bude použita nejen v Herním serveru, ale bude rozšířena i v klientské aplikaci o síťové připojení. Tato třída tedy nabízí následující rozhraní pro práci a ovládání hry:

Bet(std::string bet) – Vsadit určitou částku peněz na danou kombinaci.
Play() – Roztočení rulety, začít hru.
Limits(double* limits) – Vrací limity sázek
SetLimits(double limits[2]) – Nastaví limity sázek

Systém celé hry zde není složitý, jedná se vlastně pouze o vygenerování náhodného čísla. Pouze zpracovávání sázek a následné vyplácení výher bylo složité a pracné, protože těch kombinací všech různých sázek, na které se dá v této variantě sázet, je opravdu veliký. Ke zpracovávání sázek proto používám hashovací tabulky, které mi výrazně ulehčily práci.

5.1.3 Síťový protokol

K ruletnímu serveru se lze připojit jakýmkoliv klientem, který je schopen posílat textové příkazy. Zde je seznam a podrobný popis všech příkazů (včetně správné syntaxe), které jsou v aktuální verzi tohoto serveru podporovány:

LOGIN	Nick*Heslo
BET	Sázka je výraz: Pole@Částka[*Pole@Částka]*
LIMITS	

LOGIN:

Pokusí se na Hlavním serveru ověřit daného uživatele. Návratový kód přihlášení je vrácen uživateli. V případě úspěchu je vytvořena přihlášenému hráči nová instance třídy ROULETTE, a hráč může nyní hrát.

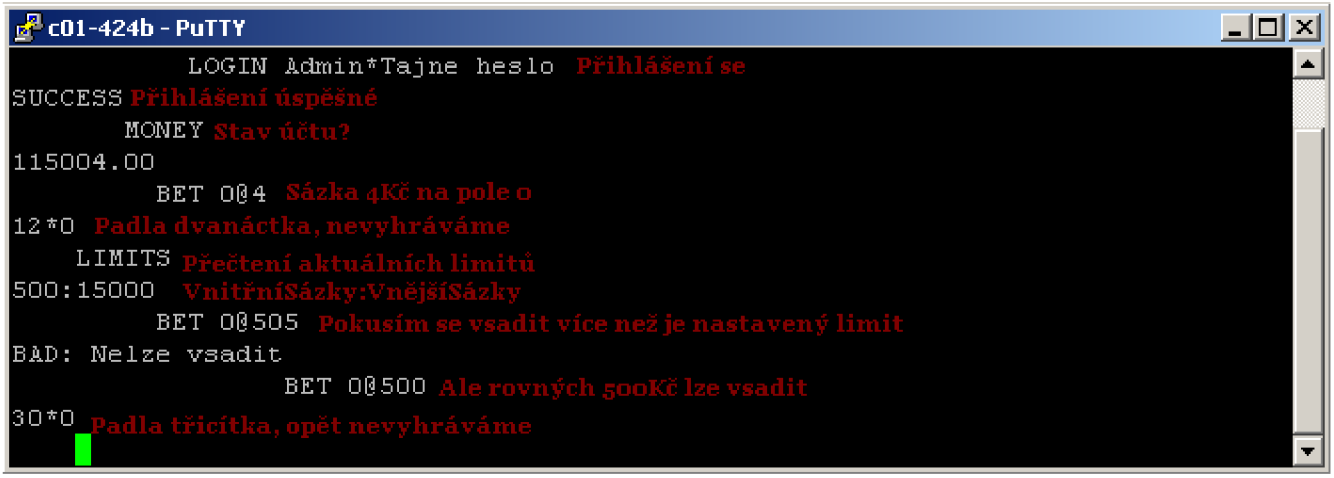
BET:

Zapíší se do hashovací tabulky všechny sázky přečtené z daného výrazu. Poté je hned spuštěna hra, ruleta vygeneruje náhodné číslo v rozmezí 0-36. A hned jsou nazpět poslány výsledky ve tvaru: TaženéČíslo*VyhranáČástka

LIMITS:

Vrátí uživateli aktuální nastavené maximální limity sázek ve tvaru: VnitrniSazky:VnejsiSazky

Dále uvádím ukázku komunikace pomocí tohoto protokolu za použití telnetového klienta PuTTY. Červeně jsou připsány komentáře k jednotlivým příkazům:



5.2 BlackJack

5.2.1 Pravidla této varianty

Je to hra určená pro dva hráče, proti hráči hraje krupiér. Hraje se se čtyřmi standardními balíčky karet zamíchanými do jednoho balíku. Všechny karty se před započetím každé hry rozmíchají. Cílem hráče je dosáhnout součtu bodového hodnocení svých karet co nejblíže a méně než je BlackJack, to je 21. Karty od dvojky do desítky mají stejné bodové hodnocení jako číslo na nich napsané(dvojka je za dva body, trojka je za tři body...), obrázkové karty (J, Q, K) jsou každá za deset bodů a eso (A) může být buď za 1 nebo za 11 bodů (na konci hry se počítá s nejvýhodnější hodnotou). Tato hra se může v průběhu hraní dostat do několika základních stavů.

Hra začíná ve stavu **připraveno**. Limit sázky je prozatím nastaven na 15000Kč. Po přijetí sázky, hra začíná. Nejdříve jsou oběma hráčům rozdány dvě karty, hráčovy karty jsou vidět obě, krupiér odhalí pouze svou první kartu, další karta zůstává prozatím skrytá (otočená lícem dolů). Toto je stav hry **po rozdání karet,** v tomto stavu jsou hráči nabídnuty tři možnosti, jak pokračovat:

Stav po rozdání karet:

HIT: Hráč dobere jednu kartu a dále se hra posune do stavu **dobírání karet.**

STAY: Stačí, hráč nedobere žádnou kartu a předá hru krupiérovi.

DOUBLE: Hráč zdvojnásobí svoji sázku , peníze jsou mu odečteny, a poté se dobere poslední hráčova karta a hned nato již pokračuje ve hře krupiér.

Stav dobírání karet:

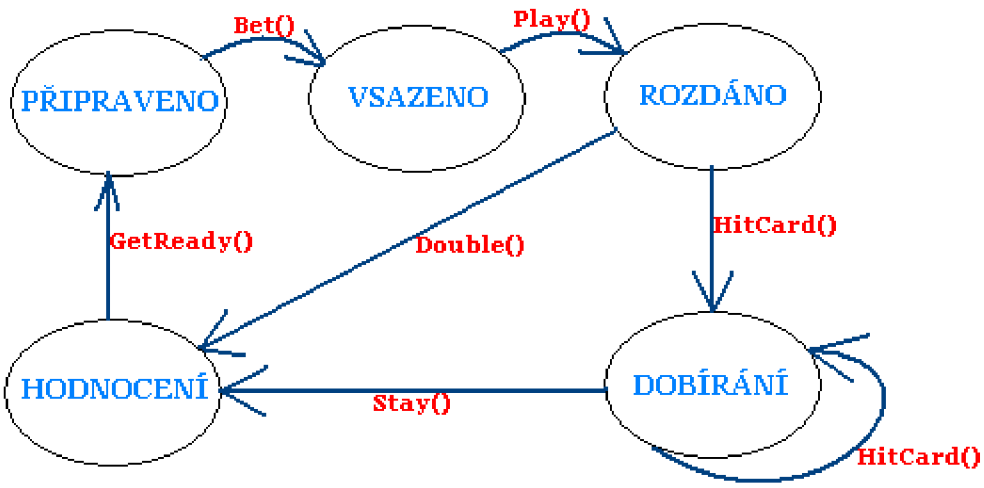
Ve stavu dobírání karet má hráč už pouze dvě možnosti, jak pokračovat, buď dobrat další kartu, nebo stát. Pokud kdykoliv po dobrání karty hráč přebere, tj. součet bodového hodnocení jeho karet přesáhne hodnotu BlackJack (21), hra automaticky končí prohrou a krupiér už další karty nemusí dobírat.

