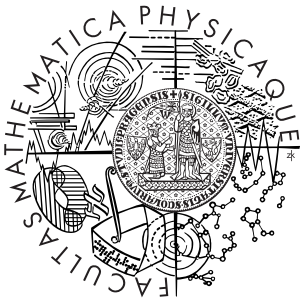
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Michal Čevora

KBang

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Bernard Lidický
Studijní program: Informatika, programování

2009

Rád bych poděkoval Mgr. Bernardu Lidickému za vedení ročníkového projektu a bakalářské práce a za poskytnutí mnoha cených rad. Dále bych chtěl poděkovat všem lidem, kteří se zúčastnili testování projektu.

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

V Praze dne 28. 5. 2009 Michal Čevora

Obsah

Název práce: KBang
Autor: Michal Čevora
Katedra (ústav): Katedra aplikované matematiky
Vedoucí bakalářské práce: Mgr. Bernard Lidický
E-mail vedoucího: bernard@kam.mff.cuni.cz

Abstrakt: Cílem této práce je převést karetní hru Bang! do počítačové hry hratelné přes počítačovou síť. Na začátku práce představuji základní koncepty hry Bang! a zabývám se řešením problémů spojených s implementací hry. V následujícím textu popisuji architekturu aplikace z hlediska serveru i klienta. Od obecné architektury se dostávám k popisu konkrétní implementace. Nejprve představím použitou knihovnu Qt, včetně důvodů pro její použití, poté popisuji implementaci všech částí projektu. V závěru práce se nachází uživatelská dokumentace.

Klíčová slova: Bang! online karetní hra

Title: KBang
Author: Michal Čevora
Department: Department of Applied Mathematics
Supervisor: Mgr. Bernard Lidický
Supervisor's e-mail address: bernard@matfyz.cz

Abstract: The aim of this work is to transform the card game called Bang! into a computer game that will be playable over computer network. In the beginning of the work I present the basic concepts of the game and I deal with solving problems related to implementation of the game. In the subsequent text I outline the architecture of the application with respect to server and client. From general architecture I proceed to the description of concrete implementation. First I acquaint the reader with the Qt, mentioning the reasons for its use, afterwards I describe the implementation of all parts of the project. In the conclusion of the work there is the user manual.

Keywords: Bang! online card game

Kapitola 1

Úvod

Bang! je karetní hra, která se především mezi hráči karetních a deskových her stala velmi populární. Vytvořil ji italský herní návrhář Emiliano Sciarra v roce 2002 a důkazem jejího úspěchu je nejen vydání celkem tří oficiálních rozšíření, ale i několik ocenění, které tato hra za dobu své existence posbírala.

Pozadí hry se odehrává v malém městečku na divokém západě, kde hráči v rolích šerifa, banditů, pomocníků šerifa a odpadlíka svedou bitvu o moc. Šerif a jeho pomocníci chtějí odstranit všechny bandity i jejich bývalého kolegu odpadlíka. Bandité chtějí pro změnu zabít šerifa, protože ví, že jakmile zemře šerif, nastane ve městě bezvládí. Nakonec odpadlík má nejtěžší úkol — musí zabít všechny bandity i pomocníky šerifa a samotného šerifa si musí podat jako posledního. Pouze tak se sám může stát novým šerifem.

Velmi důležitým prvkem ve hře je skutečnost, že kromě šerifa jsou všechny role tajné a tedy na začátku hry zná každý hráč pouze svou roli a ví, který hráč je šerif. Právě tento prvek vnáší do hry napínavou atmosféru a nejspíš stojí za vysvětlením, proč je tato hra tolik oblíbená.

Nápad o vytvoření sítové počítačové hry Bang! vznikl už dávno, ať už v hlavách technicky zdatných náruživých hráčů tak i u samotného Emiliana, který se několikrát nechal slyšet, že nakladatelství DaVinci pracuje na online počítačové verzi. Nakonec i na diskusním fóru serveru `www.bang.cz`[1] se nejednou objevilo oznámení různých fanoušků, že začínají pracovat na převedení této hry do počítačové podoby. Bohužel všechna tato diskusní vlákna po čase utichla a žádná funkční implementace Bangu se na fóru nikdy neobjevila.

[1] Ačkoli má tento server českou doménu, jedná se pravděpodobně o největší neoficiální server o Bangu! a na fórum často přispívá sám Emiliano.

Cílem bakalářské práce bylo vytvořit program, který by lidem umožnil hrát Bang! na počítači přes počítačovou síť.

Kompletní pravidla Bangu! najdete na přiloženém CD nebo je můžete získat na oficiálních stránkách hry[2].

Kapitola 2

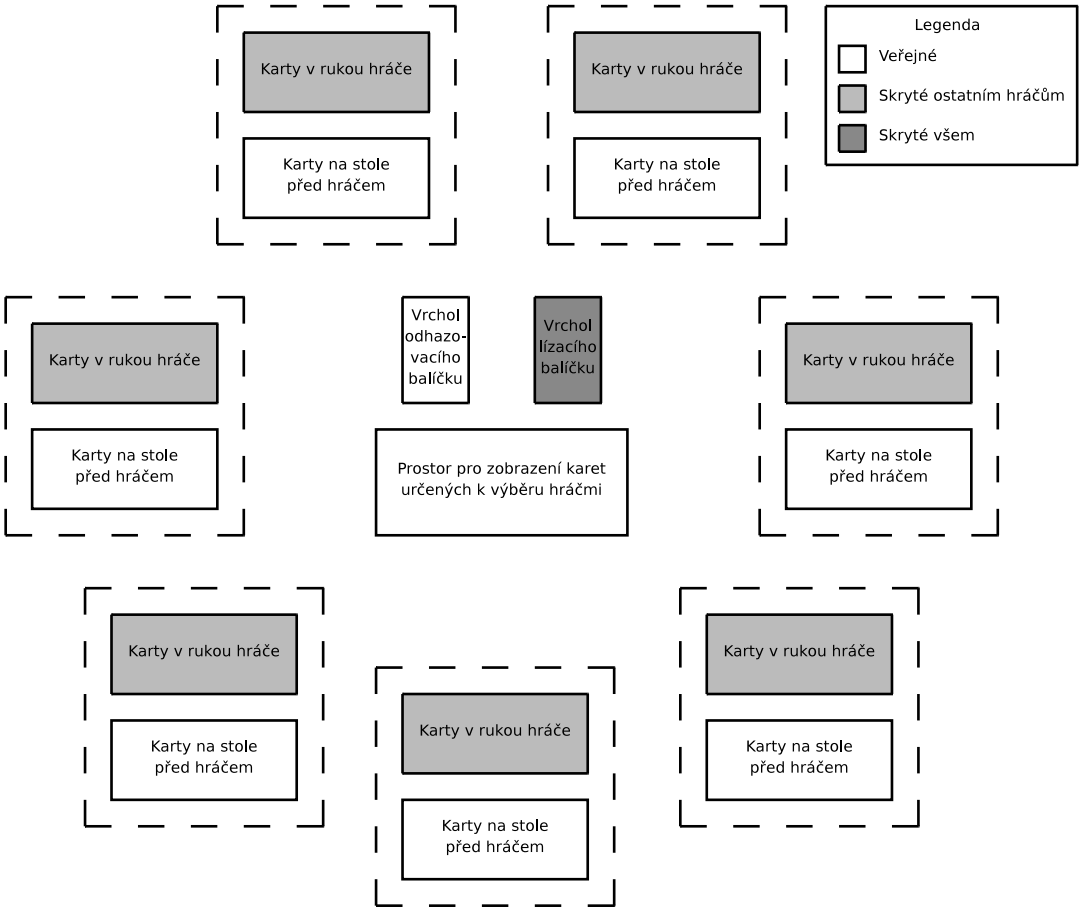
Základní koncepty a analýza problémů hry

Tato kapitola se věnuje problémům, které je nutné zohlednit při návrhu architektury projektu. K daným problémům jsou předložena a porovnávána různá řešení a v závěru každé sekce je popsáno použité řešení.

2.1 Herní plocha

Herní plocha v Bangu je tvořena oblastmi, ve kterých se shromažďují karty. Karty mohou být umístěny na hrací ploše buď lícem nahoru a nebo rubem nahoru. V případě, že je karta umístěna lícem nahoru (a za předpokladu, že na této kartě neleží jiná karta), je tato karta pro daného hráče *známá*. Opakem jsou *skryté* karty, ležící rubem nahoru. Podobně jako v jiných karetních hrách, také v Bangu drží jednotliví hráči některé své karty v ruce. Hráči by si měli dávat pozor, aby jim do karet nekoukal protivník. Z pohledu jednoho konkrétního hráče jsou proto všechny karty v ruce protivníků považovány za skryté. Stejně tak jsou i skryté všechny karty v lízacím balíčku a také v odhazovacím balíčku, kromě vrchní karty. Viditelnost karet na herní ploše shrnuje obrázek 2.1.

Hrací plocha je do jisté míry sdílená mezi hráči. Všichni hráči vždy vidí ty samé karty, jediný rozdíl je, že někteří vidí některé karty jako *známé*, kdežto jiní jako *skryté*.



Obrázek 2.1: Viditelnost karet na herní ploše

2.2 Tahy a střídání hráčů

Hráči se v Bangu střídají po tazích. Jakmile se hráč rozhodne ukončit svůj tah, pokračuje hráč po jeho levici.

Hráč má možnost ve svém tahu hrát karty, které vyžadují reakce ostatních hráčů. Například když hráč *A* zahraje kartu *Bang!* na hráče *B*, tento hráč se musí rozhodnout, zda zahraje kartu *Vedle* a nebo si ubere život. Existují i karty, které vyžadují reakci všech hráčů. Příkladem budiž karta *Indiáni*, po jejimž zahrání musí každý hráč zahrát kartu *Bang!* (zastřelit si svého indiána) a nebo si ubrat život. V tomto případě pravidla nespecifikují pořadí, kdy mají jednotliví hráči reagovat a při běžné hře „na živo“ se většinou reaguje v náhodném pořadí.

