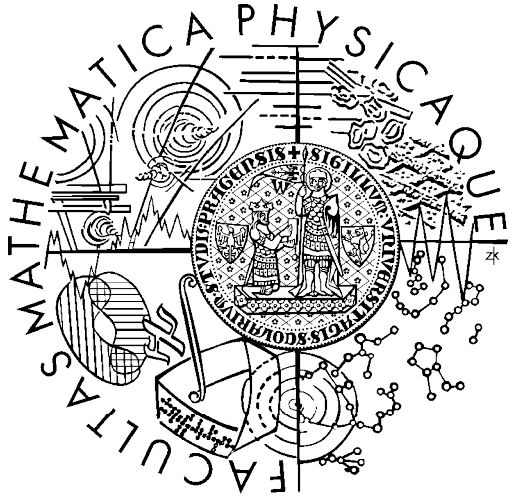
Universita Karlova v Praze
Matematicko-fyzikální fakulta

Bakalářská práce



Martin Urza

RDX2 Server - Red Dragon NeXt Generation,
online tahová strategie z fantasy prostředí.

Středisko informatické sítě a laboratoří
Vedoucí bakalářské práce: Dan Lukeš
Studijní program: Informatika, Správa počítačových systémů

2009

Poděkování

Děkuji svému vedoucímu bakalářské práce, Danu Lukešovi, za vedení tohoto projektu a přínosné diskuse.

Děkuji firmě HIAX.cz s.r.o., především Martinu Janderovi a Pavlu Andrysovi, za cenné rady a poskytnutí potřebných knihoven.

Děkuji všem vyučujícím Matematicko-fyzikální fakulty, kteří mě učili, čímž mi předali potřebné znalosti k napsání projektu.

Děkuji své manželce, Janičce Urzové, za podporu, kterou mi poskytla během tvorby projektu, i za tolerantní postoj, který zaujala k mému pracovnímu vytížení a nedostatku času během tvorby RDX2.

Prohlášení

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

V Praze dne 24.7. 2009 Martin Urza

Obsah

Název práce: RDX2 Server – Red Dragon NeXt Generation, online tahová strategie z fantasy prostředí

Autor: Martin Urza

Katedra (ústav): Středisko informatické sítě a laboratoří

Vedoucí bakalářské práce: Dan Lukeš

e-mail vedoucího: dan@obluda.cz

Abstrakt: RDX2 Server je program, který umožňuje provozování hry Red Dragon NeXt Generation. Jedná se o online tahovou strategii z fantasy prostředí, kterou mohou hrát zároveň stovky tisíc hráčů. Hra navazuje na již existující Red Dragon 2, kterou v mnoha ohledech rozvíjí a zlepšuje (jak po stránce herního systému, tak ohledně možnosti serveru pojmout řádově více hráčů). Se serverem lze komunikovat například pomocí klienta, kterého přikládám k bakalářské práci, ale který však **není** její součástí (nejedná se ani o mé dílo).

Klíčová slova: tahová strategie, online hra, herní logika, komunikace server-klient

Title: RDX2 Server – Red Dragon NeXt Generation, turn-based online strategy fantasy game

Author: Martin Urza

Department: Network and Labs Management Center

Supervisor: Dan Lukeš

Supervisor`s email address: dan@obluda.cz

Abstrakt: RDX2 Server is a program which allows operation of Red Dragon NeXt Generation game. This game is turn based online strategy with fantasy background and it can be simultaneously played by hundreds of thousands players. Red Dragon NeXt Generation is a continuation of Red Dragon 2 which in many aspects exceeds and enhances (in terms of game system and also in terms of performance i.e. possibility of server to seat much more players). Communication with server can be done e.g. via client, which is appended to this bachelor thesis, however is not part of it.

Keywords: turn-based strategy game, online game, game logic, server-klient communication

Kapitola 1 - Úvod

1.1 Cíl

Cílem bakalářské práce bylo vytvořit nástupce hry Red Dragon 2, která vznikla v roce 1999 ze hry Red Dragon 1 a postupně zastarala. Bylo především nezbytně nutné hru výrazně změnit po stránce matematického modelu herních mechanismů, neboť kvůli jejich nedostatkům docházelo k postupnému odlivu hráčů RD2. Dalším cílem bylo zlepšit hru po technické stránce, což si vyžádalo (z programátorského hlediska) její kompletní znovuvytvoření.

V rámci zlepšování herních mechanismů bylo především nutné nahradit stávající systém herních odměn, který se ze začátku zdál dobrým, ale po několika letech hraní se objevily některé jeho nedostatky, které se projevily jako zcela fatální. Krom toho bylo třeba hru oživit o některé nové a zajímavé prvky.

Nedostatky původního technického řešení RD2 byly především v tom, že celý server byl napsán jen jako skript (nikoliv program), což ho činilo pomalým. Kromě tohoto nedostatku jsem objevil ještě několik dalších, které RDX2 také odstraňuje.

Vytvořením Red Dragon NeXt Generation se mi podařilo odstranit klíčové nedostatky a problémy RD2 (jak herního tak technického rázu) a vytvořit plnohodnotného nástupce této hry.

1.2 Online tahové strategie

Z hlediska hráčů je tento typ her oblíben především ze dvou důvodů. Prvním je, že hra není tzv. real-time, což znamená, že nezáleží na tom, kdo má jak rychlé reflexy a jak šikovně dokáže

mačkat klávesy, nýbrž je možno si každý svůj krok dobře rozmyslet. Red Dragon například vyhodnocuje všechny interakce hráčů (typicky to, jak na sebe útočí) pouze jednou denně (veškeré akce, které hráči vůči sobě během dne provedou, se ukládají a jsou pak vyhodnoceny naráz). Druhým důvodem oblíbenosti tohoto typu her je možnost porovnání svých schopností s desítkami až stovkami tisíc živých oponentů. Každý, kdo někdy hrál delší dobu nějakou strategii, určitě přišel po čase na způsob, jak obelstít umělou inteligenci, zjistil její slabiny. Když najdete slabinu živého hráče, ten svou herní taktiku změní. A když hrajete s tolika hráči naráz, je (při dostatečné variabilitě hry) počet herních strategií prakticky neomezený.