Pokud se hráč v průběhu hry rozhodne nedobírat další karty, přijde na řadu krupiér. Nejprve odhalí svoji druhou kartu. Krupiér má v této fázi hry dvě hlavní omezení (Tato omezení jsou naznačena na plátně anglickým textem „Dealer must draw to 16 and stand on all 17's"). Jedná se o to, že krupiér je povinen dobírat další karty, dokud je jeho bodové hodnocení menší než 16, a v případě, že už je jeho bodové hodnocení vyšší nebo rovno číslu 17, krupiér musí stát. Tato omezení jsou výhodná pro hráče. Např. když hráč má 20 bodů, a krupiér má 18, potom krupiér už nesmí dobrat další kartu! I když je zde malá pravděpodobnost, že by krupiérova následující karta mohla být dvojka nebo trojka. Prostě krupiér při dosažení nebo překročení čísla 17 již dál nepokračuje.

Na konci hry vyhrává hráč, pokud jeho bodové hodnocení je blíže BlackJacku než krupiérovo. Hráči je vyplácena výhra v poměru 1:1 ke vsazené částce. Pokud hráč zdvojnásobil svoji sázku ve stavu **po rozdání karet**, je mu vyplacena pochopitelně dvojnásobná výhra!

5.2.2 Třída pro hru BlackJack

Třída dědí opět všechny vlastnosti i rozhraní bázové třídy GAME. Herní třída pro tuto hru reprezentuje stavový automat nakreslený na obrázku níže:



Na tomto orientovaném grafu jsou červeně napsány příkazy, kterými se přepínáme mezi těmito stavy. Rozhraní této třídy nabízí kompletní práci se všemi stavy hry a se všemi příkazy uvedenými na obrázku. Dále je uvedeno celé rozhraní třídy včetně popisu všech funkcí:

Limit() – Nastaví aktuální limit sázky.
Player() – Vrátí vektor možných bodů hráče.
Dealer() – Vrátí vektor možných bodů krupiéra.
Bet(std::string bet) – Provede se sázka. Hra se přesune do stavu VSAZENO.
Play() – Spuštění hry, rozdání karet. Hra se přesune do stavu ROZDÁNO.
Hit() – Vzít další kartu z balíku, pokud hráč přebere, automaticky prohrává.
Double() – Zdvojnásobení sázky, dobrání karty a předání hry krupiérovi.
Stay() – Předání hry krupiérovi.

Procedury událostí (CALLBACK)
void (*PlayerHit)(std::string Nick, t_card card) – Když hráč dostal jednu kartu.
void (*DealerHit)(std::string Nick, v_card cards) – Když krupiér dobral karty.
void (*CardsTaken)(std::string Nick, v_card card) – Karty byly rozdány.

5.2.3 Síťový protokol

Zde je seznam a podrobný popis všech příkazů (včetně správné syntaxe), které jsou v aktuální verzi tohoto serveru podporovány:

Příkaz	Parametry
LOGIN	Nick*Heslo
BET	Částka
HIT	
DOUBLE	
STAY	
LIMITS	

LOGIN:

Pokusí se na Hlavním serveru ověřit daného uživatele. Návratový kód přihlášení je vrácen uživateli. V případě úspěchu je vytvořena přihlášenému hráči nová instance třídy BLACKJACK, a hráč může nyní hrát.

LIMITS:

Vrátí aktuální nastavený limit pro jednu sázku.

BET:

Zapíší se do hashovací tabulky všechny sázky přečtené z daného výrazu. Poté je hned spuštěna hra, jsou vytaženy čtyři karty z balíku, přičemž nazpět jsou hře zaslány pouze tři karty, protože druhá krupiérova karta je prozatím skrytá. Toto je opatření k tomu, aby uživatel nemohl zjistit, jakou druhou kartu dostal krupiér. Odpověď hernímu klientovi je ve tvaru: **NázevKarty|hodnota[,hodnota]*NázevKarty|hodnota[,hodnota]*NázevKarty|hodnota[,hodnota]**

HIT:

Když je hra ve stavu rozdáno nebo dobírání, je hráčovi vytažena další karta z balíku a ta je poslána zpět hernímu klientovi ve tvaru: **PLAYER_HIT NázevKarty|hodnota[,hodnota]**

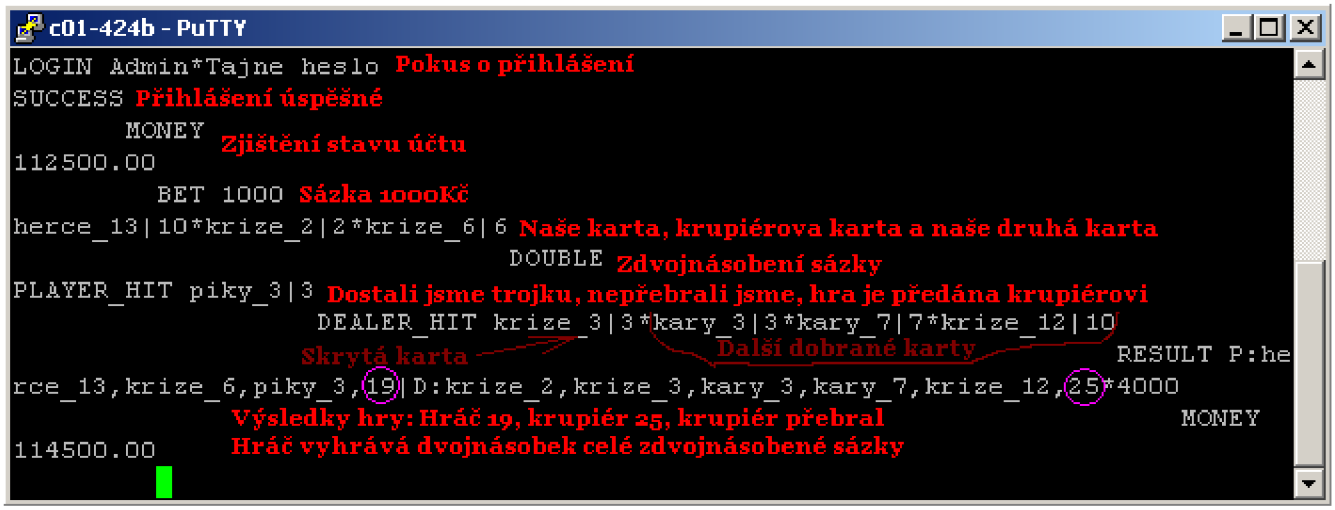
DOUBLE:

Když je hra ve stavu rozdáno, je uživateli zdvojnásobena sázka (pokud má dostatek financí na účtu), a dále rozdána jedna karta a nakonec začne hrát krupiér. Nejdříve je hráči zaslána zpráva o jeho vytažené kartě ve tvaru: **PLAYER_HIT NázevKarty|hodnota[,hodnota]**, a poté je předána hra krupiérovi, který může dobírat karty a nakonec je podána zpráva o ukončení hry, obě poslední zprávy jsou ve stejném tvaru jako následující příkaz.