Abychom mohli popsat momentální stav hry, je důležité si pamatovat, který hráč je na tahu. V případě zahrání karet, které vyžadují reakci hráče si také musíme pamatovat, od kterého hráče v daný moment očekáváme reakci. V případě karet, které vyžadují reakci od více hráčů nastává problém.

Pokud bychom chtěli umožnit hráčům reagovat v libovolném pořadí, museli bychom si pamatovat množinu hráčů, od kterých očekáváme reakci. Postupně jak by hráči reagovali bychom tyto hráče z množiny vyškrtávali. Toto řešení se na první pohled zdá velmi dobré — nakonec by přece nejvíc odpovídalo běžným hrám „na živo“, nicméně podívejme se ještě na další řešení.

Jakmile bude zahrána karta, která vyžaduje reakci více hráčů, můžeme se těchto hráčů zeptat po jednom ve směru hry. Toto řešení má jednu výhodu, která na první pohled není tolik patrná. Pořadí reakcí hráčů totiž není nezanedbatelné (např. pokud si hráči lížou kartu na *Barel*) a pro některého hráče může být výhodnější zareagovat jako první, nebo naopak jako poslední. To by mohlo hráče svádět k závodění, což u této hry není příliš vhodné. Nakonec musíme brát v úvahu, že hráči mohou hrát s pomalým připojením a takto by mohli být znevýhodněni.

2.3 Problém zapomenutí vlastnosti

V běžných hrách (obzvláště mezi ne příliš zkušenými hráči) se často stává, že nějaký hráč zapomene na svou vlastnost a nevyužije ji. Například hráč hrající za *Willyho Kida*, ačkoli má možnost hrát neomezený počet *Bangů* za kolo, zapomene na svou vlastnost a odhodí dva *Bangy* do odhazovacího balíčku, ačkoli by je mohl použít k útoku na jistého nepřítele.

U některých vlastností však vzniká problém, který se pokusím ilustrovat na následujícím příkladu. Uvažme postavu *Vulture Sam*, jejíž vlastnost ji dovolí získat všechny karty mrtvých hráčů. Při normální hře musí hráč na tuto vlastnost pamatovat a při smrti nějakého hráče se připomenout a tohoto hráče požádat o karty. Pokud na to zapomene, umírající hráč odhodí všechny svoje karty do odhazovacího balíčku a hra pokračuje dál. Konkrétní situace by mohla vypadat následovně.

Hráč A zahraje *Bang!* na hráče B, který má poslední život. Hráč B se nemůže bránit, protože nemá kartu *Vedle* a proto umírá. Nyní má hráč C hrající jako *Vulture Sam* právo uplatnit svůj nárok, ale dřív, než to stihne udělat, hráč A pokračuje ve hře a nárok hráče C je nyní promlčený. Objevuje se zde stejný problém závodění hráčů jako v předchozí sekci.

Jedna z možností řešení tohoto problému je vložit do hry prodlevu, která dá hráči C například pět vteřin na využití své vlastnosti. Problémem je, že jakékoli pevně nastavené časové prodlevy mohou dělat v síťových hrách problémy. Klient může mít například pomalé internetové spojení nebo třeba pomalý počítač, na kterém se pomaleji animují pohyby karet, což by mu nakonec bránilo ve využití jeho vlastnosti.

Další možné řešení je počkat na reakci hráče C. Ten by se mohl v klidu rozhodnout, zda svou vlastnost použije či ne. Očividně toto není správné řešení, protože jakmile se hráče zeptáme, jestli chce použít svou vlastnost, musel by být hloupý, aby řekl ne.

Poslední možné řešení je používat vlastnosti trpící tímto problémem automaticky. K tomuto řešení jsem se nakonec přiklonil.

2.4 Problém identifikace karet

Důležitým problémem ve hře je vyřešit způsob identifikace karet. Server bude muset posílat klientu informace o pohybu karet a zároveň bude muset umožnit klientu některé tyto karty používat. Jedna z možností je posílat karty jako čtveřice (I, T, H, B), kde I je jednoznačný číselný identifikátor karty, T je typ karty, H je hodnost karty (2 – 10, J, Q, K, A) a B je barva karty (piky, kříže, káry, srdce).

Server musí také zajistit, že se žádný klient nedozví informace o kartách, u kterých vidí pouze rub (lízací balíček, karty v rukou oponentů). V případě takových karet určitě nesmíme posílat informace T, H a B. Otázkou je, zda můžeme posílat informaci I. Pokud by klient potřeboval manipulovat s touto kartou, jistě by se hodilo znát identifikátor karty. Například kdyby hráč zahrál kartu *Panika* za účelem vzít oponentovi kartu z ruky, klient by serveru poslal identifikátor karty *Panika* a identifikátor dané karty oponenta.

Jakmile však dovolíme klientovi znát identifikátory neznámých karet, otvíráme tím prostor k podvádění. Klient si tak může vytvořit mapování známých karet a jejich identifikátorů a toto mapování pak používat k odhalování oponentových karet. Hráč – programátor by si pak mohl upravit klienta tak, aby se mu skryté karty, jejichž identifikátory jsou již namapované, zobrazovaly odkrytě.

Jeden z možných způsobů je používat více identifikátorů a volit je tak, aby nedocházelo k výše uvedeným problémům. Například je možné zavést dva typy identifikátorů — *absolutní identifikátory*, které přiřazují každé kartě jedinečné číslo a *popisné identifikátory*, které popisují, kde se karta nachází.

Při odkazování na oponentovu kartu v ruce bychom pak mohli použít pouze popisný identifikátor, který by např. říkal „druhá karta v rukou hráče B“.

Pokud si promyslíme všechny případy, kdy je třeba odkazovat se na neznámou kartu, zjistíme, že jich není mnoho. V podstatě se jedná pouze o použití karet *Cat Balou* nebo *Panika* na neznámou kartu v rukou oponenta. Vezmeme-li v úvahu fakt, že karta z rukou oponenta se vybírá náhodně (oponent by si měl své karty zamíchat, než nechá hráče vybrat kartu), můžeme nechat skutečný výběr karty na serveru. Klient by pak pouze poslal, že cílem *Cat Balou* nebo *Paniky* je ruka hráče X. Touto metodou bychom mohli velmi elegatně vyřešit problém identifikace neznámých karet.

Kapitola 3

Architektura

Základním rozhodnutím implementace síťové karetní hry je rozdělení povinností mezi klienta a server.

Jedna z možností je soustředit veškerou herní logiku na server a klienta navrhnout co nejjednodušeji. V takovém návrhu bude server kromě základních úkolů spojených s obsluhou spojení a udržování stavu hry také zodpovědný za kontrolu vstupů od jednotlivých klientů. Klient naopak bude rozumět hře na velmi nízké úrovni. Když pak bude chtít hráč zahrát nějakou kartu, klient pošle daný požadavek na server a až ten se rozhodne, jestli tato akce neodporuje pravidlům.

Tento přístup má několik výhod, ale také i pár nevýhod. Především nám umožní implementovat pravidla pouze na jednom místě a nemusíme je tedy psát dvakrát. Dále díky toho, že klient nezná pravidla, je možné upravovat (opravovat) pravidla pouze na severu bez nutnosti měnit klienta. Tento poznatek je velmi důležitý v typickém síťovém prostředí, kdy se při objevení chyby v pravidlech pouze opraví a překompiluje server.

Hlavní nevýhodou je pak zvýšené množství přenášených dat a vyšší zátěž serveru. Jedná se však o malé navýšení, a proto je možné ho zanedbat.

3.1 Obecné koncepty

3.1.1 Komunikační protokol

Komunikační protokol jsem se rozhodl postavit na základě obecného jazyka XML. Narozdíl od binárních protokolů použití čistě textového protokolu netrpí problémy s nejednotnou reprezentací dat (např. problém různé endi-

anity) a také umožňuje pohodlné ladění. Použití XML má zase výhodu v automatické validaci přijímaných dat pomocí XML Sceme nebo Relax NG.

Bezespornou výhodou binárního protokolu je menší množství přenášených dat, než je tomu u textového protokolu. Nicméně tato výhoda je zanedbatelná, protože velikost informací, které je nutné posílat, je relativně malá.

Po navázání spojení pošle klient serveru hlavičku XML dokumentu a otvírací `stream` element. Server odpoví stejně. V tomto okamžiku má každá strana vytvořený jeden vstupní a jeden výstupní proud a komunikace probíhá do doby, než jedna strana pošle uzavírací `stream` element.

Komunikace probíhá pomocí posílání top level elementů, zvaných *stanza*. Protokol rozlišuje tři druhy stanzy.

První je stanza `query`, která slouží k dotazovaní se. Klient používá tuto stanzu například k zjištění informací o serveru. Nejprve pošle stanzu `query` s atributem `type="get"` a jedinečným identifikátorem `id="1"` a dovnitř zabalí konkrétní požadavek. Server tento požadavek zpracuje a vrátí výsledek zabalený opět do stanzy `query` s atributem `type="result"` a se stejným identifikátorem, jaký měl požadavek.

Další typ stanzy je `action`. Tuto stanzu používá klient k posílání všech typů akcí, ať už se jedná o založení nové hry, připojení se ke hře, nebo zahrání karty.

Poslední typ stanzy je `event`, který používá server k informování klienta o událostech které nastaly.

3.1.2 Identifikace hráčů

Pro identifikaci hráčů použijeme celočíselné identifikátory. Číslo 0 zůstane rezervované pro speciální případy (neplatný hráč, žádný hráč).

3.1.3 Identifikace karet

Jednotlivé karty budou identifikovány také pomocí celočíselných identifikátorů. Číslo 0 zůstane rezervované pro hráči neznámou kartu. Jak jsem již rozebral v sekci 2.4, neznámé karty nemusí mít identifikátory, protože je lze určit pomocí *kapsy*, ve které se nachází. Kapsy jsou popsány v sekci 3.1.4.