Z hlediska vývoje online her je velmi důležité si uvědomit několik věcí. Především to, že požadavky na vyváženost herního systému jsou daleko kritičtější než u her ostatních, a to především proto, že ve hře, které se účastní jeden živý hráč a bojuje s několika oponenty s umělou inteligencí, příliš nezáleží na tom, kdo z nich má lepší výchozí podmínky, neboť obrovský vliv na konečný výsledek mají právě použité algoritmy pro „myšlení“ umělých protivníků. Tam, kde spolu soupeří živí hráči, by měly být jejich výchozí podmínky pokud možno vyrovnané; v opačné případě nebývají hráči spokojeni. Dalším velmi důležitým požadavkem je variabilita (přičemž však typicky platí, že čím více je hra variabilní, tím těžší je zajištění vyváženosti) – když hru hrají tisíce hráčů, kteří se mohou navzájem sledovat a komunikovat spolu, velmi rychle z výběru možných strategií zjišťují, které jsou jak efektivní a pokud je možností málo, není již po nějaké době co vymýšlet. Dále je nutné, aby se hra dynamicky vyvíjela, neboť hráči od provozovatelů typicky nekupují hru „na doživotní hraní“, nýbrž si platí za časové úseky. U her, které si člověk jednou koupí a pak je

hraje kolikrát chce, nezáleží provozovateli na tom, kolikrát (či jak dlouho) hráč hru hraje (jde mu jen o to, aby se zákazníkovi líbila a zakoupil si případně další díl). Provozovatelé online her však potřebují zákazníky držet co nejdéle, protože jejich výdělek nezávisí pouze na počtu hráčů, ale je ještě u každého zákazníka vynásoben časem, po který hru hrál a platil za ni. Z toho důvodu je nutno stále dělat v herním systému změny, aby se u hry mohl jeden člověk bavit i několik let (zkušenosti ukazují, že jen minimum hráčů je ochotno roky platit za hru, která se nemění). S ohledem na to je samozřejmě nutno navrhovat matematický model herního systému, stejně jako je to třeba uvážit při samotném programování.

1.3 Projekt RDX2

Práce na vývoji celého projektu Red Dragon NeXt Generation je značně rozsáhlá týmová práce vycházející z projektů Red Dragon 2 a Red Dragon 1. Celkem se na těchto projektech podílelo (od roku 1996) značné množství lidí (počínaje těmi, kdo původní hru RD1 vymysleli a navrhli, přes programátory, až po grafiky). RDX2 je tedy vyústěním práce mnoha týmů, jejichž poznatky jsou v něm použity. Nejedná se tedy o původní autorské dílo mé osoby.

Má práce začala před několika lety, kdy jsem patřil k týmu, který utvářel herní pravidla RD2. Mimo jiné jsem se podílel i na vzniku projektu RD4, který však nakonec nebyl úspěšný. Na projektu RDX2 má práce spočívala především v návrhu matematického modelu herních mechanismů, při kterém jsem však vycházel z některých principů RD2 (z nichž některé byly původně mé, jiné cizí). Dále jsem naprogramoval herní server RDX2, který však není čistě mou prací, nýbrž jsem do něj převzal

(a případně upravil) části kódu jiných programátorů. Nejednalo se však o žádné rozsáhlé ani důležité pasáže.

Ačkoliv k bakalářské práci přikládám i klienta, tento není její součástí, neboť není ani mým dílem (navíc není ještě zcela dokončen a odladěn, nicméně na zběžné prohlédnutí hry stačí). Důvodem přiložení je, že bez něj si není možno hru ani zkusit zahrát (nicméně i s ním jsou možnosti velmi omezené, neboť k online hře jsou zapotřebí především živí soupeři). Dále přikládám i uživatelský manuál, který mým dílem sice z části je, nicméně rozhodně ne výhradně (je převzatý z manuálu RD2, který je zas převzatý z RD1) a na jeho tvorbě se podílely desítky lidí (často bez znalostí psaní HTML, což je důvodem, proč se výše zmíněné HTML stránky validnímu kódu ani vzdáleně neblíží). K bakalářské práci ho přikládám proto, aby bylo jasné, o co vůbec ve hře jde. Přiložený manuál však není součástí bakalářské práce.

1.4 Prezentace RDX2 Serveru

Podmínky nutné pro běh programu lze najít v uživatelské dokumentaci, která je součástí bakalářské práce. Po samotném spuštění serveru se však nebude nic pozorovatelného dít, dokud někdo nezačne hru hrát. Pro tento účel lze využít přiloženého klienta a k serveru se připojit. Ani poté však server sám o sobě nevykazuje žádnou „zajímavou“ aktivitu – pouze zobrazuje čítače zodpovězených dotazů, přihlášených uživatelů a podobně.

Herní činnost serveru lze pozorovat z klienta poté, co si ve hře založíte gubernát (nabídka je hned na úvodní stránce) a ten pak hrajete podle pokynů v uživatelském manuálu.

Kapitola 2 - Analýza a vývoj

2.1 Jazyk

Původní server RD2 byl napsán jako skript jazyka Perl, což samozřejmě není příliš efektivní řešení z hlediska spotřebované paměti i procesorového času. RDX2 dosahuje výrazného zrychlení použitím programovacího (neinterpretovaného) jazyka.

Co se týče samotné volby jazyka, je RDX2 psán v Delphi (jazyk Pascal). Tato volba byla dána především tím, že v době, kdy jsem projekt začal tvořit, ovládal jsem ho výrazně lépe než ostatní jazyky. Kdybych měl jazyk volit dnes znovu, použil bych pravděpodobně C nebo C++, ačkoliv kód Pascalu považuji za přehlednější a lépe laditelný, nenahrazuje to dle mého názoru chybějící výkon.

