VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HERNÍ PORTÁL V CLOUDU

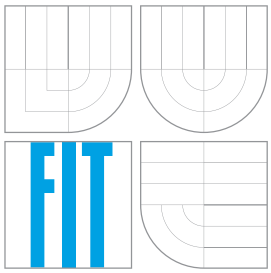
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZDENĚK PEČÍNKA

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HERNÍ PORTÁL V CLOUDU

GAME PORTAL IN CLOUD

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ZDENĚK PEČÍNKA
AUTHOR

VEDOUCÍ PRÁCE — RNDr. PETR ŠKODA
SUPERVISOR

BRNO 2015

Abstrakt

Tato bakalářská práce se zabývá tvorbou herního portálu v Cloudu navrženého pro zvoleného poskytovatele PaaS. Informuje o možnostech, jež nabízí cloudové počítání. Shrnuje výhody a nevýhody poskytovatelů PaaS. Věnuje se návrhu a implementaci aplikace se zaměřením na její škálovatelnost a modularitu. Výsledkem práce je herní portál umožňující hraní her Tanky a Lodě, které je možné hrát proti jiným uživatelům i proti počítačem ovládanému protihráči. Součástí řešení je zhodnocení škálovatelnosti nejnáročnějších komponent, založené na výsledcích zátěžových testů.

Abstract

This thesis deals with the development of the gaming portal in Cloud designed for selected PaaS provider. Informs about the possibilities offered by cloud computing. Summarizes the advantages and disadvantages of PaaS providers. It pursues the design and the implementation of the application, focusing on its scalability and modularity. The result is the gaming portal providing games Tanks and Ships, which you can play against other users or against a computer controlled opponent. The solution contains the evaluation of the scalability of the critical components, based on the results of the stress tests.

Klíčová slova

web, hra, lodě, tanky, cloud, škálovatelnost, PaaS, Node.js, MongoDB

Keywords

web, game, ships, tanks, cloud, scalability, PaaS, Node.js, MongoDB

Citace

Zdeněk Pečínka: Herní portál v Cloudu, bakalářská práce, Brno, FIT VUT v Brně, 2015

Herní portál v Cloudu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana RNDr. Petra Škody.

.....
Zdeněk Pečínka
20. května 2015

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce RNDr.Petru Škodovi za rady a konzultace, které mi pomohly s dokončením práce.

Obsah

Kapitola 1

Úvod

Cloudové počítání je fenoménem současnosti, který silně ovlivnil vývoj moderních informačních systémů. Přináší mnoho výhod jak pro malé projekty, tak pro mohutné firemní systémy. Usnadňuje vývoj aplikací i následný přístup k nim a současně pomáhá ekonomicky využívat výpočetní výkon poskytované infrastruktury.

Tato práce se zaměřuje na tvorbu herního portálu v cloudu, pro vybraného poskytovatele PaaS. Ve druhé kapitole se dozvíte více informací o tom, co pojem cloud v informatice znamená. Jaký je jeho význam a možnosti pro různé druhy uživatelů. Kapitola slouží jako teoretický úvod do problematiky projektu.

Vývoj aplikace je zaměřený především na škálovatelnost a modularitu. V kapitole věnované návrhu jsou rozebrány nejdůležitější myšlenky, spojené s výběrem technologií pro implementaci portálu, a problémy ovlivňující výslednou architekturu aplikace. Součástí návrhu je i popis očekávané poměrné zátěže jednotlivých komponent.

Pátá kapitola popisuje nejzajímavější části implementace, mezi které patří například systém komunikace mezi klientem a serverem, způsob přidávání her a v neposlední řadě implementace Lodí a Tanků.

Kapitola testování popisuje vznik nové testovací aplikace, umožňující simulování libovolného počtu připojení na server herního portálu, způsob hodnocení jednotlivých testů a vyhodnocení všech získaných dat.

Kapitola 2

Cloud

2.1 Co je to cloudové počítání?

Ve své podstatě se jedná o poskytování služeb informačních technologií, jen o něco výhodnějším způsobem, než bylo kdy dříve možné. Cloudové počítání umožňuje uživateli přistoupit k požadované službě prakticky odkudkoliv a to bez nutnosti instalace speciálního software na vlastním přístroji. Ve důsledku je užívání služeb příjemnější a jednodušší, omezuje zodpovědnost koncového uživatele, a zároveň šetří výdaje, jak na straně poskytovatele, tak na straně uživatele.

2.2 Příklady cloudových služeb

Cloudové služby se již teď vyskytují všude kolem nás, přitom se každým dalším dnem pole jejich působnosti stále rozšiřuje na další a další oblasti. Jakých oblastí se tedy cloud týká dnes? Už delší dobu platí, že jde o služby, které jsou pro uživatele přístupné po internetu, jde tedy o všechny emailové služby, vyhledávače, osobní blogy, sociální sítě, internetové obchody, registry, datová uložiště, databáze, servery pro sdílení multimédií a mnohé další. Položky předchozího výčtu mají společné dvě věci. První z nich je potřeba propojit více klientů mezi sebou a umožnit jejich vzájemnou komunikaci, druhou je potřeba spravovat ukládání dat, tak aby byla pro uživatele dostupná dle bližší specifikace služby. V dnešní době se objevuje čím dál více cloudových služeb, zaměřených spíše na šetření výkonu uživatelova přístroje a usnadnění jeho přístupu ke službě. Jde například o antivirové programy běžící v cloudu, přes nástroje pro provádění náročných výpočtů, jako je analýza vědeckých dat, až po poskytování celé platformy, primárně za účelem usnadnění práce pro uživatele služby.

2.3 Omezení cloudových služeb

Cloudové služby sice nabízejí velké množství výhod, zároveň ale mají (mimo jiné) jednu velice vážnou nevýhodu. Pro využívání jakékoliv cloudové služby je bezpomínečně nutné, aby uživatel měl stálý přístup k internetovému přípojení, a tato podmínka jej činí v podstatě závislým. V případě, že dojde k narušení internetového pokrytí, nebo třeba jen jeho dostatečnému oslabení, je poskytování služby dočasně nemožné, což může mít různé důsledky na koncového uživatele.

Tyto důsledky se samozřejmě liší podle poskytované služby, jedná-li se o sociální síť, pak se uživatel prostě bude nějakou dobu více věnovat reálnému světu. Problém nastane když

například nedojde důležitý email, nebo například když uživatel cloudovou službu používá ke své obživě. Mezi další nevýhody patří třeba možné omezení pravomocí uživatele.

Na jednu stranu je dobré odstínit část toho, co se děje na pozadí, a uživateli zobrazit jen to co chce. Ovšem v případě, že by uživatel potřeboval pozměnit detaily v konfiguraci, může narazit na problémy, které by nemusel potkat v případě, že by poskytovaná služba byla nainstalovaná přímo na jeho vlastním přístroji.

2.4 Požadavky na cloud

Každá cloudová služba by měla splňovat jisté požadavky. Existuje celá řada různých popisů cloudu. Vzhledem ke komplexnosti tohoto řešení není překvapením, že každý z těchto popisů na problematiku nahlíží jiným způsobem a tím pádem většinou nezachycuje všechny její podstatné aspekty.

Já bych rád zmínil informace obsažené v definici cloudu napsané pro *Národní institut standardů a technologií Spojených států amerických*[15]. Následují v ní vyjmenované požadavky, které by měly v cloudu být řešeny:

- **Služba na požádání** - služba musí být pro uživatele přístupná bez zpoždění a automaticky, bez nutnosti interakce s jinou osobou na straně poskytovatele.
- **Přístupnost po síti** - služba musí být přístupná odkudkoliv, ať už pomocí internetového prohlížeče, nebo použitím klienta dané služby.
- **Sdružování zdrojů** - všechny fyzické i virtuální zdroje musí být sdruženy, a podle potřeby dynamicky přiřazovány uživateli, který tedy ani nemusí vědět, kde se počítač, na kterém běží jeho aplikace, nachází.
- **Pružnost přidělování zdrojů** - výpočetní kapacita přiřazená uživateli se musí rychle přizpůsobovat jeho požadavkům. Toto přizpůsobování by mělo probíhat v reálném čase a pokud možno automatizovaně. Služba by měla poskytovat zdánlivě až nevyčerpatelnou výpočetní kapacitu.
- **Měření služby** - je nutné umožnit měření a s tím spojené upravování výpočetní kapacity přidělené uživateli. Monitorování a zprávy o činnosti musí být přístupné jak pro poskytovatele, tak pro uživatele.

Jednoduše řečeno, cloud je model, umožňující pomocí internetu ekonomicky a spravedlivě využívat výpočetní kapacitu, jež nám nabízí tisíce počítačů po celém světě. Díky cloudu je možné sdílet nejen prostor pro ukládání dat, ale také samotný výpočetní výkon, což umožňuje zcela nový přístup k tvorbě i využívání aplikací. Cloudové řešení je možné používat jak v soukromé firmě, tak v poskytování služeb veřejnosti. Poskytované služby se objevují na různých úrovních abstrakce - poskytovatel může mabízet k pronájmu již hotovou službu, jejíž modifikace pro uživatele ale není možná, na druhou stranu jde sdílet samotný výpočetní výkon bez programové výbavy, ale uživatel si sám vybere, jak ho nejlépe využije. Následující podkapitoly se budou věnovat právě dělení cloudových služeb.

2.5 Dělení podle nasazení

Podle způsobu nasazení je možné klasifikovat následující infrastruktury cloudu.[15]

- **Soukromý cloud** - služba je poskytována jedné konkrétní soukromé organizaci, za konkrétním účelem. Může být vlastněna nebo spravována touto organizací, externím poskytovatelem, nebo je možné i rozdělení zodpovědnosti mezi obě možnosti.
- **Komunitní cloud** - služba je poskytována za jedním konkrétním účelem, ale její vyžívání je umožněno více osobám nebo organizacím podle jejich potřeb. Správu služby může mít na starosti jedna z organizací nebo externí poskytovatel.
- **Veřejný cloud** - služba je určena pro každého, kdo ji může potřebovat. Veřejné cloudy mohou být založeny za spoustou různých účelů, ovlivněných především poptávkou veřejnosti. Veřejné cloudy jsou poskytovány obchodními, vládními nebo akademickými organizacemi.
- **Hybridní cloud** - je cloudová infrastruktura využívající kombinaci předchozích způsobů nasazení cloudových služeb. Hybridní cloud je používám například, když organizace mimo svůj vlastní soukromý cloud používá i služeb jiného veřejného, nebo komunitního cloudu.