3.1.4 Kapsy

Pro reprezentaci umístění karet si pomůžeme zavedením *kapes (angl. pockets)*. Kapsou budeme rozumět jakoukoli oblast herní plochy, kde se můžou shromažďovat karty. Některé kapsy jsou společné a některé kapsy jsou vlastněny konkrétním hráčem. Mezi společné kapsy patří *lízací balíček*, *odhazovací balíček* a *výběr*. Kapsa obsahující karty v rukou hráče (dále jen *ruka*) a kapsa obsahující vyložené karty před hráčem (dále jen *stůl*) jsou kapsy vlastněné konkrétními hráči.

Libovolná kapsa lze jednoznačně určit dvojicí (T, H), kde T je *typ kapsy* a H je *identifikátor hráče*. Identifikátor hráče má význam pouze v případě kapes vlastněných hráčem (*ruka*, *stůl*).

Každá hrací karta ve hře náleží v jednom okamžiku právě do jedné kapsy a na pořadí karet v rámci jedné kapsy nezáleží. Výjimku tvoří kapsy *lízací balíček* a *odhazovací balíček*, kde jsou karty uspořádané. V případě těchto kapes se však vždy pracuje pouze s vrchní kartou a tedy lze z pohledu klienta tyto dvě kapsy chápat jako kapsy velikosti jedna.

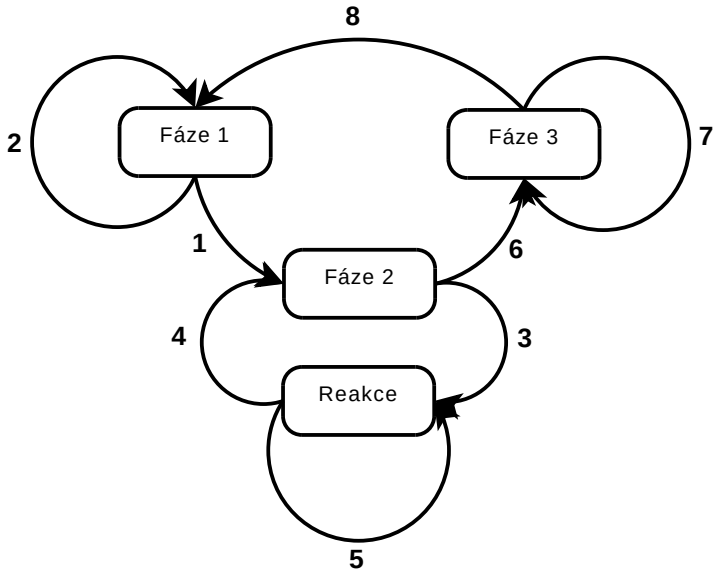
3.1.5 Herní stavy a kontext hry

Průběh hry je provázen přechodem mezi různými herními stavy (viz obrázek 3.1). Kontext hry je reprezentován trojicí (S, C, R), kde S je aktuální herní stav, C je identifikátor hráče na tahu a R je identifikátor dotazovaného hráče.

Po rozdání karet hra začíná ve stavu *fáze 1* a C i R je nastaveno na hráče, který zahajuje hru (šerif). Hráč si lízne dvě karty a přesune se po šipce *1.* do stavu *fáze 2*. V případě, že má hráč před sebou kartu *Dynamit* nebo *Vězení*, musí před líznutím karet otočit vrchní kartu balíčku, čímž ověří efekt dané karty (přesun po šipce *2.*).

Hráč ve *fázi 2* může zahrát některé karty z ruky. Pokud zahraje kartu, která vyžaduje reakci ostatních hráčů (*Bang!*, *Emporio*, aj.), přejde se po šipce *3.* do stavu *reakce* a R se nastaví na cílového hráče. V případě, že byla zahrána karta, vyžadující reakci pouze jednoho hráče, vrátíme se šipkou *4.* do stavu *fáze 2* a nastavíme $R = C$. V případě, že byla zahrána karta vyžadující reakci více hráčů, půjdeme po šipce *5.* a nastavíme R na dalšího reagujícího hráče. Po reakci posledního hráče se vrátíme šipkou *4.* do stavu *fáze 2* a nastavíme $R = C$.

Jakmile se hráč ve *fázi 2* rozhodne ukončit svůj tah, přesuneme se po šipce *6.* do *fáze 3*. Pokud má hráč více karet než je jeho počet životů, musí



Obrázek 3.1: Diagram herních stavů

odhodit přebytečné karty (šipka *7.*). V opačném případě se pokračuje po šipce *8.*, C a R se nastaví na následujícího hráče a začíná nový tah.

Je důležité uvědomit si, že hráč, který je aktuálně na tahu může ve svém tahu zemřít, např. pokud má smůlu při líznutí na *Dynamit* nebo pokud prohraje *Duel*. V těchto případech se hra z libovolného stavu přesune po šipce *8.* do *fáze 1* a pokračuje další hráč.

3.2 Server

Server je program, který se spravuje připojené *klienty* a probíhající *hry*.

3.2.1 Klient

Klient je z pohledu serveru entita, se kterou server komunikuje pomocí komunikačního protokolu. Server se stará o obsluhu některých požadavků klienta. Jedná se o požadavky o *vytvoření hry* a *připojení se ke hře*. Jakmile se klient připojí do hry jako hráč, bere na sebe roli *ovladače hráče*.

3.2.2 Vytváření her

Každý klient má možnost vytvořit novou hru. Při vytváření nové hry má pak možnost nastavit některé parametry hry, jako je název a popis hry, minimální a maximální počet hráčů, počet AI[1] a také se může rozhodnout, zda si přeje po spuštění hry zamíchat pozice hráčů.

Navíc lze každá hra zabezpečit heslem. Každý klient, který se chce připojit k dané hře, pak musí znát dané heslo.

Po vytvoření hry se čeká na připojení hráčů. Jakmile se ve hře nachází dostatek hráčů k spuštění hry (dáno parametrem „minimální počet hráčů“), tvůrce hry může tuto hru odstartovat, čímž se zahájí rozdávání karet.

3.2.3 Hráč

Jakmile se klient připojí do hry, vytvoří se nový hráč a jako jeho *ovladač* se nastaví onen klient. Hráči ovládáni umělou inteligencí se vytvoří podle nastavení parametru hry ihned po vytvoření hry. Pokud se klient ještě před spuštěním hry rozhodne opustit hru, jím ovládaný hráč se odstraní. Ovšem pokud by klient opustil hru po spuštění hry, pouze se odpojí ovladač hráče. K hráči se pak může připojit jiný klient (pokud zná heslo).

3.2.4 Vytváření hráčů

Hráč se vytvoří připojením klienta do hry. Pro zjednodušení se hráč vytvoří také při založení hry a tento zakládající hráč je v roli *tvůrce hry*. Při vytváření hráče je třeba zadat jméno hráče. Volitelně je možné zadat heslo hráče. Toto heslo slouží k opětovnému připojení se k hráči, například po výpadku spojení se serverem. Pouze klient, který zadá správné heslo, se může připojit k tomuto hráči.

3.2.5 Průběh hry

Před spuštěním hry se hráčům náhodně rozdělí role a každému hráči se přidělí jedna postava. Dále se připraví balíček karet, zamíchá se a začnou se rozdávat karty.

[1] Artifical Intelligence - hráči ovládání umělou inteligencí

3.3 Klient

Po připojení klienta k serveru může uživatel buď vytvořit novou hru a nebo se připojit do již existující hry. Pokud se rozhodne pro vytvoření nové hry, stává se automaticky *tvůrcem hry* a rozhoduje kdy bude hra spuštěna. Jakmile se připojí dostatek hráčů, *tvůrce hry* může odstartovat hru.

3.3.1 Pohyb karet

Veškerý pohyb karet ve hře je přímo ovládán serverem, který posílá klientům události o pohybu karet. Každá tato událost je tvořená trojicí *(karta, zdrojová kapsa, cílová kapsa)*. Karta je pak reprezentovaná čtveřicí (I, T, H, B) (v případě neznámé karty je karta *nulová* — všechny položky čtveřice jsou nulové) a kapsy jsou jednoznačně určeny *typem kapsy* a *identifikátorem vlastníka kapsy* (v případě sdílených kapes je to 0).

Takto definovaná událost je schopna reprezentovat téměř libovolný pohyb karty v Bangu. Existuje pouze jediný případ, kdy je třeba tuto strukturu rozšířit.

V případě že zdrojovou kapsou je odhazovací balíček, klient neví (nemusí si pamatovat) jaká karta leží pod vrchní kartou, která se bude hýbat. Klient by si tuto informaci mohl pamatovat, ale jelikož bychom mohli teoreticky odebírat karty z odhazovacího balíčku donekonečna (dokud tam nějaká karta bude), klient by si tak musel ukládat celý odhazovací balíček a také by musel správně interpretovat přesunutí karet z odhazovacího balíčku do lízacího balíčku v případě, že v lízacím balíčku dojdou karty. Tyto komplikace na straně klienta by mohly vést ke ztrátě synchronizace, kdy by si klient myslel, že je na vrcholu odhazovacího balíčku jiná karta, než ve skutečnosti je.

Danou strukturu doplníme o ještě jednu kartu, která se použije pouze v případě, kdy zdrojová kapsa bude *odhazovací balíček* a bude značit nový vrchol odhazovacího balíčku. V případě, že tato karta bude *nulová*, znamená to, že odhazovací balíček bude prázdný.

3.3.2 Změna počtu životů

V případě, že dojde k změně počtu životů hráče, klienti jsou o tom informování skrze událost o změně životů. Tato událost obsahuje *identifikátor klienta* a nový *počet životů*.

