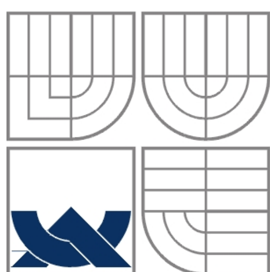


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

VIRTUÁLNÍ SVĚT
VIRTUAL WORLD

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE **BC. DANIEL BALCÁREK**
AUTHOR

VEDOUCÍ PRÁCE **ING. JAN ROUPEC, PH.D.**
SUPERVISOR

BRNO 2015

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2014/2015

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Daniel Balcárek

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Aplikovaná informatika a řízení (3902T001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Virtuální svět

v anglickém jazyce:

Virtual World

Stručná charakteristika problematiky úkolu:

Cílem projektu je vytvořit aplikaci, spadající do kategorie online hry pro více hráčů s využitím herního enginu Unity3d. Aplikace bude rozdělena na 3 hlavní části: server, svět určený pro pohyb hráčů a závodní hra, odehrávající se v tomto světě. Za základ světa bude použit areál FSI v Brně a okolí.

Cíle diplomové práce:

1. Seznamte se s problematikou tvorby počítačových her.
2. Seznamte se s herním enginem Unity3d.
3. Navrhněte a realizujte herní server včetně aplikačního programového rozhraní a knihoven pro jeho využívání.

Seznam odborné literatury:

1. Blackman, S.: Beginning 3D Game Development with Unity 4. Springer, New York, 2013.
2. Jirkovský, J. a kol.: Game Industry : Vývoj počítačových her a kapitoly z herního průmyslu. D.A.M.O., 2011.

Vedoucí diplomové práce: Ing. Jan Roupec, Ph.D.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2014/2015.

V Brně, dne 24.4.2015

L.S.

_______________________________	_______________________________
Ing. Jan Roupec, Ph.D. Ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. Děkan fakulty

ABSTRAKT

Cílem této diplomové práce je seznámit se s procesem tvorby počítačových her a herním enginem Unity3D a poté realizovat herní server, včetně aplikačního programového rozhraní a knihoven pro jeho využívání. V první části se práce věnuje historii žánrů a procesu vývoje počítačových her. Ve druhé části je popsán engine Unity3D s důrazem na Unity3D networking. Ve třetí části jsou popsány nástroje použité při realizaci práce a návrh herního serveru, včetně knihoven pro jeho využívání. Poslední kapitola popisuje nejdůležitější třídy realizované aplikace.

ABSTRACT

Main goals of this work are to present game development process and game engine Unity3D and develop game server with application programming interface including framework for using the server. The first part of this work is tageted on history of game genres and game development process. The second part of the thesis is dedicated to describing Unity3D game engine with focus on Unity3D networking. Development tools used in implementation and server desing included framework for use of the server are described in the third part of the thesis. In the last chapter of this work are described most important classes used in developed application.

BIBLIOGRAFICKÁ CITACE

BALCÁREK, D. Virtuální svět. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2015. 64 s. Vedoucí bakalářské práce Ing. Jan Roupec, Ph.D..

KLÍČOVÁ SLOVA

Photon framework, Unity3D, Herní server, Tvorba PC her

KEYWORDS

Photon framework, Unity3D, Game server, Creation of PC games

PROHLÁŠENÍ O ORIGINALITĚ

Prohlašuji, že jsem práci vypracoval samostatně dle rad a pokynů vedoucího práce pana Ing. Jana Roupce Ph.D. s použitím literárních pramenů a bibliografických citací uvedených v práci.

PODĚKOVÁNÍ

Chtěl bych poděkovat svému vedoucímu práce panu Ing. Janu Roupci Ph.D. za pomoc a rady při tvorbě této diplomové práce.

