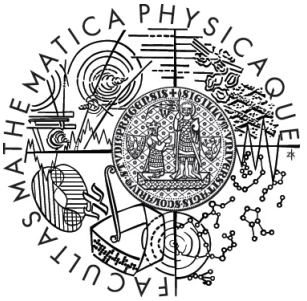
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Marek Vlk

Hazardní hra ruleta

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Michal Malohlava
Studijní program: Informatika, Obecná informatika

2010

Děkuji panu Michalu Malohlavovi za odborné vedení mé práce, za rady, které mi poskytl, a především za to, že mi umožnil věnovat se tomuto tématu. Dále děkuji Dmytru Nikitinovi, studentovi fakulty architektury, za tvorbu grafických modelů.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 4. 8. 2010 Marek Vlk

Obsah

Název práce: Hazardní hra ruleta
Autor: Marek Vlk
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Michal Malohlava, Katedra distribuovaných a spolehlivých systémů
e-mail vedoucího: Michal.Malohlava@d3s.mff.cuni.cz

Abstrakt: Tato práce se věnuje hazardní hře ruletě. Jejím stěžejním cílem je navrhnout a implementovat tuto hru. Aplikace poskytuje dva režimy hraní: „cvičná hra“ pracující off-line a „skutečná hra“, která umožňuje hru o více hráčích on-line. Práce se zaměřuje na rychlost a spolehlivost síťové komunikace i na uživatelsky přívětivé 3D prostředí kasina. V programu je dále umožněno hraní a simulování herních systémů, které jsou implementovány, a lze importovat vlastní. Aplikace je napsána v jazyce C# pro platformu Windows. Síťová komunikace je uskutečněna pomocí technologie .NET Remoting.

Klíčová slova: Ruleta, Herní systémy, .NET Remoting

Title: Roulette Gambling
Author: Marek Vlk
Department: Department of Software Engineering
Supervisor: RNDr. Michal Malohlava, Department of Distributed and Dependable Systems
Supervisor's e-mail address: Michal.Malohlava@d3s.mff.cuni.cz

Abstract: This work goes into roulette gambling. Its pivotal intension is to design and implement this game. The application provides two playing modes: "training game" working off-line and "real game" that performs multi-player game on-line. The work takes a focus on speed and reliability of network communication and on user-friendly 3D casino environment. Furthermore, the program provides playing and simulation of game systems implemented in the program and allows importing own system. The application is written in C# language for Windows platform. Network communication is realized via .NET Remoting.

Keywords: Roulette, Game systems, .NET Remoting

Kapitola 1

Úvod

Kouzlo rotujícího kola, typický kovový zvuk dopadající kuličky, hedvábný zelený kobereček a decentní chrastění žetonů láká a nikdy lákat nepřestane masy lidí. Vidina snadného a velkého finančního zisku přebíjí obavy z rychlého a ještě snadnějšího bankrotu. Mnoho lidí mělo to štěstí a vyhrálo obrovskou hromadu peněz. Mnoha lidem však ruleta zničila život.

Myšlenka hazardních her má své kořeny v 18. století v Londýně a v Paříži [6]. Narozeniny evropské rulety, jak ji známe dnes, lze však přisuzovat až roku 1843, kdy Francois Blanc spolu se svým bratrem Louisem otevřeli v německém Bad Homburgu luxusní hernu Kursaal. Milníkem v historii rulety se stal rok 1863, kdy Francois přenesl své podnikání do slavného (tehdy ovšem bezvýznamného) Monte Carla. Během patnácti let proměnil tuto hernu v nejslavnější kasino a ruleta se začala šířit po celém světě.

1.1 Cíle

Náplní této práce je vytvořit aplikaci, která umožní hrát ruletu jak v režimu off-line, tak v režimu on-line pro více hráčů. Při hraní on-line bude cílem, aby sázky byly umísťovány ihned, bez jakékoli latence, ale na druhou stranu spolehlivě a bezpečně.

Všechny rulety dostupné na webu stanovují limity sázek, a to i v cvičných režimech. Od našeho programu tedy budeme požadovat, aby umožňoval nastavit vlastní sázkové limity a vstupní kapitál. Zároveň budou implementovány vybrané herní systémy. Mnoho lidí věří, že herní systémy jim pomohou v ruletě vyhrát. Aniž by si cokoli spočítali nebo vyzkoušeli, jdou hrát o skutečné peníze. Možnost si daný systém odsimulovat a vidět výsledek by jistě změnila jejich přístup. To

je motivace pro umožnění simulace systémů a jejich hraní přímo v programu.

V neposlední řadě je cílem vytvořit uživatelsky přívětivé prostředí, které hráčům poskytne veškerý herní komfort.

Konkrétní cíle práce jsou:

1. Síťová komunikace: rychlost, spolehlivost, bezpečnost.
2. Hraní a simulace herních systémů.
3. 3D herní prostředí.

1.2 Struktura

Druhá kapitola popisuje, jak se hraje ruleta a co jsou vůbec herní systémy. Třetí kapitola rozebírá nástroje, které budou použity pro tvorbu aplikace. Kapitola čtyři pojednává o problémech, kterým je třeba čelit, a navrhuje jejich řešení. Uživatelská a programátorská dokumentace je umístěna v páté a šesté kapitole a porovnání s podobnými programy se nachází v kapitole sedmé. Poslední kapitola shrnuje práci a navrhuje možná vylepšení.

V následujícím textu budeme pojmem Ruleta (s velkým R) označovat tuto aplikaci a budeme-li se bavit o hazardní hře obecně, budeme jí říkat ruleta (s malým r).

Kapitola 2

Pravidla a herní systémy

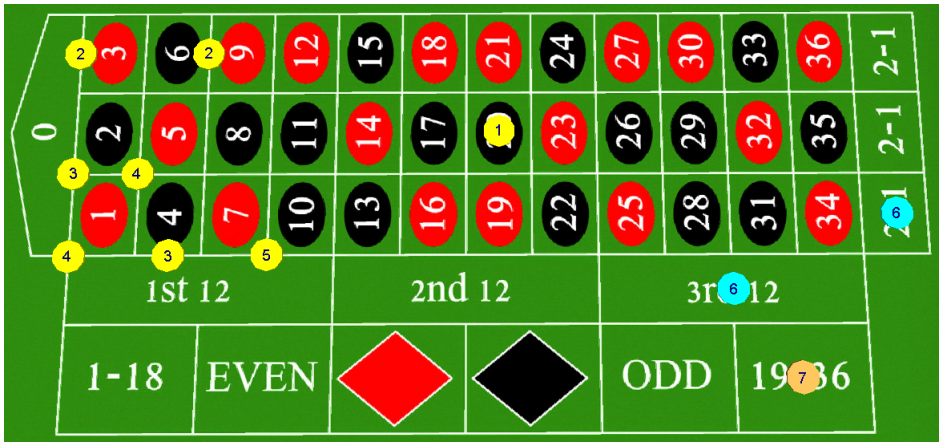
2.1 Pravidla rulety

Otáčivé zařízení rulety obsahuje čísla od 0 do 36 (americká ruleta má navíc tzv. dvojnulu, čímž ještě zvyšuje svou ziskovost, proto se jí nebudeme zaobírat). Hráči umísťují své sázky na herní plátno. Krupiér roztočí kolo a vhodí do něj kuličku. Číslo, u něhož přistane, se stává výherním a podle tohoto čísla jsou vypláceny výhry. Jednomu otočení rulety a padnutí čísla se říká spin.

Možnosti sázkových příležitostí jsou v následujícím seznamu [7, 8] odkazujícím na polohy znázorněné na obrázku 2.1. Jestliže se příležitost vztahuje na n čísel, potom je případná výhra vyplacena s kurzem $\frac{36}{n}$.

1. Sázka na číslo (straight up, plein)
2. Rozdělená sázka (split, cheval): 2 čísla
3. Sázka na uličku (street, transversale plein): 3 čísla
4. Roh (corner, carré): 4 čísla
5. Řada (line): 6 čísel
6. Tucet (dozen) a sloupec (column): 12 čísel
7. Jednoduché šance (event bets), tedy černá/červená, nízká/vysoká a sudá/lichá: 18 čísel

Nutno dodat, že číslo nula není ani nízké, ani sudé.



Obrázek 2.1: Typy sázkových příležitostí.

2.2 Herní systémy

Hráč může v ruletě pochopitelně sázet, jak se mu zlíbí, třeba podle toho, jak v danou chvíli „cítí“, kde kulička přistane. Může ale taky hrát důmyslně (systematicky) a žetony pokládat v závislosti na padnutém čísle podle předem stanovených podmínek. Pojmem systém tedy označujeme jakousi soustavu podmínek, které podle padnutého čísla a předchozích sázek určí, kam a kolik se má vsadit.

Ruleta má 37 čísel, ale kurzy jsou vytvořeny, jako by čísel bylo 36. Z tohoto faktu plyne, že při dlouhodobém hraní se ziskovost kasina na ruletě blíží k jedné sedmatřicetině z obratu, tedy přibližně 2,07 %. To ale ani zdaleka nepřipomíná důkaz, že neexistuje sázecí progrese vedoucí k stabilnímu vyhrávání. Proto se lidé odnepaměti snaží takový systém najít. Dosud se to však nikomu nepodařilo a kupodivu ani tato práce v tomto směru nepřinese nic nového.

Na webu se to samozřejmě jen hemží lákadly typu:

> „Genuine winning roulette strategy that has won millions. Learn how to win at roulette, and beat the casinos guaranteed.“ [9]

Touto nekalou cestou se snaží z lidí vymámit elektronickou adresu nebo dokonce peníze.

2.3 Martingale

K pochopení, jak systémy fungují, poslouží vůbec nejjednodušší a nejznámější systém: Martingale. Spočívá v sázkách na jednoduché příležitosti. Vybereme si jednu, třeba lichou, a budeme sázet jen na ni. Nejprve vsadíme základní sázku, tedy jeden žeton. V dalších spinech vždy sázku při prohře zdvojnásobujeme, při výhře vracíme na základní sázku.

Po každé výhře — návratu na základní sázku — získáme všechny investice zpět a k tomu vyhrajeme základní sázku. Takto bychom mohli teoreticky stále profitovat. Aby tomu tak nebylo, stanovují kasina limity sázek a my v průběhu hry tohoto systému dříve či později narazíme na limit (tedy nebudeme moci „navýšit“) a hra skončí nezdarem. Prohrajeme mnohem více, než stihneme systémem vyhrát.

