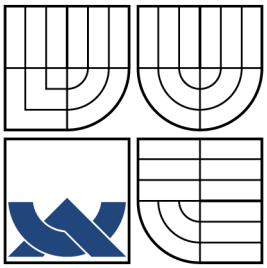
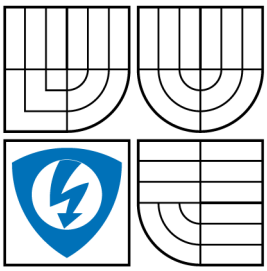


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

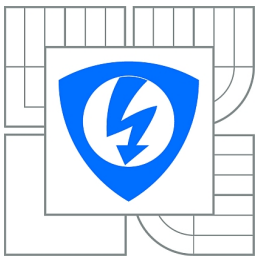
SÍŤOVÁ KOMUNIKACE V .NET FRAMEWORK
NETWORK COMMUNICATION IN THE .NET FRAMEWORK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
DAVID MÜLLER

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. IVO LATTENBERG, Ph.D.

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	David Müller	***ID:***	126761
Ročník:	3	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Síťová komunikace v .NET Framework

POKYNY PRO VYPRACOVÁNÍ:

S využitím programovacího jazyka C# a vývojového prostředí Microsoft Visual Studio 2010 navrhněte složitější webovou aplikaci, která bude sloužit jako vzorový program pro výuku. Realizovaná aplikace by měla představovat webovou implementaci hry Qwirkle pro 2 až 4 hráče. Program řádně okomentujte a při návrhu pamatujte na názornost a transparentnost. Aplikaci otestujte na kompatibilitu s prohlížeči Internet Explorer a Firefox.

DOPORUČENÁ LITERATURA:

[1] PROSISE, J. Programování v Microsoft .NET, Nakladatelství Computer Press, a.s. 2003, 736 s., ISBN 8072268791
[2] TROELSEN, A. C# a .NET 2.0 profesionálně, Zoner press, 2006, 1200 s., ISBN 8086815420
[3] LACKO L. ASP.NET a ADO.NET 2.0 Hotová řešení, Computer Press 2006, 385 s., ISBN 80-251-1028-1

Termín zadání:	6.2.2012	***Termín odevzdání:***	31.5.2012

Vedoucí práce: doc. Ing. Ivo Lattenberg, Ph.D.

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Bakalářská práce seznamuje s prostředím .NET, s jeho prostředky pro síťovou komunikaci a s principy webových aplikací a služeb. Stručně je také popsán jazyk C#, JavaScript a zásady objektového programování. Na uvedených základech je ve druhé části práce popsán vývoj webové aplikace – hry Qwirkle pro 2 až 4 hráče. Vývoj je dokumentován od analýzy, přes návrh a implementaci až po testování hotové aplikace.

KLÍČOVÁ SLOVA

klient, server, .NET, C#, webové služby, JavaScript, JSON

ABSTRACT

This bachelor thesis provides an introduction to the .NET environment and its tools for network communication as well as the principles of web applications and services. It also briefly describes the C# programming language, JavaScript and the principles of object-oriented programming. On this basis, the second part of the thesis describes the development of a web application: The Qwirkle game for 2 to 4 players. Its development is documented from analysis through design and implementation up to testing of the finished application.

KEYWORDS

client, server, .NET, C#, web services, JavaScript, JSON

MÜLLER, David *Síťová komunikace v .NET Framework*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2012. 53 s. Vedoucí práce doc. Ing. Ivo Lattenberg, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Síťová komunikace v .NET Framework“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

Rád bych poděkoval panu doc. Ing. Ivo Lattenbergovi, Ph.D. za cenné postřehy a rady. Také chci poděkovat rodině za podporu při psaní práce a zejména za aktivní účast při jejím testování.

OBSAH

ÚVOD

Cílem bakalářské práce je demonstrovat prostředky síťové komunikace v prostředí .NET při naprogramování webové verze hry Qwirkle.

Pro přehlednost je práce rozdělena do několika kapitol. Nejdříve si představíme síťové komunikace teoreticky a dále jejich implementaci v prostředí .NET. Následovat bude přehled technologií, na kterých bude řešení založeno a nástrojů, které k tomu budeme používat. Přitom nelze vynechat ani pravidla hry, ty dotvářejí společný rámec znalostí nutných pro vyřešení úkolu zadání. V dalších kapitolách bude v postupných krocích popsán vývoj aplikace. Začneme analýzou, ve které se budeme snažit definovat požadavky a zvolit optimální způsob jejich řešení. Pokračovat budeme návrhem a vlastní implementací, abychom nakonec mohli přistoupit k testování.

Vzhledem k velkému rámci výchozího teoretického základu a omezenému rozsahu práce, budou jednotlivá témata často probrána velmi stručně. Detailnější pohled však bude věnován klíčovým nebo implementačně zajímavým momentům řešení.

1 TECHNOLOGIE

Než začneme s vývojem aplikace pokusíme se o nezbytný úvod do technologií, které pak také z větší části budeme v našem řešení používat.

1.1 Síťové komunikace

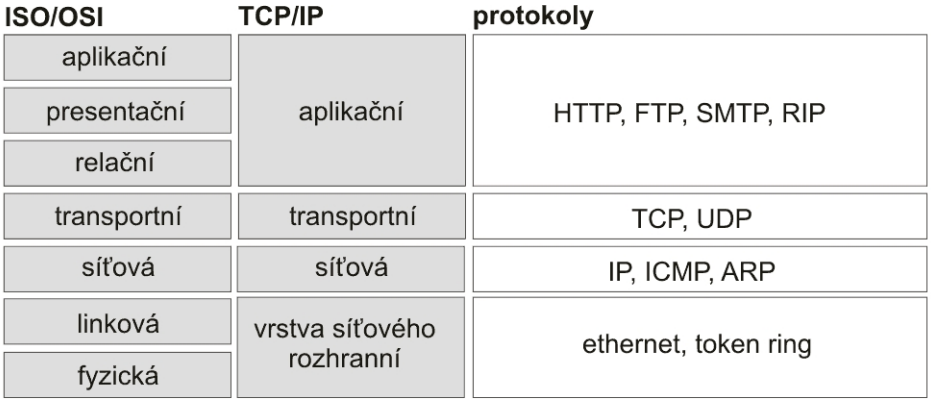
Síťové komunikace jako výměnu informací mezi účastníky můžeme popisovat z mnoha úhlů pohledů. Například podle způsobu přenosu informací je dělíme na simplexní (přenos probíhá jen jedním směrem – dětská chůvička), duplexní (plně obousměrný přenos – telefon) nebo poloduplexení (směry přenosu se střídají – pár Walkie-talkie vysílaček).

Jiným kritériem může být rozsah šíření. Podle toho rozlišujeme unicastové přenosy (směrem k jedinému cíli – načtení www stránky), přenosy multicastové (směrem ke skupině – videokonference) nebo broadcastový přenos (všesměrové vysílání – televizní vysílání).

1.1.1 Referenční modely

Když popisujeme, analyzujeme nebo projektujeme složitější systém, pomáháme si často jeho rozdělením na menší celky s jednoduší funkcionalitou. Podobně i komunikační systémy bývá zvykem dělit do jednotlivých vrstev, které plní svou specifickou úlohu a proti sousedním vrstvám vystupují s jasně definovaným rozhranním. Obecně platí, že nižší vrstva je vždy poskytovatelem služeb pro vyšší vrstvu a současně využívá služeb vrstvy nižší. Mimo tento „vertikální“ popis je možný i pohled „horizontální“ – při komunikaci si totiž spolu musí rozumět vždy stejné vrstvy protějších zařízení, to zajišťují tzv. komunikační protokoly.

Pro obecný komunikační systém byl Mezinárodní organizací pro normalizaci navržen sedmivrstvý referenční model ISO/OSI. Tento model je však velmi komplexní a pro potřeby komunikace v prostředí internetu se více hodí tzv. referenční model TCP/IP, který se skládá ze 4 vrstev a jednotlivé protokoly tohoto modelu byly vyvinuty agenturou ARPA už koncem 60. let. Model TCP/IP vychází z předpokladu, že spolehlivost přenosu si zajistí až koncové body komunikace. Principiální odlišnost od robustnějšího modelu ISO/OSI je také v tom, že model TCP/IP byl navržen pro sítě s nespojovým charakterem služeb. [1]



Obr. 1.1: Referenční modely a protokoly

1.1.2 Klient server a P2P

V teorii síťové komunikace se musíme podívat na síťovou architekturu také z pohledu organizačního, kde se nabízí dvě hlavní uspořádání.

Architektura klient/server – je model ve kterém jsou dva druhy uzlů klienti a servery. Klienti jsou aktivní a své požadavky směřují k serverům, jejichž úkolem je naslouchání a vyřizování příchozích požadavků. V uvedených rolích se může nacházet např. webový prohlížeč a webový server, ale např. i dva různé programy na jednom počítači nebo dokonce dvě části jednoho programu. Hlavní výhodou tohoto uspořádání je snadná údržba a to hlavně u internetových aplikací, kde lze za chodu provádět úpravy programu nebo i výměnu hardware. Nevýhodou pak závislost na běžící službě serveru nebo možnost jeho zahlcení nárůstem počtu požadavků od klientů.

Architektura P2P – je naopak modelem, kde jsou si všechny uzly rovny a plní tak současně roli serverů i klientů. Na tomto principu fungují různé služby pro sdílení obsahu nebo komunikační programy. Hlavní výhodou tohoto řešení je, že rostoucí počet uzlů znamená zvyšování kapacity sítě. Na tomto principu stojí i infrastruktura internetu.

1.2 .NET Framework obecně

.NET Framework je prostředí spjaté s vývojovým nástrojem Visual Studio firmy Microsoft. Dnes už však existují vývojová i běhová řešení založená na této platformě i od jiných subjektů. Verze 3.5 je součástí posledních verzí Windows (Windows 7, resp. Server 2008 R2) a současnou verze 4.0 lze doinstalovat do všech „novějších“

verzí Windows operačních systému (od verze XP, resp. Server 2003).

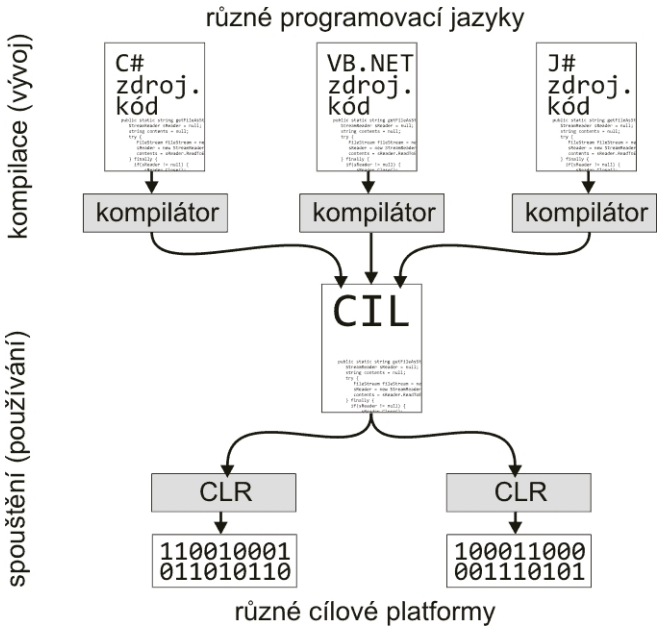
.NET Framework dává vývojářům několik velkých výhod:

- plná interoperabilita s existující kódem,
- naprostá jazyková volnost, a to nejen v samotné volbě jazyka, ale v možnosti jejich kombinací až do té míry, že lze například do jiného jazyka podědit třídu, propagovat výjimku nebo napříč jazyky ladit běh aplikace,
- propracovaná knihovna tříd, která skrývá komplikovaná API volání,
- jednoduché rozmisťování.

Základními stavebními pilíři platformy jsou:

- CLR (Common Language Runtime) tj. společné běhové prostředí, které se stará o správu paměti, bezpečnostní kontroly atd.,
- CTS resp. CLS (Common Type System, resp. Common Language Specification) tj. společná knihovna typů, která zajistí jazykovou nezávislost programových konstrukcí a použitých datových typů,
- CIL (Common Intermediate Language) tj. společný „mezijazyk“, který je procesorově i platformě nezávislý.

Výsledkem kompilace je tzv. assembly (sestavení), což může být např. spustitelná aplikace nebo modul (např. služba operačního systému).



Obr. 1.2: Princip vzniku spustitelného kódu

Obrázek 1.2 dokumentuje otevřenost a univerzálnost celého řešení. Zdrojový kód může být pořízen nejen v uvedených zdrojových jazycích, které jsou součástí Visual Studia, ale v libovolném jazyce, jehož kompilátor bude dodržovat pravidla CLS (Common Language Specification). Stejnou otevřenost má běhové prostředí, které musí jen prostě správně přeložit CIL do svého nativního vykonávacího kódu. [2]

Poznamenejme, že assembly .NET mohou být vyvíjeny i spouštěny pod různými operačními systémy jako Linux, Mac OS apod.

1.3 Síťová komunikace v .NET Framework

.NET disponuje rozsáhlými knihovnami pokrývající všechny běžně řešené problematiky. Síťovou komunikaci podporuje hned celá řada zabudovaných tříd v několika jmenných prostorech. Komunikaci tak můžeme snadno navázat v síťové, transportní i aplikační vrstvě za použití předdefinovaných i vlastních protokolů.