2.2 Práce s daty

Původní server RD2 ukládal všechna data do databáze, odkud je také četl, když bylo potřeba. Po provedení analýzy jsem však zjistil, že přes 90% objemu všech dat (jak RD2 tak RDX2) se ukládá do pěti z přibližně padesáti tabulek databáze (přesný počet tabulek neuvádím proto, že je v RD2 a RDX2 různý, navíc se mění v průběhu vývoje). Další analýza ukázala, že 75% z výše zmíněných 90% dat, se ukládá do tří tabulek, ze kterých jsou průměrně přečtena 1-2x a téměř vždy směřuje dotaz k nějaké jejich agregaci (ať už je tato agregace provedena již na úrovni SQL dotazu nebo výše, k uživateli dojde v drtivé většině případů „malý“ výstup nějaké funkce, do které vstoupí „hodně“ dat).

RDX2 dosahuje velmi výrazného zrychlení tím, že všechna data, která nepatří do výše zmíněných pěti tabulek, ukládá do write-throught cache v operační paměti, která je realizována objekty „listů“ (seznamy, jejichž fyzická realizace je rozvedena v dalším odstavci), jejichž každý prvek obsahuje jednu řádku tabulky. Když je třeba data zapsat, děje se tak do paměti i databáze. Je-li třeba data číst, děje se tak pouze z paměti (databáze tedy slouží pro většinu tabulek jen k tomu, aby se z ní při startu programu data načetla do paměti).

Další analýzy ukázaly, že řádky v drtivé většině databázových tabulek jsou poměrně často tvořeny, velmi často modifikovány, ale jen zřídka mazány po jedné (typicky bývají mazány všechny řádky v tabulce, například když se jednou denně vyhodnotí všechny akce hráčů za předchozích 24 hodin, jsou následně vymazány všechny tabulky, do kterých se budou ukládat akce dalšího dne). Z toho mimo jiné vyplývá, že tabulky budou „husté“ ve smyslu, že skoro každá možná hodnota klíče (menší než aktuálně nejvyšší) bude v nějakém záznamu obsažena. Tato vlastnost přináší další možnost zrychlení - ukládání dat do dynamických polí, kdy indexem každého prvku bude jeho klíč, což umožňuje vyhledávání podle klíče v konstantním čase (navíc se jedná o konstantní čas přístupu do paměti, nikoliv na disk). Proto jsou výše zmíněné „listy“ v paměti realizované jako dynamická pole.

2.3 Komunikace server-klient

Původní RD2 server komunikoval s uživateli tak, že generoval HTML stránky, které si oni zobrazovali ve svém prohlížeči. Velikost těchto stránek se většinou pohybovala od 20kB do 100kB (ve výjimečných případech i výš, až 1MB), protože stránky musely

obsahovat i informace o formátování (a všechna statická data, například různé popisky).

RDX2 server využívá ke komunikaci s uživateli externího flashového klienta, ve kterém jsou obsaženy všechny formátovací informace a statická data, takže mu stačí posílat malé XML soubory, které obsahují pouze dynamická data. Jejich velikost se prakticky vždy vejde do 1kB, povětšinou mívají kolem 200B (a kdyby u XML tagů nebyl kladen důraz na čitelnost, mohly by se ještě zmenšit).

2.4 Způsob psaní kódu

Při psaní programu jsem kladl důraz především na jeho přehlednost, snadné psaní kódu a „pohodlné“ programování (stejně jako případné budoucí psaní změn). Vzhledem k tomu, že se jedná o webovou hru, nelze očekávat, že projekt bude jednou napsán, spuštěn, odladěn a ponechán osudu. Ze zkušenosti vím, že hráči chtějí především změnu a nové věci, jinak u hry moc dlouho nevydrží. Bude tedy stále třeba projekt měnit a zasahovat do něj. S ohledem na to byl také napsán. Jména proměnných a funkcí mají leckdy kolem třiceti znaků, aby pokud možno co nejpřesněji popisovaly svůj význam a nebyl zapotřebí dlouhých komentářů (které tam beztak jsou, především u konstant). Kód je rozsekán do mnoha krátkých funkcí, aby se dal co nejsnáze modifikovat.

Kód jsem psal s ohledem na to, že není třeba optimalizovat detaily, pokud to významně ušetří čas při programování či zpřehlední kód za cenu minimálního zpomalení celé aplikace (nemyslím si, že při programování her je bezpodmínečně nutné šetřit každou mikrosekundou procesorového času a každým

bajtem paměti – zastávám názor, že do úvah o optimalizaci je třeba zahrnout i čas, který programátor věnuje danému projektu a – v případě programování her – nepovažuji za správnou optimalizaci, která zvýší výkon o 5% nebo ušetří několik sekund procesorového času denně, ale programátor musí na napsání projektu vynaložit dvakrát více času). Zadání projektu v podstatě vycházelo ze stávající RD2, kterou má RDX2 nahradit a výrazně vylepšit jak po stránce herní, tak i výkonové. Herní vylepšení jsou víceméně popsána v uživatelské dokumentaci (manuálu). Celá filosofie zvyšování výkonu oproti RD2 se opírá o myšlenky, které urychlí hru o několik řádů (použití programu místo skriptu znamená obrovskou úsporu procesorového času na parsování textu; ukládání menších a často používaných databázových tabulek do datových struktur v paměti, které jsou navíc organizované jako pole, takže umožňují konstantní přístup, znamená řádově rychlejší přístup k datům; flashový klient (znovu podotýkám, že ten není součástí bakalářské práce) zase umožní posílat 20x-100x méně dat než HTML stránky, které musí obsahovat mnoho formátovacích údajů), naopak dílčí optimalizace, které urychlí běh programu o několik málo procent (přepočítáno na dotaz klienta a odpověď serveru se jedná o čas tak krátký, že ho teoreticky možná nelze ani měřit, neboť je zanedbatelný proti samotné odezvě při síťové komunikaci), leckdy nebyly cíleně udělány, zejména díky tomu, že by znepřehlednily kód, ale v některých případech i proto, že by neúměrně prodloužily čas, který bych strávil jejich programováním.