2.4 Limity

Na světě neexistuje kasino nebo herna, kde by bylo možné sázet bez omezení. Proto ani Ruleta nebude výjimkou. V cvičné hře a pro systémy si však můžeme limity i kapitál nastavovat libovolně. V programu jsou rozlišovány následující limity:

- Nejvyšší možná celková sázka na celé herní plátno: `maxBet`
- Maximální a minimální sázka na jednoduché příležitosti: `maxBetEven`, `minBetEven`
- Max. a min. sázka na třetiny (tucty, sloupce): `maxBetThird`, `minBetThird`
- Max. a min. sázka na vnitřní sázky (inside bets), tedy všechny ostatní: `maxBetStraight`, `minBetStraight`

2.5 Implementované systémy

V této sekci jsou co možná nejstručněji popsány systémy zahrnuté v programu Ruleta.

Great Martingale Tento systém se od Martingalu liší pouze tím, že při prohře nejen zdvojnásobíme sázku, ale ještě přičteme sázku základní. Tím dosáhneme zisku základní sázky za každý uplynulý spin.

D'Alambert Opět začínáme jedním žetonem (základní sázkou) na jednoduchou příležitost. Pokud prohrajeme, vsadíme o žeton více. Pokud vyhrajeme, vsadíme o žeton méně (nebo alespoň jeden žeton). Zde je snadno vidět, že se na původní sázku nemusíme nikdy dostat.

Fibonacci Začínáme jedním žetonem. Při prohře se posuneme na další číslo Fibonacciho posloupnosti, při výhře se vracíme o dvě čísla zpět.

Labouchere Na začátku mějme libovolnou posloupnost čísel, například pouze jedničku. Součet prvního a posledního čísla odpovídá hodnotě, kterou vsadíme. Při výhře tato čísla odstraníme, při prohře přidáme na konec posloupnosti jejich součet (prohranou částku).

Oscar's grind Opět začínáme jedním žetonem. Při výhře vsadíme o žeton více, při prohře sázku neměníme. Při tomto systému si ale pamatujeme, kolik jsme prohráli od začátku série, a kdybychom měli sázet větší hodnotu, než činí dosavadní prohra, vsadíme právě tolik, abychom vyhráli vše zpátky a jeden žeton navíc.

Superior Nyní navyšujeme o jeden žeton v každém spinu, ale opět sázíme nejvýše tolik, aby čistý přínos od začátku série činil jeden žeton.

Systém 63 Tento systém spočívá v sázení na tucty, a to vždy na ty dva, které v posledním spinu nepadly. Pokud následně prohrajeme, sázku ztrojnásobujeme, při výhře se vracíme na původní hodnotu žetonů.

Systém 4 spinů Konečně systém sázející přímo na čísla. Vychází z představy, že se čísla nerovnoměrně opakují a některá padají velmi rychle po sobě. Sledujeme hru, vyčkáváme. V momentě, kdy se zopakuje číslo z posledních čtyř, začneme sázet na čtyři poslední čísla. Pokud dva spiny po sobě prohrajeme, skončíme a opět čekáme.

Nálet Tentokrát vsadíme na poslední padnuté číslo jeden žeton. Pokud padne jiné číslo, vsadíme i na ně a hodnoty navýšíme o jeden žeton. Takto pokračujeme, dokud nepadne jedno z čísel, které sázíme. Potom začínáme opět od jednoho žetonu na jedno číslo. Zde je třeba si uvědomit, že největšího zisku dosáhneme v šestém a sedmém spinu. Naopak od deseti sázených čísel začínáme silně prodělávat.

2.6 Tercius

Při simulaci některého z výše uvedených systémů se ukáže, jak rychle systém prohraje celý kapitál nebo narazí na limit, tj. nebude moci vsadit tak, jak má předepsáno. V Ruletě je implementován ještě jeden systém: Tercius. Jiří Kahan [7] jej považuje za profesionální a zasílá jej na žádost prostřednictvím e-mailu. Tento systém je opravdu velice komplikovaný. Spočívá v sázení na jednoduché šance a má skutečně opatrnou progresi. Nebude to tedy limit, na který narazí, ale po nějaké době vždy dojde kredit. Ani tento systém tedy nepřináší revoluci, proto mu zde nebude věnována další pozornost.

Kapitola 3

Technické aspekty

3.1 .NET Remoting

Pro komunikaci mezi serverem a klienty existují různé technologie. Lze použít sokety nebo pomocné třídy jmenného prostoru `System.Net`, které zjednodušují práci s protokoly, adresami a podobně. Vždy však musíme posílat data po síti a postarat se o to, aby adresát přenášený balík správně interpretoval a zavolal tak správnou metodu. Kromě samotného posílání dat je třeba se postarat o práci s vlákny. Tyto problémy se značně redukují použitím technologie .NET Remoting [1].

Aplikace spuštěná v jednom procesu nemůže přistupovat k datům jiného procesu a poškodit je. Procesy tradičně představovaly hranice zaručující izolaci. V aplikacích postavených na platformě .NET Framework jsou aplikační domény novou bezpečnostní hranicí uvnitř procesu. V jednom procesu tedy můžeme spustit různé aplikace, každá však musí běžet v oddělené aplikační doméně.

Objekty v jedné aplikační doméně k sobě mohou přistupovat přímo, avšak komunikace mezi objekty v různých aplikačních doménách již tak triviální není. K tomuto účelu právě vznikla technologie .NET Remoting, která umožňuje přístup k objektům z jiné aplikační domény, a to lhostejno, jsou-li spuštěné objekty v tomtéž procesu či v oddělených procesech, nebo dokonce na vzdálených systémech. Proto model .NET Remoting poslouží i ke komunikaci mezi serverem a klienty Rulety.

Technologie .NET Remoting nabízí pružnou architekturu, které lze užít v každém typu aplikace, s libovolným přenosovým protokolem a s různým kódováním dat. Téměř každá část je navržena tak, aby ji bylo možné nahradit vlastní implementací.

Třídy objektového modelu .NET Remoting jsou umístěny ve jmenném prostoru `System.Runtime.Remoting` a jeho podprostorů. Některé třídy jsou v sestavení výkonného jádra platformy .NET Framework `mscorlib`.

3.2 Průběh volání

Když klient volá metodu vzdáleného objektu, volá ve skutečnosti metodu *transparentního objektu proxy*. Z pohledu klienta vypadá transparentní objekt jako skutečný vzdálený objekt, tj. implementuje jeho veřejné metody, které zná díky mechanismu reflexe, jímž načítá metadata ze sestavení. Transparentní objekt proxy volá metodu `Invoke()` na *skutečném objektu proxy*, který je zodpovědný za předání zprávy *kanálu* pomocí *příjemce zpráv* (*message sink*).

Příjemce zpráv (message sink) je záchytný objekt, který se nachází na obou stranách komunikace. Příjemce je přidružen ke kanálu (channel), který zprostředkovává komunikaci mezi klientem a serverem a je zodpovědný jak za připojení se k soketu na serveru, tak za odesílání (již formátovaných) dat. .NET Framework 2.0 poskytuje kanály komunikující prostřednictvím protokolů HTTP, TCP a IPC.

Komunikace mezi klientem a serverem je zajišťována vytvářením *zpráv* (*messages*), které jsou posílány do kanálu. Tyto zprávy nesou informace o vzdáleném objektu, volané metodě a jejích argumentech.

Způsob, kterým příjemce odešle zprávu do kanálu, závisí na *formátovači* (*formatter*). Platforma .NET nabízí formátovač SOAP, který je vhodný spíše ke komunikaci s webovými službami, a binární formátovač, jenž je rychlejší a efektivnější zejména v intranetovém prostředí. Formátovač je ke kanálu přidružen *poskytovatelem formátovače* (*formatter provider*).

Na straně serveru kanál přijímá formátovaná data od klienta a používá formátovač k jejich dekódování na zprávy, které předává příjemci. Ten, respektive poslední z řetězce příjemců, konečně zavolá metodu vzdáleného objektu.

3.3 Konfigurační soubory

Přístup ke vzdáleným objektům tedy neprobíhá nikterak jednoduše, nicméně po vytvoření kanálů a zástupných proxy můžeme jednat s objekty, jako by se nacházely v jedné aplikační doméně, a nemusíme se téměř o nic starat. Při práci se vzdálenými objekty tedy můžeme užívat všech jazykových konstrukcí, jako jsou konstruktory, delegáty, vlastnosti atd.

Nastavení kanálů a objektů nebude provedeno ve zdrojovém kódu, ale pomocí konfiguračních souborů. Ty přinášejí tu výhodu, že můžeme přidávat a měnit konfigurace kanálů a objektů bez zásahu do zdrojového kódu, tedy bez nutnosti opětovného překladu. Konfigurační soubor v kódu aplikujeme statickou metodou `Configure()` třídy `RemotingConfiguration`. Kanály potom vytváří a konfiguruje běhová knihovna technologie .NET Remoting.

3.4 Zpřístupnění objektu a aktivace

Možností zpřístupňování a aktivace objektů poskytuje model .NET Remoting opravdu hodně. Jedná se o *klientem aktivované objekty* (instance se vytvoří na serveru pro každého klienta nová) a *známé objekty*, které zahrnují režimy *SingleCall* (při každém volání objektu se vytváří nová instance) a *Singleton* (objekt uchovává svůj stav, je k dispozici všem klientům). Dále jsou nám nabízeny možnosti správy životního cyklu, tj. po určité době nečinnosti objekt zrušit apod.

Přestože jsou to možnosti pěkné, v Ruletě potřebujeme jednoduše vytvořit objekt při spuštění serveru a ten zpřístupnit (navždycky). Tomuto postupu se říká *Direct Remoting*. Zpřístupnění již vytvořeného objektu provádí příkaz `RemotingServices.Marshal`, kterému jako parametry předáme zpřístupňovaný objekt a URI.

3.5 Parametry vzdálených metod

Typy parametrů vzdálených metod mohou být nejen základní datové typy, ale také vlastní třídy. Při předávání argumentů je potřeba rozlišovat tři základní typy tříd:

- *Marshal-by-value classes*: Objekty těchto tříd nemají vzdálenou identitu, jsou kanálem přenášeny celé. Objekt je po přijetí zcela nezávislý na druhé straně. Aby mohly být tyto třídy takto serializovány, musejí být navíc ozdobeny atributem `[Serializable]`. (V Ruletě se jedná například o strukturu `Bet`.)

- *Marshal-by-reference classes*: Tyto objekty mají vzdálenou identitu. Po síti se nepředá skutečný objekt, ale zástupný objekt proxy. Takovéto třídy musejí být potomky třídy `MarshalByRefObject`. (Na serveru objekty `Lobby` a `Chair`.)