OBSAH

1 Úvod....13
2 Slovník pojmů....15
3 Problematika tvorby počítačových her....17
3.1 Historie PC her....17
3.2 Důležité dovednosti pro tvorbu her....19
3.3 Gamedesing dokument....20
3.3.1 Příklad návrhu serveru pro gamedesign dokument....21
3.3.2 Příklad návrhu klienta pro gamedesign dokument....22
3.3.3 Příklad návrhu databázového modelu pro gamedesign dokument....22
3.4 Zhodnocení navržené hry....23
3.5 Procesy vývoje....24
4 Herní engine Unity3D....25
4.1 Unity3D editor....25
4.2 Unity3D scripty....27
4.3 Unity3D GUI....28
4.4 Unity3D Networking....29
4.4.1 Servery....29
4.4.2 Komunikace....31
5 Návrh aplikace....35
5.1 Výběr vhodných nástrojů....35
5.1.1 Výběr serveru....35
5.1.2 Výběr databázového systému....36
5.2 Photon Server....36
5.2.1 Architektura Photon serveru....37
5.2.2 Základní pojmy....39
5.2.3 Požadavky....41
5.2.4 Konfigurační soubor....41
5.2.5 Ukázka využití Frameworku....43
5.3 MySQL....47
5.4 Návrh databáze....48
5.5 Návrh Serveru....48
5.6 Návrh knihoven pro Unity3d klienta....49
6 Realizace aplikace....51
6.1 Serverovská aplikace....51
6.2 Klientské knihovny....56
7 Závěr....61
8 Seznam použité literatury....63

1 ÚVOD

Cíle této diplomové práce jsou: seznámit se s problematikou tvorby počítačových her, seznámit se s herním enginem Unity3D a navrhnout a realizovat herní server, včetně aplikačního programového rozhraní a knihoven pro jeho využívání.

Proces tvorby počítačové hry je velmi náročný a pokud má hra v současnosti zaujmout, měla by být propracována v každém aspektu. Proces vývoje začíná návrhem hry, kde gamedesignéři vymýšlejí obsah, příběh, úrovně, grafiku a mnoho dalších částí, ze kterých vzniká podoba hry. Navrženou hru gamedesignéři předají programátorům, zvukařům a grafikům v podobě game design dokumentu, což je dokument popisující detailně hru. Při předání hry do rukou programátorů a umělců, se hra dostává do fáze implementace, kde se navržená hra realizuje. V této fázi se do game design dokumentu píší obecné vlastnosti realizace hry, například programátoři popisují abstraktní třídy. Jakmile je hra dokončena, dostává se do fáze testování, kde profesionální testeři nebo beta testeři hru testují a předávají informace vývojářům o chybách. Proces vývoje je tedy cyklus, vracející se stále do fáze implementace nebo návrhu, dokud není výsledná hra dovedena do dokonalého stavu.

Unity3D je multiplatformní herní engine vyvinutý společností UnityTechnologies. Díky své propracovanosti, uživatelskému rozhraní a velké komunitě vývojářů, se jedná o velmi oblíbený herní engine, ve kterém je možné vyvíjet singleplayerové i multiplayerové hry. Pro vývoj multiplayerových her Unity3D obsahuje vlastní nástroj nazvaný Unity networking. Unity3D se do vydání nejnovější verze 5.0 dělilo na personal a professional edici, kdy professional byla placená verze, obsahující více funkcí pro vývoj hry. Od vydání nejnovější verze je i professional edice zdarma a může být použita pro vývoj komerčních projektů s omezením, že hranice hrubého příjmu nesmí překročit 100 000,- dolarů, jinak je nutné zakoupit si licenci. Toto omezení platí i pro personal edici.