STAY:

Začne hrát krupiér. Pokud bere nějaké karty, je to oznámeno klientovi ve tvaru:
DEALER_HIT **NázevKarty|hodnota[,hodnota]*[NázevKarty|hodnota[,hodnota]]***
I kdyby krupiér už nedobíral žádnou kartu, musí být tímto příkazem předána minimálně jedna krupiérova karta, to je ta karta, která byla zatím před hráčem skryta.
Poté jsou vráceny klientovi všechny výsledky hry ve tvaru:
RESULT: P: NázevKarty|hodnota[,hodnota][,NázevKarty|hodnota[,hodnota]]*,BodyHráče|D: NázevKarty|hodnota[,hodnota][,NázevKarty|hodnota[,hodnota]]*,BodyKrupiéra*Výhra

Dále uvádím ukázku komunikace pomocí tohoto protokolu za použití telnetového klienta PuTTY. Červeně jsou připsány komentáře k jednotlivým příkazům:



5.3 Šťastné kostky

5.3.1 Pravidla této varianty

Jedná se o jednoduchou a velmi rychlou hru. Hráč hází dvěma kostkami a sází na výsledný součet bodů na obou kostkách. Na herním plátně je nakresleno 11 sázkových polí, každé pro jinou hodnotu. Pravděpodobnost úspěšného tipu není na všech polích stejná, proto i vyplácené výhry jsou na každém poli jiné. Následující tabulka ukazuje, jak veliká výhra je vyplacena, při sázce na příslušné pole:

Výherní číslo	2	3	4	5	6	7	8	9	10	11	12
Vyplacená výhra navíc k sázce (čistý zisk)	34:1	16:1	10:1	7:1	5:1	4:1	5:1	7:1	10:1	16:1	34:1

Poté, co hráč vsadí na libovolnou kombinaci možných sázek, může hodit kostkami. Aktuální limit pro tuto hru je 1000Kč na jedno pole. Hází se pouze jednou a hod se okamžitě vyhodnotí, tj. nevýherní sázky sebere (virtuální) krupiér a případná výherní sázka je vrácena hráči spolu s jeho výhrou.

5.3.2 Třída pro hru Šťastné kostky

Herní třída CUBES může trochu připomínat hru ROULETTE, jelikož ani tato hra nemá žádné stavy, pouze herní funkce Bet() a Play() a dále jsou to funkce Limit(), vracející limit sázek a SetLimit nastavující tento limit. Dále jsou rozepsány příkazy této třídy:

Bet(std::string bet) – Vsadit určitou částku peněz na danou kombinaci.
Play() – Hodit kostkami, začít hru. V této funkci dojde i k vyhodnocení a skončení hry.

Limit(double* Limit) – Vrací se limit maximálních sázek.

SetLimit(double Limit) – Nastavuje limit maximálních sázek.

5.3.3 Síťový protokol

Zde je seznam a podrobný popis všech příkazů (včetně správné syntaxe), které jsou v aktuální verzi tohoto serveru podporovány:

Příkaz	Parametry
LOGIN	Nick*Heslo
BET	Pole@Částka[*Pole@Částka]*
LIMIT	

LOGIN:

Pokusí se na Hlavním serveru ověřit daného uživatele. Návratový kód přihlášení je vrácen uživateli. V případě úspěchu je vytvořena přihlášenému hráči nová instance třídy CUBES, a hráč může nyní začít sázet a hrát.

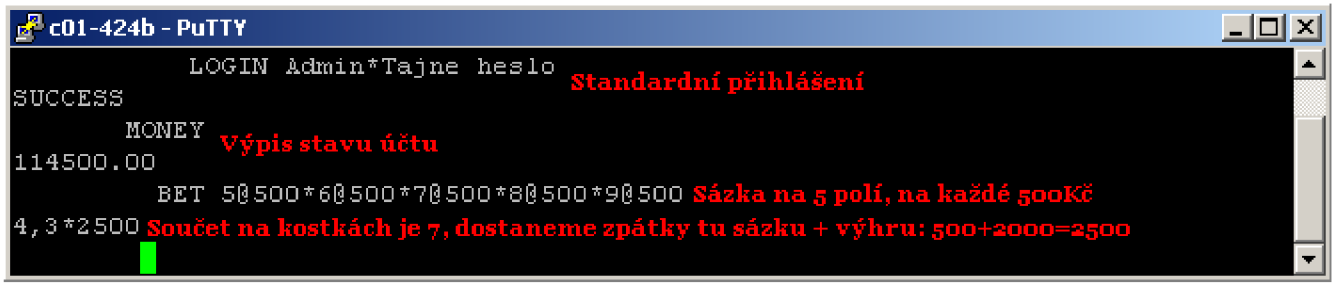
LIMIT:

Vrátí aktuální nastavený limit pro jednu sázku.

BET:

Server přijme sázku, pokud má uživatel na účtu dost peněz, poté se vygenerují dvě čísla, nazpět je klientské aplikaci vrácen výsledek hry ve tvaru: **Číslo1,Číslo2*Výhra**

Dále uvádím ukázku komunikace pomocí tohoto protokolu za použití telnetového klienta PuTTY. Červeně jsou připsány komentáře k jednotlivým příkazům:



5.4 Válka

5.4.1 Pravidla této varianty

Nejdříve hráč položí svoji sázku na stůl. Maximální možná sázka je v této verzi nastavena na 10000Kč. Zamíchají se dva balíčky karet do jednoho a potom jsou vytaženy 2 karty, první dostává hráč, druhou krupiér. Vyhrává ten, kdo má vyšší kartu (nejnižší je dvojka, nejvyšší je ESO).

Pokud mají oba hráči stejnou kartu, hra je ve stavu WAR, a hráč se nyní může rozhodnout, zda půjde s krupiérem do války, či ne. Pokud nechce jít hráč do války, krupiér bere polovinu vsazené částky, druhou polovinu bere zpět hráč. Pokud se hráč rozhodne jít do války, musí to stvrdit další válečnou sázkou (ke své aktuální sázce přidá ještě jednu sázku ve stejné výši, čili jakoby zdvojnásobil svoji první sázku). Dále se zahodí tři karty z balíku a krupiér poté rozdá hráči i sobě poslední kartu. Hráč vyhrává, když bude mít vyšší nebo stejnou kartu jako krupiér. Pokud hráč vyhraje, je mu vrácena válečná sázka, a pouze jeho primární sázka je zdvojnásobena.

Mimo standardní sázky se hráči nabízí ještě sázka na remízu (TIE). Pokud hráč vsadí na remízu, tak hned po rozdání prvních dvou karet se rozhodne o výhře nebo prohře této sázky, a dále potom pokračuje případná válka. V tomto případě se vyplácí výhra v poměru 11:1 ke vsazené částce. Hráč není omezen, na co musí sázet. Může tedy vsadit např. pouze na remízu nebo pouze na standardní hru a nebo na obě tyto sázky naráz.

5.4.2 Třída pro hru Válka

Jsou zde následující 4 stavy, do kterých se může hra dostat:
STATE_READY: Balík karet je rozmíchán a hra může začít. V tomto stavu se dá vsadit.
STATE_BET: Stav po vsazení částky. Může se hra spustit.
STATE_WAR: Oba hráči obdrželi stejnou kartu, hráč se nyní rozhoduje, zda půjde do války.

Dále je zde rozepsáno rozhraní použité třídy:
Limit() – Vrátí maximální povolenou sázku na jednu hru.
Bet(std::string bet) – Vsadit určitou částku peněz na danou kombinaci.
Play() – Spuštění hry. Rozdání karet, a v případě shody je tu vyhlášení války.
War() – Hráč jde do války, musí dát do hry válečnou sázku, poté jsou vytaženy další dvě karty.
NoWar() – Hráč se rozhodl vzdát válku, dostane nazpět pouze polovinu vsazené částky.
Procedury událostí (CALLBACK)
void (*ResultWar)(std::string Nick, t_card card[2], double money) – Funkce spuštěna, když jsou rozdány první dvě karty shodné. Informuje vnější funkce o válečném stavu.
bool (*PlayerBetsWar)(std::string Nick, double money) – Je spuštěna, když hráč chce jít do války.

5.4.3 Síťový protokol

Zde je seznam a podrobný popis všech příkazů (včetně správné syntaxe), které jsou v aktuální verzi tohoto serveru podporovány:

Příkaz	Parametry
LOGIN	Nick*Heslo
BET	Pole@Částka[*Pole@Částka]*
WAR	
NOWAR	
LIMIT	

LOGIN:

Pokusí se na Hlavním serveru ověřit daného uživatele. Návratový kód přihlášení je vrácen uživateli. V případě úspěchu je vytvořena přihlášenému hráči nová instance třídy WAR, a hráč může nyní začít sázet a hrát.

LIMIT:

Vrátí aktuální nastavený limit pro jednu sázku.