- *Non-remotable classes*: Takové objekty nelze předávat jako parametry vzdálených objektů. Nejsou ani serializovatelné, ani nedědí od třídy `MarshalByRefObject`.

3.6 Společné rozhraní

Jak plyne z kapitoly 3.2, aby si klient mohl vytvořit odkaz na vzdálený objekt na serveru, musí, zjednodušeně řečeno, znát, jak objekt vypadá. Sestavení obsahující tyto informace je tedy použito jak na straně serveru, tak na straně klienta. Umístit ale implementaci vzdáleného objektu do společného sestavení přináší nezanedbatelné nevýhody a je to také proti zásadám distribuovaných aplikací.

Jelikož klientská aplikace potřebuje znát jen metadata, stačí do společného sestavení jen definovat rozhraní a to implementovat až na serveru (resp. u klienta). Tímto dosáhneme čistého oddělení serverového a klientského kódu.

Drobná daň za to je taková, že nemůžeme odkaz na vzdálený objekt vytvořit operátorem `new`, neboť nelze vytvářet instanci rozhraní, ale musíme použít příkaz `Activator.GetObject()`.

3.7 Rychlost volání

Voláme-li vzdálené metody běžným (synchronním) způsobem, bude nám vráceno řízení až po jejím dokončení. Pokud je potřeba vrátit řízení ihned, je vhodné použít asynchronních volání.

Při volání vzdálených metod asynchronně je řízení vráceno okamžitě. Delegátem lze volat vzdálený objekt stejně jako místní. Pokud je navíc volaná metoda typu `void` a má pouze vstupní parametry, může být navíc okrášlena atributem `[OneWay]`. Tento způsob, kterému se říká *fire-and-forget*, zajišťuje, že zpětné potvrzení (přestože ve skutečnosti stejně chodí) bude zcela ignorováno [2]. Tyto metody jsou volány asynchronně bez ohledu na způsob volání.

Pokud jde o spolehlivost, tu zajišťuje na transportní vrstvě protokol TCP. Síťový protokol IP sice může zahazovat pakety, když je síť zahlcena, ale TCP se postará o jejich odeslání znovu. Pokud je však síť nedostupná nebo je klient z nějakého důvodu odpojen, ani TCP to pochopitelně nezachrání. Pro odstranění nekomunikujících klientů je při některých asynchronních voláních jako parametr předán odkaz na obslužnou metodu `AsyncCallback`, díky níž jsou visící klienti odebráni.

3.8 Zabezpečení

Platforma .NET verze 2.0 podporuje důvěrnost dat posílaných po síti a autentizaci uživatele. Zabezpečení technologie .NET Remoting podporuje každý kanál implementující rozhraní `ISecurableChannel`, a těmi jsou TCP a IPC kanály. Pro zajištění bezpečnosti kanálu HTTP je třeba vzdálený objekt nakonfigurovat na serveru IIS (*Internet Information Services*).

Pomocí atributu `protectionLevel` je možné v konfiguračním souboru nastavit požadovanou úroveň šifrování dat přenášených po síti. Tento atribut může nabývat hodnot `None`, `Sign` nebo `EncryptAndSign`. `Sign` zajistí vytváření digitálního podpisu k ověření, že při přenosu dat nedošlo k jejich změně. Pokud je ale žádoucí, aby informace nemohl kdokoli odchytit a přečíst, je vhodné použít možnost `EncryptAndSign`, která navíc zajistí šifrování dat.

Při použití bezpečného kanálu, je nezbytné metodě `RemotingConfiguration.Configure()` jako druhý parametr předat hodnotu `true`, čímž se zajistí zapnutí bezpečnosti. Potom však každý nakonfigurovaný kanál musí podporovat bezpečnost, jinak dojde k výjimce.

3.9 Managed DirectX

Přistupovat přímo k DirectX v jazyce C# není možné jako v jazyce C++. DirectX API je k dispozici vývojářům pracujícím s technologií .NET od vydání Managed DirectX. Seznámit veřejnost s touto technologií je úkol knihy Programujeme 3D hry v jazyce C# od Toma Millera, vydané roku 2007 [4]. Tato kniha by měla zasvětit do tvorby 3D her i naprosté začátečníky.

Dnes již je však Managed DirectX vytlačeno novější iniciativou Microsoftu, a tou je XNA. Protože členství v XNA Creators Club je placené, je v zájmu firmy Microsoft co nejvíce XNA rozšířit.

XNA je předně API pro herní konzole Xbox. Zde je nevyhnutelný důsledek rozhodnutí, aby aplikace postavené na XNA byly vzájemně kompatibilní mezi systémy Windows a Xboxy: Konzola je z těchto dvou systémů přirozeně ta více omezující svými schopnostmi a tato omezení se přenášejí na XNA a ovlivňují tak všechny aplikace postavené na této technologii.

XNA je sice na klikaté cestě, má však bezesporu velký potenciál. Cílem tohoto odstavce bylo pouze ospravedlnit použití Managed DirectX pro tvorbu Rulety.

3.10 Typická kostra aplikace Managed DirectX

Pro použití 3D grafiky v projektu je potřeba přidat reference na soubory Managed DirectX, tj. `Microsoft.DirectX`, `Microsoft.DirectX.Direct3D` a `Microsoft.DirectX.Direct3DX`. Nyní lze užívat podpůrných tříd z DirectX SDK, jejichž zdrojové kódy jsou ve složce `Samples/Managed/Common`. Tyto třídy jsou navrženy pro snadné použití a řeší spoustu problémů spojených například s výběrem a vytvořením zařízení.

3.10.1 IDeviceCreation

Aby třída `Framework` uvedla do chodu příslušné zařízení, musí herní jádro rulety (třída `HerniJadro`) implementovat rozhraní `IDeviceCreation`, které zahrnuje tyto metody:

1. `public bool IsDeviceAcceptable(Caps caps, Format adapterFormat, Format backBufferFormat, bool windowed)`
2. `public void ModifyDeviceSettings(DeviceSettings settings, Caps caps)`.

Když třída `Framework` hledá vhodné zařízení v systému, zavolá první z těchto metod pro každé zařízení. Vrátí-li metoda `true`, je zařízení vhodné. Těsně před vytvořením vybraného zařízení volá `Framework` druhou metodu a umožní tak změnit jeho nastavení. Parametr `Caps` (capabilities) je struktura obsahující spoustu informací o příslušném zařízení, podle kterých můžeme rozhodnout, je-li zařízení vhodné pro daný účel. V herním jádře rulety jsou touto cestou odmítána zařízení nepodporující alfa-míchání nebo osvětlení. V metodě `ModifyDeviceSettings` je nastavení zařízení upraveno na ryzí Direct3D zařízení, pokud je grafická karta podporuje.

3.10.2 Události zařízení

V průběhu života zařízení nastávají tyto události:

1. `private void OnCreateDevice(object sender, DeviceEventArgs e)`
2. `private void OnResetDevice(object sender, DeviceEventArgs e)`
3. `private void OnLostDevice(object sender, EventArgs e)`

4. `private void OnDestroyDevice(object sender, EventArgs e)`

Metoda `OnCreateDevice` je zavolána jen jednou za celý běh aplikace, a to bezprostředně poté, co je vytvořeno zařízení. Proto se užívá k alokování tzv. `Pool.Managed` zdrojů, tedy těch, které stačí alokovat jednou. Při ukončení aplikace se volá naopak metoda `OnDestroyDevice`. V ní jsou uvolněny zdroje vytvořené v metodě `OnCreateDevice`.

Metoda `OnResetDevice` je volána také po vytvoření zařízení a dále při každé změně velikosti okna nebo přepnutí mezi celoobrazovkovým a okenním režimem. Zde je tedy místo pro úpravu objektů závislých na rozměrech kreslicí plochy, jako například změna matice projekce, a alokaci zdrojů z třídy `Pool.Default`. Tyto zdroje jsou umístěny v paměti grafické karty a je nezbytné se postarat o jejich uvolnění. K tomu slouží metoda `OnLostDevice`, která je volána vždy před metodou `OnResetDevice` (kromě úvodního vytvoření zařízení).

3.10.3 IFrameworkCallback

A konečně, abychom něco zobrazili, je potřeba třídě `Framework` předat objekt implementující rozhraní `IFrameworkCallback`, které implementuje metody:

1. `public void OnFrameMove(Device device, double appTime, float elapsedTime)`

2. `public void OnFrameRender(Device device, double appTime, float elapsedTime)`

První metoda je volána na začátku každého snímku a slouží pro aktualizaci scény v závislosti na čase, tedy například pootočení kola či změnu polohy kuličky. Všechny objekty scény jsou vykresleny v druhé metodě, která je určena skutečně pro kreslení, a proto by v ní neměly být složité animační výpočty, neboť by mohlo docházet k rušivému blikání.

Po inicializaci aplikace zavoláme metodu `MainLoop()` třídy `Framework`, čímž jí předáme možnost obsluhovat systémové zprávy, a ta v době, kdy žádné události nezpracovává, volá metody rozhraní `IFrameworkCallback`.

3.11 Výkon a rychlost Managed DirectX

Každého jistě nejvíc zajímá, jaký je výkon Managed DirectX. Ve vší obecnosti — bez ohledu na kvalitu Managed DirectX — řízený i neřízený kód musejí běžet

na témž DirectX API, avšak s řízeným kódem jsou samozřejmě spojeny režie navíc, proto pro některé hry bude použití řízeného kódu vždy nevhodné. Zda-li převáží jeho nedostatky nad jeho výhodami, to závisí na konkrétní aplikaci.

Podle [4] většina příkladů z SDK rychlostně odpovídá verzím v neřízeném kódu. Pokud řízený kód vykazuje pozorovatelně pomalejší běh, svědčí to o nějaké nevhodné konstrukci. Kvůli neustále se aktualizujícím API a chybějící dokumentaci zabývající se výkonem prostě není jasné, jak napsat řízený kód tak, aby byl efektivní.

Důležité je dbát na jevy typické pro .NET framework, například boxing a unboxing. V případě, že metodě, která očekává parametr typu `object` (referenční typ), předáváme hodnotový typ, provádí framework převod mezi těmito dvěma typy. Tomu se říká boxing. Aby tento převod mohl být proveden, alokuje se místo na haldě, na které se data zkopírují ze zásobníku. Opačnému postupu se říká unboxing. Voláme-li nějakou takovou metodu každý snímek, dojde ke znatelnému zpomalení nejen kvůli zbytečnému alokování paměti a kopírování dat, ale navíc začne uklízet garbage collector.