3.3.3 Změna kontextu hry

Jakmile dojde ke změně *kontextu hry*, server informuje všechny klienty zasláním nového kontextu. Klient na základě aktuálního kontextu vizuálně označuje hráče na tahu a dotazovaného hráče. Pokud je klient sám dotazovaným hráčem, umožní uživateli provádět *herní akce*.

3.3.4 Herní akce

Klientská aplikace poskytuje uživateli rozhraní pro vykonávání *herních akcí*. Klient by měl posílat serveru herní akce pouze, pokud je daný hráč specifikován jako *dotazovaný hráč* v *kontextu hry*. Pokud není odeslaná herní akce platná (např. pokud akce odporuje pravidlům nebo není daný hráč na tahu), server takovou akci ignoruje. Z tohoto důvodu klient nikdy sám po odeslání herní akce nemění stav hry, ale vždy čeká na danou událost od serveru.

Klient může odesílat tyto herní akce:

- *DrawCard* - hráč si lízne 2 karty
- *PlayCard* - hráč zahraje danou kartu
- *PlayCard* (s cílovým hráčem) - hráč zahraje danou kartu s cílovým hráčem
- *PlayCard* (s cílovou kartou) - hráč zahraje danou kartu s cílovou kartou
- *UseAbility* - hráč použije svou vlastnost
- *UseAbility* (s cílovým hráčem) - hráč použije svou vlastnost
- *UseAbility* (s cílovými kartami) - hráč použije svou vlastnost
- *Turn* - hráč ukončí svůj tah
- *Pass* - hráč se rozhodne nereagovat na efekt karty na něj zahrané
- *Discard* - hráč odhodí svou kartu do odkládacího balíčku

Kapitola 4

Implementace

KBang je implementován v objektově orientovaném jazyce C++ a používá převážně prostředky knihovny Qt4. Projekt je rozdělený do tří podprojektů:

- *common* — knihovna se společným kódem pro *server* i *client*
- *server* — serverová aplikace
- *client* — klientská aplikace

4.1 Knihovna Qt4

Knihovna Qt[1] je multiplatformní knihovna pro vytváření (nejen) aplikací s grafickým uživatelským rozhraním. Vyvíjí ji norská společnost *Qt Software* (dříve *Trolltech*), která je od léta 2008 vlastněna finskou firmou *Nokia*. Knihovna je vydávána v několika edicích pod různými licencemi. Pro vývoj svobodného softwaru/open source je k dispozici edice *Qt Open Source Edition* licencována pod licencemi, které vychází z GNU/GPL[2] a GNU/LGPL[3].

Knihovna Qt4 byla vydána v roce 2005 jako nástupce knihovny Qt3. Tato verze přinesla mnoho změn a stala se tak zpětně nekompatibilní s její starší verzí. Pro vývojáře znamená použití knihovny Qt4 značné zjednodušení vývoje software. Knihovna jde až tak daleko, že se snaží rozšířit v jistých ohledech omezenou funkcionalitu jazyka C++. Zavádí tak prostředky jako *signály a sloty*, *Qt RTTI*[4] nebo třeba *metaobjekty*. Tyto pomůcky jsou v

[1] Vyslovováno jako anglické slovo „cute“ - roztomilý
[2] GNU General Public Licence
[3] GNU Lesser General Public Licence
[4] Run-Time Type Identificaion

C++ realizovatelné pouze pomocí generování kódu. Knihovna *Qt4* přichází s tzv. *kompilátorem metaobjeků* (*Meta Object Compiler*), který se stará o převod těchto vyšších prostředků do standardního C++.

Dále poskytuje Qt knihovna nástroje, které konkurují standardní *STL* knihovně. Často se jedná o vylepšení příslušných protějšků z STL. Například třída `QString` oproti běžnému stringu ze standardní knihovny poskytuje implicitní sdílení (*copy-on-write*) a podporu *Unicode*. V neposlední řadě poskytuje Qt multiplatformní rozhraní pro použití vláken.

Důvodem k použití Qt4 v projektu byla především ona multiplatformnost této knihovny. Jakožto uživatel minoritního operačního systému nemůžu očekávat, že by mohlo dojít k většímu rozšíření hry bez možnosti hrát ji na majoritních operačních systémech a architekturách. Další bezespornou výhodou je skutečnost, že knihovna je nyní zaštiťována úspěšnou společností, což má vliv na budoucí vývoj této knihovny.

4.2 Common

Knihovna se společným kódem obsahuje především společný `Parser`, který implementuje *komunikační protokol*. Dále pak obsahuje třídu `Config`, která slouží k načítání a ukládání konfiguračních souborů.

4.2.1 Parser

Třída `Parser` poskytuje společný parser jak pro klientskou aplikaci, tak pro server. Každý pak používá pouze část jejího rozhraní. Po vytvoření socketu pak každý účastník připojí na socket parser. Pro odesílání zpráv se volají jednotlivé metody parseru a pro příjem zpráv je třeba připojit signály parseru na příslušné sloty. Parser pak dává vědet o příchozích zprávách pomocí signálů, jejichž zavolání pak zavolá připojené obslužné metody.

Jelikož mohou jednotlivé zprávy přicházet fragmentovaně, parser je zodpovědný za jejich spojování a příslušný signál se zavolá až tehdy, když je přijímaná zpráva kompletní.

V hlavičkovém souboru `parser/parserstructs.h` jsou navíc definovány pomocné struktury pro reprezentaci posílaných informací. Např. struktura `PublicPlayerData` slouží k reprezentaci veřejných informací o hráči. Pokud chce server například poslat klientu informaci, že se do hry připojil nový hráč, nejprve vytvoří novou instanci `PublicPlayerData` a nastaví ji podle informací o hráči. Nakonec zavolá na parseru metodu `eventPlayerJoinedGame`

`(const PublicPlayerData&)`, což zajistí odeslání daných informací klientovi.

Parser na straně klienta pak příjme zprávu, identifikuje ji a vytvoří podle ní nový objekt `PublicPlayerData`. Tento objekt je pak pomocí signálu `sigEventPlayerJoinedGame(const PublicPlayerData&)` poslán do příslušné obslužné metody klienta.

Třída `Parser` tvoří tedy *adaptér* mezi herními strukturami a herním protokolem. Pokud by v budoucnu došlo ke změně protokolu (např. na binární protokol), bylo by potřeba pouze změnit implementaci parseru — kód klienta i serveru by zůstal nezměněn.

4.2.2 Config

Třída `Config` se stará o ukládání různých nastavení do konfiguračního souboru. Tato třída je implementována jako *singleton* a její jediná instance je přístupná přes statickou metodu `Config::instance()`. Při její konstrukci (která je odložena až do momentu prvního použití třídy) se automaticky načte konfigurační soubor a v případě, že tento soubor neexistuje, nastaví se výchozí hodnoty.

Třída pak poskytuje metody na čtení a zápis jednotlivých položek. Položky v konfiguračním souboru jsou buď textové řetězce nebo seznamy textových řetězců. Všechny tyto položky jsou navíc seskupovány do skupin. Pro načtení položky je tedy potřeba zadat kromě jejího jména i jméno skupiny, ve které se nachází. Pro pohodlný přístup k číselným parametrům nabízí třída také metody pro zápis a čtení celočíselných položek, které jsou implementovány pomocí přetypování z/na textový řetězec.

Po zápisu nových položek do `Configu` je třeba uložit změny pomocí metody `store()`.

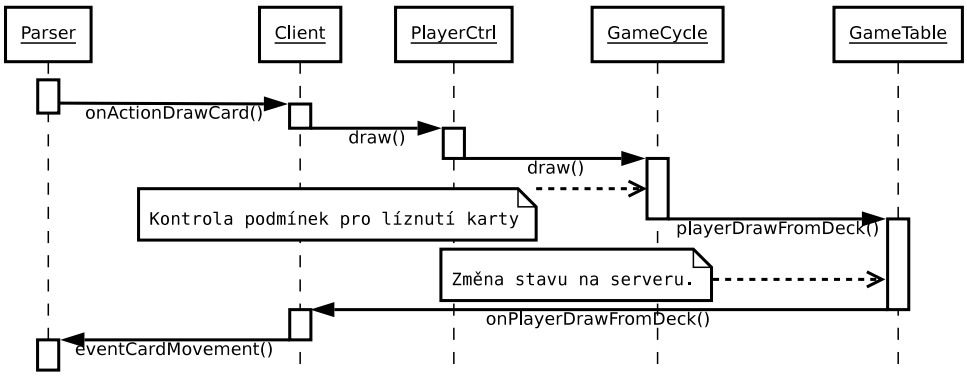
Umístění konfiguračního souboru je závislé na použité platformě. V případě Unix-like systémů je to `$HOME/.kbang/kbang.conf`, na systémech Windows je to typicky `C:\Documents and Settings\$USERNAME\kbang\kbang.conf`. Soubor je ukládán v textovém formátu, tudíž není problém editovat ho jednoduchým textovým editorem.

4.3 Server

Server, ačkoli může najednou obsluhovat více klientů a více her, je napsán jako jednovláknová aplikace. Využívá se při tom systému událostí poskyto-

vaných knihovnou Qt4. Jednotlivé akce na serveru probíhají vždy v reakci na příchozí zprávy z klientů. Server je tedy nutno implementovat tak, aby veškeré metody byly neblokující.

Jako příklad si uvedeme požadavek klienta o líznutí karet, který je znázorněn na obrázku 4.1. Jakmile přijde zpráva od klienta na server, probudí se Parser (je zavolán ze smyčky událostí řízené knihovnou Qt), rozpozná příchozí zprávu a vyšle signál `sigActionDrawCard()`. Na tento signál je připojený slot `onActionDrawCard()` ve třídě `Client`. Tato metoda se zavolá, zkontroluje, zda je klient připojený ke hře a pokud ano, zavolá metodu `draw()` na instanci třídy `PlayerCtrl`, která požadavek propaguje dál do vnitřních částí serveru. Nakonec v metodě `playerDrawFromDeck()` třídy `GameTable` dojde ke změně stavu (dvě karty z vrcholu lízacího balíčku se přesunou do rukou hráče) a k oznámení této události klientům.