2.6 Dělení podle služby

Podle poskytované služby je možné cloud rozdělit na tři základní třídy, lišící se hlavně v druhu poskytovaných služeb, což úzce souvisí i s odlišným přístupem a možnostmi uživatele.[15]

- **Software as a Service** (*SaaS*) je cloudový model založený na poskytování software uživateli. Již v dřívějších dobách bylo možné nakupovat programy. Stará koncepce ovšem s sebou nesla mnoho nepříjemných potíží. Zakoupený program bylo nutné nainstalovat a zprovoznit na lokálním počítači. Což pro obyčejného uživatele nemusel být jednoduše vyřešitelný probém. I v případě, že se podařilo software nainstalovat a zprovoznit, mohlo stále dojít k potížím. Z těchto důvodů bylo nutné, aby producent softwaru platil peníze zaměstnancům technické podpory.

  Řešením tohoto problému je použití služeb poskytovaných pomocí cloudu. V takovém případě uživatel není nucen zakoupit si vlastní software, ani ho zprovozňovat na vlastním počítači. Místo toho platí menší poplatky za využívání softwaru vlastněného poskytovatelem SaaS. Výhodou je to, že uživatel zaplatí pouze za to, co opravdu využije, navíc nemusí řešit žádné problémy s instalací a tím pádem se k vlastní práci dostane mnohem rychleji. Tento systém je zároveň výhodný i pro poskytovatele služby. Jelikož software běží na jeho serverech, není nutné řešit technickou podporu pro uživatele, a všechny problémy se řeší mnohem rychleji.

- **Infrastructure as a Service** (*IaaS*) je cloudový model založený na poskytování samotné infrastruktury. Jedná se o již dlouho používaný koncept, umožňující uživateli pronájem výpočetního výkonu za účelem zpracování nebo uložení dat, nebo poskytování hostingu. Uživatel dostane plný přístup k výpočetním zdrojům a může na ně aplikovat libovolný vlastní software od operačních systémů, přes databáze až k serverovým firewallům.

Hlavní výhodou je, že uživatel nemusí řešit problémy s napájením a údržbou fyzických zdrojů. Za tento problém je odpovědný poskytovatel IaaS, což ulehčuje uživateli práci. Vzhledem k poskytování samotné infrastruktury je nutné, aby uživatel dodal vlastní software, na druhou stranu má při jeho výběru naprostou svobodu.

- **Platform as a Service** (*PaaS*) je cloudový model, jež bychom při jistém stupni abstrakce mohli popsat jako kombinaci modelů SaaS a IaaS. Pokud zhodnotíme poskytovanou službu podle toho, kdo ji může přímo využívat, byl by model IaaS na vrcholu pomyslné pyramidy odbornosti. Když uživatel dostane k dispozici samotnou infrastrukturu, čeká na něj stále spousta práce s nasazováním vlastního softwaru a související konfigurací. Model je tedy vhodný převážně pro technicky zdatné uživatele.

  Na druhé straně by byl model SaaS, který poskytuje již připravenou službu. Tu velmi často mohou využívat i lidé, kteří o informačních technologiích nevědí vůbec nic. Pokud jde například o schránku elektronické pošty, může se jejím uživatelem stát opravdu téměr kdokoliv, bez toho aby se staral o další detaily.

  Model PaaS se nachází v pomyslném středu mezi těmito dvěma extrémy. Umožňuje uživateli zapomenout na problémy IaaS díky tomu, že mu nabídne část operačního systému, který je předem připravený k používání vybraných programovacích jazyků, datových uložišť a dalších služeb a nástrojů. Oproti SaaS ale neposkytuje žádnou konkrétní službu - je tedy stále vhodný pro relativně obecné využití, které je omezeno právě pevně danými podporovanými technologiemi. Tento problém ale není v důsledku až tak markantní, díky tomu, že existuje stále rozrůstající se pole poskytovatelů PaaS. Každý poskytovatel nabízí služby trochu odlišné, než ostatní, proto věřím, že platí staré dobré pravidlo *"Kdo hledá, najde"*. Rozboru technologií, jež nabízejí jednotliví poskytovatelé se budeme podrobně věnovat v následující kapitole.

2.7 Databáze v Cloudu

Ukládání dat je často řešený problém, ať už při tvorbě interaktivních webových stránek, nebo v informatice obecně. Týká se i tohoto konkrétního projektu, kvůli potřebě pamatovat si nejen přihlašovací jména, ale i další informace jako jsou statistiky nebo dosažené úspěchy uživatelů.

V cloudovém prostředí se tento problém řeší různými způsoby. Někteří poskytovatelé nabízejí mimo samotného výpočetního výkonu i zabudovanou podporu pro určitý typ databáze. Někteří to řeší podobným způsobem, pomocí tzv. addonů, které za příplatek rozšiřují základní poskytované služby, kde jedním z těchto rozšíření může být právě používání databáze.

V případě, že řešení nabízené poskytovatelem cloudu není vhodné, je nutné sáhnout po externím poskytovateli hostované databáze. Tito poskytovatelé ve většině případů nabízí ukládání dat v souladu s filosofií cloudového počítání, v tzv. distribuovaných databázích. Tento systém umožňuje, jak je v cloudu běžné, optimálně využívat fyzické zdroje a současně usnadnit práci uživateli. Data mohou být uložena na více různých počítačích, spravována lokálním systémem. Mezi jednotlivými uložišťmi neexistuje žádná hierarchie řízení, aby bylo možné k nim jednotně přistupovat z vyšší úrovně řízení, která je virtuálně sjednotí pro uživatele. Z pohledu uživatele se tedy vše jeví, jako by pracoval s databází jako s jedním velkým celkem.

V nejčastěji používaných databázových systémech se používají následující dva koncepty.[14]

- **Relační databáze** (*SQL*) prošly již mnoha léty používání a práci s němi usnadňuje velké množství podpůrných materiálů. Jsou založeny na velmi striktním uspořádání dat. Data jsou ukládána do předem připravených tabulek. Existují metodologie návrhu těchto tabulek tak, aby se omezil výskyt redundantních dat a byla zachována vysoká úroveň bezpečnosti. Použití klasické relační databáze je vhodné hlavně v případě, že je nutné udržovat data v bezpečí a kdy je tvar ukládaných dat znám předem a není tedy nutné provádět komplikované změny později.

  Příkladem SQL databázových systémů mohou být SQLite, MySQL a PostgreSQL.

- **Nerelační databáze** (*NoSQL*) jsou poněkud novějším konceptem. Jejich hlavním cílem je zbavit uživatele veškerých omezení vyplývajících ze striktního strukturování používaného v SQL a nabídnout ještě větší flexibilitu při ukládání dat. V NoSQL databázi jsou data ukládána podle potřeby pomocí dvojic klíč - hodnota. Neexistuje možnost vytvořit něco jako předpis tabulky, je možné ukládat libovolná data. Díky jednoduchosti tohoto přístupu je zapisování i čtení dat rychlejší než v SQL, díky čemuž je možné dosáhnout u NoSQL horizontální škálovatelnosti, což je v cloudovém prostředí vhodné. Slabinou NoSQL je možnost výskytu redundantních dat díky volné struktuře jejich ukládání a také nižší úroveň bezpečnosti dat. NoSQL se vyplatí v případě, že je důležité dosáhnout co nejlepší škálovatelnosti, nebo když je pravděpodobné, že struktura ukládaných dat se bude s postupem času měnit.

  Příkladem NoSQL systémů jsou MongoDB, CouchDB nebo Redis.

Kapitola 3

Poskytovatelé PaaS

V dnešní době existuje mnoho poskytovatelů PaaS. Každý z nich se snaží oslovit co nejvíce potenciálních zákazníků, z toho logicky vyplývá, že se všichni snaží být o krok dál před konkurencí a nabízet něco jiného, než ostatní. V této kapitole se budeme věnovat několika významným poskytovatelům a rozdílům v nabídkách jejich služeb.

3.1 AppFog

AppFog[1] nabízí programovací jazyky Java, Python, Node.js, PHP, Ruby a další. Umožňuje použití SQL i NoSQL databázových systémů, například MySQL, PostgreSQL nebo MongoDB. Další služby mohou být přidány pomocí add-onů. Pro práci s platformou nabízí jak grafické uživatelské rozhraní, tak možnost použití příkazové řádky. Doručení zdrojových kódů je možné pomocí verzovacích systémů, jako je Git, SVN nebo Mercurial.

Uživatel má možnost vybrat si poskytovatele infrastruktury, na které poběží jeho aplikace. Může si vybrat mezi Evropským, Americkým nebo Asijským AWS[1], nebo CenturyLink Cloudem.

Ceny služeb AppFog se odvíjejí především od množství pronajímané RAM a počtu běžících aplikací, v rozmezí $20 až $720 měsíčne. Existuje možnost využití 30ti denní zkušební verze, ve které má uživatel k dispozici 2GB RAM a až 200MB na databázi.

3.2 dotCloud

DotCloud[4] je na trhu PaaS relativně novým poskytovatelem. Nabízí uživateli možnost vybrat si z široké nabídky služeb ty, o které má zájem, a z nich poskládat výslednou platformu. Podporované programovací jazyky jsou PHP, Node.js, Python, Ruby, Perl, Java a další. Je možné využít databázových systémů MySQL, PostgreSQL, MongoDB či Redis.

Cena služby je založena na zátěži RAM, která se vyhodnocuje každou hodinu. Měsíční poplatek vychází přibližně na $4.32 a více, paměťový prostor je započítán v ceně. Větším zákazníkům nebo těm se speciálními požadavky nabízí dotCloud možnost dohodnout se na vlastní výší poplatků. Je možné připlatit si za přednost zpracování u technické podpory.

[1] Amazon Web Services

3.3 Engine Yard

Engine Yard[6] umožňuje použití programovacích jazyků PHP, Ruby on Rails a Node.js, je možné využívat mnoho databázových systémů, jako je například MySQL nebo Sphinx, základní služby je možné rozšířit pomocí velkého množství add-onů.

Nabízené prostředí fyzicky běží na infrastruktuře AWS nebo Microsoft Azure[3]. Engine Yard zodpovídá za monitorování a správu prostředí a komponent platformy jako je operační systém, databáze, nebo rozhraní pro komunikaci jádra systému s add-ony rozšiřujícími základní služby. Engine Yard udržuje vlastní optimalizovanou linuxovou distribuci, která běží na každé pronajímané instanci.

V základním plánu je cena $0.04 za hodinu za tři servery, nebo $29.00 za měsíc za server. Pro vyzkoušení je možné využít zdarma prvních 500 instančních hodin.