Prvním krokem při návrhu serverovské aplikace je výběr a seznámení se s vhodnými nástroji pro jeho realizaci podle zvolených parametrů. Tento krok můžeme samozřejmě nahradit vývojem vlastních nástrojů pro realizaci serveru, ale jedná se o velmi náročný a zdlouhavý proces. Dalším krokem je samotný návrh, kde vymýšlíme základní funkcionalitu, strukturu a vlastnosti serveru a po ukončení návrhu přecházíme do fáze realizace. Při návrhu a realizaci serveru musíme brát zřetel na spoustu aspektů, například na velikost dat posílaných při komunikaci nebo na možnou vytíženost serveru. Realizovaný server by měl být stabilní, rychlý a ošetřený proti možným útokům nebo chybám odeslaným ze strany klientů. Tyto požadavky mohou být částečně podpořeny vytvořením vlastní klientské knihovny pro využívaní serveru.

Nástroje, které mají být realizovány, jsou součástí projektu online hry pro více hráčů s možností hraní závodního simulátoru z virtuálního světa této hry. Na projektu spolupracuji s kolegy Jakubem Hamalem a Robertem Kováčem.

2 SLOVNÍK POJMŮ

GUI - uživatelské rozhraní, umožňuje ovládat počítač pomocí interaktivních grafických ovládacích prvků

Unity3D Komponenta – jedná se o prvky, které dávají hernímu objektu funkčnost, patří mezi ně například script ovládající herní objekt nebo nějaká fyzikální vlastnost předdefinovaná v unity3D

Prefab – je v Unity3D soubor předdefinovaných komponent a herních objektů, které jsou používány ve hře

Engine - počítačový termín, znamenající jádro počítačové hry, databázového stroje nebo programu

MMO hry – z anglického *massive multiplayer online game*, jedná se o počítačovou online hru pro velké množství hráčů

On-premise software – nazývané jako lokální softwarové řešení. Jedná se o software, který je nainstalovaný a běží u osoby nebo organizace využívající tento software. Pojem můžeme chápat i tak, že jde o software nad kterým má osoba nebo organizace plnou kontrolu (nemusí se tedy vždy jednat o software běžící u využívatele)

Interface – v programování se rozhraním rozumí souhrn přístupných informací o objektu a jakým způsobem lze s objektem komunikovat

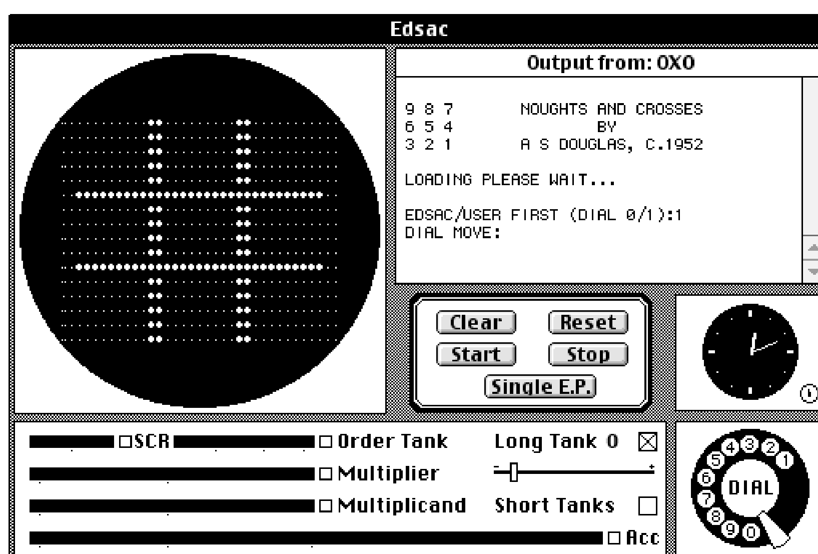
ER diagram – z anglického *entity-relationship diagram,* jedná se o abstraktní, vztahové a souvislostní zobrazení dat v diagramu

3 PROBLEMATIKA TVORBY POČÍTAČOVÝCH HER

První část této kapitoly je věnována počátkům počítačových her a poté historicky prvním hrám, které vytvořily dnes nejvyužívanější žánry. Další část se věnuje dovednostem gamedesignéra, které pokud ovládá mu usnadní proces návrhu hry a výsledný projekt může být mnohem lepší. Třetí část popisuje jednu z nejdůležitějších věcí celého projektu a to game design dokument, ve kterém jsou popsány příklady návrhu serveru, klienta a databáze. Předposlední část této kapitoly se věnuje zhodnocení hry po úspěšném návrhu, kdy se snažíme projít jednotlivými body hodnocení a odfiltrovat možné špatné nápady. Poslední část kapitoly je věnována celému procesu vývoje her.