Dále je třeba mít na paměti, že Managed DirectX zachytává a spouští mnoho událostí. Grafické objekty potřebují zachytávat události zařízení pro jejich správné chování. Kdybychom objekt, který zachytává nějaké události, vytvářeli při každém snímku, došlo by brzy k zahlcení paměti, protože garbage collector jej z paměti nemůže uvolnit. Podobných záludností je jistě více a je třeba na ně dávat pozor.

Kapitola 4

Analýza a návrh řešení

4.1 Scénář použití

Myšlenka hraní v režimu více hráčů on-line znamená možnost vidět sázky všech hráčů sedících u stolu. Potenciální hraní o peníze přináší nutnost zavedení nějaké centrální autority, která bude kontrolovat přípustnost sázek, generovat náhodné číslo (pochopitelně stejné pro všechny hráče u stolu) atd. Z tohoto důvodu není vhodné uvažovat o architektuře peer-to-peer. Aplikace tedy sestává ze serverové a klientské části. Serverová část je vytvořena jako konzolová aplikace, klientská část je grafická aplikace.

Poskytovatel, tj. osoba, která se rozhodne poskytovat hraní rulety, musí jednak zajistit běh serveru — jedné instance serverové části, na jednom a stále stejném místě —, druhak zpřístupnit instalační balík klientské části tak, aby ji mohli uživatelé získat a nainstalovat.

Uživatel Rulety (hráč) si nainstaluje a spouští klientskou část aplikace. Aby mohl hrát v režimu off-line, není potřeba provoz žádného serveru ani připojení k síti. Pro hraní více hráčů on-line musí být spuštěný a dostupný server. Uživatel se připojí k serveru prostřednictvím klientské aplikace, aniž by si to příliš uvědomoval, tedy pouhým kliknutím. Nebude si za běhu aplikace vybírat z žádné množiny serverů ani nebude nic vyplňovat.

Ze záměru snadného připojování plyne nutnost někde *předem* uložit skutečnou IP adresu nebo doménu serveru. K tomu slouží konfigurační soubory.

Když se uživatel připojuje do libovolného internetového kasina dostupného na webu, je pečlivě kontrolován, nepoužívá-li špatnou verzi aplikace či nějak modifikovanou. Cílem této práce je napsat servervou část tak, aby nebylo nutné kontrolovat, jakou verzi klientské části uživatel používá, a aby bylo myslitelné

zveřejnit její zdrojový kód — ať si ji eventuelně každý může upravit podle svých představ.

4.2 Model serveru a komunikace

Mohlo by se zdát, že navrhnout objektovou strukturu serveru bude banální. Lobby bude objekt, který bude zpřístupněn při startu serveru, a jednotlivé stoly budou opět zpřístupněné objekty (každý pod jiným URI). Když bude hráč u stolu pokládat sázky, předá volané metodě id pozice u stolu, kde sedí, aby server věděl, kdo to sází. Na tom je na první pohled něco špatně. Směřujeme k tomu, aby si kdokoli teoreticky mohl upravovat klientskou aplikaci. Potom ale stačí, aby jako id pozice u stolu uvedl nepravé id a už pokládá sázky za někoho jiného.

Zavést židle coby objekty patřící stolům je tedy nevyhnutelné. Otázkou zůstává, jakým způsobem židle zpřístupňovat. Kdyby židle byly dobře známými objekty, nelze zabránit možnosti sednutí si více uživatelů na tutéž židli (takzvaně na klín). Tento problém se vyřeší zveřejněním židle jako klientem aktivovaného objektu. Kdyby si totiž klient chtěl vytvořit židli na místě, kde už někdo sedí, vyvolá server výjimku, která bude serializována a poslána klientovi, a tím se objekt nevytvoří. Když si ale klient odsedává, musí zrušit svůj odkaz na židli a už na ní žádné metody nevolat, aby ji uklidila automatická správa paměti. Proto může zákeřný klient tuto židli nechat žít a čekat, až si někdo vytvoří židli na tomtéž místě, a potom sázet za něho. Tato cesta je tedy také slepá.

Ke správnému řešení vede až konstrukce z [3, strany 42–48]. Když klient volá metodu, že si chce sednout na dané místo, vytvoří pro něj server židli a vrátí mu odkaz na ni. Pokud si chce sednout na obsazené místo (buď za zlým účelem, nebo je obsadil jiný klient ve zlomku vteřiny před ním), vrátí server `null`. Jiný klient odkaz na cizí žídli nemůže získat, a dokonce, když si klient odsedne nebo se ztratí, existuje prostředek jak pro daný objekt zamezit všem příchozím voláním. Tím je příkaz `Disconnect()`.

Při hraní Rulety po síti je snahou, aby byly sázky umísťovány *bezpečně* i *rychle*. Pojem bezpečnost v tomto případě znamená maximalizovat šanci, že když hráč položí sázku, server ji vezme na vědomí a hráč se také dozví, že ji server bere na vědomí. Rychle znamená, že když hráč umístí na plátno žeton, bude ihned zobrazen a okamžitě bude možné položit další žeton. Snahou je najít kompromis mezi těmito dvěma cíly, které jdou přirozeně proti sobě. Postup je rozebrán v programátorské dokumentaci (kapitola 6.6).

4.3 Herní systémy

Součástí Rulety jsou implementovány vybrané herní systémy (popsané v kapitole 2.5) a v programu je umožněna jejich *simulace* a *hraní*. Pojmem simulace systému se míní nechat počítač opakovaně generovat náhodná čísla a umísťovat sázky podle podmínek stanovených systémem. Hraním systému se myslí možnost, kdy při hře, ať už off-line nebo on-line, budou po spinu nejprve umístěny sázky podle systému a uživatel je pak bude moct upravit.

Cílem práce je rovněž umožnit nějakým způsobem vložit do programu systém, který není v Ruletě implementovaný. Přestože se některé systémy mohou jevit jako podobné, může systém spočívat v použití nejrůznějších struktur, nejrozmanitějších postupů a algoritmů. Proto není možné vkládání systému do programu uskutečnit pomocí nějakých klikátek, nastavitelností množiny parametrů nebo pomocí nějakého jazyka na úrovni abstrakce vyšší než C#.

Možnost vložit systém do programu Ruleta spočívá v jeho samotném naprogramování v projektu MujSystem, který je součástí řešení Ruleta, a jeho následném zkompilování do podoby dynamicky linkované knihovny .dll. V Ruletě pak ve volbě systému vybereme možnost *Vlastní*, čímž se nám zviditelní a zpřístupní pole pro vložení absolutní cesty k souboru se systémem (viz obrázek 5.1 na straně 26).

4.4 Žetony na plátně

Trojrozměrné herní prostředí umožňuje uživateli měnit pozici kamery a směr pohledu. Jak uživatel pohybuje myší nad sázkovým plátnem, objevuje se pod kurzorem žeton a po kliknutí se položí. Díky možnosti pohybu kamery vzniká problém, jak zjistit, nad kterou sázkovou příležitostí se kurzor nachází.

Nejprve je potřeba si uvědomit, co se děje s každým bodem před jeho zobrazením. Takový bod je nejprve transformován z lokálních souřadnic objektu na souřadnice světa, např. posunutím, rotací apod. Potom je transformován maticí pohledu, která je určena polohou kamery, směrem pohledu a směrem nahoru, a nakonec maticí projekce, která určuje vlastnosti čočky kamery a převádí tak svět do uříznutého jehlanu (viewing frustum), který je určen přední a zadní rovinou, poměrem šířky a výšky plochy pro zobrazení a úhlem ve svislé rovině, který kamera snímá.

Takováto transformace převádí body 3D světa do roviny, v níž x-ové i y-ové souřadnice nabývají hodnot z intervalu $[-1; 1]$ a každý bod nese informaci o hloubce, tj. nese hodnotu z intervalu $[0; 1]$, kde 1 znamená nejhlubší, tedy nejdál

od kamery. Takovéto body jsou následně převedeny na souřadnice obrazovkových pixelů a vykresleny.

Abychom získali souřadnice bodu na plátně pod kurzorem myši, musíme postupovat přesně obráceně. V objektu `MouseEventArgs` nalezneme souřadnice `x` a `y`, a tak je nejprve převedeme do intervalů $[-1; 1]$. Vezmeme takový bod v hloubce 0 a označíme jej P_0, podobně v hloubce 1 vezmeme bod P_1. Tyto body transformujeme maticí M, kterou získáme invertováním transformační matice, tedy

$$M = (maticeSveta \cdot maticePohledu \cdot maticeProjekce)^{-1}$$

Nyní máme body 3D světa, které určují přímku procházející plátnem pod kurzorem. Potřebujeme spočítat průsečík, a k tomu nám poslouží vztah odvozený v [5]:

$$t = -\frac{P_0 \cdot N + D}{P_1 \cdot N}$$

Rovina plátna splývá s rovinou z = 0, a proto se výpočet výrazně zjednodušuje na vztah

$$t = -\frac{P_0.Z}{P_1.Z}$$

Nyní stačí k bodu P_0 přičíst t-násobek vektoru daného rozdílem bodů P_1 a P_0 a máme hledané souřadnice na plátně. Zjistit, jaké sázkové příležitosti bod náleží, je už jen titěrná práce.

4.5 Padající kulička

Od naší Rulety požadujeme, aby server oznámil, jaké číslo má padnout, a aby skutečně padlo. (Čekat na padnutí čísla a pak je sdělit serveru je nemyslitelné nejen z bezpečnostních důvodů, ale taky potřebujeme, aby padlo všem hráčům u stolu totéž číslo.) Navíc chceme, aby byla kulička vystřelena nějakou chvilku před ukončením možnosti pokládat sázky. Tohoto efektu dosáhneme drobným porušením zákonů fyziky: kulička po svém vržení bude vykonávat rovnoměrný pohyb po kružnici a v momentě, kdy server určí číslo, které má padnout, začne rovnoměrně zpomalovat tak, aby přistála tam, kde má. K odvození výpočtu zpomalení si vystačíme se znalostmi středoškolské fyziky.