BET:

Server přijme sázku, pokud má uživatel na účtu dost peněz, poté se spustí hra. Oba hráči dostanou po jedné kartě a hned se vyhodnotí, zda uživatel něco nevyhrál. Zde jsou možné hned dvě výhry. Když bude mít uživatel vyšší kartu než krupiér, a nebo když vsadil na remízu, a oba hráči dostali stejnou kartu. V obou případech je vrácen stav hry ve tvaru:

RESULT HráčovaKarta,KrupiérovaKarta*Výhra

WAR:

Odečte se hráči válečný poplatek a začíná se druhé kolo hry. Každý hráč dostane poslední kartu a výsledky hry jsou vráceny klientovi ve tvaru:

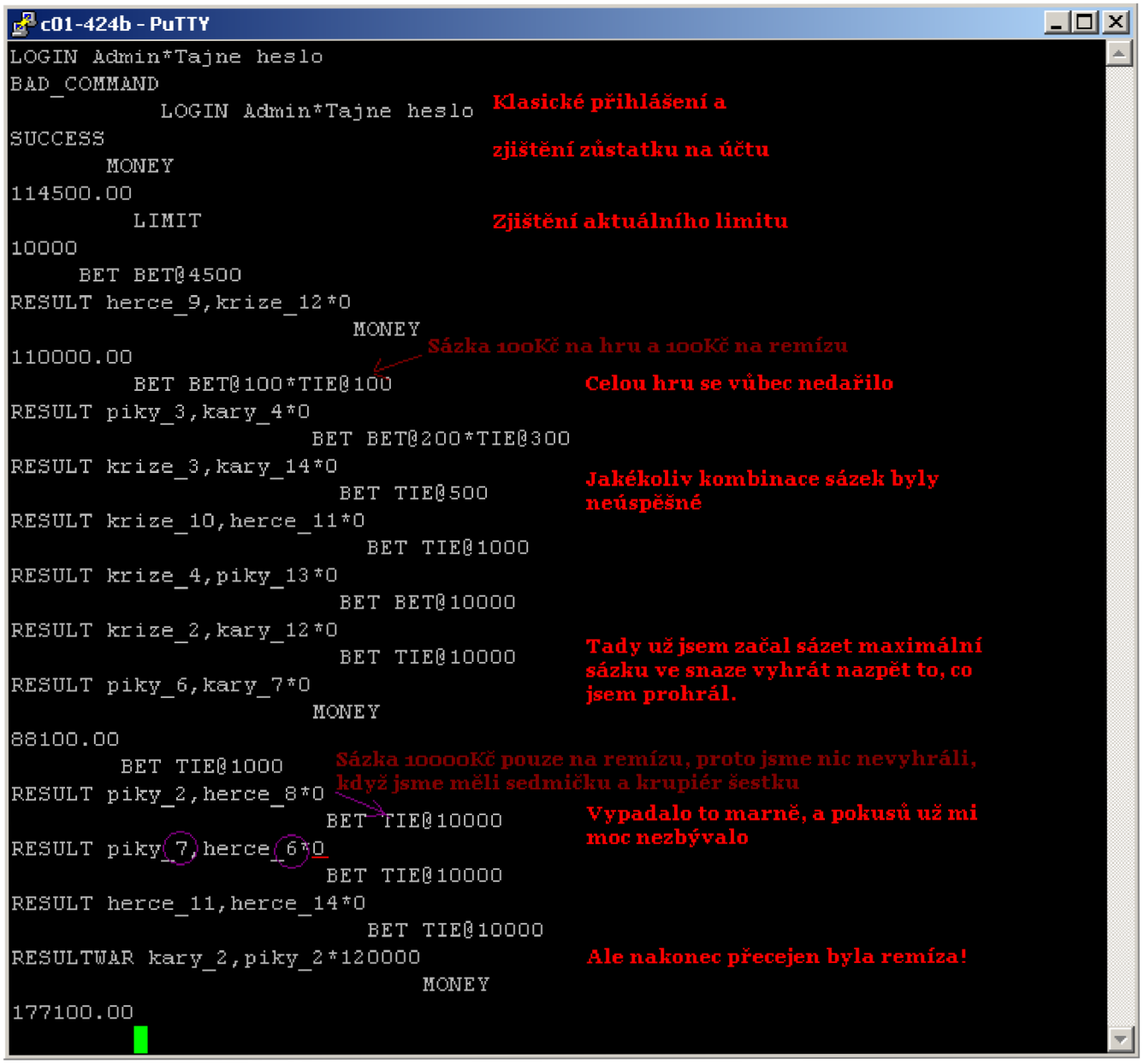
RESULT_WAR HráčovaKarta,KrupiérovaKarta*Výhra

NOWAR:

Hráči bude přičtena zpět polovina vsazené částky, klientovi bude vrácena odpověď ve tvaru:

REUSULT NOWAR*VrácenáČástka

Dále uvádím ukázku komunikace pomocí tohoto protokolu za použití telnetového klienta PuTTY. Červeně jsou připsány komentáře k jednotlivým příkazům. Tato ukázka je trochu delší než ty předcházející, ale zato je z hráčského pohledu nejzajímavější a nejnapínavější:



5.5 Postup instalace

Pro kompilaci zdrojových souborů a následné spuštění některého z herních serverů je potřeba nejdříve mít připravený svůj systém. Databázový systém MySQL spolu s C API konektorem bychom měli už mít nainstalovaný, když jsme spouštěli Hlavní server. Ve Windows použijeme vývojové prostředí Dev-C++ s nainstalovaným balíčkem libmysql, který nám dovolí komunikovat s databází. V Linuxu žádné speciální vývojové prostředí nepotřebujeme, vše zkompilujeme za pomocí souboru Makefile.

5.5.1 Prerekvizity

Pro každý herní server vytvoříme vlastní databázi. Budou sice zatím nést informace pouze o výsledcích jednotlivých her, ale v budoucnu najdou mnohem lepší uplatnění, zejména při vytvoření informačního systému. Pro herní databáze jsem připravil skript, který vytvoří jednu tabulku s výsledky, do níž se budou zaznamenávat všechny odehrané hry.

Před kompilací je potřeba, nastavit správně konstanty v programu. Jsou to 4 databázové konstanty, pomocí nichž se bude spuštěný program připojovat do databáze (**gameROULETTE.cpp**, **gameBLACKJACK.cpp**, **gameCUBES.cpp**, **gameWAR.cpp**). Všechny tyto konstanty nalezneme přibližně na začátku programu, takže by neměl být problém je identifikovat a změnit podle přihlašovacích údajů naší databáze:

64. **DB_HOST** - DATABÁZOVÁ KONSTANTA Jméno počítače
65. **DB_NAME** - DATABÁZOVÁ KONSTANTA Jméno databáze
66. **DB_USER** - DATABÁZOVÁ KONSTANTA Jméno uživatele
67. **DB_PASS** - DATABÁZOVÁ KONSTANTA Heslo uživatele

5.5.2 Kompilace zdrojových souborů

Linux

Pro kompilaci každého serveru zvlášť jsem vytvořil čtyři soubory Makefile, pomocí nichž přeložíme tyto servery. Můžeme použít soubor
Makefile.gameROULETTE pro ruletní server,
Makefile.gameBLACKJACK pro BlackJackový server,
Makefile.gameCUBES pro server Šťastné kostky a také
Makefile.gameWAR pro herní server Válka.
Takže při překládání napíšeme do terminálu příkaz: **make –f [NázevSouboruMakefile]**

Windows

Ve Windows se projekt přeloží ve vývojovém prostředí Dev-C++, celý projekt je uložený a připravený v souborech gameROULETTE.dev, gameBLACKJACK.dev, gameCUBES.dev a gameWAR.dev. Ještě zkontrolujeme, zda máme nainstalovanou knihovnu libmysql (**Nástroje->Package Manager**). Pokud se v seznamu tato knihovna nenachází, je potřeba si ji před kompilací stáhnout a nainstalovat, stačí spustit **Nástroje -> Zjistit updaty**, zde si zvolíme nějaký funkční server a z něj si stáhneme a nainstalujeme balíček **libmysql** v aktuální verzi.