Obrázek 4.1: Šíření požadavku o líznutí karet uvnitř serveru

Po provedení každé reakce na příchozí zprávu z klienta se kontext programu vždy vrátí do smyčky událostí a čeká na další zprávy.

4.3.1 Třída GameServer

Třída `GameServer` je *singleton*, který představuje celý server. Stará se o vytváření a správu klientů (třída `Client`) a vytváření a správu her (třída `Game`). Jakmile přijde na server příchozí TCP spojení, tato třída vytvoří novou instanci třídy `Client` a předá ji socket. Třída `Client` se pak už sama stará o spojení (připojí na něj parser ...).

4.3.2 Vnitřní implementace hry

Třída `Game` představuje jednu hru. Tato třída se stará o vytváření nových hráču a o udržování seznamu hráčů. Jednotliví hráči jsou reprezentováni třídou `Player`, která se stará o veškeré atributy hráčů, jako je například role a charaktér hráče, karty v rukou a vyložené karty na stole a počet životů hráče.

Třída `GameCycle` se stará o udržování *kontextu hry*. Tato třída kontroluje, zda příchozí *herní akce* neodporuje *kontextu hry*. V případě platných akcí je tato třída propaguje dále, buď voláním metod konkrétních zahraných karet a nebo přímo voláním metod třídy `GameTable`.

Třída `GameTable` udržuje všechny tři sdílené kapsy — *lízací balíček*, *odhazovací balíček* a *výběr*. Kromě toho je to také jediná třída, která má právo manipulovat s kartama. Při manipulaci s kartama tato třída automaticky posílá příslušné *herní události* (pomocí třídy `GameEventManager`).

Třída `GameEventManager` se stará o šíření *herních událostí*. Třídy, implementující rozhrání abstraktní třídy `GameEventListener` se mohou zaregistrovat k odběru *herních událostí* a tato třída jim pak tyto události posílá.

Vnitřní implementace hry je oddělena od okolního světa (*ovladačů hráčů*) pomocí fasády, která je tvořena třídami `PublicGameView`, `PublicPlayerView`, `PrivatePlayerView` a `PlayerCtrl`. Více o této fasádě najdete v sekci 4.3.4.

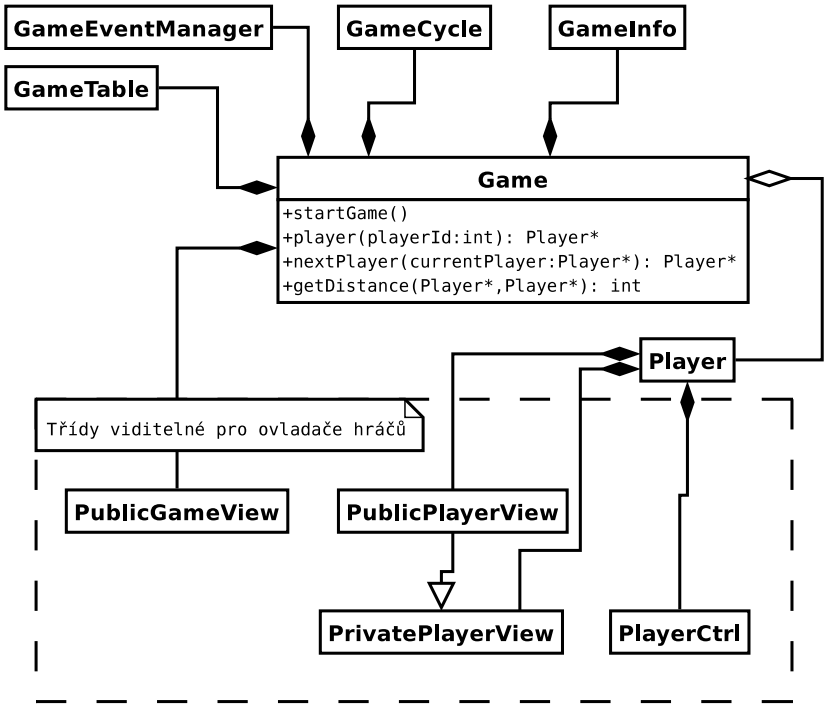
4.3.3 Implementace karet

Funkce jednotlivých typů karet jsou implementovány přímo v potomcích třídy `PlayingCard`. Tato třída se stará o udržení společných rysů všech karet (identifikátor, znak, hodnost, ...) a zároveň poskytuje základní rozhrání pro konkrétní karty.

```
virtual void play();
virtual void play(Player* targetPlayer);
virtual void play(PlayingCard* targetCard);
```

Výchozí implementace těchto metod vyvolá vyjímku `BadUsageException`. Třídy konkrétních karet pak některé z těchto metod překryjí svými implementacemi.

Třída `TableCard` rozšiřuje třídu `PlayingCard` a slouží jako nadtřída pro karty, které se dají zahrát na stůl (karty s modrým okrajem). Poskytuje metody `registerPlayer()` a `unregisterPlayer()`, které jsou zavolány v



Obrázek 4.2: Třída Game a její okolí

momentě, kdy se karta objeví vyložená před hráčem (resp. kdy karta opustí prostor před hráčem).

Třída `ReactionCard` rozšiřuje třídu `PlayingCard` a slouží jako nadtřída pro karty, které po zahrání očekávají reakci ostatních hráčů. Třída konkrétní karty použije metodu `setResponseMode()` třídy `GameCycle`, které kromě ukazatele na sama sebe pošle i ukazatel na hráče, po němž žádá reakci. Jakmile daný hráč zareaguje, třída `GameCycle` zavolá na kartě jednu z následujících metod, podle dané reakce.

```
virtual void respondPass();
virtual void respondCard(PlayingCard* targetCard);
```

Funkce některých karet jsou velmi podobné, a proto je občas několik typů karet sdruženo do jedné třídy. Například všechny karty zbraní jsou implementovány ve třídě `WeaponCard`. Karty *Mustang* a *Appaloosa* jsou zase implementovány ve třídě `CardHorse`.

4.3.4 Ovladače hráče

Ovladače hráče jsou třídy, které ovládají připojeného hráče. Tohoto hráče mohou ovládat samy (umělá inteligence) a nebo mohou delegovat tuto činnost na jiné třídy. V projektu jsou zatím dva ovladače hráče - třída `VoidAI` a třída `Client`.

Každý *ovladač hráče* musí být schopen odesílat *herní akce* a přijímat *herní události*. K odesílání *herních akcí* slouží instance třídy `PlayerCtrl`, kterou ovladač obdrží jakmile se připojí do hry. K příjmu *herních událostí* musí ovladač implementovat rozhrání třídy `GameEventListener`. Při připojování se do hry odešle ukazatel na sama sebe a *vnitřní implementace hry* se postará o přihlášení k odběru *herních událostí*.

Navíc mají ovladače možnost samy si zjišťovat informace o hře. K tomu slouží třída `PublicGameView`, která poskytuje veřejné informace o hře. Tato třída také umožňuje získat pro každého hráče ve hře příslušnou instanci třídy `PublicPlayerView`. Skrze metody této třídy pak může ovladač zjišťovat veřejné informace jednotlivých hráčů.

V serveru ještě existuje třída `GameLogger`, která také implementuje rozhrání `GameEventListener`, ale nejedná se o ovladač. Tato třída pouze přijímá *herní události* a zapisuje je do log souboru. Aby log soubory mohly obsahovat také neveřejné informace, tato třída je v `GameEventManageru` zaregistrovaná jako *supervizor*, což jí umožní získávat `herní událostí` se soukromými údaji (např. informace o líznutých kartách).

4.4 Klient

Hlavní okno klientské aplikace obsahuje kromě hlavní nabídky v horní části pouze hrací plochu Bangu!. V levé spodní části je oblast pro psaní zpráv a v pravé spodní části je prostor, kde se zobrazují herní události (*log*) a nebo ladící informace (*XML*). Dále jsou pak v okně rozmístěny oblasti pro zobrazování jednotlivých hráčů. Celé toto okno je vizuálně vytvořeno pomocí aplikace Qt Designer.

4.4.1 Třída MainWindow

Třída `MainWindow` představuje hlavní okno aplikace. Hlavní okno má v horní části hlavní nabídku, skrze kterou se uživatel připojuje k serveru a vytváří nebo připojuje se ke hrám. Pro připojení se k serveru, vytvoření hry a

připojení se ke hře jsou vytvořena dialogová okna (třídy `ConnectToServerDialog`, `CreateGameDialog` a `JoinGameDialog`).

Třída `MainWindow` používá instanci třídy `ServerConnection` ke komunikaci se serverem. Tato třída na požádání vytvoří spojení se serverem a v případě úspěchu nasadí na nově vzniknutý *socket* parser ze společné knihovny (třída `Parser`).

Jakmile přijde událost ze serveru o vstupu do *game módu* (v reakci na požadavek klienta o připojení se do hry), vytvoří se nová instance třídy `Game`, která existuje do doby, než přijde požadavek na opuštění *game módu*. Instance třídy `Game` dostane při inicializaci referenci na *herní widgety*, které po dobu svého života spravuje. Dostane také referenci na `ServerConnection`, u které si zaregistruje příjem *herních událostí* a dále ji používá k odesílání *herních akcí*.