Důležité komunikační třídy

Následující přehled obsahuje důležité komunikační třídy a jejich možnosti:

- Třída `Socket` je třída z prostoru `System.Net.Sockets`, která umožňuje na úrovni transportní vrstvy propojit místní a vzdálený koncový bod. V konstruktoru máme na výběr řadu možností jak zvolit způsob adresování, typ soketu a typ protokolu. Můžeme tak přenášet stream, datagramy nebo dokonce přistupovat v režimu `Raw` a přímo vytvářet IP hlavičku.
- Třídy `TCPClient`, `TcpListener` a `UDPClient` jsou další třídy z jmenného prostoru `System.Net.Sockets`. Umožňují efektivnější práci, než poskytuje třída `Socket` tím, že obstarávají rutinní záležitosti s navazováním spojení a odstiňují tím aplikaci od těchto činností. Na straně serveru zahájíme naslouchání pomocí instance třídy `TcpListener`, TCP spojení zahajuje klient a na serveru pak příchozí spojení můžeme převzít jako odkaz na objekt `TCPClient`. Při UDP komunikaci spojení nevzniká a na obou stranách pracujeme pouze s metodami třídy `UDPClient`.
- Třídy `WebRequest`, `WebResponse` ze jmenného prostoru `System.Net` umožňují klientovi komunikovat se serverem na úrovni protokolů aplikační vrstvy (http, ftp, atd.). Typické použití je např. pro stažení souboru ze vzdáleného serveru. Požadavek je vytvořen jako instance `WebRequest` a příchozí odpověď převezmeme jako `WebResponse`.

Třída `HttpListener` umožňuje přímo a jednoduše zabudovat do aplikace webový server. Nabízí možnosti synchronního i asynchronního vyřizování příchozích klientských požadavků pomocí tříd `HttpListenerContext`, `HttpListenerRequest` a `HttpListenerResponse`. [3]

1.3.1 Webová aplikace

V případě, že budeme vyvíjet webovou aplikaci můžeme dříve uvedený „nízkoúrovňový“ pohled zapomenout a přejít rovnou ke třídám, které dovolují straně webového serveru vystavět funkční aplikaci pomocí ovládacích prvků ze jmenného prostoru `System.Web.UI.WebControls`.

Webovou aplikací rozumíme řešení, kdy je podstatná část aplikační logiky soustředěna na webovém serveru, a klient k ní přistupuje přes webový prohlížeč. Ten pak obstarává především činnosti spojené s obsluhou uživatelského rozhranní. Takové řešení se snadno udržuje a je široce dostupné bez nutnosti instalace klientské části aplikace.

Webová aplikace funguje na principu odeslání klientského dotazu protokolem `http` a po zpracování na webovém serveru vrácením vygenerované odpovědi nejčastěji HTML stránky. Odpověď může obsahovat mimo prvků vlastního uživatelského rozhraní např. data načtená z databáze, multimediální obsah nebo třeba skript pro spuštění ve webovém prohlížeči.

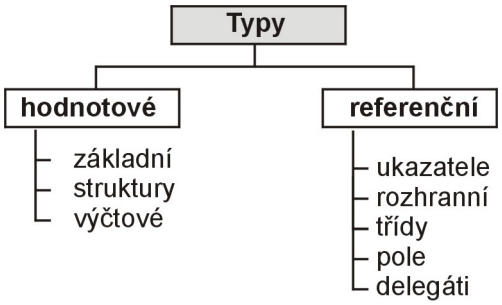
1.3.2 Webová služba

Často si vystačíme z úspornější variantou než nabízí webová aplikace a aplikujeme webovou službu. Webová služba obvykle poskytuje přístup k nějakému datovému zdroji a její volání může být např. jednou z částí při zpracování klientského požadavku na webovém serveru (webová stránka cestovní kanceláře zobrazující aktuální kurzovní lístek). Může však být volána také přímo z webového prohlížeče (našeptávač při vyplňování formulářového pole) nebo dokonce z běžné desktopové aplikace (kontrola aktualizací).

Webová služba může existovat samostatně většinou jako celá kolekce různých metod které nabízejí řešení platformě nezávislých úloh pro jiné servery nebo aplikace. Mimo tyto „plnohodnotné“ služby existuje i úsporné řešení, kdy mohou být metody služeb dokonce přímo zabudované do kódu generujícího webovou stránku. Taková metoda označená atributem `WebMethod` nepotřebuje deklarace přístupové infrastruktury (endpoint, binding, behavior) a bude také využita v našem řešení.

1.4 Jazyk C#

Veškerý programový kód na straně serveru je napsán v programovacím jazyku C# . Název napovídá, že se vyvinul z jazyka C, ale možná ještě více se podobá jazyku Java. Vznikl v roce 2002 společně s Frameworkem .NET a je proto pro něj optimalizován – většina základních typů totiž přesně odpovídá CTS viz obr. 1.3.



Obr. 1.3: Typový systém C# [4]

C# je orientován na objektové programování, o což se také opírá řešení bakalářské práce. Mezi základní pilíře OOP (objektově orientovaného programování) patří:

- **Zapouzdření** znamená, že na každý objekt existuje vnější a vnitřní pohled. Zvenku objekt voláme pomocí metod a pracujeme s jeho vlastnostmi aniž bychom znali skutečnou implementaci. Zevnitř naopak řešíme všechnu potřebnou funkcionalitu ve vztahu k vnějšímu světu. Vedlejším, i když významným faktorem, je také zabezpečení vnitřních stavů objektu proti nežádoucí modifikaci zvenku.
- **Dědičnost** je způsob jak se vyhnout duplicitnímu kódu. Při dobrém návrhu můžeme s výhodou vytvářet nové třídy na základě už existujících s tím, že funkčnost předka můžeme doplňovat i překrývat. Pro pohled zvenku to znamená další výhodu v tom, že se k potomkům dá přistupovat vlastnostmi i metodami jeho předků.
- **Polymorfismus** umožňuje různě implementovat stejné rozhranní. Pro vnější svět se mohou dva objekty jevit jako stejné, ale uvnitř mohou fungovat rozdílně.

1.5 JavaScript

Zvyšující se požadavky na uživatelské rozhranní webových aplikací lze řešit různými způsoby. Jedním z řešení je klientské skriptování, které přenáší část aplikační logiky

do režie webového prohlížeče. Tímto způsobem je možné snížit zátěž webového serveru, ale především velmi výrazně zvýšit rychlost odezvy uživatelského rozhranní a rozšířit jeho možnosti.

JavaScript v prohlížeči Firefox stejně jako JScript v Internet Exploreru jsou dialekty standardu ECMAScript. V této práci bude používáno tradiční pojmenování JavaScript [1], pod kterým se bude rozumět klientský skript napsaný tak, aby fungoval shodně v prohlížečích Internet Explorer i Firefox [2].

Syntaxe JavaScriptu je příbuzná jazykům založeným na syntaxi C. JavaScript je jazyk interpretovaný – překládá se za běhu a je na rozdíl od přísné typovosti jazyka C# dynamicky typovaný – proměnné se deklarují bez udání typu a jsou automaticky přetypovány podle kontextu použití.

1.5.1 Objektové programování v JavaScriptu

JavaScript podporuje objektové programování i když se jedná o poněkud odlišnou implementaci než např. v C# . Objekt je zde chápán jako asociativní pole klíč/hodnota a takto je ho možné také přímo založit. Jiným způsobem je definice funkce, která se následně při použití klíčového slova `new` dostává do role konstruktoru. Navíc je možné pomocí klíčového slova `prototype` libovolně rozšiřovat tuto definici. Další odlišností je, že obecná specifikace nezná modifikátory přístupu, i když jsou v některých implementacích podporovány. I v JavaScriptu lze však dosáhnout ochrany privátních členů a skrytí implementace, a to pomocí různých návrhových vzorů. V práci je takto použit návrhový vzor *modul* [5].

Poznamenejme, že i interprety JavaScriptu mají, podobně jako je to v běhovém prostředí .NET, zabudovaný Garbage collector, čímž se značně zjednodušuje práce při psaní zdrojových kódů.

1.5.2 Knihovny jQuery a jQuery–UI

V řešení klientské části je použita knihovna jQuery. Jedná se svobodný software, který usnadňuje programování v JavaScriptu. Knihovna se od svého vydání v roce 2006 velmi rychle rozšířila a stala se součástí mnoha významných internetových projektů. Výhodou jQuery je především oddělení chování od sktrukuty HTML, nezávislost na prohlížeči, výhodný přístup k elementům DOM pomocí selektorů nebo snadná rozšiřitelnost o další funkce.

[1] Uveden poprvé v prohlížeči Netscape Navigrator v roce 1995

[2] Kompatibilita pro Internet Explorer a Firefox je definována v zadání práce

V klientské části je použito také rozšíření jQuery–UI, které je zaměřeno na podporu tvorby uživatelského rozhranní a obsahuje funkce pro různé efekty a interakce mezi objekty.

1.6 Ajax

Ajax (původně *Asynchronous JavaScript and XML*) je dnes obecné označení pro asynchronní komunikaci s webovým serverem. Tradiční princip, kdy na základě svého požadavku dostává webový prohlížeč kompletní obsah stránky se všemi obsaženými prvky nemusí být vždy výhodné. Technologie Ajax umožňuje jiný přístup – komunikaci jednotlivých částí webové stránky s webovým serverem bez toho, že by došlo k ovlivnění okolních prvků. Moderní interaktivní uživatelská rozhranní se bez technologie Ajax prakticky neobejdou (našeptávače, validátory apod.) a tato technologie je přímo zabudována do některých webových kontrolů (např. `UpdatePanel`).

Jinou možností je přímé použití asynchronního volání z klientského skriptu, které bude součástí našeho řešení. Serializovaná data webového požadavku odesíláme na server a v definované metodě přijímáme a zpracováváme odpověď. Tento způsob je sice relativně pracný, ale jeho možnosti jsou prakticky neomezené.

1.7 JSON

JSON (JavaScript Object Notation) vznikl pro potřeby serializace jako alternativa k existujícímu formátu xml. Jak název napovídá, je podporován JavaScriptem a umí serializovat objekty. Jeho podpora je zabudována i v .NET, přičemž proti příliš komplexnímu xml je úspornější s rychlejším zpracováním a navíc snadno čitelný. Ukázky serializace ve formátu JSON jsou uvedeny v kapitole 5.4.1.

1.8 HTML a CSS

HTML a CSS tvoří spolu se skriptovacím jazykem JavaScript trojici základních technologií pro uživatelské rozhranní většiny webových aplikací.

Značkovací jazyk HTML vznikl v roce 1990 jako jazyk pro přenos hypertextu a i když od té doby prošel významným vývojem, je stále základním formátem pro internetovou prezentaci. Dokument je v HTML strukturován pomocí tzv. tagů, které nesou informaci o vzhledu a začlenění jednotlivých jeho částí.

Základní dělení HTML tagů:

- Strukturální – řídí výslednou strukturu dokumentu,
- Popisné – vyjadřují druh informace, kterou uvozují,
- Stylistické – přímo definují vzhled.

Trendem je náhrada stylistických tagů použitím tzv. CSS. Výhodou CSS je oddělení vzhledu dokumentu od jeho obsahu. Popis vzhledu je pak možné opakovaně použít pro různé dokumenty nebo naopak pro stejný dokument použít různý vzhled např. v závislosti na výstupním zařízení.

1.9 Modelovací jazyk UML

Výčet použitých technologií uzavírá modelovací jazyk UML. Tento analytický pomocník, vznikl sjednocením různých modelovacích technik pro objektovou analýzu a návrh systémů. Zásadní výhodou UML je převážně grafické vyjádření a tedy velká názornost. Díky tomu mohou na bázi UML komunikovat analytici se zadavateli, spolupracovat analytici s vývojáři a také vývojáři na různých platformách mezi sebou. [9]

Přehled nejfrekventovanějších UML diagramů:

- **Diagram případů užití** modeluje interakce uživatelů se systémem, tedy každou jednotlivou funkčnost a aktéra (nejčastěji roli), který danou funkci vyvolá.
- **Diagram tříd** modeluje reálný svět pomocí objektových tříd, tedy předpisu zahrnujícího datový obsah a funkčnost. Navenek komunikuje objekt dané třídy definovaným rozhranním s tím, že vnitřní implementace zůstává ukryta. Pro diagramy tříd jsou charakteristické především vazby mezi třídami (asociace, agregace, generalizace apod.)
- **Sekvenční diagram** je jedním z typů modelů objektové spolupráce, který na příslušné čáře života vyjadřuje jednotlivé aktivity příslušných objektů a zejména interakci mezi nimi (volání funkcí, zasílání zpráv).
- **Stavový diagram** vyjadřuje nejčastěji životní cyklus objektu dané třídy, tedy stavy, kterými objekt prochází na základě přijatých zpráv.
- **Diagram aktivit** zachycují aktivity a přechody mezi nimi. Výhodné jsou zejména v případech, kdy potřebujeme modelovat paralelní procesy nebo např. pro popis složitějšího chování.

2 POUŽITÉ NÁSTROJE

V následující kapitole je uveden krátký popis softwarových nástrojů, které byly použity při vývoji aplikace.

2.1 Visual Studio 2010

Základním nástrojem pro vytvoření klientské i serverové části aplikace se stalo vývojové prostředí Visual Studio verze 2010 v edici Professional od Microsoft Corporation. Projekt byl založen jako ASP.NET Empty Web Application. V prostředí Visual Studia je při psaní zdrojových kódů k dispozici zvýrazňování syntaxe, pohotové IntelliSense a propracované debuggování. Program je pro nekomerční použití studentům k dispozici zdarma.

Navíc použijeme rozšíření JScriptEditorExtensions od Microsoft Corporation, které zpříjemňuje také editaci klientských skriptů na úroveň popsanou výše. Toto rozšíření je volně k dispozici.