Ač jsou v celém kódu relativně často používány objekty, typicky nejsou využity vlastnosti, které objekty jako takové nabízí (zejména dědičnost a zapouzdření). Nejedná se o náhodné nepoužití či mou neznalost problematiky, nýbrž o to, že mi tyto

vlastnosti připadají spíše jako nevýhody. Dědičnost nepoužívám proto, že ať bych teď udělal sebelepší objektový návrh celého programu, je pravděpodobné, že mnoho věcí se bude na hře v průběhu času měnit. U projektů, které se v průběhu své existence často (a někdy velmi výrazně) mění, mám s dědičností špatné zkušenosti – původně dobrý objektový návrh může být po několika letech vývoje projektu špatný a vyžadovat určité změny. Použití dědičnosti těmto změnám často brání a to, že se změny v mateřských objektech promítají do potomků, nemusí být nutně dobře. Co se týče zapouzdřenosti, nepřijde mi u velkých projektů výhodou, protože když na projektu pak pracuje někdo jiný, nebo sám programátor zapomene, co přesně se uvnitř objektů děje (a neumím si představit, jak by to po pár letech nezapomněl), leckdy se pak nestačí divit, jaké všechny automatické jevy doprovází volání funkce, o které se v tu chvíli domnívá, že pouze nastavuje nějakou hodnotu – kód mi pak připadá nepřehledný, protože v něm není na první pohled vidět, co se vlastně děje při volání metod zapouzdřených objektů.

Kapitola 3 - Programátorská dokumentace

(ačkoliv je v dokumentaci celého projektu popsán i klient, nejedná se o součást bakalářské práce, ani to není mé dílo)

3.1 Překlad

Projekt je přeložitelný v prostředí **Delphi 6.0.2 Professional** nad operačním systémem **Windows Server 2003** s následujícími knihovnami (některé z nich jsou komerční a je nutno je před použitím zakoupit):

- ***RX Library*** *(autoři Fedor Kozhevnikov, Igor Pavluk a Sergej Korolev)* – ruská knihovna obsahující spousty šikovných komponent a funkcí, nemá žádné konkrétnější určení, ale používá ji většina programátorů, kteří pracují v Delphi.
- ***ICS*** *(autor Francois Piette; http://www.overbyte.be)* – knihovna, ze které RDX2 Server používá komponentu pro webserver.
- ***IB Objects*** *(http://www.ibobjects.com)* – knihovna pro komunikaci s databází.
- ***TurboPower Systools*** *(http://www.turbopower.com/; http://www.sourceforge.org/)* – knihovna pro manipulaci s řetězci, dále jsou z ní použity některé objekty listů.
- ***TMS ComponentPack*** *(http://www.tmssoftware.com)* – knihovna některých grafických komponent (tlačítka a podobně).
- ***HIAX.cz library*** *(hlavní autor Martin Jandera)* – interní knihovny firmy HIAX.cz s.r.o., které se nedají koupit ani stáhnout, proto je se svolením vedení firmy přikládám k práci.

3.2 Skladba projektu

RDX2 je novou generací jedné z prvních masově úspěšných webových online her. Jedná se o kompletní přepis RD2, bez využití jediné řádky původního kódu v jazyce Perl, který se stal léty modifikací naprosto nepřehledným a významně neměnitelným. Nová verze má kromě toho, že umožní vylepšování hry samotné, vyřešit další dva zásadní problémy: vylepšit herní rozhraní o nové grafické možnosti a umožnit zvládnutí většího množství souběžně připojených hráčů. Z těchto důvodů byla použita kombinace vlastního http serveru s flashovým klientem. Při orientačních testech vykazovala použitá technologie schopnost vyřizovat řádově větší množství dotazů než technologie původní (odhad jsou stovky tisíc online uživatelů oproti tisíci aktuálního maxima).

Projekt se skládá ze dvou zcela izolovaných částí : **serverové** a **klientské**.

Serverová část je vytvořena v prostředí Borland Delphi 6 a je určena k běhu pro operačním systémem Windows, konkrétně Windows Server 2003. Serverová část se opírá o databázový stroj Firebird 2 (InterBase). Serverová část je vybavena pouze jednoduchým uživatelským interfacem, který zobrazuje základní info o stavu serveru a umožňuje klikací konfiguraci, která je jinak uložena v konfiguračním INI souboru, navíc umožňuje manuální start/zastavení serveru a ruční spouštění přepočtů (funkce hry). Server je možno spouštět i z příkazové řádky.

Klientská část je vytvořena v prostředí Adobe Flex 3 a je určena k běhu v přehrávači Adobe FlashPlayer 9 a vyšším, který je volně k dispozici pro drtivou většinu browserů a je dle průzkumů Adobe k dispozici na 98% počítačů s připojením na internet. Závěry lze najít na stránkách Adobe :

http://www.adobe.com/products/player_census/flashplayer/version_penetration.html
http://www.adobe.com/products/player_census/flashplayer/

Komunikace obou částí se realizuje pomocí HTTP protokolu, přičemž si klient se serverem vyměňují data strukturovaná do primitivní formy XML. Generované XML soubory obsahují jen faktická data, žádné formátovací informace, formátování realizuje přímo klient, přenášená data jsou proto ve srovnání s HTML soubory významně menší (průměrná velikost stránky v RD2 je cca 20kB až 100kB, průměrná velikost XML souboru RDX2 je výrazně menší než 1kB). Velikost XML souborů lze navíc ještě řádově zmenšit použitím kratších tagů (ovšem na úkor čitelnosti). Použité komunikační schéma navíc umožňuje vytvoření nových klientů (ty může vytvářet kdokoliv, kdo zná rozhraní), kteří budou komunikovat přímo se serverem (server s tímto počítá a kontroluje přijímaná data proti podvrhům). Se serverem lze de-facto komunikovat bez použití klienta; server kontroluje pouze přijímaná data bez ohledu na to, jakým programem jsou odesílána.