3.4 Google App Engine

Google App Engine[7] podporuje programovací jazyky jako je Python, Java, PHP a vlastí jazyk společnosti Google zvaný Go nebo též Golang. Umožňuje vývoj aplikací v lokálním prostředím pomocí SDK specifických pro vybraný jazyk. Je možné využít různé SQL i NoSQL databázové systémy, pro uložení velkých datových objektů je možné použít Google Cloud Storage.

Programy nabízené platformy běží v tzv. "sandboxu", který je logicky izoluje od ostatních aplikací, operačního systému a používaného hardwaru. Tento model zaručuje vysokou bezpečnost celé platformy.

Někteří poskytovatelé PaaS nabízejí uživateli i některé funkce IaaS, aby bylo možné ještě lépe si přizpůsobit placené služby, Google App Engine bohužel není jedním z nich. Uživateli je umožněno soustředit se plně na kód, ale je nutné, aby se do jisté míry přizpůsobil.

Google nabízí jisté množství poskytovených služeb denně zdarma, placení je tedy nutné až při překročení těchto limitů.

K volnému použití je nabínuto až 28 instančních hodin, 50 tisíc čtení nebo zápisů do databáze, omezené na velikost 1GB. Dále je omezen příchozí a odchozí síťový provoz, oba na 1GB denně.

Po překročení některého ze zmíněných limitů stojí jedna instanční hodina $0.05. Cena za databázi se určuje podle počtu operací, $0.06 za 100 tisíc čtení nebo zápisů v databázi, za každý uložený GB navíc se platí dalších $0.18 měsíčne. Odchozí síťový provoz je zpoplatněn $0.12 za každý odchozí GB dat.

3.5 Heroku

Heroku[8] nabízí v základní výbavě Ruby, Node.js, Python, Javu, a PHP. Služby je možné za příplatek rozšířit, například o podporu vybraného databázového systému, pomocí bohatého výběru add-onů. Každý uživatelem vybraný příkaz běží v izolovaných linuxových kontejnerech zvaných "dynos"[5]. Dynos mohou být různých typů a velikostí.

- **Web Dynos** - používané pro procesy pracující s HTTP komunikací
- **Worker Dynos** - jakékoliv procesy, jiné než web dynos, většinou běžící na pozadí aplikace
- **One-off Dynos** - dočasné dynos, jejichž vstup i výstup je propojen s terminálem

Měsíční poplatky za využívání služeb se mění podle výkonosti a počtu dynos, velikosti a úrovně bezpečnosti používané databáze, a podle úrovně technické podpory.

Cena za jedno dyno se pohybuje v rozmezí $0.05 až $0.80 za hodinu, podle jeho výkonosti. Cena databáze dosahuje až $30000 měsíčně, v případě využívání prémiových služeb nabízejících například až měsíční rollback.

Zdarma je možné využívat jedno dyno nejmenší výkonosti a databázi obsahující až 10 tisíc záznamů. Standardní technická podpora je pro všechny uživatele také zdarma.

3.6 Microsoft Azure

Windows Azure[3] je odlišný v tom, že poskytuje kombinaci IaaS a PaaS. Uživatel tak má možnost kustomizovat si služby, za které platí, tak, aby mu vyhovovaly ve všech ohledech. Důležitým cílem Microsoft Azure je flexibilita, je tedy možné používat jak Linuxové architektury, tak Windows, SQL i NoSQL databáze a řadu programovacích jazyků včetně .NET, Node.js, PHP, Pythonu nebo Javy, které jsou doplněny o přístupné SDK. Pro tvorbu a doručování aplikací je možné využívat Visual Studio nebo příkazovou řádku. Uživatel může data ukládat v tzv. blobech[2].

Windows Azue nabízí hostování až deseti jednoduchých webových aplikací zdarma, s omezením na 1GB zabírané paměti a maximálně 165MB přenesených dat denně. Používání procesoru je omezeno na 60minut na instanci denně.

Podle úrovně pronajatých služeb se ceny pohybují od necelých $10 do $74 za měsíc za jedno jádro procesoru.

3.7 RedHat OpenShift

RedHat OpenShift[12] poskytuje veřejný cloud, který umožňuje vývoj i hosting aplikací, a automatizuje jejich správu a škálování. Nabízené jazyky jsou Java, PHP, Python, Perl, Node.js, Ruby a velké množství souvisejících vývojářských nástrojů a technologií. Je možné použít databázové systémy jako je MongoDB, MySQL nebo PostgreSQL. Hlavní cíle jsou urychlení nasazení aplikace, široký výběr technologií a používání standardizovaných komponent pro umožnění přenositelnosti aplikací. Při práci je možné používat jak integrovaného vývojového prostředí, tak příkazové řádky. Zdrojové kódy je možné doručit jednoduše pomocí verzovacího systému Git.

Prémioví uživatelé mají k dispozici odborně vzdělanou technickou podporu, neplatící uživatelé mohou využít služeb komunitní podpory. Pro všechny uživatele je přístupné velké množství návodů zdarma.

Nabízené služby jsou uspořádány v abstraktní struktuře[9] obsahující instance operačních systémů zvané "nodes", pod které patří "gears". Ty sjednocují jednu nebo více tzv. "cartridges", představujících konkrétní nástroje pro používání vybraného programovacího jazyka nebo například pro správu databázového systému.

Ceny se vypočítávají podle používané výbavy a využívaného paměťového prostoru. Je možné zdarma používat až tři gears a 1GB paměti zdarma.

Za každý další gear nad limit se platí $0.02 za hodinu. Při překročení limitu paměti se platí měsíčně $1.00 za 1GB.

[2] Binary Large Object

Kapitola 4

Návrh aplikace

Tvorba internetového herního portálu představuje komplexní problém. Je nutné zaměřit se současně na mnoho různých úkolů, počínaje komunikací po internetu, přes programování umělé inteligence počítačových protihráčů, až po zobrazování uživatelského obsahu v klientské aplikaci. V této kapitole jsou informace o architektuře aplikace, uživatelském rozhraní, pravidlech implementovaných her a o výběru vhodných implementačních technologií.

4.1 Návrh uživatelského rozhraní

Herní portál bude implementován formou webové stránky přístupné pomocí libovolného internetového prohlížeče. Přístup do systému bude umožněn více způsoby. Registrovaní uživatelé mohou vstoupit pomocí svého uživatelského jména a hesla. Noví uživatelé mají dvě možnosti. První možností je jednoduchá registrace, ověřená pomocí emailové adresy, za kterou následuje výše zmíněná autentizace. Druhou možností je vstup do systému jako host, s automaticky vygenerovaným jménem, které bude platné jen tak dlouho, dokud bude uživatel přihlášen.

V další fázi se uživatel dostane do hlavního menu herního portálu. V horní části budou zobrazovány aktuality. Napravo se zobrazí informace o uživateli a možnost personalizovat si nastavení. Pod zobrazenými aktualitami budou zobrazeny ikonky dostupných her, kliknutím na jednu z nich se zobrazí detail, umožňující různé způsoby zahájení dané hry. V základní verzi budou implementovány hry lodě a tanky.

Po výběru konkrétní hry uživatel uvidí seznam aktuálně vytvořených místostí, do kterých se bude moci připojit. V případě vstoupení do místnosti bude muset počkat, dokud kapitán (zakladatel místnosti) nespustí hru. Kromě vstupu do cizí místnosti má hráč možnost vytvořit svou vlastní, nastavit parametry hry podle sebe, přidat počítačového protihráče, a nebo místnost uzamknout heslem, tak, aby se k němu mohli připojit pouze jeho známí. Poslední možností bude využití rychlého vyhledávání zápasu, které započne hru
v okamžiku, kdy bude současně vyhledávat dostatečný počet hráčů, hledajících stejnou hru se stejnými parametry.

4.2 Obecná architektura cloudové aplikace

Všechny cloudové aplikace nějakým způsobem využívají komunikaci po internetu. Z toho důvodu je nutné, aby celková aplikace sestávala minimálně ze dvou hlavních částí. První z

nich, zvaný klient (v terminologii cloudu se jedná o *frontendovou* část aplikace), bude program, běžící na straně uživatele. Druhou částí je server (*backendová* část aplikace), což je nejdůležitější část aplikace, běžící na straně poskytovatele. Server zodpovídá za vykonávání služby, pro kterou je určený, takovým způsobem, že reaguje na žádosti zaslané
od klienta. Činnost klienta spočívá převážně v získávání uživatelského vstupu, zasílání příkazů na server, a zobrazování získaných odpovědí. Z této obecné architektury je možné vyvodit základní věci, jež je nutné vybrat před zahájením samotného vývoje. Základní rozhodování se tedy musí týkat přinejmenším implementačního jazyka serveru a implementačního jazyka klienta. Vzhledem k nutnosti zasílání zpráv mezi klientem a serverem je nutné zvolit též vhodný způsob komunikace. Výběru technologií se bude věnovat následující kapitola.

4.3 Výběr techologií

Výběr technologií je často tou nejdůležitější součástí tvorby jakékoliv aplikace a je úzce spojen s jejím návrhem. V případě pochybení ve fázi návrhu čeká na programátora nemalé množství problémů, které mohou zpomalit nebo dokonce i znemožnit dokončení práce.

Při výběru technologií jsem se rozhodoval na základě mnoha faktorů. Mezi nimi byl například požadavek na modularitu aplikace nebo jednoduchost její implementace, největší vliv na rozhodování ale jednoznačně měla potřeba dosáhnout dobré škálovatelnosti řešení.

Výběr implementačního jazyka byl v podstatě soubojem mezi některými z aktuálně nejpoužívanějších jazyků při psaní klient-server aplikací. Nejzajímavějšími porovnávanými jazyky byly PHP a Javascript. Oba jazyky nabízejí mnoho frameworků, ať už obecnějšího ražení, nebo těch specializovaných pro konkrétní účel. Oba dosahují dobré rychlosti při zpracovávání požadavků a je možné je použít při tvorbě multiplatformních aplikací. Největší výhodou PHP je uživatelská přívětivost a díky velkému množství vestavěných funkcí rychlá tvorba téměř jakékoliv aplikace. Velkou nevýhodou jsou omezené možnosti asynchronního zpracování a také omezení PHP na serverovou část aplikace.

Na druhé straně je Javascript, který nemá tak širokou základnu funkcí jako PHP. Výhodou je možnost v něm psát serverovou i klientskou část projektu, není tedy nutné ovládat další jazyk a výsledná práce bude jednotnější. Použití frameworku Node.js zjednodušuje tvorbu programu a jeho vykonávání probíhá asynchronně. Vzhledem k výše zmíněným výhodám jsem tedy zvolil jako implementační jazyk Javascript. Jako základní nadstavbu jsem zvolil již zmíněný Node.js. Mimo Node.js je možné využít dalších frameworků vyšší úrovně, jako jsou například Pomelo nebo Sails. Pro účely této práce mi ale všechny připadaly zbytečně robustní, proto jsem se rozhodl využít plně dostačující a hojně používanou knihovnu pro obousměrnou komunikaci klient-server, zvanou *socket.io*[13].