Nyní již přistoupíme konečně ke kompilaci. V menu **Spustit** zmáčkneme **Zkompilovat**, a projekt se přeloží do spustitelného EXE souboru.

5.5.3 Spouštění serverů

Před spuštěním jakéhokoliv Herního serveru je potřeba, aby byl spuštěný Hlavní server.

Syntaxe pro spuštění Herního serveru z příkazové řádky je pro všechny tyto servery stejná:

gameSERVER LocalPort mainSERVER MainPort

kde *gameSERVER* je název spustitelného souboru serveru,

LocalPort je číslo portu, na kterém server bude naslouchat příchozí připojení,

mainSERVER je adresa pro připojení k Hlavnímu serveru a

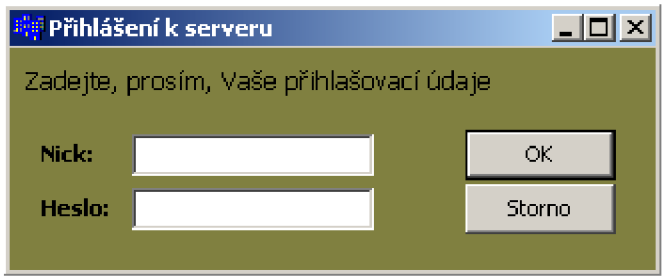
MainPort je port, na kterém poslouchá Hlavní server.

V aktuální verzi serverů není možno za běhu programu psát příkazy přímo do konzoly serveru. Prozatím se mi nepodařilo nijak zjistit, jak odchytávat stlačené klávesy. Ale při dalším vývoji tento nedostatek určitě napravím.

5.6 Klientské aplikace

Je to pouze uživatelské prostředí pro klientské aplikace. K hernímu serveru by se dalo připojit i pomocí telnetu, ale to by pro hráče nebylo nejpříjemnější prostředí. Herní aplikace jsou spustitelné pouze v systému Windows. Jsou psány programovacím jazykem C++ a k jejich vývoji jsem používal vývojové prostředí Borland C++ Builder. Všechny grafické prvky jsem namaloval použitím programu Microsoft Paint. Ostatní obrázky byly zdarma ke stažení na internetu. Ovládání všech her je na první pohled intuitivní, ale kdyby hráč náhodou neznal nějakou funkci programu, nebo potřebovat si přečíst návod, pro tyto případy jako přílohy přikládám uživatelské manuály pro všechny 4 hry.

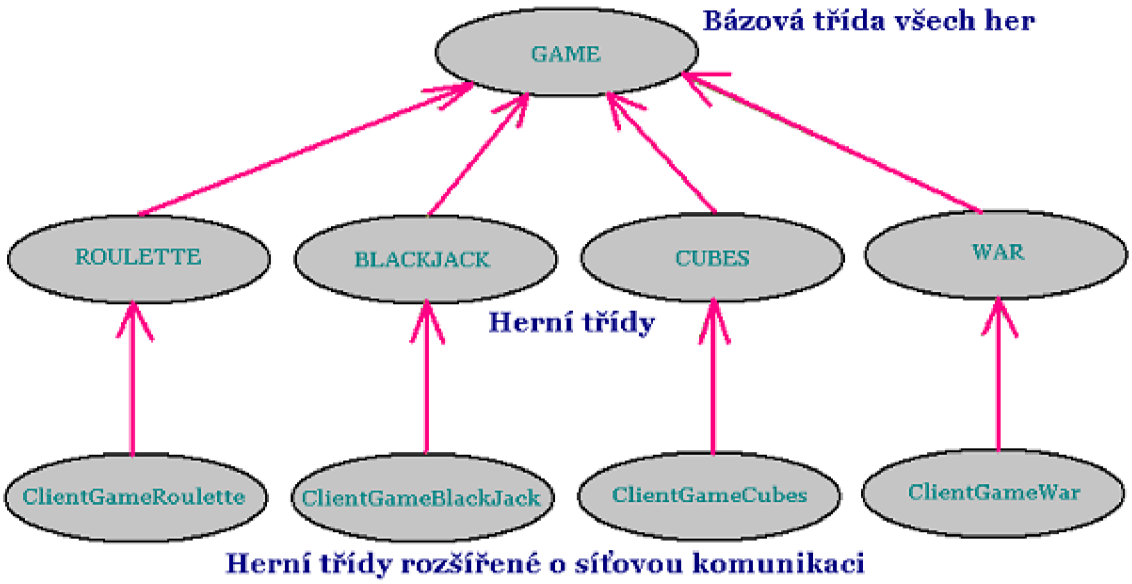
Pro všechny klienty je zde společné jedno přihlašovací okno, do kterého napíší svoje přihlašovací jméno a heslo, a poté potvrdí tlačítkem OK, nebo zruší tlačítkem Storno. Toto okno se objeví pouze v případě běžícího herního serveru. Žádné další společné prvky tyto aplikace nemají, pouze každý klient v sobě obsahuje novou herní třídu, kterou pouze rozšíří o síťovou komunikaci. Následuje obrázek přihlašovacího okna:



Nepoužívám operační systém Windows XP, ale Windows 2000, proto je vzhled okna u mě takovýto. Vzhled oken se liší pochopitelně podle verze operačního systému.

V adresáři s hrou se nachází dva konfigurační soubory pro připojení klienta k serveru. Jedná se o soubory **server** a **port**, v nich jsou uloženy informace o tom, na jakém počítači a na kterém portu běží herní servery pro danou hru Tyto soubory je nutné mít správně nakonfigurovány, jinak nebude možno se připojit k hernímu serveru.

Po úspěšném přihlášení si můžeme zahrát o své peníze, které máme uložené na účtu v Hlavní databázi. Pokud nejsme připojeni k Hernímu serveru, hra nám nabízí zahrát si i přes to, jen tak zkušebně. Při každé zkušební hře obdržíme 5000Kč, o které se ale nehraje ve skutečnosti, ale pouze cvičně. Dále uvádím UML diagram všech herních tříd:



Jediné omezení má klient rulety, a to je, že uživatelský počítač musí mít nastavené rozlišení obrazovky minimálně 1280×800, jinak se celá ruleta nevejde na plochu, i přes to má ruletní klient nejlepší hratelnost a vzbudil u hráčů veliký obdiv, díky jeho grafickému rozhraní se stala tato hra nejoblíbenější hrou celého systému.

Nezůstanu pouze u těchto čtyř her, celý systém je stavěný pro přidávání nových a nových her, a čím více her budou mít hráči na výběr, tím budou spokojenější, a tím také bude mít kasino větší výdělky. Jedním z dalších typů her, které hodlám přidat do systému, jsou výherní automaty a v neposlední řadě to bude dnes velmi oblíbený poker. Fantazii se meze nekladou, a je možno jednoduše přidávat jakoukoliv hru, která bude mému kasinu vynášet další zisk.

Závěr

Podle mého názoru nebyl tento projekt náročný na použité složité technologie, ale zato byl velmi časově náročný. Spousta modulů v tomto systému musela být zkoušena odděleně, zejména síťová komunikace musela být bezproblémově funkční, protože na ní stavěl celý systém. Pro testování síťové komunikace jsem navrhnul chatovací program. Moje síťová knihovna je dále využitelná i pro jiné aplikace než je pouze tento projekt, díky své přenositelnosti je použitelná prakticky všude. Dále každou herní třídu jsem jednotlivě otestoval v novém programu, abych měl jistotu, že tyto komponenty fungují správně.

Grafické rozhraní jednotlivých klientských aplikací nebylo sice stěžejním bodem mé práce, ale i přesto bych rád všechna tato uživatelská rozhraní v budoucnu převedl do lepšího grafického provedení, např. použitím OpenGL. Klientské programy jsem nechal testovat na několika uživatelům a alespoň trochu jsem se snažil upravit tento prozatímní vzhled, aby se uživatelům líbil.