Databázově je systém postaven tak, že drtivá většina dat se načítá při startu serveru do interní write-throught cache; analýza původního RD2 ukázala, že ač má databáze hry cca 50 tabulek (přesný počet tabulek není uveden proto, že v RDX2 je jiný než RD2, navíc s různými změnami ve hře přibývají i tabulky), 90% objemu dat se nachází pouze v pěti z nich, z čehož do třech, které obsahují přibližně 70% všech dat, proběhne typicky jeden zápis a ten je čten průměrně 1-2x (a většinou není zájem ani o konkrétní záznam, ale spíše agregaci několika desítek záznamů). V RDX2 se tedy všechny relativně malé a především často přistupované tabulky načtou do operační paměti. Vlastní výkonné části serveru,

které realizují samotnou hru, vůbec nekomunikují s databází, pouze s interní cache (výjimkou je práce s těmi několika málo tabulkami, které zůstávají v databázi a nejsou do operační paměti načteny). Každý zápis se okamžitě propisuje do databáze, bez prodlevy a server je tudíž kdykoliv schopen restartu bez potřeby provádět flush cache. Drtivá většina práce serveru je pouze s cachovanými daty, což ve výsledku přináší mnohonásobné zrychlení oproti klasické práci přímo s databází. Navíc pokud by časem bylo třeba změnit databázový stroj, vlastní práce serveru se toto vůbec nedotkne. Struktura dat je taková, že do 8GB paměti (kterou má server, na kterém má hra běžet) by se měla vejít data pro řádově několik (málo) milionů herních subjektů, přičemž nejvyšší dosud dosažený počet subjektů v RD2 byl cca 100 tisíc. Pokud by se časem ukázala nemožnost umístit všechna data do cache serveru, lze upravit implementaci cache tak, že se toto nedotkne jejího interface a funkčnost herní části tím nebude nijak dotčena.

3.3 Programové unity serveru

3.3.1 Herní unity

(k pochopení jejich fungování je třeba přečíst uživatelský manuál)

- **RDFormulas**: funkce/výpočty související se hrou. Volají se napříč celým systémem.

- **NMRoutines**: funkce pro zpracování dalšího tahu ve hře. Obvykle se nevolají přímo, pro provedení výpočtu dalšího tahu volá server hlavní funkci ***tahni***. Ta provede celý výpočet ve správném pořadí jednotlivých kroků.

- **CountRoutines**: funkce pro zpracování přepočtu. Přepočet provádí zpracování interakce mezi gubernáty, obvykle jednou denně. Zároveň připočítá nové tahy a připraví herní subjekty do dalšího dne. Funkce z tohoto unitu obvykle nejsou volány přímo, server volá hlavní funkci ***svaceni***, která provede celý přepočet ve správném pořadí jednotlivých kroků.

3.3.2 Systémové unity

- **BackServer**: unit, který řeší dvě základní funkce: práci se sessions a logování požadavků na server. Objekt ***TBackOfficeClass*** zajišťuje podporu sessions. Protože klient není bezstavová HTML stránka, ale kompaktní Flash aplikace, může si klient držet informaci o aktuální session přímo u sebe. Přihlášením je serverem vygenerován unikátní náhodný alfanumerický klíč, který je předán klientovi a při každém dalším dotazu je porovnáván ID herního subjektu, klíč a IP adresa obsažené v dotazu s daty na serveru. Pokud nedojde k plné shodě, není dotaz zpracován. I když dojde k odposlechnutí klíče, nelze toho tudíž z jiné adresy využít. Systém podporuje možnost multiloginu, kdy do jednoho herního subjektu může být souběžně nezávisle přihlášeno více hráčů (což je opět novinka oproti RD2). Platnost sessions je časově omezená, timerem vyvolávaná funkce ***CleanUpSessions*** čistí seznam session od neaktuálních – aby bylo možno zobrazovat počet aktuálně přihlášených hráčů. Objekt ***TBackOfficeClass*** zároveň provádí propis hlášení o chybách z try-except bloků produkčních metod v unitu ***Main*** (viz dále) – každá chyba je tak zaznamenána s tím, že je známý herní subjekt i adresa, ze které přišel problematický požadavek. Objekt ***TServerLogClass*** slouží k logování všech příchozích požadavků, vytváří se standardní log soubor. Zápis do logovacího souboru

vyvolává metoda ***WriteLogDirect***, volaná z metody ***HttpServerGetDocument*** v unitu ***Main*** (viz dále).

- **MainDataModule**: unit, zařizující interakci s databází. Všechny SQL dotazy jsou připravovány při startu pomoci funkce ***Prepare***, což jednak výrazně urychluje pozdější vykonávání parametrizovaných SQL dotazů a jednak zaručuje proveditelnost SQL dotazů jejich verifikací při startu serveru. Všechny dotazy jsou zpracovávány v thread-safe režimu, který je zajišťován pomocí objektů ***TCriticalSection*** a jejich přepínačů Acquire a Release. Je tak zaručeno, že během provádění SQL dotazu nedojde k dalšímu vstupu do metody, ale bude se čekat na dokončení předchozího volání. Při zakládání nových záznamů se nepoužívají triggery ani jiná forma autoincrement polí pro identifikátory záznamů, ale sekvence, kde se nejdříve najde poslední použitý identifikátor a tento se manuálně inkrementuje. Tato technika je použita pro zajištění kompatibility pro případ použití jiného DB serveru.