Další rozhodování se týkalo využití relační nebo nerelační databáze. SQL umožňuje systematické ukládání dat a zajišťují maximální bezpečnost. NoSQL nabízí programátorovi velkou svobodu při práci s daty, pro dosažení obdobné úrovni bezpečnosti jakou nabízí SQL je vyžadováno mnohem větší množství práce. Pro účely herního portálu ovšem není bezpečnost až tolik důležitým argumentem. Pokud nedochází ke sdílení tajných informací nebo třeba k transakcím skutečných peněz, je lepší rozhodovat se na základě jiných informací.
I v tomto případě tedy rozhodl opět faktor rychlosti. Výhodněji z tohoto srovnání nakonec vyšly NoSQL databáze především díky výborné škálovatelnosti.

Jelikož téměř všechny moderní prohlížeče podporují Javascript, rozhodl jsem se ho použít i pro implementaci částí aplikace, které poběží na straně klienta. Pro zobrazování uži-

vatelského obsahu jsem zvolil dnes nejvíce používané HTML5 v kombinaci s CSS, včetně některých částí frameworku Bootstrap[2]. Pro vykreslování samotných her je možné využít jeden z prvků HTML5, značku canvas.

Během vybírání jednotlivých technologií použitých pro implementaci jsem současně přihlížel k možnostem, jež nabízejí různí poskytovatelé PaaS. Požadavkům vybraným výše vyhovoval nejvíce projekt OpenShift Online firmy RedHat a cloud posktovaný společností AppFog, oba poskytující Node.js a zabudovanou NoSQL databázi MongoDB, a umožňující pohodlný přístup pomocí verzovacího systému Git. Vzhledem k ceně služby a odlišnému systému poskytování volné PaaS, jsem nakonec zvolil pro implementaci portálu RedHat OpenShift.

4.4 Architektura portálu

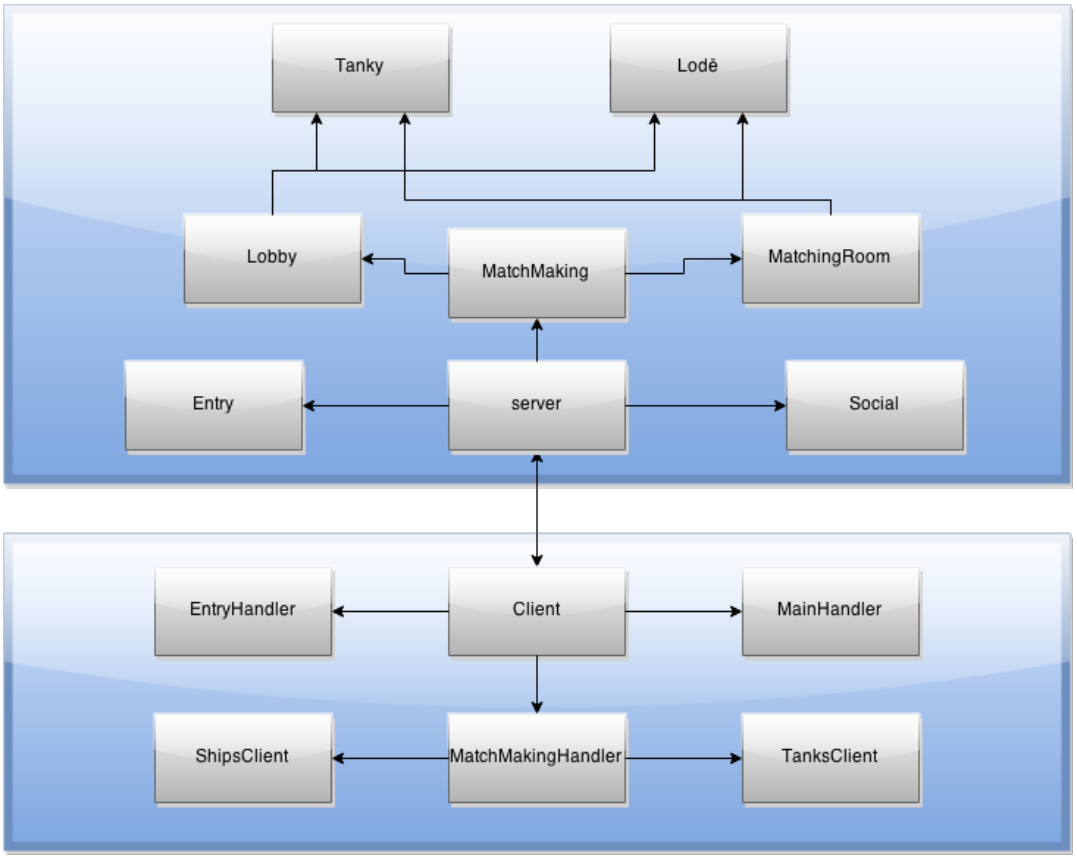
Stejně jako je tomu v obecném konceptu cloudové aplikace se bude i herní portál skládat z klientské a serverové části. Server poběží na cloudu a bude obstarávat úkoly, které jsou nejdůležitější, a zároveň potřebují největší míru zabezpečení. Hlavní funkcí tedy bude komunikace s klientskou částí aplikace, ověřování uživatelů, řízení herní logiky, a práce s databází, ve které budou uloženy informace o uživatelích a údaje pro výpočty statistických informací o hrách.

Server bude rozdělen na několik menších modulů. První z nich bude sjednocovat zprávy zaslané klientem. Podle jejich typu a případných parametrů budou volány moduly nižší úrovně, zodpovídající za příslušející akce. Mezi těmito moduly bude jeden modul řídící přihlašování a odhlašování uživatelů, druhý, odpovídající za komunikaci s ostatními hráči a systém přátel a třetí, sjednocující operace určené pro správu her. Pro každou z her bude vytvořeno také po jednom modulu.

Přijímání zpráv v klientské části aplikace bude sjednoceno v jednom modulu, který bude volat funkce z ostatních součástí, obdobně jako na serveru. Ostatní moduly budou rozděleny především v závislosti na oblasti aplikace, kterou ovládají, v klientské části bude toto dělení založeno především na zobrazování různých částí portálu.

Komunikace mezi serverem a klientem bude probíhat pomocí již zmíněné knihovny - socket.io. Zprávy budou rozlišovány pomocí určených řetězců a parametrů zaslaných v datové části zprávy.

Architektura aplikace je abstraktně znázorněna níže na obrázku číslo 4.1.



4.5 Škálování aplikace

Škálovatelnost je jedním z hlavních kritérií hodnocení cloudových služeb. Aplikace, jejíž návrh je zaměřený na kvalitní škálování, umožňuje obsloužení většího počet uživatelů, než by obyčejně bylo možné. V důsledku škálování šetří náklady, především na straně poskytovatele služby. Většina poskytovatelů PaaS nabízí využití automatického škálování hostované aplikace. V této sekci popíšeme, jaké možnosti automatického škálování nabízí RedHat Openshift a jakým způsobem tyto možnosti ovlivňují návrh aplikace.

4.5.1 Škálování v RedHat OpenShift

Jak již bylo zmíněno v druhé kapitole, základní výkonovou jednotkou v Openshiftu je "gear". Poskytované nástroje, zvané "cartridge", zabírají určitý počet gearů. Naše aplikace bude omezena na tři malé gears, jelikož budeme využívat pouze bezplatné služby Openshiftu. Jeden z těchto gearů bude zabrán pro databázi - MongoDB, zbývající dva tedy budou využity pro běh aplikace. Samotné škálování je pak založeno na využití vyrovnávače zátěže, zvaného HAProxy. HAProxy slouží jako brána, která přijímá veškeré příchozí požadavky, a rozhoduje, jak s nimi bude dále zacházeno. V našem konkrétní případě bude po spuštění aplikace aktivní pouze jeden aplikační gear. V okamžiku, kdy bude dosaženo stanovené úrovně vytížení na prvním gearu, bude aplikace naklonována na další volný gear a spuštěna. HAProxy poté bude přicházející žádosti spravedlivě rozdělovat mezi tyto dva geary, podle úrovně jejich vytížení. V okamžiku, kdy dojde k opětovnému poklesu počtu požadavků, které způsobí snížení vytíženosti gearů pod danou mez, dojde k přerušení činnosti

jednoho z gearů. Tímto způsobem je umožněno šetřit výkon v okamžicích nízké poptávky, a zároveň zajistit dostupnost služby v případě, že bude poptávka naopak vysoká. Můžete si povšimnout, že klonování se týká pouze aplikačních gearů. Databáze zůstává ve všech případech jen jedna, a všechny geary se tedy připojují ke stejnému společnému zdroji.

4.5.2 Důsledky automatického škálování

Uživatel PaaS má stále velký vliv na výslednou škálovatelnost, i přesto, že aplikace nemusí řešit úkony spojené se škálováním, jako je například monitorování zatížení nebo přepojování klientů mezi servery. Jeho hlavní přínos pro úroveň škálovatelnosti bude ve většině případů především optimalizačního rázu. Aplikace, která šetří zdroje všude, kde to je možné, bude logicky lépe škálovatelná, než ta, jejíž implementace obsahuje velké množství nadbytečných věcí.

Mezi operace, které zabírají nejvíce času, patří hlavně operace nad databází. Databáze herního portálu v základní verzi bude vyžadovat pouze ukládání informací o registrovaných uživatelích, a počet přístupů k těmto datům nebude příliš vysoký, proto se nepředpokládá velký vliv databáze na škálovatelnost systému.

Naopak je tomu u modulů zodpovídajících za jednotlivé hry. Zaprvé budou všechny hry vyžadovat zasílání velkého množství zpráv mezi klientem a serverem, zadruhé je potřeba počítat s množstvím dat, které je nutné si pamatovat, pro udržení kontextu každé instance hry, zatřetí je nutné odhadnout, jaký vliv na běh aplikace budou mít výpočty související se hrami, ať už jde o výpočty trajektorie střely nebo o výpočty odpovídající za chování počítačových protihráčů. V průběhu implementace her bude tedy velký prostor pro optimalizaci a tím pádem i pro zlepšení škálovatelnosti celku.

4.6 Lodě

Jedná se o tahovou strategickou hru pro dva hráče, též je možné hrát proti počítačovému protihráči. Na začátku hráči dostanou možnost uspořádat dané množství různých lodí na svou část moře rozdělenou na 30x20 polí, kromě manuálního rozestavení lodí mohou využít též náhodného rozdělení. Každá loď zabírá tři pole a je položena buď horizontálně nebo vertikálně. Hráč nevidí jakým způsobem jsou uspořádané lodě protihráče.