Myslím si ale že projekt jako celek se mi podařil dopracovat do takové fáze, aby mohl být spuštěn a otestován nějakými dalšími hráči. Herní systém a pravidla jednotlivých her jsem se snažil navrhnout , aby byl takový, jak jsou hráči zvyklí.

V tomto projektu hodlám pokračovat i nadále. Především je nutné rozšířit svoji síťovou knihovnu o zabezpečené připojení pomocí SSL. Dále se do systému budu snažit zaintegrovat informační systém, který by měl výrazně ulehčit práci s Hlavním serverem, s uživatelskými účty i s herními servery. Určitě se budu zaměřovat na vylepšení databázové části všech serverů. Opomenul jsem v systému zabezpečení a zálohování dat. Přece jen se jedná o velmi citlivá uživatelská data, takže by bylo vhodné nějakým způsobem zabránit případnému zneužití těchto informací.

Přidávání nových her do systému je jednoduché, proto nechci zůstat pouze na těchto čtyřech hrách, chtěl bych hráčům nabídnout široký výběr, aby se necítili omezeni pouze na pár her. Další vhodnou hazardní hrou budou výherní automaty. V neposlední řadě, velmi oblíbenou hazardní hrou je poker, je to sice karetní hra pro více hráčů, ale již existují různé automaty, které hrají proti hráči, a které nabízí spoustu herních variant. Samozřejmostí musí opět být to, že automat bude mít v pravidlech malinkou výhodu, a tím bude tento automat pro kasino výdělečný.

Nakonec bych ještě dodal pár poznámek z osobní zkušenosti s kasiny jak ve skutečnosti tak na internetu. Zkoušel jsem různá kasina na internetu, většinou jsem hrál pouze ruletu. Potvrdilo se mi, že když má člověk opravdu nějaký systém, tak ve většině případů skutečně vyhraje a odnese si s sebou i nějakou výhru. Ale bohužel je touto menší výhrou „ohromen“ natolik, že se do kasina vrátí. Bude se tam vracet tak dlouho, dokud konečně o všechny výhry nepřijde, a dokud neprohraje ještě mnohem více, než vyhrál. Proto se musíme mít na pozoru před lákavými nabídkami od jakýchkoliv kasin. Jakákoliv bonusová akce, nebo peníze navíc, jsou pouze tímto trikem, jak přilákat více hráčů.

Literatura

[1] Dostál Radim. *Sokety a C++*, 4. 11. 2002. Dokument dostupný na URL http://www.builder.cz/art/cpp/sokety_a_cpp.html (květen 2008)

[2] Dostál Radim. *TCP klient v Linuxu*, 20. 12. 2002. Dokument dostupný na URL http://www.builder.cz/art/cpp/tcp_klient_linux.html (květen 2008)

[3] Dostál Radim. *TCP klient v MS Windows*, 3. 1. 2003. Dokument dostupný na URL http://www.builder.cz/art/cpp/tcp_klient_windows.html (květen 2008)

[4] Dostál Radim. *TCP server v Linuxu*, 17. 1. 2003. Dokument dostupný na URL http://www.builder.cz/art/cpp/tcp_server_linux.html (květen 2008)

[5] Dostál Radim. *TCP server v MS Windows*, 24. 1. 2003. Dokument dostupný na URL http://www.builder.cz/art/cpp/tcp_server_windows.html (květen 2008)

[6] Dostál Radim. *Protokol UDP 1.část*, 3. 2. 2003. Dokument dostupný na URL http://www.builder.cz/art/cpp/udp1.html (květen 2008)

[7] Dostál Radim. *Protokol UDP 2.část*, 20. 2. 2003. Dokument dostupný na URL http://www.builder.cz/art/cpp/udp2.html (květen 2008)

[8] Dave Marshall. *Remote Procedure Calls (RPC)*, 1. 5. 1999. Dokument dostupný na URL http://www.cs.cf.ac.uk/Dave/C/node33.html (květen 2008)

[9] *Remote procedure call*, 1. 4. 2008. Dokument dostupný na URL http://en.wikipedia.org/wiki/Remote_procedure_call (květen 2008)

[10] Zbyněk Křivka, Dušan Kolář. *Principy programovacích jazyků a objektově orientovaného programování IPP – II Studijní opora*

[11] Pavel Tišnovský. *Nástroje pro tvorbu UML diagramů*, 4. 7. 2005. Dokument dostupný na URL http://www.root.cz/clanky/nastroje-pro-tvorbu-uml-diagramu (květen 2008)

[12] pavus. *OO, UML, analýza, metodologie*, 18. 7. 2006. Dokument dostupný na URL http://mpavus.wz.cz/index.php (květen 2008)

[13] *MySQL 5.0 Reference Manual*, 11. 5. 2008. Dokument dostupný na URL http://dev.mysql.com/doc/refman/5.0/en/index.html (květen 2008)

Použitý software

<1> CentOS 5.1

<2> Microsoft Windows 2000 Professional SP4

<3> Dev-C++

<4> Borland C++ Builder 2006

<5> KWrite

<6> PSPad

<7> Microsoft Paint – Malování

<8> Microsoft Word 2003

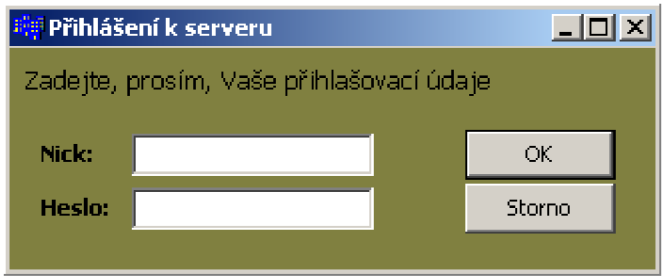
<9> Microsoft Excel 2003

<10> PuTTY

Seznam příloh

Příloha 1. Uživatelský manuál pro hru Ruleta

Vedle spustitelného souboru jsou i dva soubory s konfigurací připojení k serveru. Jsou to soubory **server** a **port**, ve kterých by měl být zapsán DNS název serveru a port, na kterém poslouchá příchozí připojení. V případě, že jsou tyto soubory správně nakonfigurovány, a server je spuštěn, objeví se po spuštění hry přihlašovací obrazovka, který Vás vyzívá k zadání uživatelských údajů.



Do pole **Nick** prosím zadejte Vaše přihlašovací jméno. Do pole **Heslo** zadejte Vaše aktuální heslo pro přístup k Vašemu účtu. Tyto Vaše přihlašovací údaje potvrdíte stiskem tlačítka OK.
V případě, že si chcete zahrát pouze zkušební hru nanečisto, můžete stisknout Storno, nebudete přihlášeni k serveru, a dostanete 5000Kč, se kterými si můžete zahrát. Tato zkušební hra se nebude nijak započítávat do Vašeho skutečného účtu.

Tato hra má zatím omezení na klientský počítač. Je potřeba, abyste měli rozlišení monitoru nastaveno minimálně na 1280×800, jinak se Vám hra nevejde na obrazovku. Po spuštění hry uvidíte následující grafické rozhraní:

Dále popíši, k čemu slouží jednotlivá tlačítka, jak se hra obsluhuje.

V tomto modrém poli se zobrazuje aktuální zůstatek peněz na účtu.


Aktuální žeton v ruce. Uživateli je zde nabídnuto šest žetonů, každý v jiné peněžní hodnotě (1, 5, 10, 25, 50 a 100Kč). Vybraný žeton je pokládán na hrací pole.

Při přejíždění sázkových polí myší, jsou označeny vždy ta čísla, kterých se dané pole týká, aby měl hráč lepší přehled o tom, na co konkrétně sází.
Při položení jednoho žetonu na hrací pole je hráči odečtena příslušná částka a tato částka je přesunuta v podobě sázky na vsazené pole. Vsazení na nějaké sázkové pole se provede pomocí levého tlačítka myši. Pokud chceme nějakou sázku ze sázkového pole zrušit, je to možno pravým tlačítkem myši.