Označme v_0 rychlost, jíž je vystřelena, a v_t rychlost otáčení kola (proti směru letu kuličky). Nechť je kulička vhozena nad místem, kde se nachází číslo n. Potom úhel (ve středu kola) mezi polohou kuličky a číslem n označíme s a jeho závislost

na čase t vyjadřuje vztah

$$s = (v_0 + v_t) \cdot t - \frac{a \cdot t^2}{2}$$

Pro výpočet zrychlení tedy potřebujeme znát celkový čas letu kuličky. Při zpomalování bude kulička nejprve obíhat po nejdelší kružnici, po dosažení určité rychlosti, kterou označme v_p, se začne blížit ke žlábkům. Dobu od začátku klesání kuličky (tedy od dosažení rychlosti v_p) do okamžiku dopadu označme t_p. Potom celkový čas bude

$$t = \frac{v_0 - v_p}{a} + t_p$$

Po dosazení do předchozího vzorce tedy máme

$$s = (v_0 + v_t) \cdot \left(\frac{v_0 - v_p}{a} + t_p \right) - \frac{a \cdot \left(\frac{v_0 - v_p}{a} + t_p \right)^2}{2}$$

a po úpravách získáme rovnici

$$a^2 \cdot t_p^2 + a \cdot [2 \cdot (s - t_p \cdot (v_t + v_p))] + [(v_p - v_0) \cdot (v_p + v_0 + 2v_t)] = 0$$

k jejíž vyřešení nám poslouží vzorec pro řešení kvadratických rovnic s přičítáním diskriminantu.

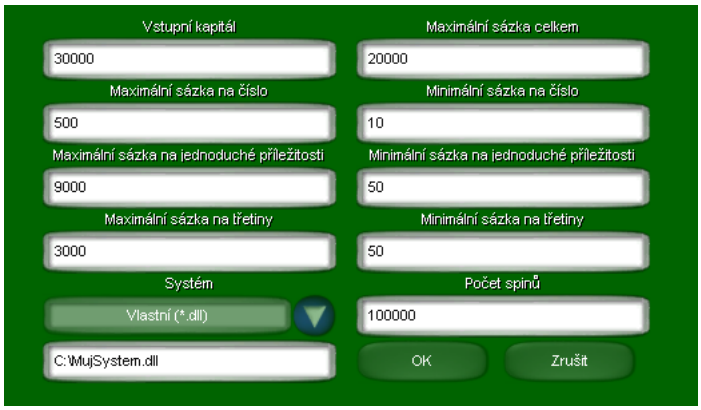
Uvažujeme-li, že se kulička od začátku klesání blíží ke středu rovnoměrně zrychleně, je výpočet t_p triviální. Ostatní rychlostní konstanty lze zvolit libovolně.

Kapitola 5

Uživatelská dokumentace

5.1 Single-player

Po spuštění programu se zobrazí hlavní nabídka obsahující mimo jiné tlačítko *Cvičná hra – off-line*. Po stisknutí tohoto tlačítka se zobrazí dialog, ve kterém lze zadat počáteční kapitál a limity hry (obrázek 5.1). Chcete-li vyplnit co největší možnou hodnotu, tedy limit v podstatě zrušit, stiskněte v příslušném poli klávesu enter. Pozor, zadáte-li do pole ještě větší číslo nebo nějaký nečíselný symbol, bude údaj interpretován jako nula.



Obrázek 5.1: Dialog nastavení limitů.

Po stisku *OK* se okamžitě ocitáme u stolu (viz obrázek 5.2). Tlačítka se šipkami slouží k pohybu kamerou, písmena A, S, D a W k rotaci pohledu.

Při pohybu myší nad plátnem se pod kurzorem objevuje žeton, který položíme stisknutím levého tlačítka. Držíme-li tlačítko stlačené, bude při změně pozice kurzoru nad jinou sázkovou příležitost na ni nanesen žeton. Naopak pravým tlačítkem danou hodnotu odečítáme. Hodnotu pokládaného žetonu vybíráme ze vzorových žetonů umístěných na stole (1).

V pravém horním rohu je seznam (2), do něhož jsou vypisována padnutá čísla, dále tlačítka *Odebrat sázky*, které z plátna odstraní všecky naše položené žetony, *Poslední sázka*, které na plátno přidá sázku posledního spinu, a konečně *SPIN!*, které roztočí kolo a střelí kuličku.

Okamžitý hráčův kapitál je zobrazen v levém dolním rohu (3), kde se také nacházejí další tlačítka. *Chat* zobrazuje a schovává dialog chatu (4), *Celá obrazovka* přepíná mezi okenním a celoobrazovkovým režimem.

Tlačítko *Hrát systém...* způsobí zobrazení téhož dialogu jako v situaci spouštění nové hry (obrázek 5.1). Navíc bude ale možno vybrat systém, který chceme hrát. Hrajeme-li systém, budou po každém spinu umístěny žetony podle systému. Můžeme je odstraňovat, přidávat a měnit dle našeho zalíbení. Systém ale v dalším spinu umístí sázky opět tak, jako by se hrálo přesně podle něj, tedy bez našeho zásahu. Instance systému má tedy vlastní kapitál i vlastní limity (podle toho, co vyplníme v dialogu). Je tedy nezávislá na skutečné hře. Proto se může stát, že systém bude mít kapitál a skutečná hra už ne. A stejně tak naopak: skutečná hra bude mít kredit, ale systém nikoli. V tomto případě nám bude oznámeno, že systém skončil, spolu s důvodem jeho selhání.

Tlačítko *Odejít* způsobí návrat do hlavní nabídky.

5.2 Multi-player

Další možností hlavní nabídky je *Skutečná hra – on-line*. Před jejím zvolením je třeba dbát na vyplnění uživatelského jména, pod nímž budeme vystupovat. To může být dlouhé nejvýše 10 znaků.

Podaří-li se navázat spojení se serverem, ocitneme se v lobby, tj. zobrazí se dialog z obrázku 5.3, kde je každé místo u stolu reprezentováno tlačítkem, jehož stisknutím si ke stolu přisedneme. Sedí-li již na daném slotu hráč, je na příslušném místě vypsána jeho přezdívka.

Pokud se serveru povede usadit nás na kýžené místo, uvidíme, že skutečná hra se od cvičné téměř neliší (viz opět obrázek 5.2). Zásadním rozdílem jsou však barvy žetonů. Vzorové žetony pro výběr jejich hodnoty (1) jsou téže barvy, neboť barvitost žetonů je nyní použita pro rozlišení svého majitele, tedy aby si každý

Obrázek 5.2: U stolu.

poznal ty své. Přezdívky jednotlivých hráčů jsou spolu s jejich kredity vypsány vlevo dole (3), a sice každý barvou korespondující se svými žetony.

Budete-li chtít roztočit kolo, shledáte, že tlačítko *SPIN!* zmizelo (2). Nyní je to totiž server, kdo rozhoduje, kdy bude vržena kulička a ukončena možnost pokládat sázky. Abychom měli představu, kolik času do exekuce zbývá, slouží nám pokroková lišta, která se rozprostírá na úplném spodku obrazovky po celé její šířce. Naplňuje se zleva doprava zelenou barvou a když se čas krátí, přechází do červena. Kulička je vystřelena těsně před jejím naplněním, nějakou vteřinku však ještě sázet můžeme.

Po stisknutí tlačítka *Odejít* se vrátíme zpět do lobby. Zde nalezneme ještě tlačítko *Odpojit*, které nás vrátí do hlavní nabídky.

5.3 Simulace systémů

Další možností v hlavní nabídce je *Simulace systémů*, která zobrazí totéž okno (opět obrázek 5.1) jako v případě *Hrát systém...*, nyní bude ale navíc zpřístupněna kolonka *Počet spinů*. Po kliknutí na *OK* začne simulace systému. To

Obrázek 5.3: Lobby.

znamená, že počítač vygeneruje náhodné číslo a sám vsadí podle systému a opět vygeneruje číslo atd. Takto učiní nejvýše tolikrát, kolik jsme zadali do pole *Počet spinů*. V případě, že systém narazí na limit nebo na nedostatek kapitálu, informuje nás, po kolika spinech k tomu došlo, a sdělí důvod, kvůli kterému systém nemohl pokračovat. Pokud k ničemu takovému nedojde, informuje nás po daném počtu spinů o stavu kapitálu. Toto okno se tímto nezavře a můžeme okamžitě simulovat znovu.

5.4 Server a nasazení aplikace

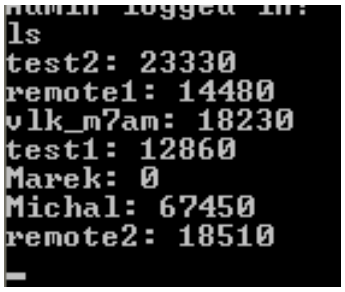
O spuštění a provoz serverové části se stará poskytovatel Rulety, který by měl rozumět scénáři použití (z kapitoly 4.1).

Pro ideální způsob nasazení aplikace by měl poskytovatel nejprve upravit v konfiguračním souboru klientské aplikace adresu serveru a následně vytvořit a zpřístupnit instalační balík. V tomto případě si uživatel (hráč) aplikaci pouze nainstaluje a nemusí se dále o nic starat. Pokud ale uživatel nainstaluje již vytvořený instalační balík nebo dojde-li ke změně adresy serveru, je nutné, aby si uživatel nastavil adresu serveru sám (návod v příloze B).

Před použitím aplikace je vhodné zvážit, který kanál použít. TCP kanál je rychlejší a efektivnější, ale na rozdíl od HTTP kanálu může vést k prob-

lémům spojeným s firewally či jinými zabezpečeními. Všecky tyto změny je nutné provést v konfiguračních souborech jak serverové, tak klientské části. Pro použití zabezpečeného TCP kanálu zajišťujícího integritu dat je možné server rovnou spustit.

Po spuštění této konzolové aplikace je hraní rulety zpřístupněno a hráči se mohou připojovat pomocí klientské aplikace. Při běhu serveru můžeme příkazem `ls` vypsat všechny hráče sedící u stolů společně s jejich kredity, příkazem `lobby` získáme všechny hráče v lobby a příkaz `end` program ukončí.

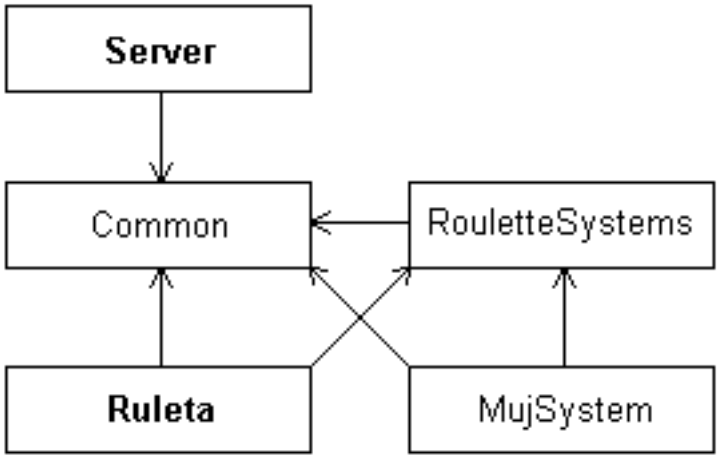


Obrázek 5.4: Výpis hráčů u stolů.