4.4.2 Třída Game

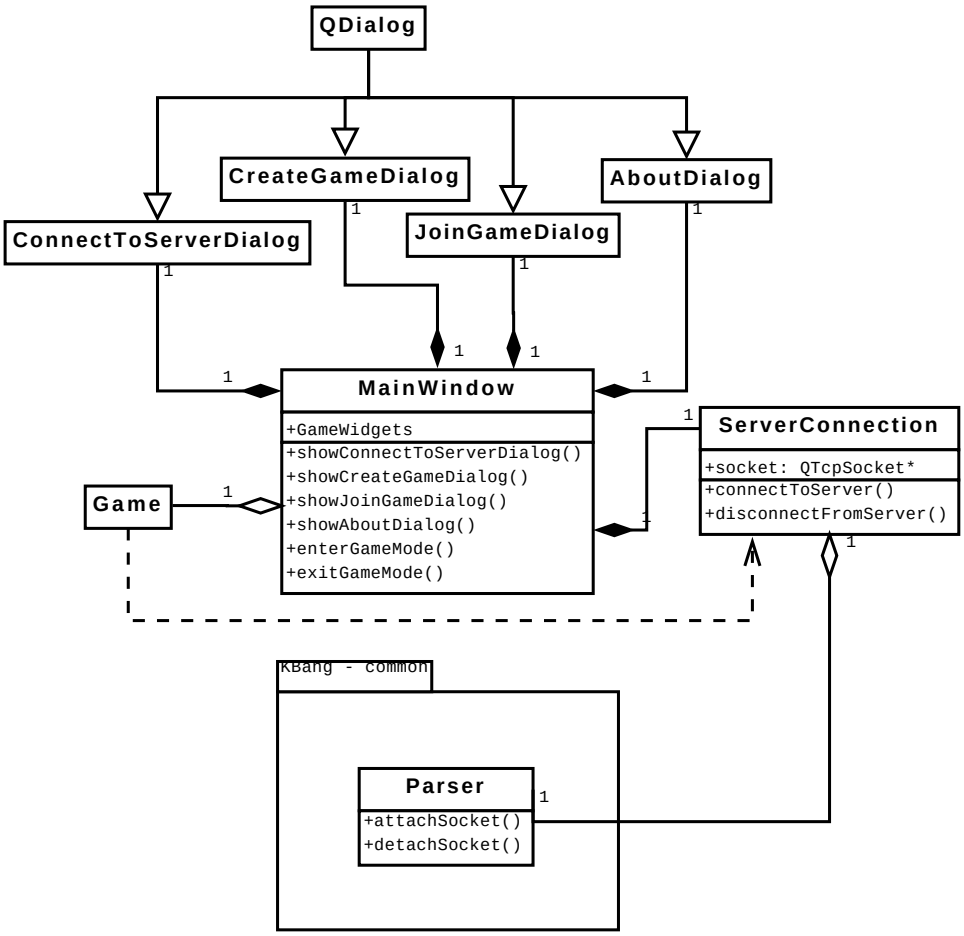
Třída `Game` se stará o zobrazování herní plochy a stavu hry (skrze widgety, které ji jsou delegovány) a také o odesílání herních akcí. Během inicializace třídy se vytvoří `GameEventHandler`, který si zaregistruje přes referenci na `ServerConnection` odběr *herních událostí* a stará se pak o řazení příchozích *herních událostí* do fronty a jejich postupné spouštění.

Dále se při inicializaci vytvoří `GameActionManager`, který se stará o obsluhu událostí myši nad *herními widgety* a o odesílání *herních akcí* (skrze `ServerConnection`) na server. Nakonec se provede inicializace všech herních widgetů.

Objekt třídy `Game` může nabývat tří stavů. Stav `NoInterface` je výchozí stav po připojení do nerozehrané hry. V tomto stavu je střední část okna prázdná. Stav `CreatorInterface` je výchozí stav po vytvoření hry. Je-li `Game` v tomto stavu, uživatel vidí uprostřed okna tlačítko sloužící k spuštění hry. Nakonec stav `GameInterface` slouží pro samotnou hru, kdy je na středu *lízací a odhazovací balíček* a prostor pro *kapsu výběr*. Hra mězi těmito stavy přechází v závislosti na přijímaných *herních událostech* ze serveru.

4.4.3 Herní widgety

Třída `Game` spravuje po dobu svého života *herní widgety*, delegované třídou `MainWindow` a zároveň (ve stavu `GameInterface`) vytváří tři widgety nové — *lízací balíček* (třída `DeckWidget`), *odhazovací balíček* (třída `GraveyardWidget`)

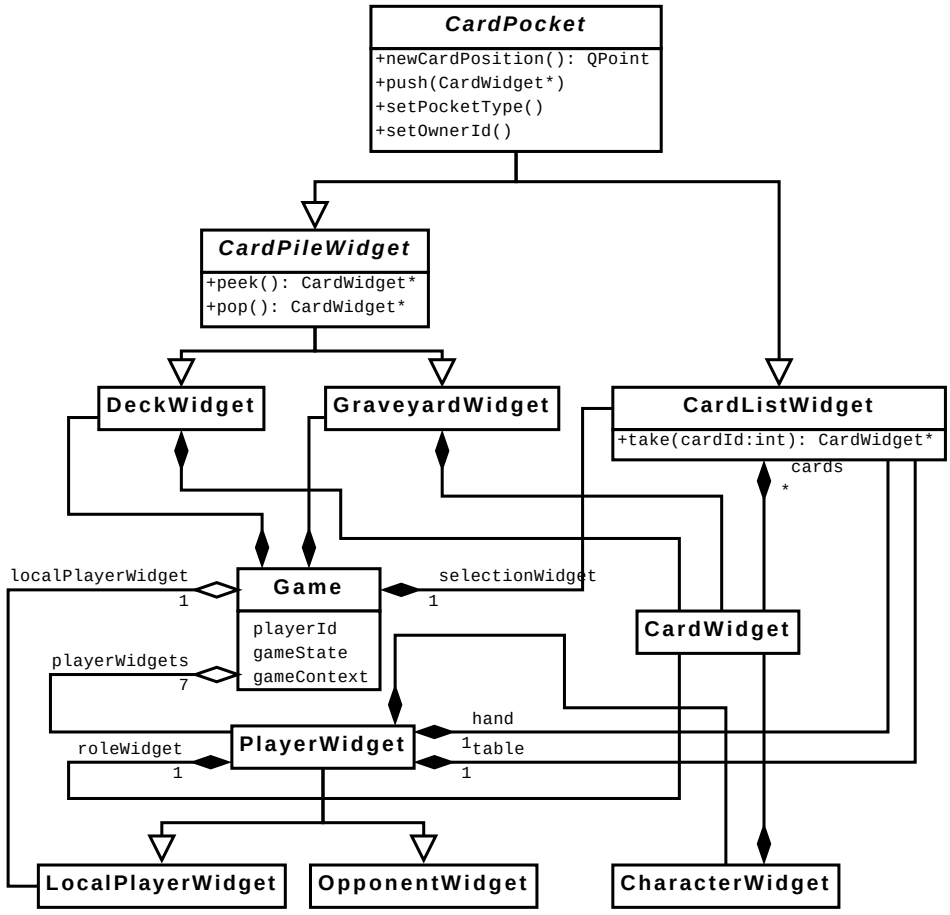


Obrázek 4.3: Třída MainWindow

a *kapsa výběr* (třída *CardListWidget*). Mezi delegovanými widgety jsou widgety všech hráčů (třída `LocalPlayerWidget` pro lokálního hráče a třída `OpponentWidget` pro oponenty). Každý z těchto widgetů obsahuje kapsu *ruka*, kapsu *stůl* a widget `CharacterWidget`, který slouží k zobrazení postavy hráče a k počítání životů. Propojení herních widgetů je zobrazeno na obrázku 4.4.

4.4.4 Herní události

O obsluhu herních událostí se stará třída `GameEventHandler` a `GameEventQueue`. `GameEventHandler` nejprve při inicializaci zaregistruje všechny konkrétní

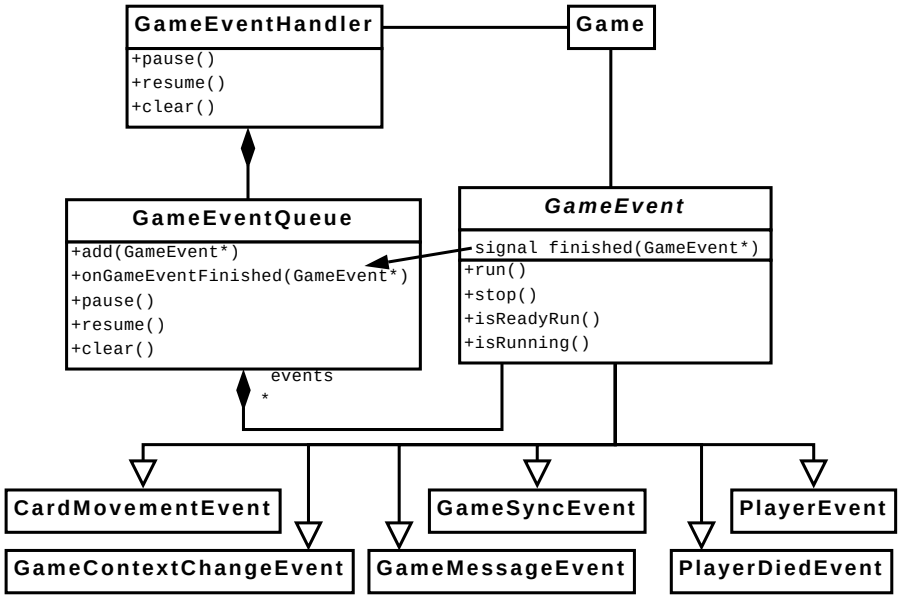


Obrázek 4.4: Třída Game a herní widgety

typy *herních událostí* a inicializuje *frontu událostí* - `GameEventQueue`. Jakmile přijde ze serveru nová *herní událost*, `GameEventHandler` vytvoří příslušného potomka `GameEvent` a vloží ho do *fronty událostí*.