Tlačítko Storno, zruší všechny sázky položené na stole. Všechny peníze jsou připsány zpět hráči.

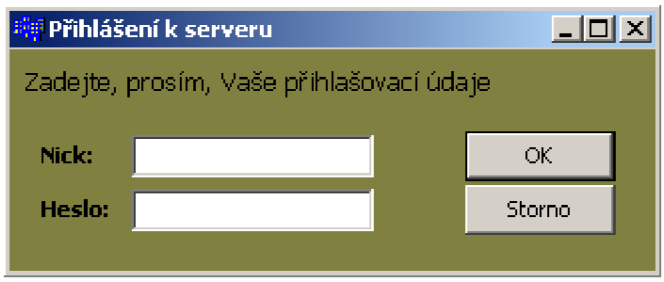

Start hry. Stiskněte toto tlačítko, až máte všechny sázky vsazené, pro roztočení rulety. Následuje vytočení čísla v rozmezí 0-36 a případné výhry jsou hned vyplaceny hráči.

Po vylosování jednoho čísla je hlasem krupiérky oznámeno tažené číslo.


Tlačítko konec. Opuštění této hry.

Příloha 2. Uživatelský manuál pro hru BlackJack

Vedle spustitelného souboru jsou i dva soubory s konfigurací připojení k serveru. Jsou to soubory **server** a **port**, ve kterých by měl být zapsán DNS název serveru a port, na kterém poslouchá příchozí připojení. V případě, že jsou tyto soubory správně nakonfigurovány, a server je spuštěn, objeví se po spuštění hry přihlašovací obrazovka, který Vás vyzívá k zadání uživatelských údajů.



Do pole **Nick** prosím zadejte Vaše přihlašovací jméno. Do pole **Heslo** zadejte Vaše aktuální heslo pro přístup k Vašemu účtu. Tyto Vaše přihlašovací údaje potvrdíte stiskem tlačítka OK.
V případě, že si chcete zahrát pouze zkušební hru nanečisto, můžete stisknout Storno, nebudete přihlášeni k serveru, a dostanete 5000Kč, se kterými si můžete zahrát. Tato zkušební hra se nebude nijak započítávat do Vašeho skutečného účtu.

Po spuštění hry uvidíte následující grafické rozhraní:

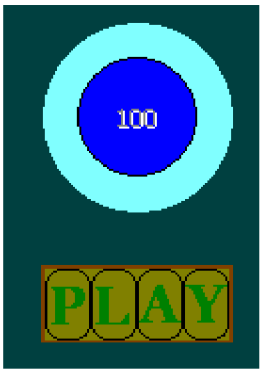
Dále popíši, k čemu slouží jednotlivá tlačítka, jak se hra obsluhuje.

V tomto modrém poli se zobrazuje aktuální zůstatek peněz na účtu.


Aktuální žeton v ruce. Uživateli je zde nabídnuto šest žetonů, každý v jiné peněžní hodnotě (1, 5, 10, 25, 50 a 100Kč). Vybraný žeton je pokládán na hrací pole.

Pole, na které hráč umísťuje své sázky

Při položení jednoho žetonu na hrací pole je hráči odečtena příslušná částka a tato částka je přesunuta v podobě sázky na toto sázkové pole. Vsazení se provede pomocí levého tlačítka myši. Pokud chceme sázku ze sázkového pole zrušit, je to možno pravým tlačítkem myši.



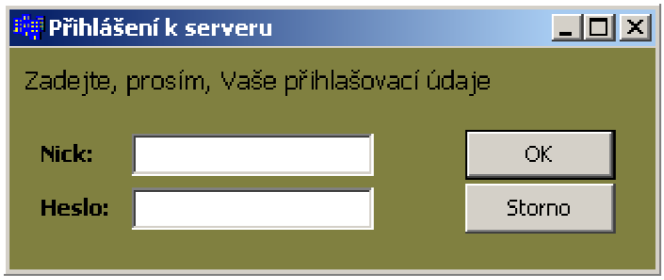
Po umístění sázky na sázkové pole se objeví tlačítko pro spuštění hry. Je to výzva pro krupiéra,že je vše připraveno a vsazeno, a aby již rozdal karty a začal hru.

Když začne nová hra, jsou hráči i krupiérovi rozdány dvě karty, hráčovy karty jsou vidět obě, krupiérova druhá karta zůstává skrytá. Vlevo od karet je vidět, jaké bodové hodnocení mají hráčovy nebo krupiérovy karty. Pro zdvojnásobení sázky stiskněte tlačítko **Zdvojnásobit sázku**, pro dobrání další karty stiskněte **Vzít Kartu**. V případě, že již nechcete dobírat karty, stiskněte tlačítko **Stačí**. Na konci hry je hlasem krupiérky oznámen výsledek hry.


Tlačítko konec. Opuštění této hry.

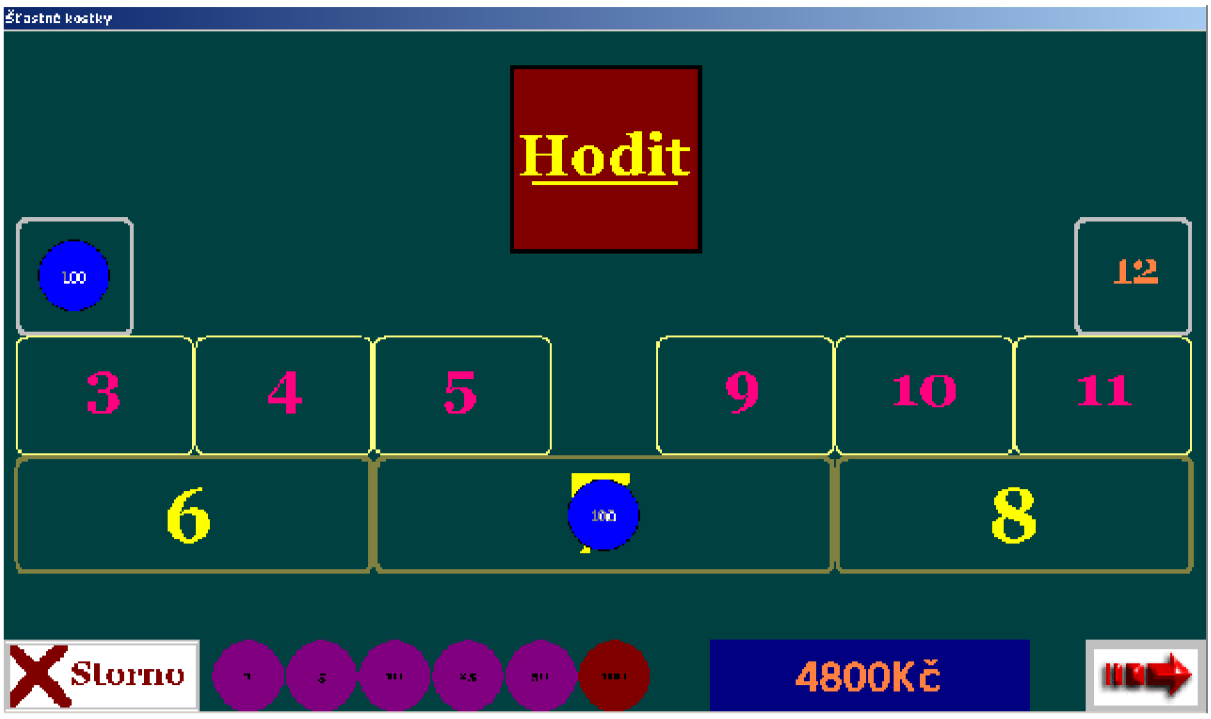
Příloha 3. Uživatelský manuál pro hru Šťastné kostky

Vedle spustitelného souboru jsou i dva soubory s konfigurací připojení k serveru. Jsou to soubory **server** a **port**, ve kterých by měl být zapsán DNS název serveru a port, na kterém poslouchá příchozí připojení. V případě, že jsou tyto soubory správně nakonfigurovány, a server je spuštěn, objeví se po spuštění hry přihlašovací obrazovka, který Vás vyzívá k zadání uživatelských údajů.