Kapitola 6

Programátorská dokumentace

Řešení Ruleta[1] sestává celkem z pěti sestavení (assemblies). Reference mezi nimi zachycuje obrázek 6.1. Spustitelné programy jsou tučně, zbylá sestavení jsou knihovní aplikace.



Obrázek 6.1: Jednotlivé projekty řešení Ruleta.

[1]Na adrese http://code.google.com/p/hazardnihraruleta/ lze stáhnout zdrojové kódy i instalační balíky.

6.1 Common

Sestavení `Common` je knihovní aplikace. Obsahuje třídy a rozhraní používané serverovou i klientskou částí Rulety. Jedná se o rozhraní `IKlient`, `ILobby` a `IChair`. Mezi třídy patří výjimky, tedy `LoginException` a `LimitException`, a konečně třída `Hra`.

6.1.1 Hra

Instance třídy `Hra` uchovává hodnoty limitů (viz kapitola 2.4), které jsou jen pro čtení, a kapitálu. Dále obsahuje proměnnou `vyhra`, která nese hodnotu výhry při posledním spinu, a `celkem`, která obsahuje hodnotu, kolik je celkem vsazeno na plátně. Důležitou složkou této třídy je `platno[]`, což je pole hodnot typu `int` o velikosti 157 (tedy od 0 do 156). Index tohoto pole udává kód neboli ID sázkové příležitosti a v poli na tomto indexu je hodnota sázky na danou příležitost. Například `platno[0]` nese informaci, kolik je vsazeno na nulu.

Hra má vytvořen indexer, který vrací hodnotu plátna na daném indexu. Tedy `hra[0]` vrátí hodnotu `platno[0]`. Další důležitou složkou je `vyplatit[]`, což je rovněž pole hodnot typu `int`, tentokrát ale od 0 do 36. V tomto poli je na indexu i uloženo, kolik se má vyplatit, padne-li číslo i.

K položení sázek na plátno slouží procedura `void Vsadit(int kde, int kolik)`, která vsadí na příležitost `kde` hodnotu `kolik`. Provádí kontrolu, jestli je to možné, podle limitů sázek a kapitálu. V případě, že ne, bude vyvolána výjimka `LimitException`, ale zároveň vsadí, kolik je jen možno vsadit. Kód této metody se může zdát na první pohled redundantní, protože důvody k vržení výjimky mohou nastat až tři najednou.

`Hra` obsahuje i další procedury k sázení, jejichž význam je jasný a které využívají proceduru `Vsadit`.

Nejdůležitější funkcí je `bool Padlo(int cislo)`, která přičte ke kapitálu výhru podle čísla, které padlo, uloží sázky do pole `minula[]` a odstraní sázky v poli `platno[]`. Pokud je výhra větší než nula, vrací `true`. To je využíváno v implementaci systémů.

A kde je informace o tom, na jaká čísla se daná sázková příležitost vztahuje? K tomu slouží `PrilezitostNaCisla`, což je statické pole polí typu `int`. `PrilezitostNaCisla[i]` vrátí pole čísel, na které se vztahuje příležitost `i`.

Některé systémy sázejí přímo na čísla, a proto potřebujeme metodu, která dostane číslo a vrátí id sázkové příležitosti na toto číslo. To dělá právě funkce `static int CisloNaPrilezitost(int cislo)`.

Nelehké rozhodování bylo o typu proměnných, zejména kapitálu. Může se stát, že uživatel nebo systém vyhraje opravdu hodně a nebude stačit čtyřbajtový integer. Nabízí se použít typ `decimal`, ale práce s ním je pomalejší. Desetinná čísla však nemají smysl, a proto je pro kapitál použit typ `long`, neboť jeho rozsah je dostatečně velký. Pokud kapitál, sázky nebo výhry přesáhnou maximální možnou hodnotu, budou se jí rovnat.

6.2 RouletteSystems

Sestavení RouletteSystems je opět knihovní aplikace. Obsahuje třídu `RouletteSystem` a veškeré implementace ruletních systémů.

Třída `RouletteSystem` je abstraktní. Jednotlivé systémy jsou pak od ní odvozeny a při konstrukci využívají nejprve konstruktor předka. `RouletteSystem` obsahuje proměnnou `int zaklad` (každý systém má nějakou základní sázku) a odkaz na instanci třídy `Hra hra`. To je ale nová instance, nikoli ta na plátně, která uchovává skutečné sázky hráče. Proto, když uživatel hraje systém, je volána jednak `hra.padlo` na plátně, dvak `system.hra.padlo` v systému. Vykonává se tedy stejný kód dvakrát, ale to je nezbytné, protože hráč může sázku systému libovolně měnit, tedy na plátně a v systému jsou jiné sázky.

Klíčovou procedurou je abstraktní `void Padlo(int cislo)`, kterou potomci přejíždějí. Tam se volá ona funkce `hra.padlo` v daném systému a ve většině systémů se využije její návratové hodnoty. Tato procedura taky zařídí potřebné umístění sázek.

`RouletteSystem` také obsahuje statickou funkci `DejSystem`, která vytváří instanci daného systému podle jeho názvu a vrací odkaz na ni. Pokud název neodpovídá žádnému připravenému systému, bere se tento řetězec jako absolutní cesta ke knihovně (.dll) obsahující vlastní implementaci.

Systém z knihovny .dll je vytvářen pomocí reflexe. Příkazem `Assembly a = Assembly.LoadFile(cesta)` získáme objekt reprezentující sestavení a příkazem `Type[] types = a.GetTypes()` obdržíme všechny použité typy, tedy třídy, rozhraní atd. Z typu, který se jmenuje MujSystem, se program pokusí vytvořit instanci statickou metodou `CreateInstance` třídy `Activator`.

6.3 MujSystem

Projekt MujSystem je opět dynamicky linkovaná knihovna. Slouží k dopsání si vlastního systému. Obsahuje jedinou třídu: `MujSystem`. Její název musí zůstat

takový, aby se povedlo v programu vytvořit její instanci. Může obsahovat pomocné metody i další třídy.

6.4 Ruleta – klientská část

Klientská část Rulety je grafická aplikace, která používá DirectX API. Pokud bude spuštěna na systému s grafickou kartou, která neposkytuje DirectX podporu na úrovni hardwaru, bude výkon velice slabý a hra pravděpodobně nebude hratelná.

Název klientské části `Ruleta` je dán průběhem vývoje aplikace. Vznikl totiž v dobách, kdy byla Ruleta vyvíjena jako samostatný projekt a ještě nikdo netušil, že bude podporovat hraní po síti. Obrázek 6.3 na straně 40 znázorňuje, z jakých tříd je aplikace tvořena.

6.4.1 HerniJadro

Hlavní třídou klientské aplikace je třída `HerniJadro`. Obsahuje veškerou herní logiku i vstupní bod aplikace. Implementuje rozhraní `IFrameworkCallback` a `IDeviceCreation`, takže se stará jak o vytvoření zařízení, tak o samotné vykreslování scény (podrobněji kapitola 3.10). Tato třída má na starost i zobrazení uživatelského rozhraní, což zahrnuje značné množství kódu, a tak je rozdělena do dvou souborů. V souboru GUI.cs tedy obsahuje metody a datové složky týkající se spíše zobrazovaných dialogů včetně metod pro obsluhu událostí jednotlivých tlačítek.

Aby herní jádro vědělo, co má vykreslovat, udržuje si v proměnné `stavHry` hodnotu z výčtu `StavHry`, který obsahuje volby `Hlavni`, `Lobby` a `UStolu`.

K tvorbě uživatelského rozhraní je použito ovládacích prvků z podpůrných tříd Managed DirectX (soubor dxmutgui.cs). Ty ale neposkytují nic jako *ProgressBar*, proto je pokroková lišta realizována pomocí dvou textur: spodní, která znázorňuje přechod ze zelené do červené, a vrchní, která je šedivá. Vrchní textura je umístěna tak, aby zakrývala spodní texturu plochou odpovídající zbývajícímu času. Šedivá textura je ve skutečnosti transparentní obrázek nanášen šedivě. Je to tak proto, že nelze jednoduše vykreslit jednobarevný obdélník. Kreslící metoda `Sprite.Draw2D()` totiž vždy musí dostat texturu jako parametr.

K výpisu hráčů a jejich kapitálů je použit také objekt typu `Sprite`. V tomto případě je důležité nastavit parametr z výčtu `SpriteFlags` na hodnotu `AlphaBlend`.

Údaje vyplněné v dialogových oknech jsou ukládány do registrů. K tomu se využívá prostoru `Microsoft.Win32`, kde se nachází třída `RegistryKey`, a do proměnné tohoto typu se uloží odkaz na větev v registrech například pomocí metody `Registry.CurrentUser.CreateSubkey(string)`. Přes proměnnou se pak využívá metod `GetValue` a `SetValue`. Veškeré ukládané hodnoty jsou řetězce.

6.4.2 Platno

Třída `Platno` má na starost uchování informací o hrách všech hráčů u stolu a zobrazování všech žetonů, tedy nejen sázených, ale i vzorových (včetně podložky zvýrazňující vybranou hodnotu). Mesh žetonů není nic jiného než obyčejný válec, který je vytvářen statickou metodou `Mesh.Cylinder`. (V XNA mimochodem takto jednoduše pracovat s geometrickými primitivy nelze, je nutné je vytvářet složitou cestou použitím vertexbufferů. Nutno samozřejmě podotknout, že na webových stránkách XNA Creators Club je dostupný projekt implementující tvorbu těchto primitiv.) Podložka pod žetonem je kvádr, čili `Mesh.Box`.

K vykreslování žetonů nejsou potřeba žádné textury, jsou jednobarevné. Číslo na žetonech je vykreslováno opět pomocí objektu `Sprite`, a to pouze na tom nejvrchnějším válci.

Po každé změně sázky je vsazená hodnota nad danou příležitostí přepočítána na druhy žetonů tak, aby jich bylo použito přirozeně co nejméně. Kdyby byly žetony umístěny ve štosech dokonale, působily by nepřirozeně a jednolitě. Proto je každá jejich pozice upravena (v osách x a y) o náhodnou chybu, jejíž maximální hodnota (vzdálenost od dokonalého umístění) je určena konstantou `Rozptyl`. Každý žeton (tj. instance třídy `Zeton`) nese informaci o své pozici, barvě a hodnotě a také příznak, je-li ve štosu nejvyšší, tedy že se má na něm vykreslovat jeho hodnota.