Hráči v každém kole mají možnost vystřelit na vybrané pole nepřátelského území tolikrát, kolik vlastní nepotopených lodí, výběr polí ke střelbě bude časově omezený. Podle efektu hráč pozná, zda zasáhl nepřátelskou loď nebo pouze moře. Vybrané pole je na zbytek souboje označeno. Loď je potopena v případě, že byly zasaženy všechny pole, na kterých se nachází. Hráč je poražen a vypadává ze hry, když jsou potopeny všechny jeho lodě.

Lodě nabízejí rychlé hledání hry nebo založení vlastní místnosti, pro dva hráče. V případě založení místnosti přibývá možnost hrát proti počítačovému protihráči.

Hra bude ovládaná čistě klávesnicí, pomocí kurzoru, pohybujícího se po polích mapy je možné nastavit pozici lodě, nebo určit cíl střelby, a akci potvrdit pomocí mezerníku. Možnost náhodného rozestavení lodí je dostupná pomocí tlačítka *náhodně*, nacházejícího se přímo nad herní mapou. Pro jeho aktivování je nutné vyjet kurzorem z mapy, směrem k tlačítku, a potvrdit akci mezerníkem.

V okamžiku kdy je hra u konce (všechny lodě jednoho z hráčů jsou potopeny), se hráčům zobrazí výsledek kola, a poté možnost odvetného zápasu, která je aktivována pomocí stisknutí mezerníku.

4.7 Tanky

Jedná se o tahovou hru pro více hráčů, která bude probíhat v dvojrozměrném vygenerovaném terénu, na kterém se budou pohybovat tanky dvou nepřátelských stran. Ke každé straně bude patřit jeden nebo více hráčů.

Hráč, jež bude zrovna na tahu bude mít omezený čas na pohyb po mapě, nastavení úhlu a síly střelby, a také samotné vystřelení projektilu. Při zasažení terénu dojde k jeho porušení, při zasažení tanku dojde k snížení počtu jeho životů, tank bez životů je zničen a vyřazen ze hry.

Rychlé vyhledávání ve hře Tanky bude nabízet vyhledávání týmu o jednom nebo dvou hráčích. V místnosti bude možnost nastavení velikosti týmů na jednoho až tři hráče, přičemž počet hráčů v týmech nebude muset být stejný.

Ovládání hry bude, podobně jako u jiných verzí této hry, závislé jen na klávesnici. Pohyb po mapě bude možný pomocí tlačítek vlevo a vpravo. Nastavení úhlu střelby pomocí tlačítek nahoru a dolů. Síla střelby odpovídá tomu, jak dlouho uživatel podrží mezerník, při jeho uvolnění dojde k výstřelu.

V prvním tahu bude hrát jeden hráč prvního týmu, poté první hráč druhého týmu, pak druhý hráč prvního týmu, a tak dále. Cílem týmu je zničit všechny nepřátelské tanky.
V okamžiku kdy je jeden z týmů poražen je hráčům zobrazen výsledek kola, a posléze možnost hrát odvetný zápas, pomocí stisknutí mezerníku.

Kapitola 5

Implementace

Kapitola implementace se bude věnovat technickému řešení herního portálu. Hlavním cílem bude podchycení nejdůležitějších částí implementace aplikace, jako je třeba řešení komunikace mezi klientem a serverem, nebo algoritmy využité při tvorbě logiky her.

5.1 Zahájení komunikace

Jak již bylo zmíněno v předešlých kapitolách, jedná se o webovou aplikaci přístupnou z libovolného internetového prohlížeče, podporujícího Javascript. Celá aplikace se skládá z klientské a serverové části. Serverová část je samostatná aplikace běžící na straně poskytovatele PaaS, ke které se přes prohlížeč připojují jednotliví uživatelé. Připojenému uživateli je na začátku komunikace zaslán, pomocí **HTTP** požadavku **GET**, soubor **index.html**, včetně formátovacích souborů **.css** a klientského Javascriptu, který bude řídit následující komunikaci se serverem.

5.2 Komunikace klient-server

Komunikace klienta se serverem je implementována pomocí knihovny **socket.io**. Na straně klienta i serveru je vytvořen **socket** - schránka obsahující informace o propojených objektech. Pomocí těchto schránek je možné zasílat zprávy na server i z něj. Z pohledu aplikace každá zpráva obsahuje dvě části. První z nich je řetězec, sloužící především k tomu, aby bylo možné rozeznat o jaký typ zprávy se jedná. Typ zprávy volí programátor dle vlastního uvážení. Podmínkou je, aby cílová schránka dokázala daný typ zprávy rozpoznat a zařídit se podle něj. Druhou částí zprávy je libovolný Javascriptový objekt, umožňující předání dat. Následující ukázka ukazuje, jakým způsobem je možné zaslat resp. přijmout zprávu pomocí socket.io:

```
//server očekává na schránce zprávu typu "ping"
socket.on("ping",function(data){
  //při jejím doručení odpoví klientovi zprávou typu "pong"
  io.to(socket.id).emit("pong",data);
});
```

Předchozí ukázka by pracovala na serveru, kde **io** je objekt umožňující komunikaci na straně serveru takovým způsobem, že k odeslání zprávy danému klientovi postačí pouze

identifikátor jeho schránky. Na straně klienta existuje pouze jediná schránka, k odeslání zprávy tedy stačí použít socket.emit(..).

Kvůli usnadnění komunikace mezi serverem a skupinou uživatelů jsou v objektech her implementovány funkce, určené pro zasílání hromadných zpráv všem zahrnutým hráčům - **notifyRoom**, nebo jen jednomu konkrétnímu týmu - **notifyTeam**.

Jak na straně serveru (**server.js**), tak na straně klienta (**client.js**), existuje modul, který zodpovídá za přijímání zpráv ze schránek a vyvolání odpovídajících akcí. Typy přijímaných zpráv odpovídají, pokud je to možné, rovnou celé skupině podobných akcí. Tento přístup sice vyžaduje sekundární rozbor zprávy, kvůli rozpoznání parametrů umístěných v její datové části, ale umožňuje snadné a rychlé napojování nových modulů, obzvláště
v případě jednotlivých her. Deatily o tomto přístupu budou zníněny v sekci *Řízení herních událostí.*

5.3 Správa uživatelů a jejich dat

Pro každého uživatele existuje odpovídající položka v databázi, obsahující informace o jím zadaných přihlašovacích údajích, přátelích, osobních nastaveních, herních úspěších a také název aktuální místnosti, ve které se nachází. Kvůli snaze sjednotit implementaci portálu jsou ukládány do databáze i položky dočasných účtů (hostů), přestože jejich záznamy jsou po jejich odhlášení vymazány.

Hlavním modulem pracujícím s databází je **entry.js**. Tento modul zodpovídá za registraci a přihlašování hráčů, generování a přihlašování hostů, a také úkony společně
s opuštěním portálu, týkající se všech druhů hráčů. Po přihlášení je do záznamu uživatele nahrána informace o jeho stavu (přihlášen) a o identifikátoru jeho schránky, díky kterému je později možné zasílat uživateli zprávy i v jiné souvislosti, než jen jako odpověď na jeho žádost. Při odhlášení uživatele je nutné nejen přenastavit stav v jeho záznamu, ale také zajistit, aby všechny části serveru, jakkoliv související s tímto hráčem, byly o jeho odhlášení informovány. Jedná se například o jeho automatické odhlášení z místnosti nebo ze hry. Řešení tohoto problému bude popsáno v následující sekci.

Při generování jmen hostů je nutné využít komunikaci s klientem. Na žádost klienta server vygeneruje nové jméno hosta. Klient se následně automaticky pokusí pod zaslaným jménem přihlásit, na straně serveru dojde k ověření, že vygenerované jméno není uložené v databázi. Je-li to možné, klient se přihlásí do portálu, v opačném případě musí požádat server o vygenerování nového jména a celý proces se opakuje. Vygenerování unikátního jména na poprvé není možné, vzhledem k asynchronnímu vyhodnocování Javascriptových příkazů. Kdyby se server snažil, například v cyklu *while*, generovat jméno tak dlouho, než výsledek nenalezne v databázi, došlo by k zacyklení celého serveru. Tento problém je způsobem tím, že než by se vyhodnotila podmínka, vylučující výskyt jména v databázi, proběhlo by mezitím generování dalších jmen a čekalo se na vyhodnocení dalších podmínek atd..

Dalším modulem vyžadujícím častou spolupráci s databází je **social.js**. Ten implementuje vyhledávání uživatelů, systém přátel pro registrované uživatele, nebo například chat. V databázi je nutné pamatovat si kromě seznamu přátel též seznam nových žádostí o přátelství.

Ostatní moduly s databází pracují pouze zřídka, pouze v případě, že je nutné zjistit například identifikátor schránky nebo jméno místnosti, ve které se uživatel nachází.

5.4 Řízení herních událostí

Řízení herních událostí je implementováno takovým způsobem, aby bylo umožněno co nejjednodušší napojení nových her do portálu. Z tohoto důvodu nejsou využívané samostatné typy zpráv pro jednotlivé hry, místo nich existuje omezené množství obecných typů zpráv, vyvolávajících obdobné akce v různých hrách. Mezi tyto typy zpráv patří *find_game, join_lobby, leave_room, game_start* a *game_update*.

Při doručení zprávy tohoto typu je zavolána odpovídající abstraktní funkce z modulu **matchmaking.js**. Úkolem těchto funkcí je analyzovat parametry, uložené v hlavičce datové části zprávy, a na jejich základě rozhodnout o konkrétní akci, která bude provedena. Při instalování nové hry do systému je tedy nutné vytvořit jen specifickou funkcionalitu podle předepsané architektury, tím pádem odpadá opětovné "vynalézání kola", v podobě implementování již existující obecné funkcionality, týkající se především propojení jednotlivých modulů.

Konkrétní vyvolané akce se týkají především dvou problémů. Prvním je vyhledávání zápasů a správa místností vytvářených uživateli. Druhým je už samotné spuštění hry a kontrolování jejího dalšího průběhu.

5.5 Správa místností

Jak hráče, kteří využívají rychlé vyhledávání hry, tak ty, kteří si založili místnost, je nutné v serverové reprezentaci nějakým způsobem spojit do skupin. K tomuto slouží virtuální místnosti implementované v modulech zvaných **matchingRoom.js** a **lobby.js**. Implementace obou modulů má mnoho společných prvků, přestože jejich funkcionalita, nejen tak jak ji vidí koncový uživatel, je velice odlišná.