3.1 Historie PC her [4]

Historie počítačových let spadá až do roku 1952, kdy byla napsána jedna z prvních počítačových her na světě. Napsal ji student Alexander *S.* Douglas jako svoji dizertační práci na univerzitě v Cambridge. Douglas svou hru napsal na počítači EDSAC a pojmenoval ji OXO. Jednalo se o zjednodušenou verzi piškvorek na ploše 3x3 zobrazenou na osciloskopu.



Obr. 1: emulovaná varianta hry OXO i s ovládacími prvky [7]

Druhým důležitým bodem ve vývoji počítačových her byl rok 1958, kdy William Higinbotham se svými spolupracovníky vytvořil hru Tenis for Two, pro kterou byl speciálně sestrojen digitální počítač a zobrazena byla stejně jako u hry OXO na osciloskopu. Osciloskop zobrazoval celou herní scénu z bočního pohledu.

V roce 1962 přišla na svět hra Spacewar!, kterou vytvořil Stephen Russell v asembleru na minipočítači PDP-1. Na této hře se také podíleli i další autoři. Peter Samoson, který vytvořil pozadí hry složené ze souhvězdí viděné ze země. Martin Graetz, autor programové rutiny. Wayne Witanen, Dan Edwards autoři rutiny pro výpočet gravitačních sil. Ve hře proti sobě stáli dva hráči, kdy každý hráč ovládal svou vesmírnou loď a mohl vypálit rakety na protivníka. Ve středu scény byla černá díra, která vytvářela silné gravitační pole a byla dalším zdrojem nebezpečí pro hráče. Hra Spacewar! byla na svou dobu revoluční hlavně v tom, že nabízela vlastní originální svět.

V roce 1969 byly vytvořeny tři hry Hamurabi, Lunar Landers a Space Travel. Tyto hry měly poměrně velký vliv na další vývoj a zavedly dvě nové herní kategorie, simulátor a strategický simulátor. První z těchto her s názvem Hamurabi je strategická hra, ve které vládnete jako panovník státu a snažíte se o jeho co nejdelší rozvoj a prosperitu. Komunikace s hráčem byla realizována jako textové rozhraní. Lunar Landers byla taktéž hra s textovým rozhraním, kdy hráč zadával tah motorů v reakci na zobrazené informace. Hru napsal středoškolský student Jim Storer v jazyku FOCAL. Space Travel byla hra vytvořená podle modelu sluneční soustavy a hraní spočívalo v cestování mezi planetami. Na vývoji se podílel Ken Thompson a jeho kolegové. Vytvořená hra vedla nepřímo k vývoji operačního systému Unix.

Hra, která velmi výrazně ovlivnila další vývoj strategických her byla hra Empire, která vznikla v roce 1971. První verze hry byla napsána v jazyku BASIC. Herní plocha byla rozdělena na čtverce. Každé pole bylo rozděleno na souš nebo vodní plochu. Po vodní ploše se dalo pohybovat pomocí bitevních lodí, po souši pak pomocí jednotek, které se vyráběly ve městech a ta bylo možné dobývat.

První závodní hrou byla hra Space Race vydaná firmou Atari v roce 1973. Jednalo se o arkádu, ve které dva hráči ovládají vesmírné lodě a snaží se vyhnout překážkám. Hra byla ovládána pomocí obousměrného joysticku na herním automatu.