2.2 CorelDraw X3

Pro vytvoření hracích kamenů a herní plochy byl použit grafické nástroj CorelDraw ve verzi X3 od Corel Corporation. Tento komplexní nástroj je určen pro vytváření a úpravy vektorové i bitmapové grafiky a podporuje export do různých grafických formátů. Program se lze naučit ovládat poměrně snadno, i když dokonalé zvládnutí všech nabízených funkcí je již poměrně komplikované. Program existuje také ve verzi pro nekomerční použití.

2.3 Software ideas modeler - verze 5.04

Software ideas modeler je UML modelovací nástroj českého autora. Zvládá 16 druhů diagramů, automatizované generování zdrojového kódu a do určité míry i reverzní inženýring ze zdrojových kódů i ze zkompilovaných .NET assembly nebo z databází. Ovládání je snadné a dá se velice rychle zvládnout. Pro nekomerční použití je program k dispozici zdarma.

3 HRA QWIRKLE

3.1 Stolní hry

Stolní hry se po určitém období útlumu, kdy byly vytlačovány hrami počítačovými, dostávají zpět na své tradiční pozice. Ani virtuální realita moderních her podpořená mimořádnou výkonností dnešních počítačů nedokáže konkurovat obyčejným vjemům z klasických materiálů jako jsou papír, dřevo nebo kámen při posezení s rodinou nebo přáteli u společného stolu. Důkazem je rostoucí nabídka a prodej těchto her, přičemž jejich ceny jsou mnohdy vyšší než ceny her počítačových.

Moderním pojmem se tedy opět začíná stávat tzv. desková hra. Podle definice by mělo jít o hru, která je založena na nějakém hracím plánu, který je součástí hry. V dnešním významu slova se však stávají hry stolní a deskové víceméně synonymem nahrazujícím dříve oblíbený výraz „společenské“.

Stále více deskových, karetních i jiných klasických her dostává postupně svou počítačovou implementaci – pomyslný kruh se uzavírá. Motivací je možnost hrát s někým, s kým u stolu sedět nemůžeme nebo např. to, že danou hru prostě fyzicky k dispozici nemáme a v extrémním případě vlastní pohodlnost (hru se nám nechce hledat nebo chystat, při hře se nám nechce počítat body apod.)

Pro tyto případy existují webové servery, které nabízí mnoho různých her často i v mnoha různých kategoriích. Výhodou je, že taková hra je okamžitě k dispozici bez nutnosti instalace nebo složitého nastavování. Konkrétní řešení tedy bývá většinou webová stránka obsahující Flash, Java plugin apod.

3.2 Qwirkle

Hra Qwirkle byla publikována firmou Mindware v roce 2009 a v roce 2011 se stala vítězem Spiel des Jahres [1]. Hra je přitom velmi jednoduchá jak z hlediska pravidel, tak i vlastního provedení. Celou hru tvoří 108 hracích kamenů s barevnými symboly a látkový pytlík, pro počáteční losování, hraje se na stole bez hracího plánu. Princip hry se podobá hře Scrabble – přikládáním kamenů se vytvářejí řady a sloupce v pravidelném pravoúhlé síti podle daných pravidel. Jde o hru strategickou i když s určitým podílem náhody.

[1] Ocenění předních recenzentů společenských her z německy mluvících zemí, které se uděluje od roku 1979 [6]

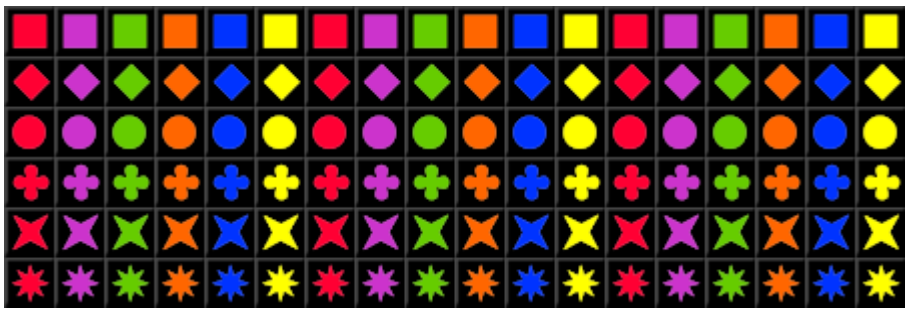
V následující kapitole budou popsána pravidla hry, která byla zpracována podle překladu pravidel [7] přiložených k originální verzi hry. Původní ilustrační obrázky jsou nahrazeny výřezy z obrazovek hotové aplikace.

3.2.1 Pravidla hry

Cílem hry je dosažení co největšího počtu bodů vhodným doplňováním řad a sloupců herními kameny.

Kameny

Herní kameny jsou celé černé a mají z jedné strany barevný symbol. Tyto symboly tvoří 36 jedinečných kombinací 6 barev a 6 tvarů, které se 3 krát opakují a dohromady tvoří celou herní zásobu $6 * 6 * 3 = 108$ kamenů viz obr. 3.1.



Obr. 3.1: Kompletní sada 108 kamenů

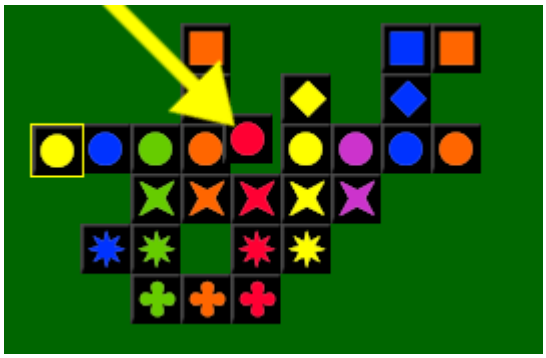
Zahájení hry

Každý hráč si vylosuje 6 kamenů a položí si je tak, aby na ně ostatní hráči neviděli. Každý hráč vyhlásí jak dlouhou řadu je schopen z těchto kamenů sestavit a hráč s nejdelší řadou hru začíná vyložením celé této řady, při shodě začíná hráč starší. Řada musí obsahovat buď kameny se stejným symbolem nebo stejnou barvou symbolu, nikoliv však obojí – stejné kameny v řadě být nesmí.

Průběh hry

Hráči se střídají v přikládání kamenů tak, že vždy pokračuje hráč sedící vlevo. Každý hráč musí v každém kole hry přiložit minimálně 1 kámen, pokud je jich více, musí vždy jít o kameny stejné barvy nebo stejného symbolu a tyto kameny musí vytvářet souvislou řadu s již vyloženými kameny, i když samy spolu sousedit nemusí. Kameny

se kladou do pravoúhlé sítě, přičemž jednotlivé souvislé úseky řad mohou obsahovat pouze kameny stejného symbolu nebo stejné barvy, ale nesmí obsahovat kameny shodné. Přiložení na obrázku 3.2 nelze provést, protože by se do souvislé řady dostaly 3 shodné kameny.



Obr. 3.2: Kámen nelze položit, výsledná řada by obsahovala několik stejných kamenů

Po přikládání si hráč dobere kameny ze společné zásoby do celkového počtu 6, případně méně, pokud již společná zásoba kameny neobsahuje.

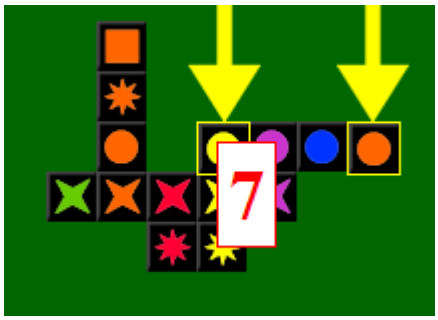
Jestli společná zásoba obsahuje kameny, může hráč namísto přikládání provést výměnu kamenů. V případě že přikládat nemůže, je výměnu povinen provést. Výměna kamenů probíhá tak, že hráč namísto přikládání odloží libovolný počet vlastních kamenů (maximálně však do počtu kamenů ve společné zásobě), stejný počet kamenů si vezme ze společné zásoby a následně odložené kameny vmíchá do společné zásoby zpět. V případě výměny hráč přikládání kamenů v daném kole neprovádí.

Bodování

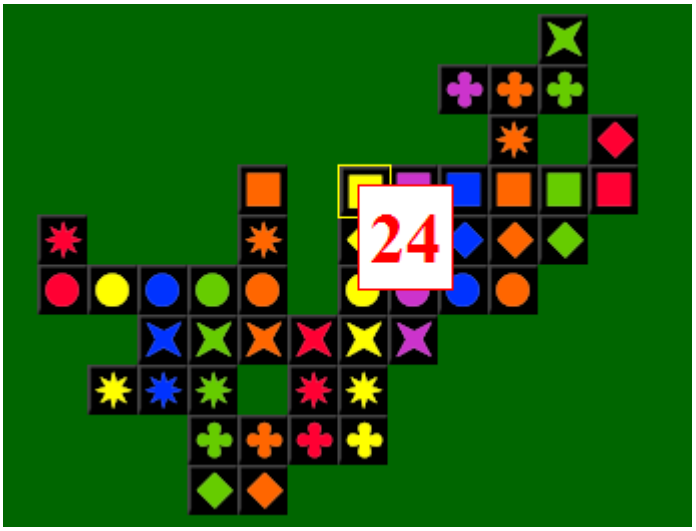
V každém kole získává hráč body za všechny přiložené kameny a navíc také za kameny, které už byly položeny a tvoří s vyloženými kameny souvislou řadu. V případě, že přiložený kámen tvoří dvě řady (řada a sloupec) započítávají se za něj 2 body (obr. 3.3). Souvislá řada 6 kamenů se nazývá Qwirkle (obsahuje všechny barvy daného symbolu nebo všechny symboly dané barvy) a hráč za ni získává dalších 6 bodů navíc. Takto lze získat přiložením jediného kamene i 24 bodů (obr. 3.4 – doplnění dvou řad na celkový počet 6 kamenů + dvakrát Qwirkle).

Ukončení hry a vítěz

Hra končí, když některý z hráčů vyloží poslední kámen. Za ukončení hry získává tento hráč navíc 6 bodů (obr. 3.5). Vítězem hry se stává hráč s nejvyšším celkovým



Obr. 3.3: 4+3 body za doplnění dvou řad



Obr. 3.4: 6+6+6+6 bodů za doplnění na dvojnásobný Qwirkle

počtem bodů.

3.2.2 Aplikace pravidel

V mém projektu se budeme držet všech klíčových pravidel hry (průběh hry, kameny, umísťování, počítání). Při zprostředkování hry počítačem se však nabízí nové možnosti nebo naopak určitá omezení při jejich aplikování. V této kapitole budou případy popsány:

Společná zásoba kamenů

Určitou vlastní interpretaci si dovolíme u společné zásoby kamenů pro dobírání. V překladu pravidel [7] přiložených k originální verzi hry (©2010 MindWare) tvoří společnou zásobou kameny ležící na stole obrácené lícem dolů. Jiná pravidla [8]

Obr. 3.5: Ukončení hry

používají pro tento účel ke hře přiložený plátěný pytlík, což se jeví z ohledem na potřebu promíchání kamenů při akci výměna kamenů účelné.

Úvodní úvaha byla využít v aplikaci symbol pytlíku, toto řešení je sice implementačně jednoduché, ale omezuje subjektivní pocit hráče z toho, že si vybírá svůj kámen sám. Konečné řešení je proto matice 6x18 kamenů umístěná v levé části obrazovky, ze které jsou zvolené kameny odebírány hráčem technikou drag–and–drop. Toto řešení dává pocit volby a zároveň přirozeným způsobem informuje o množství zbývajících kamenů.

Výměna kamenů

Výměna kamenů je činnost, kdy hráč namísto přikládání může libovolný počet držených kamenů vyměnit za kameny nové ze společné zásoby. Pravidla vyžadují, aby hráč musel napřed určit, které kameny odevzdá, následně si vybral nové kameny a nakonec zamíchal odevzdané kameny do společné zásoby.

Po různých pokusech zvítězilo nakonec vtipné řešení, kdy odevzdávané kameny budou prostě taženy na místa kamenů, které si chceme ze společné zásoby odebrat, tyto budou automaticky nahrazovat odevzdané kameny, ale zůstanou prozatím lícem dolů (k odhalení dojde po ukončení celého tahu výměny). Tím je zajištěno, že naplánované kameny hráč vymění, při výměně si kameny vybírá, ale výměna není ovlivněna tím, které kameny průběžně získal.

Zamíchání bude nakonec provedeno klientským skriptem.

Pořadí hráčů ve hře

Podle pravidel je pořadí hráčů určeno tak, že celou hru začíná hráč, který si na začátku vybral kameny, ze kterých je schopen sestavit nejdelší řadu, při shodě nejstarší z těchto hráčů a následně se střídají podle jejich pozice stolu.

Pro rozběh hry je důležité, aby začínal hráč, který vyloží nejvíce kamenů, proto je pořadí hráčů, které vzniklo postupným připojením ke hře po nultém kole přeuspořádáno na základě zvolených kamenů. Při shodě rozhoduje původní pořadí při přihlášení s tím, že pomocná pravidla jak určit pořadí (věk a pozice) nejsou uvažovány.

Informace v průběhu hry

Hráči spolu mimo polohy kamenů na stole (vyložených i ve společné zásobě) sdílí v reálné hře také přímý pohled na vytváření řad ostatními hráči.

Přenášet mezi všechny hráče každý jednotlivý tah v reálném čase by sice bylo řešitelné, ale zvolíme komunikačně méně náročné řešení, které nakonec přinese lepší přehled – všechny kameny přiložené v průběhu uplynulého kola budou označeny. Navíc bude mít každý hráč v průběhu celé hry k dispozici souhrnnou informaci o celkovém počtu bodů, o bodech získaných v uplynulém kole i o počtu kamenů v držení jednotlivých hráčů (pokud bude nižší než 6) viz obr. 3.6.