Aby server mohl volat metody klienta, je nutné, aby nějaký objekt, který bude serveru předán, implementoval rozhraní `IKlient`. Jelikož se třída `Platno` stará o sázky všech hráčů u stolu, je to právě ona, kdo implementuje požadované rozhraní. Z tohoto důvodu je i tato třída rozdělena do dvou souborů. Metody a složky spojené se síťovou komunikací jsou umístěny v souboru Klient.cs.

6.4.3 Ostatní třídy

Třída `Stul` se stará o vykreslení stolu, tedy všeho nepohyblivého. Třída `Rotor` má na starost naopak otáčivou část kola a také samotné rotování. Rotor se

může nacházet v jednom ze dvou stavů: Točí se a netočí se. Na to `Kulicka` už má počítání opravdu hodně. Prochází celkem čtyřmi stavy: Pokud je kulička ve žlábku a kolo se netočí, `Stoji`. Je-li vystřelena a pohybuje-li se po své nejdelší kružnici rovnoměrně, `Leti`. Jestliže její rychlost klesá, `Zpomaluje`. Konečně stav, kdy dopadne a kolo se ještě chvíli točí, se nazývá `Dozva`. Třída `Kulicka` obsahuje metodu `Spin()`, která se postará o to, aby kulička začala rovnoměrně létat, a metodu `Padni(int cislo)`, jež zařídí její zpomalování, které vyvrcholí pádem přesně na předané `cislo`.

`Stul` obsahuje navíc metodu `static int IDPrilezitosti(float x, float y)`, která na základě souřadnic `x` a `y` zjistí ID sázkové příležitosti. Naopak metoda `static Vector2 SouradnicePrilezitosti(int kde)` vrátí pozici na plátně, na níž se nelézá příležitost `kde`. Statické metody pro určení barvy čísla nebo vzdálenosti čísla na kole napravo od nuly jsou ve třídě `Rotor`.

Třída `SkladisteTextur`, která je použita jen staticky, je zodpovědná za načítání textur, jejich uchování v paměti i jejich uvolnění. Instance třídy `Kamera` uchovává kromě nezbytných matic pohledu a projekce také inverzní matici k součinu těchto matic. Ta je používána v metodě `Vector2 Invertuj(float x, float y)`, která souřadnice na obrazovce převede na souřadnice světa v rovině $z = 0$ (podrobněji kapitola 4.4).

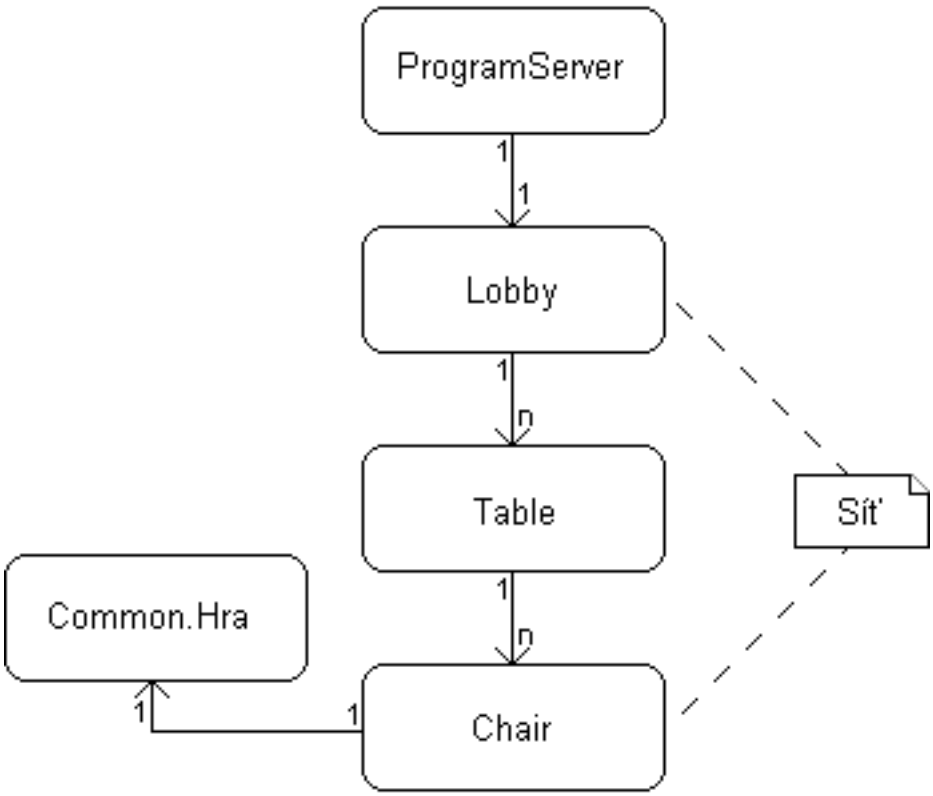
6.5 Server

Serverová část rulety je tvořena třídami znázorněnými na obrázku 6.2.

`ProgramServer` obsahuje vstupní bod aplikace. Zde se provádí vytvoření instance třídy `Lobby` a její zpřístupnění. V konstruktoru Lobby jsou okamžitě vytvořeny stoly, tj. instance třídy `Table`. Ty se starají o generování náhodných čísel v intervalech daných konstantou `Lobby.DOBA`.

Každý stůl běží v samostatném vlákně. Spiny u jednotlivých stolů jsou prováděny v pravidelných odstupech tak, aby byla výpočetní i síťová náročnost co nejlépe rozložena v čase. Odmlka mezi spiny je realizována příkazem `Thread.Sleep`, takže v danou chvíli nelze zjistit, kolik zbývá do generování čísla. Drobná daň za tuto implementační jednoduchost a robustnost je taková, že má hráč po přisednutí ke stolu funkční pokrokovou lištu informující o zbývajícím čase až po prvním zpozorovaném spinu. .NET Framework samozřejmě poskytuje prostředky pro tvorbu přesného časovače, propustnost serveru je ale přednější.

Potřebné informace o připojených klientech jsou uchovávány v objektu `Lobby`. Přisedá-li si hráč ke stolu, volá příslušnou funkci `IChair Sedni(string kdo, int stul, int zidle)`, která, pokud je kýžené místo volné, vytvoří pro klienta



Obrázek 6.2: Diagram tříd Serveru.

židli (instance třídy `Chair`) a vrátí mu na ni odkaz. Objekt židle je po odsednutí hráče odpojen metodou `RemotingServices.Disconnect(MarshalByRefObject obj)` a nastaven na `null`.

6.6 Sázení

6.6.1 Pokládání žetonů

Když uživatel někam položí sázku, je volána vzdálená metoda `chair.Vsad(int kde, int kolik, int spin)`, kde parametr `kolik` určuje kolik na danou příležitost přidat, nikoli kolik tam má být výsledně umístěno. Parametr `spin` slouží ke kontrole, patří-li sázka k právě probíhajícímu spinu. O jeho hodnotě je

klient informován při usazení a při každém oznámení, jaké číslo má padnout.

Kdyby byla metoda Vsad volána synchronně, hráč by dlouhou dobu čekal na odezvu a sázení by tak bylo příliš pomalé. Proto je tato metoda volána asynchronně. Při sázení se navíc vytvoří instance struktury `Bet`, uchovávající sázku na pozici `kde` o hodnotě `kolik`, a přidá se do seznamu `List<Bet> pending`. V tomto seznamu jsou udržovány dosud nepotvrzené sázky.

A co udělá server? Kromě nezbytné kontroly, je-li sázka přípustná, zavolá na každém připojeném klientovi metodu `Vsad(int kdo, int kde, int kolik, int spin)`. Asynchronní volání vzdálené metody pomocí delegátu má vrátit režii okamžitě. Slovo okamžitě zde ale znamená v momentě, kdy se úspěšně podaří zavolat vzdálenou metodu, tedy když klient potvrdí volání. Pokud ale klient visí (tj. nekomunikuje), trvá nekonečné dvě vteřiny, než bude program pokračovat. Čekat dvě vteřiny při každé sázce je nepřijatelné. Proto je nutné provést toto volání v samostatném vlákně, tedy nejprve zavolat asynchronně lokální metodu `PotvrdSazku()` a v ní teprve vzdálenou metodu `Vsad()` na dané židli.

V metodě `PotvrdSazku()` je potřeba kontrolovat, je-li objekt židle různý od `null`, protože u visícího klienta už mohlo dojít k jeho zrušení. K jeho zrušení může dojít i mezi kontrolou podmínky a vytvořením delegátu na vzdálenou metodu, proto je nutné objekt stolu uzamknout. Tento zámek ale musí být uvolněn před samotným vzdáleným voláním, jinak by visící klient opět na celé dvě vteřiny zablokoval stůl.

Metoda `Vsad()` na klientské straně, jedná-li se o jiného hráče, prostě přidá sázku v instanci třídy `Hra` pro daného hráče. V případě, že nám server vrací naši sázku, je odstraněna ze seznamu `pending` (= sázka je potvrzena).

6.6.2 Padnutí čísla

Uplyne-li doba mezi jednotlivými spiny, vygeneruje server náhodné číslo a na každém klientovi se zavolá metoda `void Padni(int cislo, int spin, int novySpin, long kapital)`. (Navíc samozřejmě na každé evidované `hře` je zavoláno `Padlo(cislo)` pro přičtení výher hráčům a odstranění sázek.) Od tohoto okamžiku jsou přijímány právě sázky s hodnotou `novySpin`, neboť proměnná `spin` je změněna (např. inkrementována).