Žánr first person shooter vznikl v roce 1974 a o jeho vznik se zasloužily videohry Maze War a Spasim. Jedná se o první síťové hry šířené na ARPANETu. Hra Maze war obsahovala dva módy: singleplayer, multiplayer a byla vytvořena vývojářem Stevem Colleym. Spasim obsahovala jen mód pro multiplayer a vytvořil ji Jim Bowery.

O zavedení herního žánru adventure se zasloužila hra vydaná v roce 1975, Colossal Cave Adventure. Jednalo se o textovou hru, kterou hráč ovládal pomocí slov, většinou se používala značně zjednodušená angličtina. Také šlo o jednu z prvních her, která byla šířena pomocí ARPANETu a také odpovídala dnešnímu pojetí open source.

První bojová hra se jmenovala Heavyweight Champ a byla vydána v roce 1976 firmou Sega. Jednalo se o arkádu pro dva hráče, kde každý hráč ovládal jednu boxovací rukavici a pohybem nahoru nebo dolů dával ránu vysoko nebo nízko. Scéna byla vidět z bočního pohledu a byla černobílá.

První hrou s konceptem Dungeons & Dragons byla hra Rogue. Tato hra využívala jako jedna z prvních celoobrazovkový režim. Herní svět byl vygenerován pomocí algoritmu, který vytvářel navzájem spojené herní mapy. V těchto mapách byli umístěni protivníci, předměty jako brnění, zbraně, lektvary atd.. První verzi hry napsali vývojáři Michael Toye a Glenn Wichman s využitím knihovny ncurses od Kena Arnolda. Hra byla vytvořena v roce 1980 a měla velký vliv na žánr RPG her, ve kterých se některé techniky z této hry využívají dodnes.

Žánr plošinovka spadající do arkád vznikl v roce 1980 a za jeho vznikem stojí hra Space Panic vydaná společností Universal. Hry s podobnými prvky samozřejmě existovaly už dříve, ale z důvodu technických omezení byly vyhodnoceny jako statická hrací pole. Hra je tvořena herním světem sestaveným z několika pater propojených žebříky. Hráč ovládá hrdinu, který se snaží vyhýbat překážkám a nepřátelům.

Herní scéna se stále vyvíjí a jsou vytvářeny stále nové žánry, ale vždy jsou zařazeny mezi mezi 6 hlavních žánrů: arkáda, akční, strategie, adventura, simulátor a hra na hrdiny. Ve výše popsané historii jsou uvedeny nejznámější žánry, se kterými se setkáváme dnes u nejoblíbenějších her.

3.2 Důležité dovednosti pro tvorbu her

Aby výsledná hra měla potenciál a získala si určitou herní komunitu, měl by každý gamedesignér ovládat důležité dovednosti pro tvorbu her. Čím lépe potom jednotlivé dovednosti ovládá, tím lépe se orientuje ve vývoji a o to lepší může být výsledný projekt.

Důležité dovednosti gamedesignéra:

- 2D a 3D gamedesign – grafika je vizitkou hry, proto by každý gamedesignér měl vědět jak pracovat s 2D a 3D grafikou. Znát mechanismy, které se při práci s grafikou používají a být schopen navrhnout něco nového.
- Antropologie – při vývoji hry nejde o to co hráč chce ve hře mít, ale o to jak se při hraní cítí a co chce zažít. Tedy to co v něm budí zábavu, spokojenost a radost.
- Architektura – důležitou součásti každé hry je architektura použitá ve vytvořeném světě. Proto by se každý gamedesignér měl orientovat v architektuře a tak být lépe obeznámen s životním prostorem obyvatel fiktivního světa.
- Nápad(Brainstorm) – velmi důležitá dovednost gamedesignéra. Měl by si umět představit velké množství nápadů, řešení a mechanismů, které by měly dávat smysl a zapadat do sebe. Není nutné zvládat tyto věci sám, ale je vhodné vytvořit si pravidla podle kterých by tyto nápady zapadaly do světa, který vytváří.
- Kinematografie – gamedesignér by se měl snažit dát hráči opravdový zážitek a k tomu mu může dopomoci kinematografie. To znamená umět vytvořit upoutávku, dobrý konec nebo krátkou filmovou sekvenci doplňující příběh. Zvládnutí těchto věcí se může stát velkou výhodou.