Pro oba druhy místností platí, že se jedná o objekt, obsahující informace o parametrech hledané hry a seznam aktuálně přítomných hráčů. Dalším společným prvkem je, že oba moduly v sobě ukládají seznam všech aktuálně existujících místností, včetně funkcí umožňujících manipulaci s nimi, jako je možnost vyhledání nebo smazání místnosti podle jména. Poslední společnou věcí je vlastnictví metod pro připojení popř. odpojení z místnosti, funkcionalita těchto metod je ale první rozdílnou věcí v implementaci vytváření místností, oproti implementaci vyhledávání rychlého zápasu.

5.5.1 Rychlé vyhledávání hry

Nejdůležitější částí modulu **matchingRoom.js** je funkce **findMatchingRoom**. Tato funkce přijímá jako parametr žádost o hru, což je objekt obsahující informace o tom, o kterou hru se jedná, jaké jsou její parametry, jako je například počet hráčů, časový limit nebo další specifické vlastnosti konkrétní hry, a vrací, v případě že existuje, místnost, která přesně odpovídá požadavkům žádosti a zároveň stále není plná, takže je možné se do ní stále připojit.

Proces rychlého vyhledání hry tedy probíhá tak, že napřed dojde k prohledání aktuálně existujících vyhledávacích místností. V případě, že dojde ke shodě, je uživateli, od něhož pochází žádost, umožněno vstoupit do místnosti. V opačném případě je nutné založit novou místnost s danými parametry.

V rámci metody pro vstup do místnosti je implementována kontrola na plnost místnosti. V případě, že je místnost po vstupu uživatele plná, dojde k ukončení vyhledávání

a zúčastěným hráčům je rozeslána informace o nalezení hry. Poté, co všichni hráči potvrdí, že jsou připravení, dojde ke spuštění hry.

Metoda pro odhlášení z místnosti zajišťuje, kromě samotného odhlášení, také zachování rovnoměrného rozdělení hráčů mezi týmy.

5.5.2 Zakládání místností

V modulu **lobby.js** je nutné řešit záležitosti související s rozestavením hráčů v týmu, připojením počítačových protihráčů a dalších pravomocí kapitána místnosti, jako je například vyhození jiného hráče.

Je nutné zajistit, aby v každé místnosti byl stále jeden kapitán. Může se totiž stát, že kapitán se z místnosti odpojí, ale stále jsou v ní připojení ostatní hráči. V okamžiku, kdy dojde k odhlášení hráče, proto probíhá kontrola, zda odhlášený hráč nebyl kapitánem. V případě, že tomu tak bylo, je kapitánem jnenovám jeden ze zbývajících uživatelů.

5.6 Implemetace her

V této sekci budou zmíněny informace o průběhu a fungování her, počínaje jejich spuštěním, konče technickými detaily o chování počítačových protihráčů.

5.6.1 Spuštění hry

Požadavkem pro spuštění hry je existující místnost, vytvořená uživatelem, nebo automaticky kvůli rychlému vyhledávání, která splňuje specifické startovní podmínky. Samotné spuštění probíhá tak, že se vytvoří nová instance konkrétní hry. Tato instance při inicializaci nastaví svůj kontext podle dat uložených v místnosti - jedná se o nastavení parametrů hry a seznam připojených uživatelů. Ve své podstatě je hra instancí místnosti, rozšířenou o konkrétní funkcionalitu.

Dalším krokem je zavolání metody **restartGame** dané instance. V tomto okamžiku je řízení přenecháno modulu odpovídajícímu konkrétní hře. Detaily o jednotlivých modulech budou probrány níže. Posledním úkonem, společným pro všechny hry, je rozeslání počátečních informací o hře všem připojeným uživatelům.

5.6.2 Lodě

Koncept

Lodě byly implementovány minimalistickým způsobem ve stylu klasických *tilesetových* her. Hlavní myšlenkou, z nich převzatou, bylo rozdělení herní plochy na čtvercové části, zvané *tilesety*. Každý tileset je charakterizován svými vlastnostmi a vzhledem. Takový způsob implementace zvyšuje úroveň abstrakce a značně zjednodušuje práci na tvorbě hry. Modul zodpovídající za kontrolu lodí se nazývá **ships.js**.

Přesto, že se jedná o serverem kontrolovanou hru, není nutné, aby na něm běžela žádná herní smyčka. Lodě jsou totiž koncipovány jako čistě tahová hra, jejíž kontrolování pracuje na základě zpracování příchozích událostí.

Přehled akcí

Počáteční fáze hry probíhá na klientské části aplikace. Od hráčů se vyžaduje, aby rozdělili svoje lodě do hrací plochy. V okamžiku, kdy je jejich rozdělení kompletní, je na server

zaslán seznam lodí. Těla jednotlivých lodí se skládají z tří souřadnic herní plochy, jejíž části zabírají. Není tedy nutné, aby server obsahoval vlastní reprezentaci celé mapy.

V souboru **shipsPLayer.js** je implementována reprezentace hráče pro hru lodě. Mezi nejzásadnější položky, uložené v kontextu hráče, patří seznam jeho lodí, seznam vraků, obsahující souřadnice zasažených částí lodí nebo například informace o tom, zda se jedná o počítačového protihráče, nebo připojeného uživatele.

Nejčastější akcí ve hře je střílení. To probíhá následujícím způsobem: Uživatel zvolí souřadnici, na kterou chce vystřelit. Server se pokusí hodnotu dané souřadnice vyhledat, napřed v seznamu lodí nepřítele, poté v seznamu vraků. V případě, že došlo k zasažení lodě, následuje vymazání dané souřanice ze seznamu lodí a její uložení do seznamu vraků. V poslední fázi je uživateli zaslána informace o zasaženém poli. Druhému uživateli je zaslána aktualizace také, aby i on mohl sledovat, kam jeho nepřítel mířil, a jak daleko od skutečné polohy lodí střela dopadla.

Kontrola stavu hry

Vzhledem k tomu, že se jedná o tahovou hru, je nutné, aby se hráči střídali. Implementaci tohoto chování zajišťují metoda **chooseActivePlayer**, využívaná v případě vypršení časového limitu, a **forceChoosePlayer**, která ukončí tah hráče v okamžiku, kdy mu nezbývá žádný další výstřel.

Metoda **checkGameState** zodpovídá, jak z názvu plyne, za kontrolování stavu hry. Metoda je volaná během zpracování událostí, jež mohou stav hry ovlivnit. Podle fáze hry jsou prováděny kontroly různých typů. V počáteční fázi je potřeba zjistit, kolika hráčům se již podařilo rozmístit svou flotilu, aby bylo možné přejít do fáze vlastního souboje.
Po každém zásahu lodě je nutné zkontrolovat počet zbývajících lodí uživatelů. Po poražení jednoho z hráčů se hra dostává do třetí fáze(čekání). V ní je předmětem kontroly počet hráčů připravených k odvětě.

Počítačový protihráč

Vždy, když je za aktivního hráče zvolen jeden z počítačem ovládaných hráčů (*botů*), je zavolána metoda **playAsBot**. Tato metoda využívá informace uložené v modelu ovládaného hráče k rozhodnutí, na které souřadnice vystřelit. V datovém kontextu bota musí být uloženy informace o již zasažených souřadnicích mapy a o souřadnicích, kde byly zasaženy nepřátelské lodě.

Bot se řídí třemi jednoduchými pravidly: Bot nemůže vystřelit dvakrát na stejné pole. V případě, že nevlastní ani jediný záznam o zásahu, probíhá míření čistě náhodně. Jinak vybere náhodný záznam o zásahu a vystřelí náhodným směrem do jeho okolí, přičemž v okamžiku, kdy do okolí záznamu již není možné vystřelit, je tento záznam ze seznamu vyřazen.

5.6.3 Tanky

Koncept

Tanky, implementované v modulu **tanks.js** sice obsahují jisté prvky tahových her, jako je střídání aktivních hráčů, ale přesto je nutné řešit rozesílání herních událostí v reálném čase. V případě, že se jeden z tanků pohybuje, je nutné aby tento pohyb viděli i ostatní uživatelé přítomní ve hře. Je tedy nutné aby na serveru běžela herní smyčka, provádějící pravidelné

akce a kontroly stavu. Všechny zásadní výpočty hry jsou prováděny na straně serveru. Důvodem k tomuto rozhodnutí bylo zjednodušení implementace počítačového protihráče a také snaha zajistit konzistentní stav hry pro všechny hráče.

Zprávy mezi klientem a serverem

Pro komunikaci jsou využívány dva základní druhy zpráv. Prvním z nich je aktualizace stavu tanků. V případě, že uživatel žádá pohyb tanku, nebo hlavně děla, je na server zaslána žádost o spuštění pohybu daným směrem. V okamžiku kdy uživatel uvolní klávesu, již k zaslání žádosti vyvolal, je naopak zaslána žádost o zastavení daného pohybu.
Na straně serveru je akce provedena, a klient obdrží pouze aktualizaci pozice nebo úhlu, tak aby mohl vše vykreslit. Poslední, aktualizace stavu se týká střídání aktivních hráčů, které je implementováno pomocí změn stavu proměnné *locked* jednotlivých tanků.

Druhou kategorií jsou aktualizace událostí. Mezi události, vyvolané ze strany klienta, patří žádost o vystřelení, nebo žádost o odvetu. Na straně serveru jsou tyto žádosti rozpoznány a jsou provedeny jimi vyvolané akce. V případě žádosti o střelu je nutné rozeslat všem klientům informaci o střele, aby ji mohli vykreslit pro uživatele. Dále je potřeba vypočítat místo dopadu projektilu a vytvořit kráter. V případě, že došlo k deformaci terénu, jsou klienti informování znovu, tentokrát pomocí události typu *mapa*, která provede aktualizaci zvolené části mapy.

Další události se týkají hlavně řízení hry, konkrétně jde o ukončení hry. Jedná se o události informující o výhře, nebo prohře daného uživatele.

Reprezentace mapy

Mapa v tancích je pro každou hru generována náhodně. Při generování je využita následující matematická funkce:

$$a * sin\left(\frac{x+b}{\frac{x+c}{d}}\right) + e \tag{5.1}$$

Kde proměnné *a*-*e* zastupují různé parametry terénu:

- **a** - stoupání
- **b** - horizontální posun
- **c** - šířka kopců
- **d** - rozlišnost kopců
- **e** - vertikální posun

Mapa je uložena v poli, obsahujícím na každé x-ové souřadnici výslednou hodnotu předešlé funkce, a to jak na straně serveru tak na straně klienta. Je to nutné kvůli deformacím způsobeným dopadem střel na povrch. Na začátku hry je mapa vygenerována na straně serveru, klientům jsou následně zaslány pouze hodnoty parametrů terénu, pomocí kterých si mapu mohou vygenerovat sami.