Obr. 3.6: Informace o hráčích

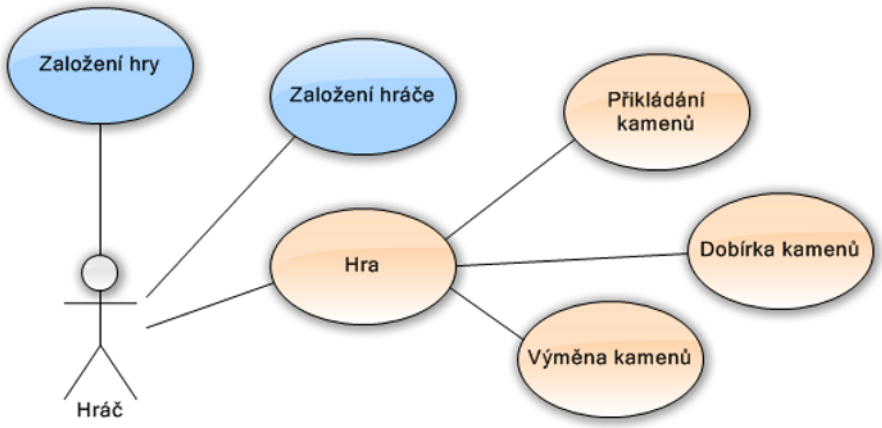
4 ANALÝZA

V zadání práce je řešení definováno jako webová aplikace. Toto řešení je výhodné zejména z pohledu uživatele, který není nucen instalovat speciální software, ale i z hlediska komunikační technologie, která bude řešena na protokolu http a bude tak snadno aplikovatelná v prostředí internetu. Předpokládá se existence webového serveru, ke kterému se uživatelé připojí. Pro vývoj, testování i praktické předvedení plně vyhoví vestavěný vývojový server v nástroji Visual Studio 2010.

V zadání je dále definováno, že aplikace je určena pro 2 až 4 hráče, což je v souladu s pravidly hry, která jsou uvedena v kapitole 3.2.1 a jsou samozřejmou součástí výchozích požadavků na aplikaci.

Požadavkem plynoucím ze zadání je také názornost řešení. Klientská i serverová část bude proto navržena v duchu OOP a bohatě komentována ve zdrojových kódech.

Analýza bude podpořena několika UML diagramy. V úvodu sestavíme diagram případů užití, který jasně zachycuje všechny interakce uživatelů se systémem a plyne z něj veškerá požadovaná funkčnost budoucí aplikace.



Obr. 4.1: Diagram případů užití

4.1 Rozdělení úloh mezi klienty a server

Jedním z požadovaných výsledků analýzy je dekompozice, z tohoto je zásadní optimálně rozdělit úlohy mezi klientskou a serverovou část aplikace.

Klientská část bude především poskytovatelem uživatelského rozhranní, serverová pak místem, které zajistí předávání dat (stavu hry, hráčů, umístění kamenů) mezi hráči. Z tohoto pohledu není zcela zjevné, kam by měla být aplikována vlastní

logika hry. V případě umístění na server je však nutné počítat s významným zhoršením odezvy uživatelského rozhranní, která by nastávala při validaci umisťování jednotlivých kamenů, případně uživatelsky nepohodlné opakování části nebo celého tahu, pokud by byla kontrola prováděna až po ukončení tahu. Z uvedených důvodů volíme distribuci herní logiky v rámci klientského skriptu do webového prohlížeče hráče.

Kontrola při vykládání kamenů na straně klienta je algoritmicky velmi podobná výpočtu bodů z tahu hráče, proto bude výhodné obě tyto činnosti spojit a provádět je u klienta. Výsledkem činnosti skriptu u klienta tedy bude předávat na server po každém tahu polohu kamenů, u kterých došlo k její změně a výsledný počet bodů, které hráč za tah obdržel.

Úkolem serveru bude především zajistit u každého hráče stejný obraz celé hry. Dále pak řídit zakládání nových her, rušení her opuštěných (dohraných nebo delší dobu neaktualizovaných) a vstup hráčů.

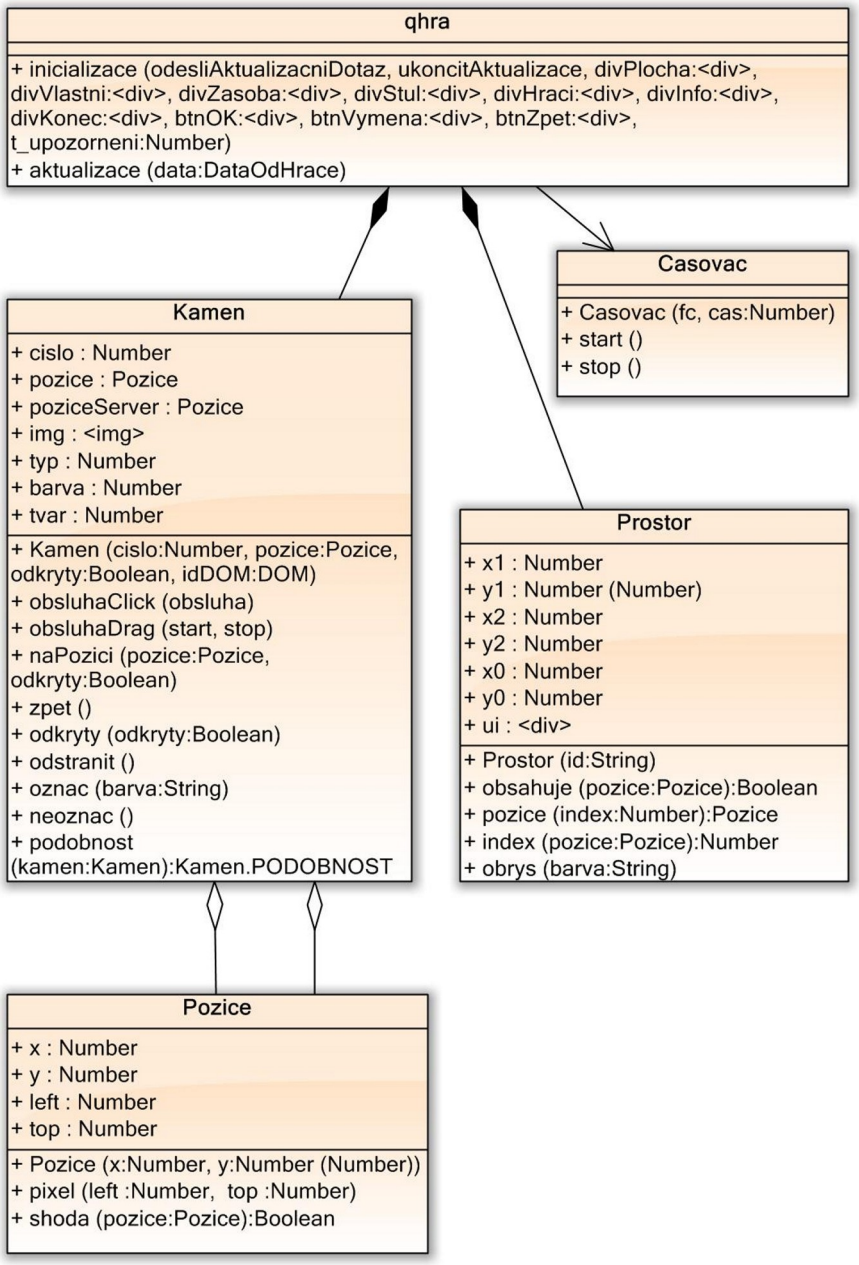
4.2 Objektové třídy

Dalším krokem bude návrh objektových tříd pro klientskou a serverovou část aplikace.

4.2.1 Klientská část

Veškerá logika hry bude řešena ve třídě `qhra` a komunikace ve třídě `qclient`. Herní logika bude operovat především s instancemi třídy `Kamen` a třídy `Pozice`. V průběhu vývoje aplikace se ukázalo, že bude výhodné udržovat u každého kamene dvě instance pozic, jedna bude vyjadřovat aktuální pozici u daného klienta a druhá pozici, které je uložená na serveru a u ostatních klientů.

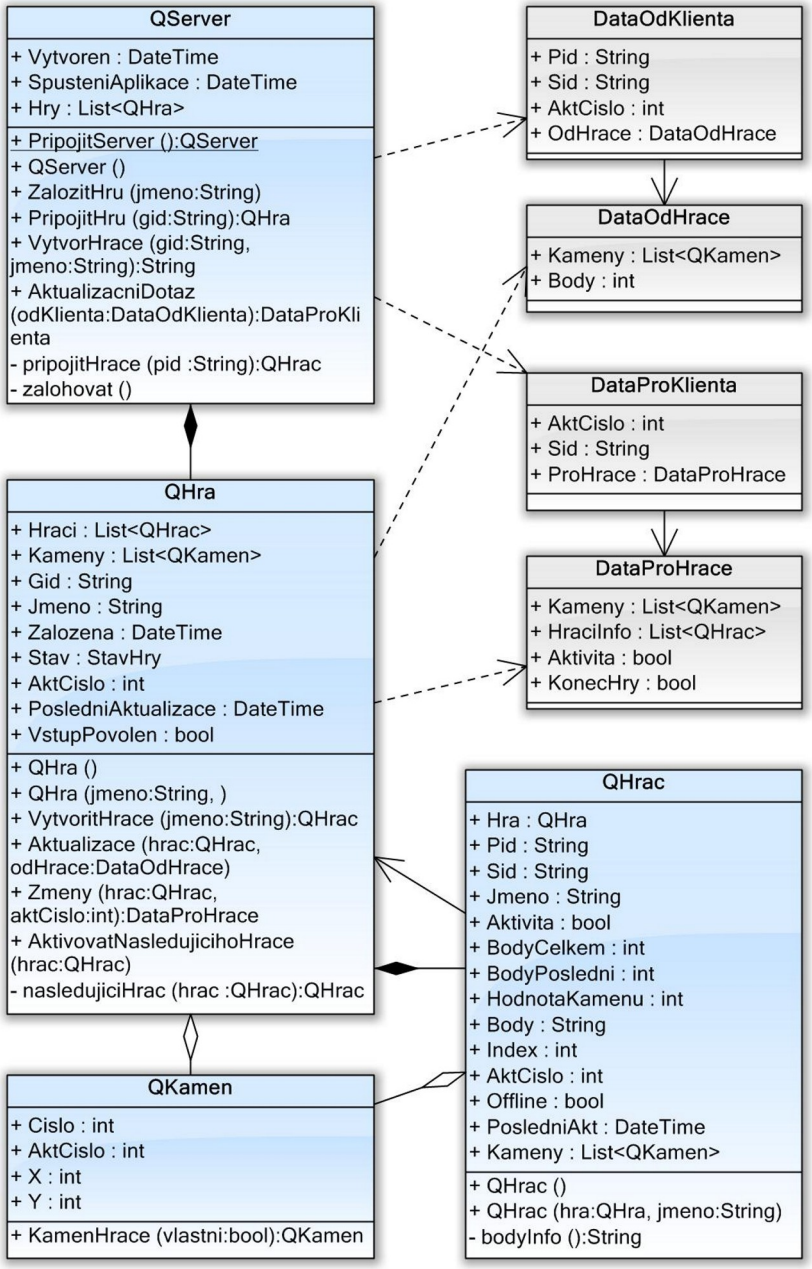
Vztahy mezi třídami jsou patrné z diagramu klientských tříd viz obr. 4.2. Diagram pro zjednodušení neobsahuje třídu `qclient`, která nebude na `qhra` třídně vázána (vazba vznikne až za běhu vložením odkazů na funkce obsluh). Dále jsou pro přehlednost z diagramu vypuštěny privátní členy tříd.



Obr. 4.2: Diagram tříd klientské části aplikace

4.2.2 Serverová část

Serverová část aplikace je zastoupena především třídou `QServer` operující s instancemi QHra, které si drží kolekce hráčů (QHrac). Instance třídy `QKamen` mohou náležet jak hráčům (kameny v zásobníku daného hráče) tak hře (vyložené kameny a společná zásoba). Ostatní třídy (`DataOdKlienta`, `DataOdHrace`, `DataProKlienta`, `DataProHrace`) se použijí při komunikaci s klientskou částí. Diagram tříd je na obrázku 4.3.



Obr. 4.3: Diagram tříd serverové části aplikace

4.3 Stavové diagramy

Abychom mohli popsat stavy objektů navrhovaných tříd, využijeme stavové diagramy. Z pohledu stavů jsou zajímavé zejména objekty kamenů a her na klientské i serverové straně aplikace.

4.3.1 Stavy hry

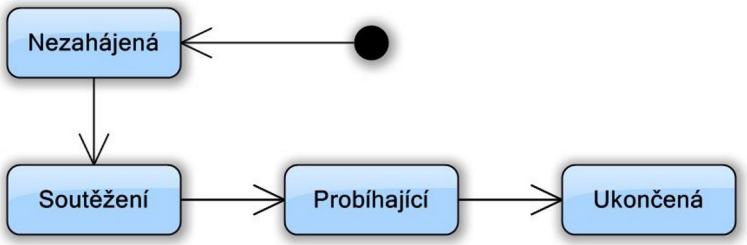
Z pohledu hry bude nutné rozlišovat tyto stavy:

- *nezahájená* – případní hráči se mohou přidávat ke hře,
- *soutěžení* – nulté kolo hry, hráči se střídají v úvodní dobírce kamenů podle pořadí připojení ke hře,
- *probíhající* – hra probíhá, hráči se střídají v aktivitě (přikládání kamenů + dobírka),
- *ukončená* – hra skončila, žádný z hráčů nemá přidělenou aktivitu.