Fronta událostí postupně odebírá události a spouští je zavoláním jejich metody `run()`. Zároveň si připojí jejich signál `SIGNAL(finished(GameEvent*))` na svůj slot `SLOT(onGameEventFinished(GameEvent*))`. Tímto způsobem se `GameEventQueue` dozví, že právě běžící *herní událost* skončila a může spustit další událost ve frontě. Je možné také dočasně pozastavit spouštění dosud čekajících událostí pomocí metod `pause()` a `resume()`.

Zavedení fronty herních událostí do programu má svůj význam. Některé události, jako třeba událost pohybu karty, jsou v klientu zobrazeny ani-



Obrázek 4.5: Herní události v klientu

movaně a tedy nějaký čas trvá, než se tyto události projeví. Pokud bychom události neřadili do fronty, mohly by události od serveru přicházet rychleji, než bychom je stačili animovat a v lepším případě by uživatel viděl změť karet lítajících do všech směrů. Jakmile by přišla událost, která by očekávala, že již byla provedena předešlá událost, ale ona předešlá událost by se stále ještě animovala, pravděpodobně by došlo k pádu aplikace.

4.4.5 Animace pohybu karet

Animace pohybu karet je zajišťována třídu `CardMovementEvent`. Po spuštění události se podle informací obdržených od serveru vyhledá karta, která se bude animovat. Součástí informace od serveru je i zdrojová *kapsa* a tedy není problém kartu jednoznačně určit i v případě, že se jedná o neznámou kartu. Třída tuto kartu odstraní z dané kapsy a určí novou, cílovou kapsu. Na této kapse zavolá virtuální metodu `newCardPosition()`, která vrací pozici na kterou je třeba kartu umístit.

Třída nyní zná zdrojovou (aktuální) i cílovou pozici pro animaci pohybu karty a pomocí klasického lineárního vzorce pro pohyb konstantní rychlostí provede animaci pohybu. Animace je implementována pomocí třídy `QTimer`. Abychom zajistili co nejplynulejší pohyb, v každém zavolání timeru

si zjistíme čas uplynulý od startu animace a ten pak použijeme k výpočtu aktuální polohy karty. Pokud se některé z volání timeru lehce opozdí, karta se jednoduše posune o větší kus a uživatel nezaznamená trhaný pohyb.

Jakmile karta doputuje na svou cílovou pozici, je vložena do cílové kapsy, která za ní dále přebírá zodpovědnost.

4.4.6 Správce herních akcí

O zachytávání akcí uživatele a generování příslušných akcí pro server se stará třída `GameActionManager`. Každá zobrazená karta (třída `CardWidget`) a také každý `PlayerWidget` reagují na události myši voláním příslušných metod `GameActionManageru`. Při hraní některých karet je potřeba zadat doplňující informaci, jako třeba *cílový hráč* nebo *cílová karta*. Uvnitř třídy `GameActionManager` je jednoduchý stavový automat, který objekt přepíná do stavů, kdy se očekává doplnění cílového hráče nebo karty. Jakmile je zadána kompletní *herní akce*, je odeslána na server.

`GameActionManager` také reaguje na stisk pravého tlačítka nad kartou. Tato akce negeneruje žádnou *herní událost*, ale provede zvětšení karty (pomocí třídy `CardZoomWidget`).

Kapitola 5

Uživatelská dokumentace

5.1 Instalace

5.1.1 Windows XP/Vista 32 bit

KBang je pro *Windows* distribuován ve formě *zip* archívu. Tento archív obsahuje kromě vlastních spustitelných souborů pro *klienta* a *server* také potřebnou grafiku karet a potřebné DLL soubory (včetně runtime knihovny Qt4). Archív stačí rozbalit do libovolné složky a spustit jeden z *exe* souborů.

5.1.2 Linux

Vzhledem k různorodosti linuxových systémů neexistuje jedno standardní řešení distribuce software. Z tohoto důvodu je zatím potřeba na linuxu celý projekt zkompilovat.

Před samotnou kompilací se ujistěte, že máte vhodný kompilátor (doporučuji aspoň *gcc 4.1*) a *dev* verzi knihovny *Qt4* (minimálně ve verzi *4.4*[1], ale doporučuji verzi *4.5* a vyšší). Pro kompilaci programů v linuxu je také doporučována alespoň základní úroveň práce s příkazovou řádkou.

Pokud nemáte k dispozici zdrojové kódy aplikace, můžete je získat na stránce projektu [4].

Samotnou kompilaci provedete vložením následujících příkazů do terminálu:

```
$ cd kbang
```

[1] V této verzi se na některých systémech objevují problémy se sekanou animací pohybu karet

```
$ qmake kbang.pro
$ make
```

Nyní by měly být vytvořeny binárky `kbang-client` a `kbang-server`, které můžete spustit.

5.2 Server

Po spuštění serveru je vše připraveno pro připojení klientů. Se serverem však lze dále pracovat pomocí konzole. Do konzole můžete zadávat příkazy — některé z nich vyžadují zadání parametrů. Parametry příkazu se píší za příkaz a oddělují se mezerou. Následuje přehled příkazů z krátkým popisem.

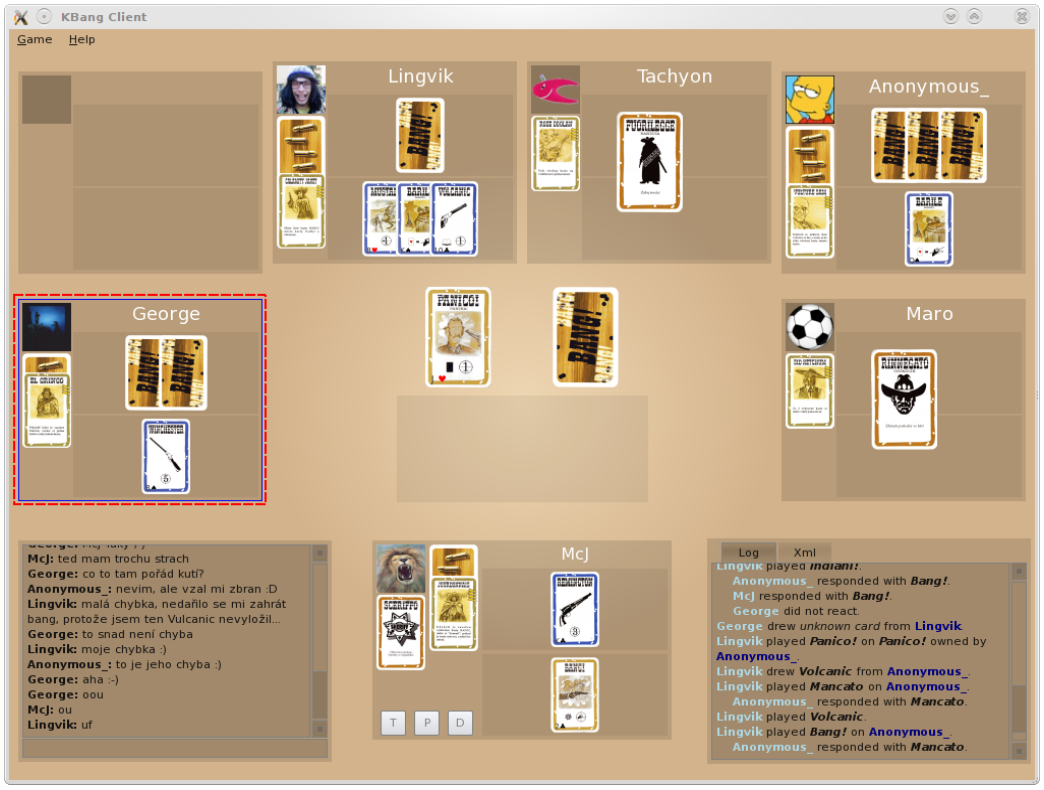
Příkaz	Popis
`help`	vypíše seznam všech příkazů včetně krátkého popisu jejich funkce
`quit`	ukončí server
`list-games`	vypíše seznam her na serveru
`list-clients`	vypíše seznam klientů připojených k serveru
`set-player-password`	umožní nastavit heslo hráče; tento příkaz vyžaduje tři parametry: *id hry*, *id hráče* a *nové heslo*

5.3 Klient

Ovládání KBangu je vesměs intuitivní a uživatel, který dobře zná Bang! by neměl mít problémy. Snad jediné části ovládání hry, se kterými mají začínající hráči KBangu problém, jsou *používání karet vyžadující „líznutí“* (5.3.10) a *používání vlastností* (5.3.11).

5.3.1 Připojení se k serveru

Po spuštění klientské aplikace je třeba přihlásit se k serveru. Pomocí volby *Connect to Server* v nabídce *Game* otevřete dialog pro výběr serveru. Pokud jste KBang spustili poprvé, bude nabídka serverů obsahovat pouze jediný server — `alderan.cz`. Tento server je určený pro veřejnost a doporučuji vám hrát právě na něm. Pomocí tlačítek *Add Server*, *Edit Server* a *Delete Server* můžete upravovat seznam serverů. Tento seznam se po uzavření dialogu



Obrázek 5.1: KBang Klient

automaticky uloží do konfiguračního souboru. Pro připojení vyberte server ze seznamu a klikněte na tlačítko *Connect*.

5.3.2 Založení hry

Po připojení ke hře máte buď možnost založit novou hru a nebo se připojit k již založené hře. Pokud se rozhodnete pro založení hry, vyberte volbu *Create Game* z nabídky *Game*. Otevře se nové dialogové okno, ve kterém můžete zadat parametry nové hry. Můžete zde nastavit jméno hry, omezit minimální a maximální počet hráčů nebo nastavit počet počítačových hráčů. Máte zde také možnost nastavit heslo pro připojující se hráče. Poslední volba umožňuje rozhodnout, zda budu hráči rozmístění okolo stolu podle pořadí, v jakém se připojovali k serveru nebo zda budou hráči při spuštění hry zamícháni.