Co se děje v metodě `Padni` na straně klienta? V první řadě je potřeba odebrat ze stolu nepotvrzené sázky, tj. sázky v seznamu `pending`. Tento seznam ale ještě nesmíme smazat, neboť se někde mohlo zatoulat potvrzení naší sázky, které teprve přijde! Proto nastavíme hodnotu `aborted` na `true` a zařídíme, aby padla kulička na správném místě. Po dokončení animace (10 - 20 vteřin) již vyčistíme

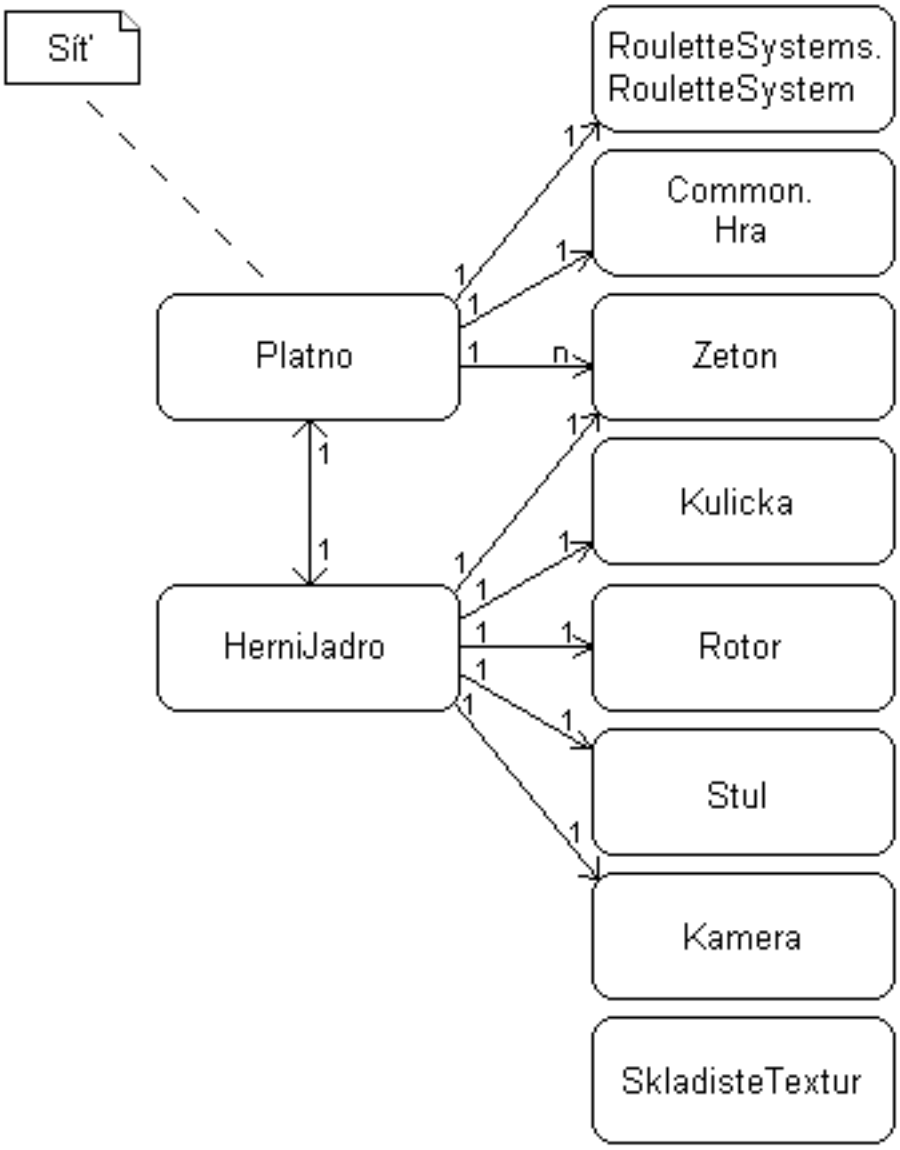
seznam `pending` a nastavíme hodnotu `aborted` na `false`.

V metodě `Vsad` je navíc kontrolováno, zda-li je hodnota `aborted` nastavena na `true`. Pokud ano, znamená to, že nepotvrzené sázky již byly odebrány a kolo se točí, a proto musíme právě potvrzovanou sázku znovu přidat. Přes všechna tato opatření je jako parametr posílána hodnota kapitálu, ke kterému by se měl klient dopočítat. V případě, že se tyto hodnoty budou lišit, bude tato informace (pro ladící účely) vypsána do chatu. Tento jev však během testování nenastal.

Tímto ale problematika sázení nekončí. Pokud se nějaký (jiný) hráč připojí v době, kdy my nemůžeme sázet, tj. kdy nám padá kulička, on pochopitelně pokládat žetony může. Mimoto kulička může někomu dopadnout dříve a ten hráč hned začne sázet. Server dělá svou práci, tedy informuje o sázkách ostatní hráče, ale ti stále čekají, až dopadne kulička. Proto nemohou sázky nového hráče zobrazovat hned, ale musejí si je pamatovat. K tomu slouží pole seznamů sázek `List<Bet>[] suspended` o velikosti maximálního počtu hráčů u stolu. Příchozí sázky jsou pro daného hráče ukládány do tohoto seznamu a v momentě, kdy se kolo zastaví a staré žetony zmiznou, jsou jeho (nebo dokonce jejich) sázky na plátně zobrazeny.

6.7 Generování náhodných čísel

Ke generování náhodných čísel je použita funkce `Next(int maxValue)` instance třídy `Random`, která vrací nezáporné celé číslo menší než předaný parametr. Během simulací systému se však ukázalo, že tato metoda *občas* vygeneruje číslo rovné hodnotě `maxValue`, tedy mimo požadovaný rozsah. Proto je příkaz pro generování čísla umístěn v cyklu, který zajistí, že bude z kýženého intervalu.



Obrázek 6.3: Diagram tříd klientské části – Ruleta.

Kapitola 7

Srovnání s podobnými programy

Věnovat se jednotlivým programům dostupným na webu by bylo zbytečné. Mají totiž mnoho společného. Jak bylo zmíněno v úvodu, žádná ruleta neumožňuje nastavit si vlastní limity ani ve cvičné hře. Žádný program neposkytuje možnost hrát systém nebo jej nasimulovat.

Obrázek 7.1: 32Red casino [10]

Oproti Ruletě jsou kasina na webu pochopitelně o poznání propracovanější, zejména co se týče uživatelského rozhraní. Některá se dokonce pokoušejí o jakési skákání kuličky při jejím dopadu, ani tyto pokusy se však realitě moc nepřibližují. Mnoho rulet se nazývá 3D, žádná však neumožňuje jakýkoli pohyb kamerou. Význam pojmenování 3D je nejasný.

Obrázek 7.2: Go Casino [11]

V čem je Ruleta opravdu jedinečná, je podpora nanášení žetonů, tj. pokládání žetonů pouhým pohybem myši při stisknutém tlačítku. Tato geniálně jednoduchá vlastnost, kterou by bezesporu ocenili všichni uživatelé hrající pod vlivem omamných látek, se nevyskytla u žádné ze zkoušených rulet. Tvrdit u všech by bylo samozřejmě troufalé, neboť vyzkoušet všechna dostupná kasina reálně nelze.

Obrázek 7.3: I ♡ roulette [12]

Kapitola 8

Závěr

Aplikace Ruleta umožňuje hraní a simulování herních systémů. Vedle implementovaných systémů lze do programu importovat vlastní systém.

Hraní on-line se zaměřilo na rychlost a bezpečnost. Sázky jsou umísťovány bez velké latence a zároveň nikdy nemůže nastat situace, že by server neakceptoval sázku a klient ji považoval za přijatou. Požadavku na trojrozměrné prostředí je také vyhověno. Uživatel může pohybovat kamerou podle libosti.

Přestože cíle dané v úvodu jsou splněny, k nasazení Rulety a hraní o skutečné peníze by bylo vhodné doimplementovat mezistupeň mezi cvičnou a skutečnou hrou, tedy hrou, která by nebyla skutečná (o peníze), ale byla by v multi-player modu. Dále při každém přisednutí ke stolu je hráčovi poskytnut výchozí kapitál, protože si server neudržuje žádné informace o uživatelích, tedy ani o konečném stavu kapitálu. Tím se redukuje problém registrace a přihlašování.

Problematika databází (ve smyslu uchovávání údajů o hráčích) není v záměru této práce. Aplikace Ruleta nevznikla za účelem skutečného užívání jakoukoli firmou poskytující libovolné internetové sázení.

8.1 Další možná rozšíření

V průběhu hraní Rulety každého jistě napadne mnoho námětů na další rozšíření nebo vylepšení.

- Co aplikace zcela postrádá, jsou zvuky: položení žetonu, padající kulička, oznámení o padnutém čísle atd. Díky DirectSound není problém je do programu přidat, nejprve je však nutné nějaké zvuky vytvořit.

- Při pohybu kamerou po místnosti by mohly být vidět ostatní stoly a u stolů přísedící v podobě avatarů nebo dokonce modelů osob. Hráči by si mohli ke stolům přisedávat přímo při „procházení“ po místnosti.
- V místnosti by navíc nemusely být jen ruletní stoly, ale různé jiné hry. Potom by se už jednalo o celé virtuální kasino. To by mohlo být vhodné téma pro týmovou práci.

Literatura

[1] Christian N., Evjen B., Glynn J., Skinner M., Watson K., Jones A.: *C# 2005 Programujeme profesionálně*, Computer Press, a.s., Brno, 2006

[2] Conger D.: *Remoting with C# and .NET*, Gearhed Press, Indianapolis, Indiana, 2003

[3] Rammer I.: *Advanced .NET Remoting*, Apress, Berkeley, California, 2002

[4] Miller T.: *Programujeme 3D hry v jazyce C#*, Computer Press, a.s., Brno, 2007

[5] Pelikán J.: *Výpočet průsečíků paprsku se scénou*, http://cgg.mff.cuni.cz/∼pepca/lectures/pdf/intersection.pdf

[6] *Historie rulety – Historie – Ruleta36.cz*, http://www.ruleta36.cz/historie/historie-rulety

[7] Kahan J.: *Ruleta – Jak přelstít casino – tipy hráče rulety J. Kahana pro ruletu i jiné hry*, http://ruleta.cz.sweb.cz/

[8] *Ruleta online – ruleta-online.cz*, http://www.ruleta-online.cz/

[9] *Roulette System – Winning Roulette Strategy – How to Win*, http://www.genuinewinner.com/

[10] *Online Casino 32Red – The Online Casino of the Decade*, http://www.32red.com/

[11] *GoCasino.Com – Online Casino*, http://gocasino.com/

[12] *Play Online Roulette Casino Games, Play Roulette on the Internet*, http://www.iloveroulette.com/

Příloha A

Obsah přiloženého CD

Součástí práce je datový disk s následujícím obsahem:

- Text této práce ve formátu PDF.
- Instalační balík klientské části aplikace: `Instalace/Klient`.
- Instalační balík serverové části aplikace: `Instalace/Server`.
- Zdrojové kódy aplikace: `Zdroje/Ruleta`.
- Projekty pro vytvoření instalačních balíků: `Zdroje/Setup`.

Příloha B

Instalace klientské části Rulety

Pro instalaci Rulety spusťte soubor `Ruleta.msi` a postupujte podle pokynů. Po úspěšné instalaci vyberte v Nabídce start položku Programy > Ruleta > Konfigurovat. Otevře se konfigurační soubor obsahující mimo jiné řádek, který vypadá takto:

```
<add key="ServerAddress" value="tcp://localhost:8824/Lobby" />
```

V tomto řádku nahraďte řetězec `localhost` skutečnou adresou serveru, kterou znáte od poskytovatele Rulety. Následně soubor uložte a zavřete.

Program spustíte v Nabídce start vybráním položky Programy > Ruleta > Hrát ruletu.