Tento model byl v rámci optimalizace (aby nebyl zaváděn další speciální stav hráče) upřesněn tak, že i u nezahájené hry je jeden hráč „na tahu“. Jde o případ, že se ke hře připojí minimálně 2 hráči a hra tedy může dobráním kamenů aktivním hráčem začít – přechod do stavu *soutěžení*. V případě, že se ke hře připojí maximální počet hráčů (4), je přechod do stavu *soutěžení* proveden automaticky. Předem tedy není nutné zvlášť definovat kolik hráčů se bude které hry účastnit.

Po nultém kole proběhne podle hodnoty dobraných kamenů přeuspořádání hráčů a hra přejde do stavu *probíhající*.

Přechod do stavu *ukončená* je v souladu s pravidly proveden v případě, že aktivní hráč už nemá a ani nemůže dobrat žádné kameny. Aby bylo možné u klienta prezentovat výsledné pořadí hry bez rozšiřování komunikačního protokolu, proběhne opět přeuspořádání hráčů, tentokrát podle celkového počtu získaných bodů. Stavový diagram je na obrázku 4.4.



Obr. 4.4: Stavy hry (na serveru)

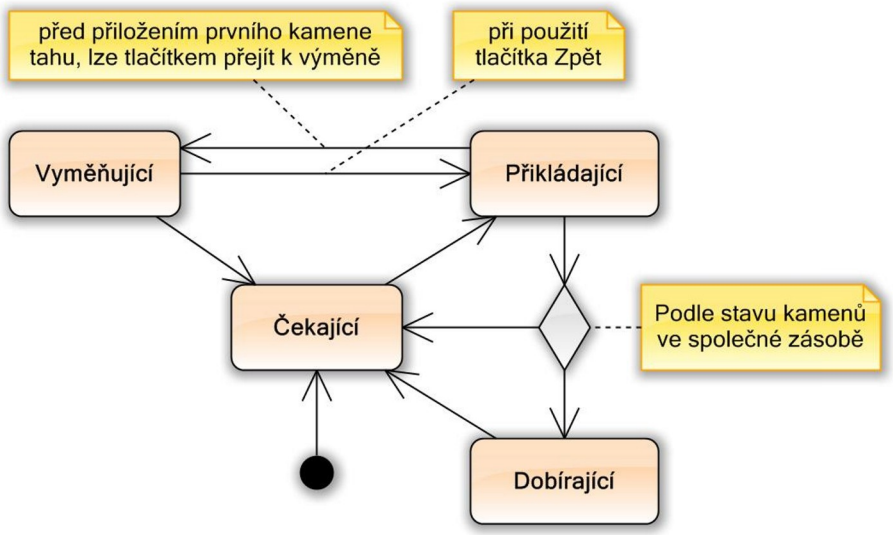
4.3.2 Stavy hráče

U hráče se pravidelně střídá stav kdy čeká až na něj přijde řada se stavem kdy hraje. Protože však vlastní hra zahrnuje různé činnosti (nesmí být např. povoleno dobírání

kamenů při jejich vykládání), budou stavy hráče u klienta zavedeny takto:

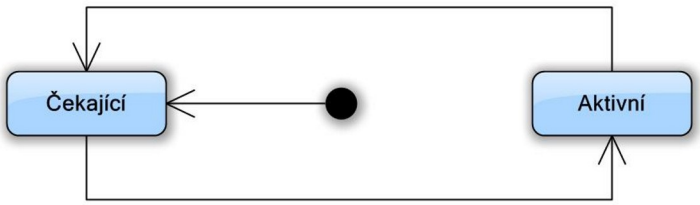
- *čekající* – nemůže nic dělat, protože hra ještě nezačala, skončila nebo není na tahu,
- *přikládající* – umisťuje kameny na hrací plochu,
- *dobírající* – vybírá si pro sebe kameny ze společné zásoby,
- *vyměňující* – mění jeden nebo více vlastních kamenů za stejný počet kamenů ze společné zásoby.

Stavový diagram je na obrázku 4.5.



Obr. 4.5: Stavy hráče (klient)

Úlohou serveru je z pohledu stavů hráče pouze předání aktivity na následujícího hráče, bude stačovat udržovat pro objekt hráče na serveru pouze 2 stavy: *čekající* a *aktivní* viz obr. 4.6.



Obr. 4.6: Stavy hráče (server)

V původním návrhu byly voleny stejné stavy také pro objekt hráče na serveru, což umožňuje mírně zrychlit hru tím, že dobírající hráč neblokuje hráče na tahu (tito spolu nemají kolizi při sdílení společných prostředků). Nicméně efekt z takového

řešení není výrazný, protože čas strávený umísťováním kamenů je obvykle výrazně vyšší než čas strávený dobírkou. Další komplikací by byla výměna kamenů – hráč, který by se rozhodl pro výměnu kamenů, by musel přejít zpět do stavu čekání, dokud by předchozí hráč nedokončil svou dobírku.

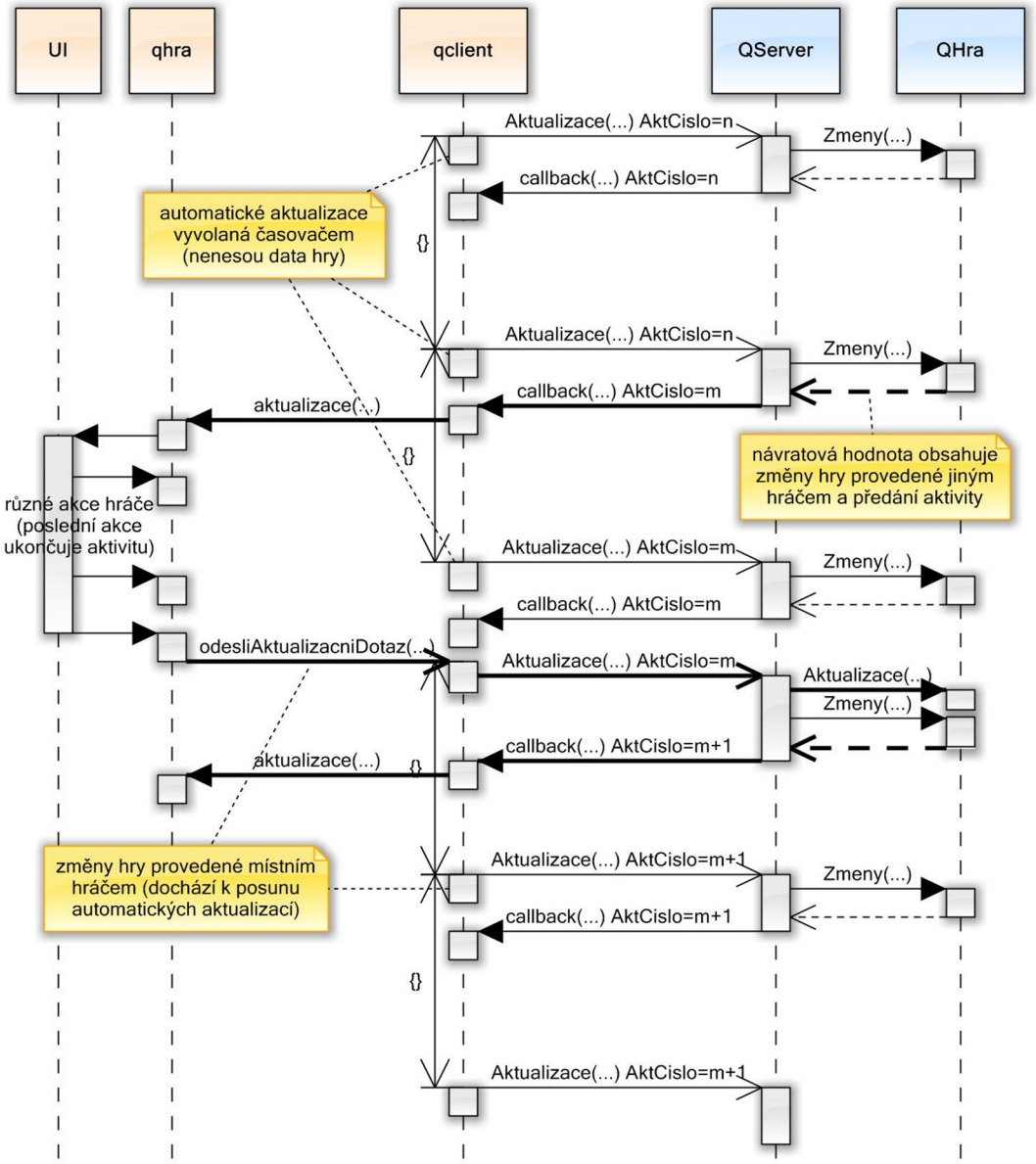
4.4 Komunikace

Komunikaci mezi klientem a serverem zachycují sekvenční diagramy. Diagram 4.7 zachycuje normální komunikaci, při které zrovna došlo ke změně ve hře, a místní hráč získává aktivitu (silnější šipky směrované vlevo). Po poslední akci provedené hráčem jsou místní změny předány na server (silnější šipky směrované vpravo). Nezávisle na tom však klient posílá na server neustále „prázdné“ aktualizační dotazy, pro zjištění změn ve hře. Tato činnost bude řízena vestavěným časovačem a v případě požadavku na odeslání dat mimo aktualizační periodu, dojde k posunu následujících aktualizací tak, jak je to na diagramu zachyceno.

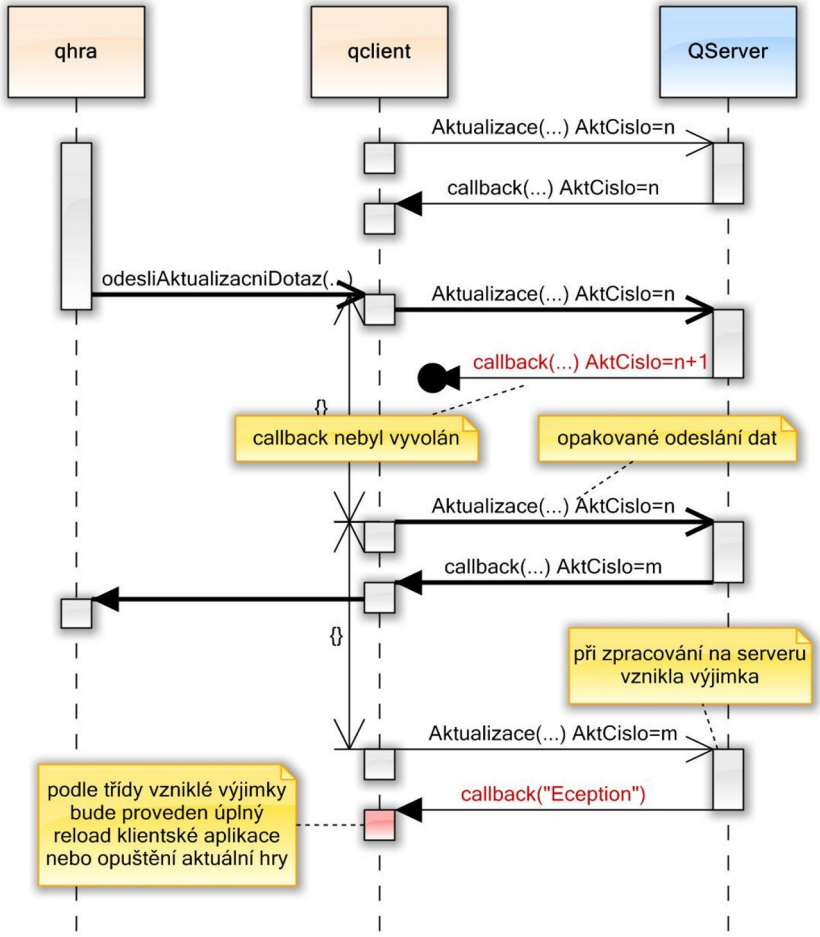
Druhý diagram 4.8 ukazuje dvě různé mimořádné situace. V horní části je naznačena chyba v komunikaci, tu bude řešit objekt `qclient`, který opakuje předchozí odesílání dat (objekt `qhra` je od této situace odstíněna, data považuje za odeslaná). Ve spodní části je situace, kdy je při zpracování na serveru vytvořena výjimka, tato bude doručena a identifikována v objektu `qclient`, který situaci vyřeší (reload klientské aplikace, opuštění aktuální hry).

4.5 Uživatelské rozhraní

U webové hry máme několik možných druhů řešení uživatelského rozhranní. Pro prezentačně náročnější bychom volili Adobe Flash nebo Silverlight, pro jednodušší si lze vystačit s kombinací JavaScriptu a technologie Ajax, jiné řešení by ještě mohl být Java applet. Naše volba míří ke kombinaci JavaScript/Ajax, která plně vyhovuje a je nativně podporována současnými prohlížeči. Zvolená technologie je již přímo zabudována v některých vizuálních komponentách nebo např. v kontejneru `UpdatePanel`. Tento princip by však byl poněkud těžkopádný, proto zvolíme přímou aplikaci těchto technologií.



Obr. 4.7: Sekvenční diagram – normální průběh hry



Obr. 4.8: Sekvenční diagram – mimořádná situace

5 NÁVRH A IMPLEMENTACE

Přestože webové aplikace různých her existují často ve větším množství variant, nebyla v době vzniku této práce zjištěna žádná webová implementace hry Qwirkle. Návrh ani vlastní implementace nejsou tedy nijak inspirovány žádným jiným řešením.

Zjednodušeně lze říci, že celé řešení sestává ze dvou webových stránek. Úvodní stránka zobrazuje výběr existujících herních instancí, ze které pak uživatel vstupuje na stránku vlastní hry. V následujících kapitolách bude postupně popsána úvodní stránka, klientská a serverová část stránky hry a v samostatné kapitole komunikace probíhající při hře.