- **CacheObjects**: unit, definující a provozující interní cache serveru. Téměř každá datová tabulka, používaná systémem, má svůj obraz v cache. Cache není ovšem přesným obrazem datových tabulek, obsahuje řadu cíleně redundantních dat, kdy různé strukturování dat slouží k rychlostní optimalizaci práce s daty. Pro faktické uložení dat jsou použity kontejnerové struktury typu TIntList, identifikátory jsou typu longint, limit MaxInt záznamů v jedné tabulce je pro potřeby systému více než dostačující; analýza útoku na server tím, že by se útočník pokusil vyčerpat identifikátory některé z tabulek ukázala, že při 1000 žádostech za sekundu, by útok trval více než tři roky, krom toho by musel útočník vést útok z mnoha různých IP adres, neboť nastavení firewallu na serveru, kde má RDX2 běžet, z jedné IP adresy tolik žádostí nenechá do programu vůbec přijít. Kontejnery nejsou nijak

tříděny, díky použitým identifikátorům se k jednotlivým prvkům přistupuje jako k poli, čili přes přímý index, což znamená, že vyhledávání není nutné, protože ke každému objektu lze přistupovat v konstantním čase (což je další velmi výrazné zrychlení oproti RD2, které data načítalo z databáze). K jednotlivým prvkům lze přistupovat pomocí property, fingující skutečné pole. Property jsou příležitostně použity i v situacích, kdy je třeba spouštět nějakou funkci při zápisu nové hodnoty. Write-through funkcionalita je realizována nikoliv automaticky, ale voláním funkcí ***Save***<*to, co se má uložit*>, které má každý jednotlivý objekt v různém počtu. U tabulek s větším počtem sloupců je efektivnější ukládat vždy jen určitou část záznamu, tu, která se měnila. Naopak není efektivní ukládat změny jednotlivých sloupců, protože obvykle dochází ke změnám více sloupců naráz a posílání dílčích SQL příkazů by zatěžovalo DB server přespříliš. Navíc by byl start systému příliš zpomalen inicializací extrémního počtu předpřipravovaných dotazů. Ostatní části programu se proto samostatně rozhodují podle situace, kdy provést uložení dat. Například při provádění přepočtu, kdy dochází k aktualizaci většiny dat, se do DB propisuje průběžně jen menší část změn, ale po skončení přepočtu se propíší paušálně všechna obvykle modifikovaná data, ať už byla nebo nebyla fakticky změněna (že je tato možnost lepší než průběžné propisování v přepočtu, nebylo zjištěno teoretickou analýzou problému, ale testováním obou možností – díky přehlednému a pružnému kódu trvá náhrada jedné možnosti za druhou jen několik hodin). Příslušné metody Cache objektů jsou koncipovány tak, aby zabezpečovaly plně automaticky konzistenci redundancí (např. při založení záznamu, který je podřízen jinému záznamu, je automaticky aktualizován seznam podřízených záznamů ve vlastnictví nadřízeného záznamu

apod). Metody pro startovní načtení dat automaticky nastavují či vypočítávají data, která jsou jen v cache, ale nemají fyzickou obdobu v DB (součtové, kalkulované položky, master-slave seznamy apod).

3.3.3 Obecné knihovny

- **MXRoutines**: zcela obecná firemní knihovna bez jakýchkoliv vazeb zpět do RDX2. Funkce pro práci s čísly, datumy, stringlisty v souvislosti s XML a řady dalších funkcí.

- **XMLRoutines**: obecná firemní knihovna pro práci s nejprimitivnější formou XML dokumentů. Zahrnuje podporu pro UTF8 kódování, pro RTL jazyky a pro maskování tagů. Bez zpětných vazeb do RDX2.

- **UStr32**: starší obecná knihovna pro práci se znakovými řetězci. Využívaná pro konverze čísel do znakových řetězců a pro formátování textů. Bez zpětných vazeb do RDX2.

- **Filer32**: starší obecná knihovna pro práci se soubory. Využívaná zejména pro práci se serverovými logy. Bez zpětných vazeb do RDX2.

3.3.4 Main

Hlavní řídící unit. Zahrnuje i veškeré GUI serveru a jeho obsluhu. Pro základ HTTP serveru je použita komponenta ze sady ICS, autora F.Pietteho. Tato komponenta odchytává příchozí požadavky na HTTP server. Pokud je požadovaný dokument typu XML (rozlišení podle přípony souboru), rozliší, o jaký XML soubor se jedná, a pokud ho zná, vytvoří dynamickou odpověď. Pokud jde o soubor jiného typu, zachová se jako standardní HTTP server a

vrátí odpovídající diskový soubor – tím se vlastní kód nemusí zabývat např. posíláním grafiky a výchozích HTML souborů. Každý známý a podporovaný XML soubor je třeba nejdříve zaregistrovat v těle metody ***SetUpRequestList***. Metoda ***HttpServerGetDocument*** (event handler HTTP komponenty) pak podle požadovaného dokumentu určuje, která metoda zpracuje odpověď. Toto rozdělení na registraci a následné vyhledávání jen registrovaných dokumentů je použito proto, že umožňuje použít konstrukci case pro určení zpracovávající metody, jinak by se musela použít nepřehledná sekvence vnořených if konstrukcí. Metody, zpracovávající příchozí požadavky, mají velmi podobnou konstrukci. Z tohoto pravidla existuje logická výjimka pro metodu, generující data pro přihlašovací obrazovku (nemá ověření session) a výjimky pro některé POST metody, kdy příslušná metoda provede zpracování požadavku, ale klientovi vrátí výsledek ekvivalentní GET metoda. Každá metoda zpracovávající požadavky provádí ověření session, pokud je v pořádku, může provést podle potřeby analýzu přicházejících dat a následně vygeneruje XML odpověď. K provádění vlastních funkcí souvisejících s hrou, se obvykle volají funkce z unitu ***RDFormulas***. Každé výkonné tělo je obestavěno try-except blokem, který má zajistit, aby žádný jednotlivý dotaz nezpůsobil nestabilitu celého serveru a zároveň loguje všechna zachycená selhání pomocí session manageru. Metody neřeší fakt, že jsou spouštěny vícethreadově, tak jak HTTP server komponenta zpracovává příchozí požadavky – vícevláknovou bezpečnost je třeba řešit v komunikaci s databází a tu si řeší přímo ty funkce, kterou s DB komunikují (unit ***MainDataModule***).