Nakonec je zde ještě třeba vyplnit údaje o hráči. Můžete zde nastavit

jméno a heslo hráče a taky avatár (malý obrázek hráče). Heslo slouží k opětovnému připojení se k hráči, například v důsledku pádu připojení během hry. Avatár nastavíte kliknutím na rámeček s textem *Avatar*. Objeví se dialog ve kterém vyberete soubor s obrázkem. Obrázek by měl mít čtvercový formát, jinak dojde k deformaci obrázku. Nezáleží na velikosti, obrázek se automaticky přeškáluje do správné velikosti.

Po nastavení všech parametrů vytvoříte hru kliknutím na tlačítko *Create*.

5.3.3 Připojení se do hry

Pokud se chcete připojit k již existující hře, vyberte volbu *Join Game* z nabídky *Game*. V levé části nově otevřeného dialogu uvidíte seznam všech her, které právě probíhají na serveru. Po kliknutí na jednu z her se objeví v pravé části dialogu informace o dané hře, včetně seznamu hrajících hráčů. V pravé spodní části se také vyskytuje oblast pro zadání informací o hráči.

Nejčastěji se nejspíš budete připojovat do ještě nespuštěných her. Takové hry mají ve sloupcí *State* napsáno *Waiting*. Chcete-li se připojit do takové hry, klikněte na danou hru a v seznamu hráčů se ujistěte, že je vybraná poslední možnost *Create new player*. Tlačítkem *Play* se připojíte do hry.

Druhá možnost je připojit se do již rozehrané hry místo hráče, který se odpojil. V tomto případě klikněte na danou hru a v seznamu hráčů vyberte hráče, místo kterého se chcete připojit. Hráči, na které se nelze připojit jsou zobrazení šedou barvou a nelze je vybrat. Abyste mohli nahradit daného hráče, je třeba vyplnit do pole *Player Password* stejné heslo, jaké zadal původní uživatel při připojování se do hry. Poté klikněte na tlačítko *Play*.

5.3.4 Zahájení hry

Po založení hry je třeba počkat na ostatní hráče. Jakmile se připojí dostatek hráčů, zakladatel hry může kliknout na tlačítko *Start Game* uprostřed herní plochy, čímž zahájí hru.

5.3.5 Kontrola karty před líznutím

Pokud má váš hráč před sebou vyloženou kartu *Vězení* nebo *Dynamit*, je třeba před líznutím si karet otočit vrchní kartu z lízacího balíčku, která rozhodne, zda se dostanete z vězení (popř. zda dynamit vybouchne či ne).

Pro tento účel je třeba kliknout na onu vyloženou kartu. V případě, že máte před sebou obě karty, nejprve se kliká na *Dynamit* a poté na *Vězení*.

5.3.6 Líznutí karty

Kliknutím na balíček karet si líznete dvě karty. Některé postavy mohou použít alternativní způsob líznutí, určený jejich vlastností. Více o používání vlastnosti najdete v sekci 5.3.11.

5.3.7 Hraní karet

Kartu z ruky zahrajete jednoduše kliknutím na ni. V případě, že se jedná o kartu *Bang!*, *Duel* nebo *Vězení*, je třeba po kliknutí na kartu vybrat cílového hráče. V případě karet *Cat Balou* a *Panika* musíte vybrat cílovou kartu.

5.3.8 Reakce

V případě, že na vás byl zahrán *Bang!* (nebo jiná útočná karta) můžete se bránit zahráním karty *Vedle* (nebo jinou obranou kartou, dle pravidel). V případě, že nemáte, nebo nechcete zahrát *Vedle*, použijte tlačítko `P`, čímž si uberete život.

5.3.9 Výběr karet

V některých případech hry (např. po zahrání karty *Hokynářství*) dojde k vyložení několika karet ve střední části herní plochy. Jakmile na vás přijde řada, vyberte si kartu kliknutím na ni.

5.3.10 Používání karet vyžadující „líznutí“

Pokud si potřebujete *líznout* na efekt nějaké karty, klikněte na tuto kartu. Tímto způsobem se například používá karta *Barel*.

5.3.11 Používání vlastností

Vlastnosti některých postav vyžadují aktivaci hráčem. U těchto postav aktivujete vlastnost kliknutím na kartu postavy. V případě postavy *Jesse Jones*, která si může první kartu líznout z ruky libovolného hráče je po

kliknutí na postavu třeba vybrat cílového hráče. Podobně při použití vlastnosti postavy *Sid Ketchum* (odhoď dvě karty z ruky a přidej si jeden život) musíte po kliknutí na kartu postavy vybrat dvě karty k odhození.

5.3.12 Odhození přebývajících karet

Na konci svého tahu nesmí mít žádný hráč více karet než je jeho počet životů. Pokud tato podmínka není splněna, musí hráč odhodit přebývající karty. V tomto případě klikněte na tlačítko `D` a vyberte karty k odhození. Jakmile se naplní uvedená podmínka, váš tah automaticky skončí a pokračuje další hráč.

5.3.13 Ukončení tahu

V případě, že chcete ukončit svůj tah (a nemáte žádné přebývající karty), klikněte na tlačítko `T`.

5.3.14 Zvětšení karty

Aby se na obrazovku vešly všechny důležité informace, karty jsou v klientu docela malé. Stisknutím pravého tlačítka myši nad kartou si můžete danou kartu zvětšit.

5.4 Konfigurační soubor

Veškerá nastavení jak klienta tak serveru se ukládají do konfiguračního souboru. Tento konfigurační soubor se (pokud neexistuje) sám vytvoří po prvním spuštění klienta nebo serveru. Umístění souboru závisí na operačním systému:

Windows: `C:\Documents and Settings\$USERNAME\kbang\kbang.conf`

Linux: `$HOME/.kbang/kbang.conf`

Konfigurační soubor je v textovém formátu, díky čemuž jej lze editovat běžným textovým editorem. Konfigurační soubor je rozdělen do skupin. Každá skupina začíná řádkem `[jméno skupiny]` a na následujících řádcích

je seznam položek. Položka může obsahovat buď textový řetězec nebo seznam textových řetězců. V případě jednoduché položky má řádek tvar `polozka=hodnota` a v případě seznamů se jednotlivé prvky seznamů zapisují `seznam[]=hodnota`.

Takto vypadá výchozí konfigurační soubor:

```
[network]
server_bind_ip=0.0.0.0
server_description=Default Description
server_name=KBang Server
server_port=6543
[player]
name=Player
password=
[server-list]
hostname[]=alderan.cz
port[]=6543
```

Kapitola 6

Závěr

6.1 Budoucnost projektu

Vzhledem k vesměs pozitivním reakcím lidí z diskuzního fóra na serveru `bang.cz` jsem se rozhodl, že budu pokračovat ve vývoji projektu. Ode dne co jsem na tomto fóru inzeroval první hratelnou verzi KBangu se začly objevovat žádosti o přidání té či oné funkcionality. Dokonce i já jsem při každovečerním „testování“ hry dostával nápady na vylepšení, jejichž realizaci jsem musel odložit, abych stihl dokončit projekt včas.

Mezi nejvíce žádaná vylepšení patří bezesporu možnost zobrazovat karty větší. Velikosti karet jsou pevně nastavené tak, aby se hlavní okno vešlo na monitor s rozlišením 1024×768. V dnešní době jsou však mnohem častější větší rozlišení, jako 1280×1024 a taky 1600×1200 již není výjimkou. Pokud se hlavní okno klienta na takových rozlišeních maximalizuje, je v okně hodně nevyužité plochy. Díky faktu, že karty jsou škálovány na konkrétní velikosti až v klientu je možné karty škálovat dynamicky. Jedna z možností je umožnit uživateli škálovat velikost karet pomocí kombinace kláves (např. Ctrl + + a Ctrl + -). Jiná možnost je karty dynamicky škálovat při změně velikosti okna.

Další často zmiňované vylepšení je implementace oficiálních rozšíření. Tento cíl bych však spíše zařadil mezi dlouhodobé cíle, jelikož bude možná potřeba upravit architekturu aplikace.

Jelikož se nyní hraje KBang výhradně na serveru `alderan.cz`, přemýšlel jsem o vytvoření autentifikačního systému. Každý hráč by se potom mohl volitelně zaregistrovat a hrát pod svým účtem. Na základě odehraných her by se pak každému registrovanému uživateli vytvářela statistika, popř. by

se mohl zobrazovat žebříček hráčů. Jako úschovna dat by se mohl používat nějaký databázový systém a dokonce by mohlo existovat webové rozhraní, umožňující prohlížet statistiku v běžném webovém prohlížeči.

Dále je v budoucnu třeba věnovat čas umělé inteligenci. Současná umělá inteligence je velmi hloupá a proto bude třeba napsat novou, lepší. Také bych rád napsal adaptér, který by převedl rozhraní pro *ovladače hráče* do nějakého jednoduchého skriptovacího jazyka. Tímto krokem bych umožnil komukoli s aspoň základní znalosti programování napsat si svou vlastní umělou inteligenci a dále bychom mohli uspořádat soutěž o nejlepší umělou inteligenci. Přihlášené umělé inteligence by pak svedli souboj v řekněme desetitisících hrách a umělá inteligence s největším skóre by byla začleněna do projektu.

Jedna s důležitých věcí pro rozšíření KBangu je jednoduchá instalace. Do budoucna bych rád přidal nějaký slušný windows instalátor a také bych rád vytvářel balíčky pro přední linuxové distribuce. Dlouhodobým cílem by pak bylo dostat se do oficiálních repozitářů linuxových distribucí.

Literatura

[1] *Blanchette J., Summerfield M. (2008): C++ GUI Programming with Qt 4 — Second Edition*

[2] *DaVinci Games — oficiální stránka hry*,
`http://www.davincigames.com/page_eng.cfm?sez=01&gioco=bang!`

[3] *Ezust A., Ezust P. (2006): An Introduction to Design Patterns in C++ with Qt 4*

[4] *KBang — stránka projektu*
`http://code.google.com/p/kbang`