Protože klientská i serverová část aplikace sestává z poměrně velkého množství kódu, nelze zde podrobně rozebrat všechny jednotlivé metody a vlastnosti. Budeme se tedy snažit o celkový pohled a podrobně popíšeme klíčové nebo jinak zajímavé detaily.

5.1 Vstupní stránka

Vstupní stránku tvoří jednoduchý webový formulář, jehož datově vázanými prvky jsou dva vnořené prvky `asp:repeater`. Vnější je navázán na seznam herních instancí (třídy `List<QHra>`) a vnitřní na přehled hráčů (třída `List<QHrac>`). U každé hry je uveden čas jejího založení, pojmenování hry, které definuje zakládající hráč a především stav hry. Do hry ve stavu `StavHry.Nezahajena` je povolen vstup nového hráče (hráč vloží své jméno, které je na straně serveru validováno, aby nekolidovalo se jménem existujícího hráče). Rovněž je na serveru znovu provedena kontrola aktuálního stavu hry (ten se mohl změnit) a nepodařený vstup generuje výjimku, která je u klienta prezentována svou zprávou ve standardním alert boxu. Novou hru lze jednoduše založit definováním jména hry.

5.2 Klient

Klient je navržen jako stavebnice se snahou co nejvíce oddělit komunikační část od herní logiky a uživatelského rozhraní. Nejdůležitějšími stavebními prvky klientské části jsou objekty [1] `qclient` a `qhra`. Při kódování obou byl použit návrhový vzor

[1] Objekty `qclient` a `qhra` sice vystupují v diagramu tříd, ale protože nejsou navrženy pro instancování budeme se u nich držet pojmenování *objekt*

modul [5], který umožňuje v duchu OOP důsledně skrýt implementaci a vystavit jen rozhraní.

Objekty `qclient` a `qhra` jsou navrženy jako nezávislé a jejich propojení vznikne až v inicializační části klientské aplikace (tedy za běhu). Přes svá rozhraní jsou potom propojeny tak, aby mezi nimi mohla probíhat datová výměna. Objekt `qhra` dostává reference na metody `qclient.odesliAktualizacniDotaz` a `qclient.ukoncitAktualizace`, zatímco `qclient` dostává referenci na metodu `qhra.aktualizace`. Podobným způsobem jsou ještě oba objekty připojeny k uživatelskému rozhranní, přesněji k jeho ovládacím a zobrazovacím prvkům.

V klientských skriptech (javascript) jsou názvy funkcí a proměnných s výjimkou funkcí kladených do role konstruktorů (`Kamen`, `Pozice`, `Prostor`, `Casovac` a `InfoOkno`), definovány s malým počátečním písmenem podle vzoru `camelCase`. Privátní členy tříd navíc začínají znakem `_`.

5.2.1 Uživatelské rozhranní

Než popíšeme jednotlivé stavební prvky klientské části, zastavíme se stručně u návrhu uživatelského rozhranní.

Jedná se webový formulář, který definuje pomocí elementů `<div>` celou hierarchii DOM potřebnou pro vlastní hru.

Jsou to především:

- `<div id="stav">` – zobrazuje stav komunikace,
- `<div id="hraci">` – zobrazuje účastníky hry,
- `<div id="vlastni">` – prostor pro kameny hráče,
- `<div id="zasoba">` – prostor pro kameny společné zásoby,
- `<div id="stul">` – prostor pro vykládání kamenů.

V hlavičce `<style>` je k nim definován vzhled a jejich identifikátory jsou předány objektům `qhra` a `qclient`. Při testování aplikace se objevily připomínky ke strohému vzhledu, proto byla aplikována textura dřeva [10], která rámuje základní prostory hry tak, aby zároveň vytvářela mírný 3D dojem. Klasická tlačítka byla nahrazena ikonovými, každé se 4 stavy vzhledu (nefunkční, v klidu, po najetí, při stisku). Veškeré chování tlačítek je řízeno aplikací kaskádových stylů.

Rozdělení obrazovky je navrženo tak, aby plocha pro vykládání kamenů mohla být co možná největší. Její absolutní velikost se odvíjí od navržené velikosti kamene a je taková, aby nedošlo k omezení hry tím, že by se přikládané kameny přiblížili

Obr. 5.1: Ukázka části obrazovky při hře

k hranici plochy. Velikost kamene byla s ohledem na dobrou čitelnost a uchopitelnost symbolu navržena 25 x 25 px. Z herních testů vyplynulo, že od položení prvního kamene se hra do žádného směru nerozšířila více než o 15 kamenů. S tímto ohledem byla navržena plocha pro vykládání 30 x 30 kamenů a první kámen hry je při vyložení vždy automaticky přesunut do jejího středu.

Rozložení jednotlivých ploch i jejich velikost může být snadno upravena, protože herní logika na skutečných souřadnicích a rozměrech přímo nezávisí. Blíže to bude popsáno v kapitole 5.4.3.

5.2.2 Objekt `qclient`

Úkolem objektu je odesílat na server změny polohy kamenů provedené místním hráčem a také přijímat ze serveru změny ve hře (změny kamenů, aktivity, bodů) a předávat je ke zpracování objektu `qhra`.

Objekt `qclient` tak poskytuje komunikační mezivrstvu vlastní hře (`qhra`) a instanci herního serveru `QServer`, podrobně to bude rozebráno v kapitole 5.4. Asynchronní komunikace je realizována voláním `PageMethods`, které tvoří proxy k metodám serveru označeným atributem `WebMethod`. V konečné verzi bylo implementován přístup přes `PageMethods._staticInstance`, které umožňuje získat referenci tak, aby mohlo být aplikováno případné zrušení volání pomocí `abort()` [11].

`qclient` má k dispozici referenci na HTML objekt, který využívá pro zobrazování aktuálního stavu komunikace.

Jedná se o tyto zprávy:

- „...“ – čeká se na odpověď serveru,
- „ON-LINE“ – odpověď serveru byla přijata (po najetí myši navíc zobrazuje čas přijetí poslední odpovědi),
- „Chyba“ – odpověď serveru obsahuje chybu (po najetí myši navíc zobrazuje druh chyby),
- „ “ – nic se nezobrazí v případě, že je komunikace ukončena (konec hry).

Objekt `qclient` má veřejné metody:

- `inicializace (serverAktualizace, defaultPage, aktualizaceHry, ...)` – slouží pro úvodní nastavení, kdy je potřeba vložit odkazy na objekt hry a na objekty uživatelského rozhranní,
- `odesliAktualizacniDotaz(data)` – zajistí odeslání aktualizačního dotazu z objektu qhra (nová data se předají na server, automatické aktualizace pro stažení nových dat ze serveru jsou iniciovány přímo objektem qclient),
- `ukoncitAktualizace()` – ukončí automatické aktualizace.

5.2.3 Objekt `qhra`

Úkolem objektu `qhra` je především správa kamenů a chování uživatelského rozhranní. V závislosti na stavu hry je proto kamenům přidělována obsluha událostí click a stop-drag podle jejich umístění ve hře. Obsluha těchto událostí pak vyhodnocuje, jaká bude nová poloha kamene a zajistí provedení této změny. Aby bylo ovládání hry pohodlné, je zajištěno automatické přiskakování kamenů do zvoleného rastru. Toto chování je přímo podporováno návrhem třídy Pozice.

Klíčovou funkcí objektu `qhra` je vyhodnocení platnosti nové pozice pro přikládaný kámen `_platnaPozice(pozice, kamen)`. Tato funkce je volána jako obsluha události drag-stop a jejím druhým úkolem je aktualizovat počet bodů probíhajícího tahu hráče. Algoritmus této funkce stojí na provádění kontrol podobnosti kamenů ve vnořených cyklech (osa, směr, vzdálenost) v okolí zamýšlené pozice umístění.

Další důležitou funkcí je `_nastaveniHrace(stavHrace)`, kde stav hráče může nabývat hodnot `STAV_HRACE = cekani:0, dobirka:1, pokladani:2, vymena:3` a která má

za úkol kromě přepnutí tohoto stavu, provedení změn v uživatelského rozhranní (obsluhy kamenů, tlačítek, označení aktivních oblastí) a někdy také vyvolání komunikace.

Komunikaci zajišťují funkce `_aktualizace(data)` a `_odeslaniStavu()` (na rozhranní vystavené se stejným názvem bez podtržítka). Funkce `_aktualizace(data)` má za úkol především přemístění všech kamenů které byly doručeny v poli `data.Kameny` na nové pozice, dále aktualizovat obsah informačního pole hráčů podle `data.HraciInfo` a provést změny spojené s přijetím aktivity nebo s informací o ukončení hry. Funkce `_odeslaniStavu()` naopak připraví místní změny pro odeslání na server. Jedná se o pole kamenů, které byly přemístěny a o počet bodů získaných při poslední akci hráče (v nultém kole hry je v poli `Body` přenášena informace o hodnotě kamenů v držení hráče).

Objekt `qhra` vystavuje metody:

- `inicializace(odesliAktualizacniDotaz, ukoncitAktualizace, ...)` – slouží především pro předání referencí na metody objektu `qclient` a na objekty HTML stránky,
- `aktualizace(data)` – pro zpravování změn ve hře dat přijatých od objektu `qclient`.

Objekt `qhra` operuje především s instancemi tříd `Kamen`, `Pozice` a `Prostor`.

5.2.4 Třída `Kamen`

Instance klientské třídy `Kamen` udržují informace o každém kamenu ve hře. Přestože je zde určitá míra redundance (barva a tvar symbolu se dá určit z typu nebo přímo z čísla kamene), ukázal se tento vzor jako optimální z pohledu funkcí, které kameny zpracovávají a také přehlednosti kódu.

Vlastnosti objektů třídy `Kamen`:

- `cislo` – jedineční číslo kamene (0..108),
- `img` – obrázek tvaru,
- `pozice` – aktuální místní pozice,
- `poziceServer` – pozice uložená na serveru,
- `typ` – typ kamene (0..36) (vždy 3 kameny jsou stejné) ,
- `barva` – číslo barvy kamene (0..6),
- `tvar` – číslo tvaru kamene (0..6).

Nejdůležitější metody objektů třídy `Kamen`:

- `naPozici(pozice, odkryty)` – přemístí kámen na zadanou pozici a skryje nebo odkryje jeho typ,
- `zpet()` – vrátí kámen na původní pozici,
- `podobnost(kamen)` – vyhodnotí míru podobnosti s kamenem zadaným v argumentu.

Ostatní metody se týkají označování kamene nebo definování obsluhy událostí.

5.2.5 Třída `Pozice`

Objekty třídy `Pozice` nenesou jen souřadnice kamenů v pixelech, ale umí také obousměrnou konverzi do/z navrženého herního rastru, která umožní tzv. přiskakování kamenů při jejich umisťování.

Vlastnosti objektů třídy `Pozice`:

- `x` – vodorovná souřadnice v rastru hry,
- `y` – svislá souřadnice v rastru hry,
- `left` – vodorovná souřadnice v px,
- `top` – svislá souřadnice v px.

Poskytované metody jsou:

- `new(x, y)` – nastaví pozici podle argumentu (rastr),
- `pixel(left, top)` – nastaví pozici podle argumentu (px),
- `shoda(pozice)` – porovná pozici se zadanou pozicí.

5.2.6 Třída `Prostor`

Poslední ze tříd v tomto výčtu je `Prostor`, která zavede určitou míru nezávislosti uživatelského rozhranní a herní logiky (viz. kap. 5.4.3). Třída se využije pro objekty všech tří prostorů ve kterých se mohou nacházet herní kameny (`_prostorVlastni`, `_prostorZasoba` a `_prostorStul`).

Důležité metody jsou:

- `new(id)` – podle identifikátoru DOM vytvoří instanci,
- `obsahuje(pozice)` – zjistí, jestli zadaná pozice náleží do prostoru,

- `pozice(index)` – převede index na pozici,
- `index(pozice)` – převede pozici na index.

5.3 Server

Serverová část aplikace odráží ve svých stavebních prvcích částečně stranu klientskou (`QKamen` – `Kamen` nebo `QHra` – `qhra`), je však vystavěna odlišným způsobem, protože jak vyplynulo z analýzy, server nemá nic vědět o vlastní logice hry a jeho úkolem je zabezpečení přenosu a persistence herních dat. Výjimkou je v tomto směru pouze generování pole kamenů do společné zásoby na začátku každé hry.

Protože jsou data na úrovni instance `QServer` i `QHra` sdíleny, mohlo by dojít ke konfliktu (současný zápis nebo zápis v průběhu čtení), proto je v obou třídách aplikováno zamykání pomocí třídy `ReaderWriterLockSlim`.

Ve zdrojových kódech na serveru (C#) jsou všechny privátní členy definovány s malým počátečním písmenem podle vzoru `camelCase` a veřejné s velkým počátečním písmenem.

5.3.1 Přístup ke službám serveru – třída `QServer`

Třída `QServer` navržená jako singleton zastřešuje veškerý přístup ke službám serverové aplikace. Její metody nabízí připojení nebo založení hry, vytvoření hráče a především zpracování aktualizačních dotazů při vlastní hře (metoda `AktualizacniDotaz`). Zpracování aktualizačních dotazů je nabízeno jako webová služba hostovaná přímo v kódu generujícím stránku hry (`WebMethod`), jedná se o jedinou metodu `public static DataProKlienta Aktualizace(DataOdKlienta odKlienta)`.

Další významné metody jsou:

- `PripojitServer(...)` – připojí nebo vytvoří (pokud neexistuje) instanci herního serveru,
- `ZalozitHru(...)` – založí novou hru (`QHra`),
- `PripojitHru(...)` – vrací instanci hry,
- `VytvorHrace(...)` – vytvoří hráče ve hře.