3.4 Programové jednotky klienta

3.4.1 MXML soubory (RDX2 a další)

Jedná se o soubory s definicí GUI, webových služeb. Až na výjimky neobsahují výkonný kód, jen formátovací definice a definici XML webových služeb. Soubory jsou v drtivé většině automaticky generované prostředím Adobe Flex.

3.4.2 AS soubory (Click, ReqResult, Routines, Central)

Jedná se o soubory ActionScriptu. Obsahují výkonný kód klientské aplikace.

- **Click.as** – funkce, představující event handlery aktivních prvků klienta. Z velké části tyto funkce připravují na základě vyplněných editovatelných položek požadavky na server a následně je odesílají na server ke zpracování. Každý požadavek obsahuje informaci o přihlášeném herním subjektu a id aktuální session – tyto informace slouží serveru k verifikaci tazatele.

- **ReqResult.as** – funkce, představující reakce na příjem odpovědi serveru na odeslaný požadavek – obvykle zobrazení výsledků a dalších informací. Komunikace se serverem probíhá asynchronně, tzn. mezi odesláním požadavku a příjmem odpovědi není přímá vazba, definice požadavku obsahuje i určení, jaká funkce se má volat po dokončení příjmu odpovědi. Každá funkce z ***ReqResult*** nejdříve provede obecné zpracování výsledku pomocí funkce ***Process_Request***, která určí, zda přišla validní odpověď. Mohou přijít dva typy nevalidních odpovědí: chybná session (pak dojde k přesměrování na přihlašovací stránku) nebo informace o tom, že právě probíhá herní přepočet a je třeba chvíli počkat.

- **Routines.as** – ostatní funkce. Zásadní funkcí zde jsou funkce ***MakeURL*** a ***InitializeURLs***. ***MakeURL*** upravuje URL tak, že předsazuje před jméno dotazu jméno serveru, ze kterého byl klient spuštěn. To umožňuje spouštět identický soubor klienta na více různých www serverech bez nutnosti cokoliv v něm měnit. ***InitializeURLs*** jen spouští ***MakeURL*** na známé typy dotazů a de-facto tím připojuje jednotlivé dotazy ke správnému serveru. Dotazy na jinou URL, než připravenou pomocí ***MakeURL*** nebudou korektně zpracovány. Toto schéma umožňuje paralelní běh více serverů na různých IP adresách, přiřazených třeba i jednomu fyzickému stroji s tím, že není třeba provádět žádné konfigurace na straně HTML stránek, ze kterých uživatelé spouští klientskou aplikaci ani na straně klienta samotného.

- **Central.as** – víceméně opuštěný soubor, který obsahuje funkce, vzniklé před historickým rozdělením funkcí podle zaměření do souboru ***Click*** a ***ReqResult***.

Kapitola 4 - Uživatelská dokumentace

4.1 Požadavky pro běh serveru

4.1.1 Softwarové požadavky

Server RDX2 byl psán pro operační systém Windows Server 2003. Byl však úspěšně otestován i na operačním systému Windows Server 2008.

Databáze, nad kterou je RDX2 server napsán, je Firebird verze 2.1. Vzhledem k tomu, že nejsou využívány žádné speciální funkce ani dotazy, které by nižší verze této databáze neuměly zpracovávat, podle dokumentace Firebirdu by mělo být možné server spustit nad libovolnou verzí databáze. Testování však ukázalo, že dokumentace Firebirdu neodpovídá realitě, takže ve skutečnosti program běží pouze nad některými verzemi (metodou pokusů a omylů bylo zjištěno, že se jedná pravděpodobně o verze 2.1 a vyšší, ale nebyly otestovány všechny).

Konkrétní databázi (tedy tabulky pro „novou hru“, které jsou většinou prázdné, jindy obsahují základní inicializační data) je možno sestavit s pomocí souborů přiložených k projektu. Většina sloupců a tabulek v databázi je řádně okomentovaná, nicméně k pochopení komentářů je pravděpodobně nutné přečíst uživatelský manuál.

Protože program nevyužívá žádných speciálních vlastností databáze, žádné autoinkrementování klíčů, sám hlídá jejich unikátnost, používá jen úplně jednoduché SQL dotazy (drtivá většina operací s daty se realizuje v paměti) a z databáze jen jednou při startu data načte a pak už do ní jen zapisuje, lze pomocí poměrně malých změn a minimálního zásahu do

programu docílit toho, že server poběží de facto nad libovolnou databází (a nemusí to být ani tak moc výkonná databáze, protože se z ní čte jen jednou při startu programu).

S operačním systémem je to podobné; RDX2 server nevyužívá žádných nestandardních systémových volání, a proto by mohl velmi pravděpodobně po minimálních zásazích běžet na většině verzí Windows (což však není otestováno, jedná se pouze o můj názor).

4.1.2 Hardwarové požadavky

Na běh samotného serveru (bez uživatelů a jejich účtů) stačí cokoliv, na čem může běžet odpovídající operační systém a databáze. S přibývajícími uživateli stoupají i hardwarové nároky.

Simulace ukázaly, že pro 100.000 a méně uživatelů rostou nároky přibližně lineárně (díky způsobu ukládání dat v paměti je k nim možno přistupovat v konstantním čase), pro více než 100.000 uživatelů simulace provedeny nebyly, ale lze očekávat, že náročnost poroste lineárně i nadále. Chování uživatelů způsobuje výkonové výkyvy (čím více spolu hráči RDX2 bojují, tím vyšší nároky jsou na server kladeny).