Reprezentace projektilu

Projektily vystřelené hráčem jsou reprezentovány pomocí podtřídy zvané **Projectile**. Při vytvoření projektilu je, z úhlu a síly střely, pomocí funkcí sinus a cosinus, vypočten horizontální a vertikální posun střely. Projektil obsahuje metodu **makeMove**, volanou v každém průběhu serverové herní smyčky. Tato metoda nastaví novou pozici projektilu přičtením hodnoty vertikálního posunu k souřadnici y a horizontální hodnoty posunu k souřadnici x. Zároveň je k vertikálnímu posunu přičtena hodnota *GRAVITY_PULL*, reprezentující gravitaci. Posledním důležitým úkolem metody je kontrola pozice projektilu v rámci mapy. Projektil je smazán v případě, že překročil její horizontální hranice. Pro detekování zásahu střelou se využívá porovnání hodnoty souřadnice y projektilu, s hodnotou souřadnice y odpovídající části mapy.

V okamžiku zasažení terénu je zavolána metoda **newCrater**, která vypočítá deformaci způsobenou střelou. Velikost poškození mapy je určena pomocí síly projektilu a rychlosti jeho klesání. Poškození v okolí dopadu je vypočteno opět pomocí funkce sinus, a hodnoty tohoto poškození jsou odečteny od odpovídajících hodnot mapy, a také od životů tanků, které se v oblasti nachází. Poškození udělené tanku je tedy maximální v případě, že se nachází přímo v místě dopadu, se vzdáleností od něj hodnota poškození rychle klesá.

Reprezentace tanku

Pro implementaci tanku byl vytvořen modul **tanksTank.js**, obsahující třídu **Tank**. Každá instance tanku musí udržovat v kontextu informace o pozici, úhlu děla, stavu, počtu životů a další. Instance reprezentující počítačem ovládaného hráče navíc musí obsahovat další proměnné, využívané při implementaci jeho chování.

Nejdůležitější metodou je metoda **tankMove**, volaná v každém průběhu herní smyčky. Zodpovídá zejména za pohyb tanku, jehož rychlost se liší v závislosti na strmosti terénu, pohyb děla a za správné fungování gravitace.

Mimo jiné je potřeba kontrolovat stav tanku. Tank, který spadne na spodní hranici mapy, nebo ten jehož počet životů klesne na nulu, je zamčen a vyřazen ze hry až do jejího konce.

Kontrola stavu hry

Podobně jako u hry lodě, i zde je nutné pravidelně střídat aktivní týmy a jednotlivé hráče v nich. Z tohoto důvodu existuje metoda **chooseActivePlayer**, která provede změnu aktivního týmu. Pomocí **chooseActivePlayerFromTeam** je vybrán konkrétní hráč z aktivního týmu. Kvůli tomuto výběru je nutné si pamatovat (na úrovni hry) pořadí naposledy zvolených hráčů daného týmu. Tyto indexy jsou při volbě hráče odpovídajícím způsobem inkrementovány, případně nulovány při výběru posledního hráče v týmu.

Metoda **checkGameState** pracuje také obdobně jako v lodích. Jejím úkolem je kontrola konkrétních věcí v závislosti na stavu hry. Rozdíl je v tom, že v tancích je kontrola volaná pravidelně herní smyčkou serveru.

Počítačový protihráč

Nejdůležitějším úkolem počítačového protihráče (bota) je míření na nepřítele. Implementace jeho chování se nachází v metodě **moveBots**, která je volaná serverovou smyčkou. Metoda sama rozezná, zda je aktivním hráčem bot, a zahájí výpočty nutné k řízení jeho akcí.

V prvním volání metody (v rámci jednoho tahu) je nutné vybrat náhodného nepřítele, určit vzdálenost, kterou bot ujede směrem v němu, a nakonec i úhel, pod jakým je nutné na nepřítele vystřelit. Do objektu reprezentujícího botům tank jsou uloženy vypočtené hodnoty jako cíle, kterých je nutno dosáhnout. Úhel střelby je před uložením ještě upraven, aby bylo hraní proti botům usnadněno. K úhlu je připočtena náhodná odchylka, jejíž rozsah závisí na vzdálenosti mezi botem a cílem.

V následujících voláních metody jsou již prováděny akce nutné k dosažení předem daných cílů. V první řadě bot provádí pohyb směrem k nepříteli, dokud nedojde k dosažení cílové pozice. Poté je nutné pohybovat dělem, dokud není dosaženo cílového úhlu střelby. Na konci je potřeba dosáhnout požadované síly výstřelu, obdobně, jako kdyby připojený uživatel držel mezerník, aby vystřelil silněji. V okamžiku, kdy dojde ke splnění všech určených cílů, bot vystřelí. Tento způsob implementace byl zvolen z důvodu, aby se hra řízená počítačem podobala hře uživatele.

5.7 Zobrazení uživatelského obsahu

Poslední sekce se bude věnovat záležitostem týkajících se čistě klientské části aplikace, konkrétně půjde o způsob zobrazování portálu. Jak bylo zmíněno v návrhu, pro zobrazování obsahu je využito kombinace *HTML5* a *css*. Soubory **index.html** a **main.html** jsou jedinými statickými zdroji. Ostatní obsah je generován dynamicky pomocí funkcí knihovny *jQuery*. Většina klientské aplikace je zaměřena právě na generování tohoto obsahu. Funkce jsou často členěny do dvojic, kdy jedna provede samotné vykreslení (**drawFindingFast(..)**) a druhá navázání odpovídajících akcí (**bindCancelFastFinding(..)**).

Obsah vykreslovaný při hraní her využívá možností značky *canvas*, podporující vykreslování základních obrazců. Zobrazování celého portálu bylo ovlivněno snahou omezit množství statických zdrojů, včetně obrázků. Proto jsou i hry vykreslovány pouze pomocí jednoduchých vygenerovaných tvarů různých barev.

Kapitola 6

Testování

Poslední kapitola se bude věnovat testování hotové aplikace a s tím souvisejícím zhodnocením škálovatelnosti. V první řadě bylo potřeba zvážit, které aspekty aplikace je nejdůležitější otestovat. Jednou z nejpomalejších částí systému jsou operace s databází. V případě této aplikace jsou ale počty těchto operací omezeny na minimum, jejich důsledky na škálovatelnost systému budou tedy také málo významné. Dalším problémem může být přetížení serverů, kvůli žádostem o statický obsah. Náš herní portál tento problém taktéž redukuje, vzhledem k tomu, že využívá malé množství statických dat. Díky těmto okolnostem bude mít na škálovatelnost největší význam právě běh samotné aplikace.

Z tohoto důvodu bylo testování zaměřeno přímo na jednotlivé hry, u kterých se očekává znatelně vyšší zatížení aplikace, než u chodu samotného portálu.

6.1 Testovací aplikace

Kvůli usnadění testování byla vytvořena samostatná aplikace, implementovaná v *node.js*, stejně jako tomu bylo u herního portálu. Pro využití aplikace bylo nutné napsat minimalistický testovací server, který po spuštění na umožní připojení k testovacímu prostředí, pomocí libovolného prohlížeče. Testovací aplikace nabízí možnost zvolit typ hry, počet instancí této hry (test jedné instance vyžaduje dva připojené hráče - celkový počet připojení se tedy rovná dvojnásobku počtu instancí) a délku jednoho testu.

Po spuštění testu se automaticky začnou k testovanému serveru připojovat jednotliví klienti. Mezi připojováním klientů je nastavená časová odezva, z toho důvodu, že testy byly zaměřené spíše na dlouhodobé měření, než na reakci na náhlý příval požadavků. Polovina klientů vytvoří hrací místnost zvolené hry, druhá polovina se do této hry připojí. Po zbytek testu probíhá simulace hraní takovým způsobem, aby se úroveň zatížení co nejvíce blížila úrovni zatížení při hře reálného uživatele.

V průběhu testu jsou sbírána testovací data. Jsou zaznamenávány počty výstřelů a počty vzorků odezvy. Na konci časového limitu, nebo po manuálním přerušení testu, jsou uživateli zobrazeny průměrné a maximální hodnoty odezvy.

6.1.1 Testování Lodí

Během simulování lodí dochází ke střelbě na stejné místo mapy. Klient, který obdrží zprávu o začátku tahu, vystřelí daný počet střel, mezi výstřely je krátký čekací interval. Vzorek odezvy je určen při prvním výstřelu každého tahu, jeho hodnota je rovna rozdílu mezi časem

výstřelu a přijetím aktualizace zaměřeného pole. Vzorek odezvy byl automaticky započítán do aktuální průměrné hodnoty konkrétního klienta.

6.1.2 Testování tanků

Kvůli napodobení uživatelského zatížení při hraní tanků bylo nutné simulovat krátký pohyb tanku, následovaný výstřelem. Každý testovací klient tedy na server zaslal žádost o pohyb doprava, poté o pohyb doleva, aby se vrátil na původní pozici, a až nakonec žádost o výstřel. Vzorky odezvy byly měřeny jako rozdíl mezi časem výstřelu a přijetím oznámení o střele. Stejně jako u lodí, i zde je aktuální hodnota odezvy připočítána k její průměrné hodnotě.

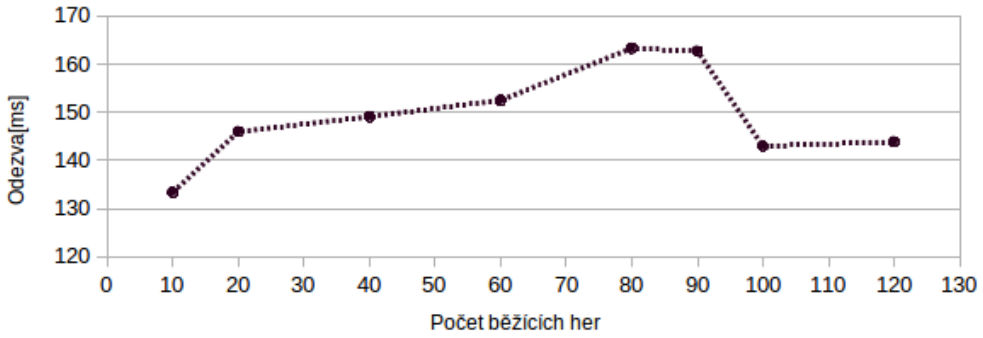
6.2 Výsledky testů

Během testování aplikace se objevily dva problémy. Prvním z nich byla nestabilita a nepřesnost výsledných hodnot. Tento problém byl způsobený množstvím faktorů ovlivňujících chod webového portálu. Tyto problémy ovšem vznikaly především na straně, odkud se klienti připojovali. Vzhledem k tomu, že testovací aplikace vysílá velké množství požadavků na server, byla výsledná odezva výrazně ovlivnitelná rychlostí připojení v místě, kde testování probíhalo. Z tohoto důvodu všechny testy probíhaly na stejném místě. Testování bylo zaměřené na určení průměrné odezvy systému pro daný počet připojení. Délka jednoho testu byla jedna hodina, kvůli tomu, aby se co nejvíce zpřesnily výsledné hodnoty a aby se omezil vliv krátkodobých výkyvů.