Protože potřebujeme instanci serveru sdílet mezi všemi dotazy všech klientů byl pro její uložení zvolen `HttpApplicationState`. Abychom uchovali stav her při odstranění objektu z paměti, je aplikováno také zálohování do souboru. Můžeme

využít vestavěný xml serializér (třída `XmlSerializer`), který je v .NET připraven tak, že stačí dekorovat zvolené veřejné vlastnosti atributy `XmlAttribute`, `XmlElement` (nebo naopak `XmlIgnore`) a publikovat u dané třídy bezparametrický konstruktor. Jméno souboru zálohy a jeho umístění určuje položka `qserver` v souboru `web.config`.

5.3.2 Třída herní instance – `QHra`

Úkolem instancí třídy `QHra` je udržování stavu konkrétní hry tak, aby mohla být sdílena všemi hráči. Za tím účelem obsahuje především kolekce `List<QHrac> Hraci` a `List<QKamen> Kameny`. Klíčové vlastnosti každé hry jsou `int AktCislo` (podrobně viz kapitola 5.4.2), unikátní identifikátor `gid` a také jméno hry, které může posloužit pro orientaci při výběru ze seznamu her.

V konstruktoru `public QHra(...)` vzniká na začátku každé hry společná zásoba kamenů, které je náhodně promíchána pomocí algoritmu Fisher–Yates shuffle [12].

Významné metody jsou:

- `Aktualizace(...)` – provádí aktualizaci herní instance daty od účastníka hry,
- `Zmeny(...)` – vrací souhrn změn ve hře pro aktualizaci účastníka hry,
- `AktivovatNasledujicihoHrace(...)` – změní hráče na tahu.

Podobně jako u třídy `QServer` jsou všechny vlastnosti, které chceme zálohovat označeny jako veřejné, a je vystaven také bezparametrický konstruktor.

5.3.3 Třída hráče – `QHrac`

`QHrac` je v podstatě pouze kontejnerem pro vlastnosti hráče. Stejně jako instance hry, tak si i instance hráče drží kolekci `List<QKamen>` – stejně jako si hráč ve hře drží vlastní kameny.

Další významné vlastnosti hráče:

- `string Pid` – unikátní identifikátor hráče, který se používá pro identifikaci hráče i hry,
- `string Sid` – ověřovací identifikátor,
- `string Jmeno` – jméno pro seznam hráčů,
- `int AktCislo` – číslo aktualizace, při které došlo u hráče k poslední změně,
- `DateTime PosledniAkt` – datum posledního kontaktu klienta hráče se serverem.

U vlastnostní třídy `QHrac` a stejně tak i u třídy `QKamen` použijeme mimo atributu pro řízení xml serializace, také atribut `ScriptIgnore`. Označená vlastnost je pak vynechána ze serializace při komunikaci s klientským skriptem – nebude na klienta přenesena. Kombinací uvedených atributů mohou tedy vzniknout až 4 druhy veřejných vlastností, podle způsobu jejich zpracování.

5.3.4 Třída herního kamene – `QKamen`

Podobně i třída `QKamen` je kontejner vlastností herního kamene, na rozdíl od reprezentace u klienta je však mnohem jednodušší, protože pro server nejsou vlastnosti kamenu rozhodující.

Existují pouze dva okamžiky, kdy server aktivně zasahuje do instancí kamenů. V prvním případě je to na začátku hry, kdy je nejdříve vygenerována a následně náhodně promíchána celá kolekce kamenů. Ve druhém případě se jedná o každou přípravu kolekce kamenů, které budou odesílány při aktualizaci na klienta, kdy je u některých kamenů modifikován příznak příslušnosti kamene do kolekce daného hráče uložený v souřadnici Y (viz kapitola 5.4.3)

Vlastnosti třídy `QKamen`:

- `int Cislo` – jedinečné číslo kamene hry (0..107),
- `int AktCislo` – číslo aktualizace, při které došlo k poslední změně ve vlastnostech kamene,
- `int X` – souřadnice vyloženého kamene, případně index (kamen je ve společné zásoba nebo v zásobníku hráče),
- `int Y` – souřadnice vyloženého kamene nebo příznak viz kap. 5.4.3

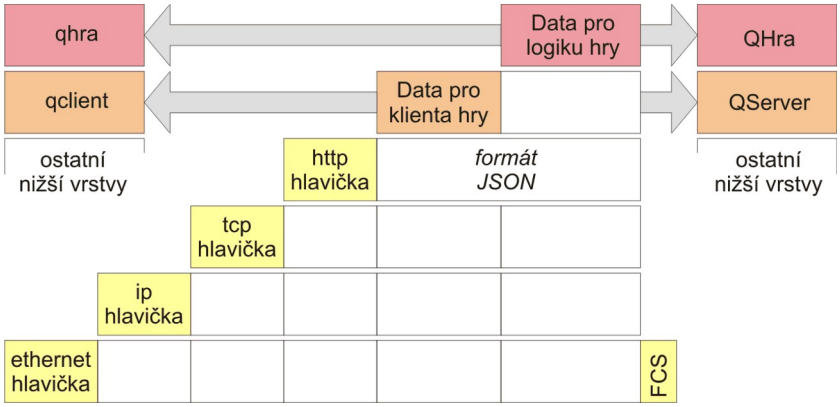
5.4 Komunikace

Veškerá komunikace mezi klienty a serverem probíhá na protokolu http. Na úvodní stránce `Default.aspx` lze po jejím načtení vstoupit do zvolené hry (přesměrování na stránku `Desk.aspx`), případně založit hru novou. Na stránce hry `Desk.aspx` probíhá po jejím úvodním načtení veškerá další komunikace na pozadí asynchronním voláním z klientského skriptu.

Data přenášená při aktualizačních dotazech jsou ve formátu JSON. Serializaci a deserializaci provádí na straně serveru zabudovaná podpora v .NET (JSON je

implicitní formát, variantně by bylo možné použít xml), automaticky jsou serializovány všechny `public` vlastnosti s výjimkou těch, které jsou označeny atributem `ScriptIgnore`). U klienta se o serializaci stará nativní podpora v interpetu JavaScriptu.

Na obrázku 5.2 je naznačeno začlenění přenášených dat do celé hierarchie přenosové infrastruktury.



Obr. 5.2: Přeprava klientských a herních dat na protokolu http

5.4.1 Ukázky komunikace

Pro lepší ilustraci uvedeme ukázky komunikace tak, jak byly zachyceny při skutečné hře. První ukázka se v komunikaci vyskytuje nejčastěji, jedná se automatický aktualizační požadavek, vysílaný klientem pravidelně s periodou `_t_aktualizace` (blok `OdHrace` nenese žádná data).

Výpis 5.1: Automatický dotaz (směr klient → server)

```
{"odKlienta":{"Pid":"b2fcbcb45f3a456ca6433af78899a925",
"Sid":"8bf81f51e4d34890a9acd7169a381149","AktCislo":5,"OdHrace":null}}
```

Další ukázka by mohla být odpovědí serveru na předchozí dotaz. Jiný hráč provedl herní akce a konečné změny ve hře jsou zapsány v bloku `ProHrace` (nová poloha některých kamenů, změna informací o hráčích, delegování aktivity).

Výpis 5.2: Odpověď se změnami ve hře (směr klient ← server)

```
{"d":{"__type":"Qwirkle.DataProKlienta","AktCislo":6,"Sid":"",
"ProHrace":{"Kameny":[{"Cislo":75,"X":23,"Y":18},{"Cislo":74,"X":22,"Y":18},
{"Cislo":41,"X":24,"Y":18},{"Cislo":54,"X":0,"Y":-3},
{"Cislo":25,"X":1,"Y":-3},{"Cislo":23,"X":3,"Y":-3],
"HraciInfo":[{"Jmeno":"Milan","Aktivita":false,"Body":"3/3",
```

```
"Index":0,"Offline":false},{"Jmeno":"Alena","Aktivita":true,"Body":"(2)",
"Index":1,"Offline":false}],"Aktivita":true,"KonecHry":false}}}
```


V poslední ukázce je vidět naopak přenos změn ve hře provedené místním hráčem směrem na server.

Výpis 5.3: Odeslání změn ve hře (směr klient → server)


```
{"odKlienta":{"Pid":"b2fcbcb45f3a456ca6433af78899a925",
"Sid":"8bf81f51e4d34890a9acd7169a381149","AktCislo":6,
"OdHrace":{"Kameny":[{"Cislo":51,"X":23,"Y":19},{"Cislo":86,"X":22,"Y":19},
{"Cislo":12,"X":21,"Y":19},{"Cislo":47,"X":0,"Y":-2},
{"Cislo":37,"X":1,"Y":-2},{"Cislo":30,"X":5,"Y":-2}],"Body":7}}}
```


5.4.2 Aktualizační číslo

Nebylo by efektivní při každém dotazu klienta na server vracet polohu všech kamenů a stavy všech hráčů, ale postačí pouze rozdílová aktualizace na základě proběhlých změn. Důležité je si uvědomit, že tyto změny předávané směrem na klienta však nemusí vždy kopírovat ty změny, které předal předchozí klient hry směrem na server. To může být způsobeno výpadkem komunikace nebo jen tím, když stihnou dva hráči odehrát svůj tah v rámci jedné aktualizační periody.

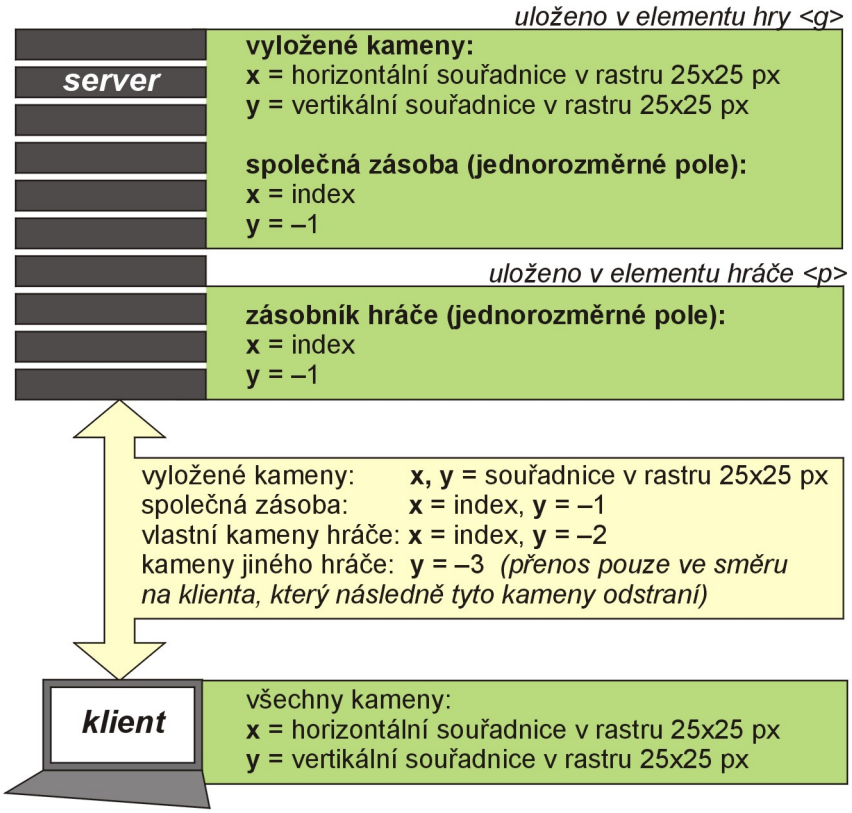
Řešením bude zavedení tzv. aktualizačního čísla, které bude při každé nové změně ve hře zvyšováno. Server bude tedy udržovat stále kompletní obraz hry, aktualizační číslo a u jednotlivých herních objektů také jejich vlastní aktualizační číslo (kdy u nich došlo k poslední změně).

Postačí tedy, aby klient v každé žádosti o nová data ze serveru uvedl své naposledy přijaté aktualizační číslo, podle kterého pak pro něj budou nová data sestavena.

5.4.3 Souřadnice kamene

Podrobnější vysvětlení si zaslouží také způsob, jakým jsou mezi klienty a serverem přenášeny souřadnice kamene a jejich rozdílný význam na obou stranách komunikace. Byl navržen způsob, který je nezávislý na skutečném uspořádání plochy u klienta (mohla by být dokonce u každého jiná) a současně je určitou optimalizací (zahrnuje i informaci o vlastnictví kamene).

Zatím co klient si drží skutečné souřadnice v rastru herní plochy u všech kamenů, na server jsou tyto souřadnice předávány přímo jen u vyložených kamenů, u ostatních proběhne transformace do jednorozměrného pole, přičemž `X` nese index polohy



Obr. 5.3: Rozdílná reprezentace souřadnic kamene

kamene v tomto poli a `Y` informuje o kterou plochu se jedná. Ve směru od serveru na klienta nese navíc `Y` informaci o tom, jestli je příjemce vlastníkem kamene, protože cizí kameny musejí být z plochy hráče u příjemce odstraněny.

5.4.4 Časování

Průběh hry je hlídán časovači, všechny hodnoty se nastavují v souboru `web.config` v sekci `<appSettings>`.

V objektu `qclient` je pomocí časovače `_timer` řízena perioda automatických aktualizací. Jeho hodnota se nastavení pomocí `t_aktualizace` a od tohoto nastavení se odvíjí timeout pro volání `PageMetods`, který se nastavuje na hodnotu o 1 s nižší.