Program byl testován na různých počítačích, počínaje starými laptopy, až po velmi výkonný server (*System Manufacturer: Intel; System Name: S5000PAL; Mainboard Vendor: Intel; Mainboard Model: S5000PAL0; BIOS Vendor: Intel Corporation; BIOS Version: S5000.86B.10.00.0085.112920071426; Number of processors: 1; Number of cores: 4 per processor; Number of threads: 4 per processor; Name: Intel Xeon E5450; Code Name: Harpertown; Specification: Intel(R) Xeon(R) CPU E5450 @ 3.00GHz; Package: Socket 771 LGA; Family/Model/Stepping: 6.7.6; Extended*

Family/Model: 6.17; Core Stepping: C0; Technology: 45 nm; Core Speed: 2992.4 MHz; Multiplier x Bus speed: 9.0 x 332.5 MHz; Rated Bus speed: 1330.0 MHz; Stock frequency: 3000 MHz; Instruction sets: MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, EM64T; L1 Data cache (per processor): 4 x 32 KBytes, 8-way set associative, 64-byte line size; L1 Instruction cache (per processor): 4 x 32 KBytes, 8-way set associative, 64-byte line size; L2 cache (per processor): 2 x 6144 KBytes, 24-way set associative, 64-byte line size; Chipset & Memory: Northbridge: Intel 5000P rev. B1, Southbridge: Intel 6321ESB rev. 09, Memory Type: FB-DDR2, Memory Size: 8192 MBytes, Memory Frequency: 332.5 MHz (1:1); Module 1: FB-DDR2, PC2-5300 (333 MHz), 2048 MBytes, Kingston; Module 2: FB-DDR2, PC2-5300 (333 MHz), 2048 MBytes, Kingston; Module 3: FB-DDR2, PC2-5300 (333 MHz), 2048 MBytes, Kingston; Module 4: FB-DDR2, PC2-5300 (333 MHz), 2048 MBytes, Kingston; HDD: RAID 10 Intel Embedded Server RAID Technology II; Software: Windows Version: Microsoft Windows Server 2008 64bit Service Pack 1 (Build 6001), Database: Firebird 2.1 64bit), který by měl být schopen podle přibližných odhadů (založených na základě výsledků simulací) při plném vytížení zvládnout 1.000.000 uživatelů (není to však potvrzeno, protože simulace tolika uživatelů je technicky velmi náročná).

4.2 Komunikační protokol

(pro potřeby napsání dalšího klienta)

Server s klienty komunikuje přes standartní HTTP protokol, kdy klient posílá GET či POST requesty serveru.

Přes HTTP protokol jsou posílány XML soubory. Seznam názvů přípustných souborů je uložen v objektu ***RequestList*** (unit

Main), který vyplňuje procedura ***TmainForm.SetUpRequestList*** (v jejím těle v unitu Main si lze prohlédnout, o které soubory se jedná).

Pokud někdo pošle serveru jiný než nadefinovaný dotaz (ať už se jedná o běžný smysluplný dotaz, nebo nějaký nesmysl), server ho ignoruje a neodpovídá. Stejně se chová v případě, že se klient dožaduje jiného souboru, než toho, který je nadefinován v ***RequestListu***.

V případě, že soubor v listu skutečně je, musí ho server vyhodnotit. O tom, jak ho vyhodnotí (nebo jestli ho zahodí), rozhoduje procedura ***TmainForm.HttpServerGetDocument*** (v unitu ***Main***), ve které je case, který pro každý typ dotazu generuje odpověď pomocí ***ClientCnx.AnswerString***, jejíž posledním parametrem je funkce, která tvoří vlastní XML soubor. I přes velmi jednoduché XML formátování (posílají se jen data, nikoliv vzhled), jsou tyto soubory různé (podle logických celků hry a jednotlivých obrazovek klienta).

Avšak formát každého jednotlivého souboru lze dohledat ve funkci, která ho generuje (která to je, se zjistí v ***HttpServerGetDocument***). Drtivá většina dotazů obsahuje tagy s ID uživatelského účtu (dále gubernátu) a ID session, což slouží k ověřování pravosti dotazu (dalším kritériem je IP adresa; musí se shodovat IP, session i gubernát, aby server povolil cokoliv v gubernátu udělat). Která trojice údajů je ta „správná“, se určuje při přihlášení (server si pak pamatuje, z jaké IP adresy, s jakým session ID a do jakého gubernátu proběhlo přihlášení). Z toho mimo jiné vyplývá, že některé dotazy nemohou tuto ověřovací trojici obsahovat (jedná se o dotazy, které klient serveru posílá před přihlášením do gubernátu).

V rámci struktury jednotlivých XML souborů je irelevantní, v jakém pořadí za sebou položky jdou, záleží pouze na tom, aby se shodovala jména tagů.

Kapitola 5 - Závěr

5.1 Dosažené výsledky

V průběhu práce na projektu jsem postupně analyzoval hlavní problémy RD2, mezi které patří především jednorozměrný systém herních odměn, stereotyp pro déle hrající uživatele, příliš pomalý běh interpretovaného skriptu, nešikovná práci s daty a neoptimální řešení komunikace server-klient. Tyto jsem postupně odstranil zavedením nového systému herních odměn (přidáním tzv. „slávy“, která se získává principiálně opačně než současná herní odměna, tzv. „pozemky“), doplněním dalších herních možností, vytvořením neinterpretovaného herního serveru, ukládáním malých a často přistupovaných dat do paměti místo do databáze a uložením statických a formátovacích dat do klienta, aby je nebylo nutno pokaždé odesílat.

5.2 Budoucnost projektu

Řešením vyjmenovaných problémů jsem splnil cíl mé práce, což bylo vytvořit nástupce hry Red Dragon 2. Jak úspěšně se to podařilo, se ukáže časem podle počtu spokojených hráčů této hry. Jak jsem uvedl v první kapitole, je každá online strategická hra záležitostí postupně se vyvíjející a dynamicky měnící. K tomu, aby hra úspěšně fungovala, bude nutné postupně doplňovat a měnit některé detaily podle zájmů a potřeb hráčů. Z toho tedy vyplývá, že se hře budu dále věnovat a odevzdáním bakalářské práce moje fungování na RDX2 nekončí.

Použitá literatura a prameny

http://www.rd2.cz/