Druhým problémem, který se projevil obzvláště u testování lodí, bylo omezení počtu současných připojení. Omezení bylo pravděpodobně způsobené vyrovnávačem zátěže (*HAProxy*), využívaným v *Openshiftu*. Ani přepisování konfiguračních souborů HAProxy nepomohlo, a tento problém se nepodařilo vyřešit. Z tohoto důvodu nejsou předkládány hodnoty zpoždění pro více jak 240 současných připojení.

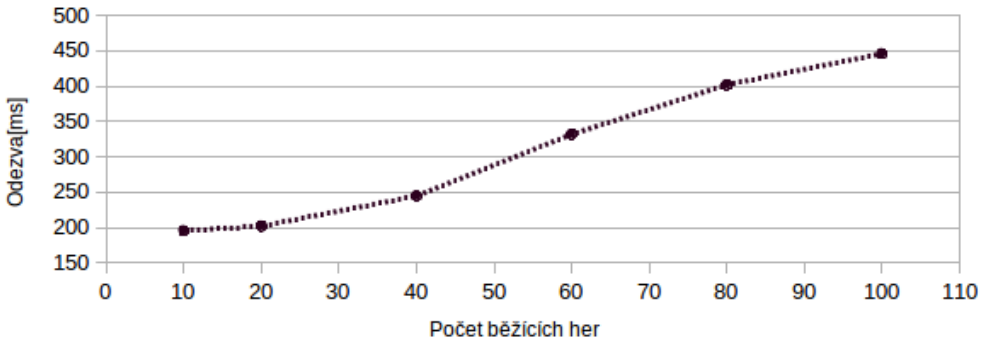
6.2.1 Lodě

Byly provedeny testy pro 10 až 120 současně běžících instancí lodí. Vzhledem k jednoduchosti hry a omezení na počet připojení se nepodařilo určit maximální možný počet instancí. I v případě, že současně probíhalo 120 her, což odpovídalo 240 připojeným komunikujím klientům, bylo stále možné hru plynule hrát a jen vzácně se objevovaly krátké nárusty zpoždění. Podle očekávání měly hodnoty odezvy růst současně s počtem běžících instancí. Objevily se ale neočekávané výkyvy, jejichž příčina nebyla odhalena. Výsledky jsou zobrazeny v následujícím grafu:



6.2.2 Tanky

Byly provedeny testy pro deset až sto současně probíhajících her. Výpočetní náročnost tanků je mnohem vyšší než je tomu u lodí, což výsledky testů jednoznačně potvrdily. Hraní tanků bylo v pořádku přibližně do šedesáti probíhajících her, po spuštění stovky instancí už dělalo problémy vytvoření dalších her. Nárůst zpoždění je znázorněn v následujícím grafu:



Tabulka přesných výsledků měření se nachází v příloze D.

Kapitola 7

Závěr

Na začátku práce bylo nutné podrobně nastudovat možnosti moderních cloudových služeb a jejich poskytovatelů. Nejdůležitější částí byl výběr správných technologií pro implementaci, ovlivněný především snahou vytvořit dobře škálovatelnou aplikaci. Mezi těmito technologiemi bylo nejdůležitější vhodně zvolit programovací jazyk a jeho nadstavby pro implementaci serveru, dále pak co nejvýhodnější databázový systém. S vybranými technologiemi souvisel i výběr samotného poskytovatele PaaS. Po zvážení informací v třetí kapitole byl vybrán RedHat Openshift.

Další část se věnovala samotné implementaci herního portálu. Výsledkem této fáze byla funkční webová aplikace umožňující hraní her Lodě a Tanky, ať už proti ostatním hráčům, nebo proti počítačem ovládanému hráči. Kromě této základní funkcionality byl implementován základ systému přátel, umožňující komunikaci mezi hráči, prozatím pouze v hlavním menu portálu.

V poslední fázi probíhalo testování portálu, zaměřené na měření odezvy během hraní her. Pomocí výsledků testů bylo možné určit přibližné limity probíhajících her při aktuálních zdrojích.

Implementace portálu usnadňuje snadné rozšiřování aplikace. Mezi možná rozšíření patří vkládání dalších her, nebo například přidávání dalších funkcí k systému přátel, jako je zvaní přátel do hry, nebo možnost komunikovat s ostatními uživateli přímo ve hře. V neposlední řadě by bylo vhodné přidat do her i zvukovou složku, která by zajisté obohatila celkový prožitek.

Literatura

[1] AppFog. https://www.appfog.com/.

[2] Bootstrap . The wolrd's most popular mobile-first and responsive front-end framework. http://getbootstrap.com/.

[3] Cloudové služby Microsoft Azure. http://azure.microsoft.com/cs-cz/services/cloud-services/.

[4] dotCloud. https://www.dotcloud.com/index.html.

[5] Dynos and Dyno manager, Heroku. https://devcenter.heroku.com/articles/dynos#dynos.

[6] Engine Yard. https://www.engineyard.com/features.

[7] Google App Engine. https://cloud.google.com/appengine/.

[8] Heroku. https://www.heroku.com/.

[9] How PaaS works, OpenShift by RedHat. https://www.openshift.com/walkthrough/how-it-works.

[10] Node.js. https://nodejs.org/.

[11] npm. https://www.npmjs.com/.

[12] RedHat OpenShift Online. https://www.openshift.com/products/online.

[13] socket.io. http://socket.io/.

[14] O.S.Tezer: Understanding SQL and No-SQL databases and different database models. https://www.digitalocean.com/community/tutorials/understanding-sql-and-nosql-databases-and-different-database-models.

[15] Peter Mell, T. G.: The NIST Definition of Cloud Computing. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2011.

Příloha A

Obsah CD

- projekt.pdf - písemná zpráva
- tex/ - zdrojový tvar písemné zprávy
- src/ - zdrojové soubory
  - server.js - startovní bod aplikace
  - server - zdrojové soubory serveru
  - client - zdrojové soubory klienta
  - html - HTML soubory
  - css - CSS soubory
  - fonts - využité fonty
  - testServer.js - startovní bod testovací aplikace
  - tests - zdrojové soubory testovací aplikace
  - package.json - popis požadovaných balíčků
- docs/ - API dokumentace

Příloha B

Spuštění aplikace

Herní portál je webová aplikace, implementovaná v Node.js[10]. Pro spuštění serveru na lokálním počítači je tedy nutné mít nainstalované Node.js včetně správce javascriptových balíčků npm[11]. Aplikace bude spouštěná z prostředí terminálu. Po přesunutí do adresáře *game*, obsahujícího zdrojové soubory aplikace, je nutné nejprve stáhnout potřebné balíčky pomocí příkazu:

```
npm install
```

Nyní je již možné spustit samotný server pomocí:

```
node server.js
```

Po spuštění serveru je aplikace přístupná pomocí internetového prohlížeče na adrese *http://localhost:8080/*.

Pro spuštění testovací aplikace je nutné použít:

```
node testServer.js
```

A poté se v prohlížeči připojit na adresu *http://localhost:8080/*.

Příloha C

Uživatelské příručka

Připojení k hernímu portálu je možné pomocí libovolného internetového prohlížeče podporujícího javascript. Aplikace běží na adrese http://game-cloudportal.rhcloud.com/. Po načtení vstupní obrazovky má uživatel možnost vstoupit do portálu jako host, registrovat si nový účet, nebo použít již dříve registrované přihlašovací údaje.

Když se uživatel dostane přímo do portálu, je mu umožněno vybrat si hru pomocí ikonek ve středu obrazovky. V současné verzi jsou implementovány hry Lodě a Tanky. Registrovaný uživatel může využít panelu na pravé straně k nalezení svých známých
a požádat je o přátelství. Uživatelé, kteří uzavřou přátelství, mohou mezi sebou komunikovat pomocí chatu.

Systém vyhledávání her je u všech her jednotný. V okamžiku kdy uživatel zvolí konkrétní hru, zobrazí se mu vyhledávací menu, obsahující založené místnosti, do kterých je možné se připojit. V pravé části vyhledávacího menu jsou tlačítka umožňující rychlé vyhledávání zápasu nebo založení vlastní místnosti, v obou případech má uživatel před potvrzením akce možnost zvolit parametry hry, které mu nejvíce vyhovují.

Všechny zatím implementované hry se ovládají čistě pomocí klávesnice. Při hraní tanků jsou šipky doleva a doprava použity k pohybu tanku doleva popř. doprava. Šipky nahoru a dolů jsou využity k zvedání popř. snižování děla. Při stisknutí mezerníku dojde k nárůstu síly výstřelu, střela je vypuštěna v okamžiku, kdy je klávesa mezerníku uvolněna.

Hraní lodí využívá stejné kombinace kláves. Šipky jsou použity pro pohyb kurzoru po hracím poli. Mezerník je použit pro potvrzení akce, která vyplývá z aktuální situace. Mezi akce patří uložení pozice lodě, zvolení náhodného rozmístění lodí, nebo vystřelení na nepřátelskou část mapy.

Všechny hry nabízejí možnost odvety po ukončení zápasu. Přijetí odvety je provedeno stisknutím mezerníku v době, kdy k tomu hra podá výzvu.

Pro opuštění hry je možné využít odkazu, nacházejícího se v levé horní části obrazovky, který umožňuje návrat do hlavního menu portálu.

Příloha D

Výsledky testů

D.1 Lodě

Počet instancí	10	20	40	60	80	90	100	120
Průměrná odezva	133.4	146	149.1	152.5	163.3	162.7	143	143.9
Počet vzorků odezvy	2400	4793	9540	14409	19323	21451	23617	28360

D.2 Tanky

Počet instancí	10	20	40	60	80	100
Průměrná odezva	195.5	202.1	244.6	331.8	401.7	445.6
Počet vzorků odezvy	7606	20283	36206	62880	73446	79747

Zajímavost: Během testování her bylo nasimulováno celkem 1513944 výstřelů. V průběhu určování průměrné odezvy bylo započítáno celkem 400294 vzorků odezvy.