Do pole **Nick** prosím zadejte Vaše přihlašovací jméno. Do pole **Heslo** zadejte Vaše aktuální heslo pro přístup k Vašemu účtu. Tyto Vaše přihlašovací údaje potvrdíte stiskem tlačítka OK.
V případě, že si chcete zahrát pouze zkušební hru nanečisto, můžete stisknout Storno, nebudete přihlášeni k serveru, a dostanete 5000Kč, se kterými si můžete zahrát. Tato zkušební hra se nebude nijak započítávat do Vašeho skutečného účtu.

Po spuštění hry uvidíte následující grafické rozhraní:



Dále popíši, k čemu slouží jednotlivá tlačítka, jak se hra obsluhuje.

V tomto modrém poli se zobrazuje aktuální zůstatek peněz na účtu.


Aktuální žeton v ruce. Uživateli je zde nabídnuto šest žetonů, každý v jiné peněžní hodnotě (1, 5, 10, 25, 50 a 100Kč). Vybraný žeton je pokládán na hrací pole.

Hráč sází na pole označená čísly 2 – 12, jedná se o sázku na součet bodů na oboukostkách.
Při položení jednoho žetonu na hrací pole je hráči odečtena příslušná částka a tato částka je přesunuta v podobě sázky na vsazené pole. Vsazení na nějaké sázkové pole se provede pomocí levého tlačítka myši. Pokud chceme nějakou sázku ze sázkového pole zrušit, je to možno pravým tlačítkem myši.

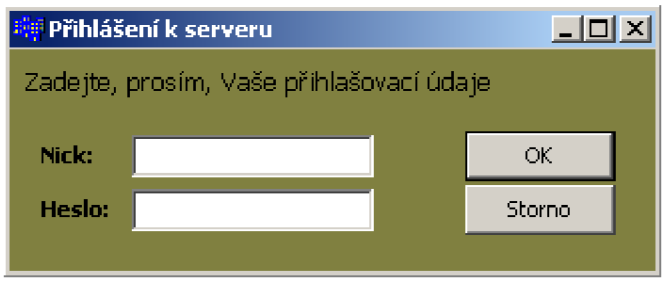
Tlačítko Storno, zruší všechny sázky položené na stole. Všechny peníze jsou připsány zpět hráči.

Start hry. Stiskněte toto tlačítko, až máte všechny sázky vsazené. Následuje hození kostkami a případné výhry jsou hned vyplaceny hráči. Po hození kostkami je hlasem krupiérky oznámen výsledek hry.


Tlačítko konec. Opuštění této hry.

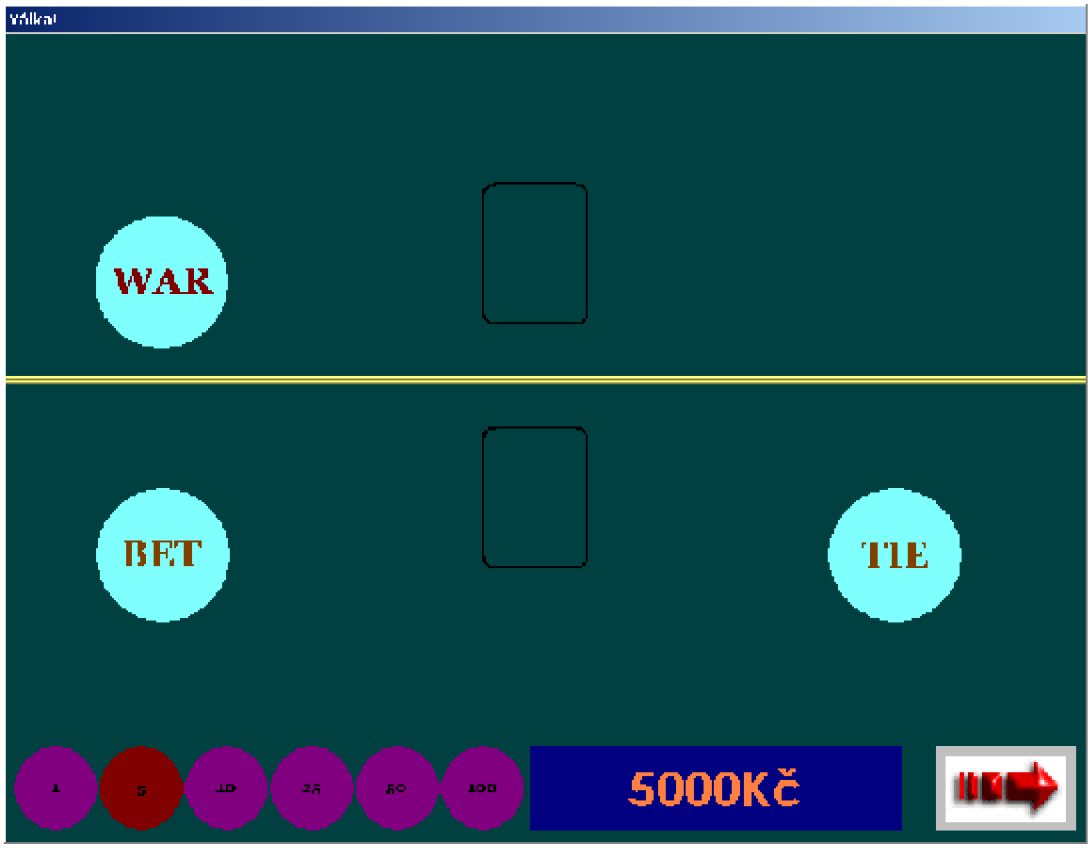
Příloha 4. Uživatelský manuál pro hru Válka

Vedle spustitelného souboru jsou i dva soubory s konfigurací připojení k serveru. Jsou to soubory **server** a **port**, ve kterých by měl být zapsán DNS název serveru a port, na kterém poslouchá příchozí připojení. V případě, že jsou tyto soubory správně nakonfigurovány, a server je spuštěn, objeví se po spuštění hry přihlašovací obrazovka, který Vás vyzívá k zadání uživatelských údajů.



Do pole **Nick** prosím zadejte Vaše přihlašovací jméno. Do pole **Heslo** zadejte Vaše aktuální heslo pro přístup k Vašemu účtu. Tyto Vaše přihlašovací údaje potvrdíte stiskem tlačítka OK.
V případě, že si chcete zahrát pouze zkušební hru nanečisto, můžete stisknout Storno, nebudete přihlášeni k serveru, a dostanete 5000Kč, se kterými si můžete zahrát. Tato zkušební hra se nebude nijak započítávat do Vašeho skutečného účtu.

Po spuštění hry uvidíte následující grafické rozhraní:



Dále popíši, k čemu slouží jednotlivá tlačítka, jak se hra obsluhuje.

V tomto modrém poli se zobrazuje aktuální zůstatek peněz na účtu.


Aktuální žeton v ruce. Uživateli je zde nabídnuto šest žetonů, každý v jiné peněžní hodnotě (1, 5, 10, 25, 50 a 100Kč). Vybraný žeton je pokládán na hrací pole.

Pole, na které hráč umísťuje své sázky, může být buď sázka na hru (BET), nebo sázka na remízu (TIE).
Při položení jednoho žetonu na hrací pole je hráči odečtena příslušná částka a tato částka je přesunuta v podobě sázky na toto sázkové pole. Vsazení se provede pomocí levého tlačítka myši. Pokud chceme sázku ze sázkového pole zrušit, je to možno pravým tlačítkem myši.

Po vsazení na jakékoliv sázkové pole se objeví toto tlačítko **Play**. Stiskem tohoto tlačítka spustíte hru. Budou rozdány karty a hlas krupiérky Vám hned poví, zda vyhráváte.

V případě války položte na toto pole válečnou sázku.

V případě, že nechcete jít do války, můžete hru ukončit tímto tlačítkem **Vzdát**.


Tlačítko konec. Opuštění této hry.