Metoda hry `public DataProHrace Zmeny(...)` volaná při zpracování aktualizačního dotazu zase hlídá klientské aktualizace tak, že pokud čas od poslední aktualizace přesáhne dobu 3 aktualizačních period (3∗`t_aktualizace`), je pro instanci hráče na serveru nastaveno `Offline=true`. To způsobí, že se jeho jméno v informačním poli u všech hráčů obarví šedě.

Další časovač hlídá hráče, který je na tahu. Časovač je nulován akcí hráče spojené s kamenem (kliknutí, začátek tažení), stiskem některého tlačítka nebo kliknutím na upozorňující text, který se zobrazuje vždy po vypršení časovače. Výchozí nastavení časovače určuje hodnota `t_upozorneni` v souboru `web.config`.

Samostatné nastavení má také čas, po který jsou drženy v paměti serveru opuštěné hry. Vždy při zakládání nové hry, proběhne kontrola všech existujících her a pokud čas od poslední akce ve hře přesáhne nastavenou dobu `t_uklid` v souboru `web.config`, je hra odstraněna.

5.5 Mimořádné stavy

Bude nutné uvažovat různé mimořádné stavy, které mohou nastat a které bude následně nutné řešit.

5.5.1 Výpadek serverové aplikace

Při provozu webové aplikace může dojít z různých příčin jejímu selhání a následnému restartování. Díky zálohování stavu serveru lze očekávat celkem spolehlivé obnovení původního stavu všech her.

Pro ilustraci uveďme část výpisu ze souboru codeqwirkle.xml, ve kterém jsou na serveru zálohovány všechny herní instance.

Výpis 5.4: Ukázka části souboru s uložením stavu serveru

```
<?xml version="1.0" encoding="utf-8"?>
<QServer xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    Vytvoren="2012-02-29T23:34:21.08775+01:00">
  <g Gid="37c12bb6ceaa410f959984c0b1a54dab" Jmeno="sobota"
      Zalozena="2012-04-07T07:53:09.9945+02:00" Stav="Bezi" AktCislo="9"
      PosledniAktualizace="2012-04-07T08:43:50.572625+02:00">
    <p Pid="7c2ae91b1ac74c9e87f79e3336d5cdf7"
        Sid="1022857ea4fa43e6b3a9acc005a2b758" Jmeno="Dominika"
        Aktivita="false" BodyCelkem="2" BodyPosledni="2" HodnotaKamenu="0"
        AktCislo="9">
      <k c="48" a="8" x="0" y="-1" />
      <k c="2" a="8" x="4" y="-1" />
      <k c="4" a="8" x="5" y="-1" />
      <k c="100" a="8" x="2" y="-1" />
      <k c="103" a="9" x="1" y="-1" />
```

```
    <k c="26" a="9" x="3" y="-1" />
  </p>
  <p Pid="4a70d463ac1a4787bdd47c326db61123"
      Sid="f5b8d4f0e56a464aa05df0986bf19ed1" Jmeno="Veronika"
      Aktivita="true" BodyCelkem="0" BodyPosledni="0" HodnotaKamenu="2"
      AktCislo="9">
    <k c="79" a="6" x="5" y="-1" />
    <k c="94" a="6" x="4" y="-1" />
    <k c="39" a="6" x="3" y="-1" />
    <k c="50" a="6" x="2" y="-1" />
    <k c="1" a="6" x="1" y="-1" />
    <k c="106" a="6" x="0" y="-1" />
  </p>
  <k c="23" a="1" x="18" y="-1" />
  <k c="88" a="1" x="19" y="-1" />
  <k c="44" a="1" x="22" y="-1" />
  <k c="98" a="7" x="23" y="18" />
  <k c="72" a="8" x="25" y="17" />
```

5.5.2 Výpadek klienta

Výpadek klienta ve smyslu jeho „zamrznutí“ nebo třeba i nechtěného zavření uživatelem bude vyřešen pomocí dříve navrženého aktualizačního čísla. Nově otevřený prohlížeč (případně znovunačtení stránky) bude žádat o aktualizaci s hodnotou aktualizačního čísla 0 a dostane tedy kompletní stav celé hry.

Toto řešení však nabízí jistou možnost podvádění a to tak, že hráč na tahu ještě před odesláním změn stavu hry provede reload a zahraje znovu. Když se v této opakované hře budou lišit dobírané kameny, získá značnou výhodu v tom, že bude znát polohu původně vybraných kamenů ve společné zásobě. Z tohoto důvodu je kompletní reload sice povolen, ale pokud proběhne u hráče, který je na tahu, je současně penalizován předáním tahu na následujícího hráče.

V této souvislosti se nabízí ještě jeden způsob hry, kterému je nutné pro možnost získání neoprávněné výhody zabránit – spuštění na více prohlížečích současně. Bude aplikováno jednoduché řešení, kdy pro klienta bude vždy v rámci plné aktualizace (nové připojení do hry, po výpadku, po reloadu) vygenerován ověřovací identifikátor (SID). Klient bude tento identifikátor vkládat do svých požadavků směřujících na server (nikoliv však do URL). Server bude reagovat pouze na platnou kombinaci identifikátoru hráče v URL (PID) a ověřovacího identifikátoru (SID).

Tato míra zabezpečení je pro zamýšlený účel dostačující, vyšším stupněm by pak mohlo být nepřenášet identifikátor vůbec, ale např. jeho hashování s přispěním proměnné části dodávané serverem.

5.5.3 Přerušení komunikace

Iniciátorem komunikace je vždy klient, proto bude sledovat doručování odpovědí a vhodně reagovat na případný výpadek komunikace. V případě předávání změn ve hře směrem na server (po tahu hráče) bude při nedoručení odpovědi serveru nutné odesílání příslušných dat opakovat (klient nemůže rozhodnout jestli byla data skutečně doručena a výpadek se vyskytl např. pouze při odpovědi serveru). Naopak pokud zrovna nebyly předávány změny směrem na server, nebude nutné výpadek řešit, protože případná nedoručená data směrem ze serveru budou doručena v odpovědi na následující aktualizační dotaz, což je zajištěno použitím aktualizačního čísla.

5.5.4 Výjimky

Při běhu však mohou nastat i jiné mimořádné stavy jak v klientské, tak i v serverové části aplikace. Zdrojem těchto tzv. výjimek mohou být běhové chyby, ale tyto výjimky budou také záměrně generovány ve vlastní aplikaci na základě různých nepřípustných stavů (např. při pokusu o vstup do již zahájené hry).

Ošetření výjimek serveru

Výjimky vzniklé na serveru při aktualizačním volání jsou automaticky serializovány a doručovány na klienta v rámci odpovědi serveru.

Výpis 5.5: Přenos výjimky ze serveru

```
{"Message":"Neznámý hráč (chyba PID)","StackTrace":"   at
    Qwirkle.QServer.AktualizacniDotaz(DataOdKlienta odKlienta)\r\n   at
    Qwirkle.Web.Desk.Aktualizace(DataOdKlienta
    odKlienta)","ExceptionType":"Qwirkle.FatalniException"}
```

Klientská část aplikace má několik možností jak reagovat:

- **Opuštění hry**

  Toto nejradikálnější řešení musí být použito např. při chybném přístupovém identifikátoru hráče.

- **Úplná aktualizace hry**
  Některé výjimky mohou vyplynout z nesouladu v informacích uložených u klienta a na serveru, tyto stavy lze řešit pouze úplným reloadem – klient si obnoví celý obraz hry ze serveru.
- **Normální aktualizace hry**
  Ostatní výjimky vzniknou z chyb komunikace nebo z mimořádných stavů serveru a nelze je u klienta řešit, tj. budou běžným způsobem opakovány aktualizační dotazy.

Z uvedeného vyplývá, že bude výhodné na serveru definovat speciální třídy pro první dva druhy výjimek tak, aby mohly být u klienta odlišeny a zpracovány příslušným způsobem.

Jiným druhem budou výjimky vzniklé mimo aktualizační dotazy, nejčastěji při založení hry, založení hráče a při vstupu hráče do hry. Jejich textový popis bude u klienta zobrazen pomocí funkce `alert()`.

Výjimky u klienta

Jiným druhem budou výjimky pocházející přímo ze zpracování klientského skriptu. Důvody jejich vzniku mohou být různé, způsob řešení však může být pouze jeden – pokusit se zcela obnovit klientské prostředí. Bude proto provedeno znovunačtení celé stránky hry tak, aby byly obnoveny všechny objekty stránky včetně skriptů a následně bude provedena úplná aktualizace stavu hry.

6 TESTOVÁNÍ

Ladění a testování aplikace probíhalo v prostředí Visual Studia na vestavěném vývojem serveru. Důležitým nástrojem se ukázalo zejména krokování, které je možné provádět jak na klientské tak i na serverové straně. Hojně bylo využito také logování. Na straně serveru pomocí `Debug.WriteLine(...)` a na straně klienta pomocí `console.log(...)` (výpis je prováděn na konzolu dostupnou z menu Internet Exploreru přes nástroje pro vývojáře).

Při vývoji aplikace se potvrdilo ukládání stavu serveru do xml souboru jako velmi výhodné, protože tímto způsobem bylo možné obnovovat z připravených záloh různé herní situace, aniž by se k nim muselo dospět skutečnou hrou, nebo dočasnou úpravou aplikace. Díky existenci souboru je také možné monitorovat obraz hry v jejím průběhu nebo ho i snadno modifikovat.

Při ladění aplikace bylo vyzkoušeno velké množství herních fragmentů a celkem také asi 20 kompletních her. Přičemž byly také simulovány různé mimořádné stavy (přerušení spojení, zdržování reakce serveru pomocí `Thread.Sleep(...)` apod.) a schopnost zotavení se z nich.

Odladěná aplikace byla také pro závěrečné testování zkušebně nasazena na webhostingu asp2.cz pod operačním systémem Windows Server 2008.

7 ZÁVĚR

V práci byly představeny možnosti komunikace v prostředí .NET a také několik konkrétních technologií na jejichž základě byla nakonec vytvořena síťová aplikace pro hru Qwirkle. Vývoj aplikace byl v práci zdokumentován po jednotlivých etapách a výsledné zdrojové kódy okomentovány. Tím bylo naplněno zadání a cíl práce.

V průběhu prací na analýze byl návrh rozšířen o možnost běhu více paralelních her, což bylo nakonec úspěšně aplikováno.

Při testování hotové aplikace vyplynulo ještě několik další námětů pro možná budoucí rozšíření:

- **Ochrana před roboty**

  V případě veřejného nasazení by bylo vhodné zavést registraci a přihlašování uživatelů nebo alespoň ochránit vstup do hry před internetovými roboty.
- **Neveřejná hra**

  Takto založená hra umožní vstup pouze pozvaným hráčům, např. těm, kteří od zakládajícího hráče obdrží unikátní URL (návrh s tímto rozšířením počítá, existuje samostatný identifikátor hry a hráče).
- **Textová komunikace**

  Textová komunikace hráčů bývá někdy součástí síťových her. Ve hře Qwirkle není tato komunikace pro průběh vlastní hry podstatná, ale mohla by být aplikována.
- **Divák**

  Role divák by mohla být alternativou k aktivní účasti ve hře. Divák by byl hráčem, jehož klient provádí aktualizace pouze ve směru ze serveru na klienta a hry se tak přímo neúčastní.

LITERATURA

[1] Herman, I. *Komunikační technologie.* Skriptum VUT FEKT, Brno 2003.

[2] Troelsen, A. *C# a .NET 2.0 profesionálně.* Zoner press, 2006, 1200 s., ISBN 80-86815-42-0

[3] Microsoft *Microsoft Developer Network - Library.* [online]. Poslední aktualizace 2011 [cit. 1. 11. 2011]. Dostupné z: <msdn.microsoft.com/cs-cz/library/>.

[4] Běhálek, M. *Programovací jazyk C#.* [online]. Poslední aktualizace 2007 [cit. 30. 9. 2011]. Dostupné z: <www.cs.vsb.cz/behalek/vyuka/pcsharp/text/>.

[5] Osmani, A. *Essential JavaScript Design Patterns.* [online]. Poslední aktualizace 2012 [cit. 14.3.2012]. Dostupné z: <addy-osmani.com/resources/essentialjsdesignpatterns/book/#modulepatternjavascript>.

[6] Wikipedie *Spiel des Jahres.* [online]. Poslední aktualizace 31. 12. 2010 [cit. 1. 4. 2012]. Dostupné z: <cs.wikipedia.org/wiki/Spiel_des_Jahres>.

[7] (autor neuveden) *Qwirkle — Pravidla hry.* DESKO KOUTEK

[8] Pavel Kohoutek *Qwirkle Pravidla hry.* [online]. Poslední aktualizace 2011 [cit. 1. 4. 2012]. Dostupné z: <www.zatrolene-hry.cz/spolecenska-hra/qwirkle-2381/k-stazeni/pravidla-2682/>.

[9] Kanisová, H., Müller, M. *UML srozumitelně.* Computer press, 2004. ISBN: 80-251-0231-9

[10] TutsKing.com *High–Quality Wood Textures for Web Design.* [online]. Poslední aktualizace 2009 [cit. 14. 3. 2012]. Dostupné z: <www.tutsking.com/photoshop/amazing-wood-textures-for-web-design>.

[11] Eggers, J. *Random Acts of Coding.* [online]. Poslední aktualizace 14. 6. 2008 [cit. 27. 3. 2012]. Dostupné z: <randomactsofcoding.blogspot.com/2008/06/look-at-canceling-page-method-call.html>.

[12] Wikipedie *Fisher–Yates shuffle.* [online]. Poslední aktualizace 5. 3. 2012 [cit. 1.4.2012]. Dostupné z: <en.wikipedia.org/wiki/Fisher-Yates_shuffle>.

A PŘÍLOHA - DVD

Příloha obsahuje DVD se zdrojovými kódy aplikace.