- Komunikace – většina gamedesignérů pracuje s týmem a řeší s ním spoustu problémů, stejně jako s klientem nebo se zájemci o hru. Proto je výhodné umět s okolím dobře komunikovat a tím získat opravdový názor týmu a veřejnosti na hru.
- Papírování – respektive tvůrčí psaní. Při vytváření světa a příběhu je nutné dát postavám, územím a událostem příběh, který hráče vtáhne více do hry. Také je třeba každou myšlenku technického směru jasně vyjádřit, protože pokud ne, nemusí být nikdy realizována.
- Obchod – pokud chce gamedesignér svou hru prodat, měl by umět jednat s investory, vydavateli, producenty atd. Měli by chtít koupit hru.
- Ekonomika – dnešní hry pracují s ekonomikou víc než kdy jindy. Hráč nesmí být silný ani příliš slabý a musí se cítit hodnotný ve hře. Také je nutné vytvořit vhodný cenový systém výbav, které je možné ve hře koupit.
- Informatika – gamedesignér by měl rozumět technologii, kterou ve své hře využívá. Pokud totiž ví o výhodách a nevýhodách zvolené technologie, má nad ní větší kontrolu a možnost vytvořit něco nového.
- Historie – znalost historie je velkou výhodou pro inspiraci.
- Management – gamedesignér by něco málo o řízení týmu vědět měl. Protože pokud vede projekt hrozný manažer, je stále možné projekt úspěšně dokončit.
- Matematika – každá hra využívá matematiku a dobrý gamedesignér by ovládat matematiku měl, ať už pro zlepšení mechanismů ve hře nebo pro zlepšení samotné hry.
- Design zvuku – velmi důležitá dovednost pro vtažení hráče do hry, zvuky a hudba hráče přesvědčí o příběhu a místě, kde se hra odehrává.
- Psychologie – pochopení lidské mysli je výhoda při tvorbě příběhu a chování NPC postav
- Chování na veřejnosti – je nutné umět komunikovat s týmem a vědět, který nápad je možné ve hře použít a který ne, protože každý v týmu by rád realizoval svůj nápad.

Není možné, aby někdo všechny tyto dovednosti ovládal na vysoké úrovni. Hlavní myšlenka spočívá v tom, že čím lépe gamedesignér tyto dovednosti ovládá, tím snazší je pro něj proces návrhu hry. [5]

3.3 Gamedesing dokument

Pravděpodobně jedna z nejdůležitějších fází vývoje hry je vytvoření gamedesign dokumentu. Jedná se o detailně popsaný dokument obsahující veškeré údaje o hře. Na tomto dokumentu pracuje celý tým a používá se v celém procesu vývoje hry. Definuje jak koncept hry, tak i funkcionální a technická specifika hry. Mezi tři hlavní části dokumentu patří: koncept hry, design hry a technický design hry. [8]

Obecná struktura dokumentu:

- Základy hry – v této části se popisují herní vlastnosti, hratelnost, charaktery, herní prvky, umělá inteligence
- Popis úrovní – popis vztahů mezi jednotlivými úrovněmi, obecný popis návrhu úrovní
- Uživatelské rozhraní – funkční požadavky hry, objekty uživatelského rozhraní
- Vývoj – základní popis objektů ve hře, z hlediska programování například:
  - popis abstraktních tříd – základní fyzika, základní fyzika hráče, nepřítele, objektu, překážky ve hře, základní interakce
  - popis odvozených tříd – základní hráč, nepřítel, objekt, překážka
- Grafika a video – grafika a animace
- Zvuky a muzika - obecný popis, zvukové efekty, muzika
- Zápletka hry – příběh hry

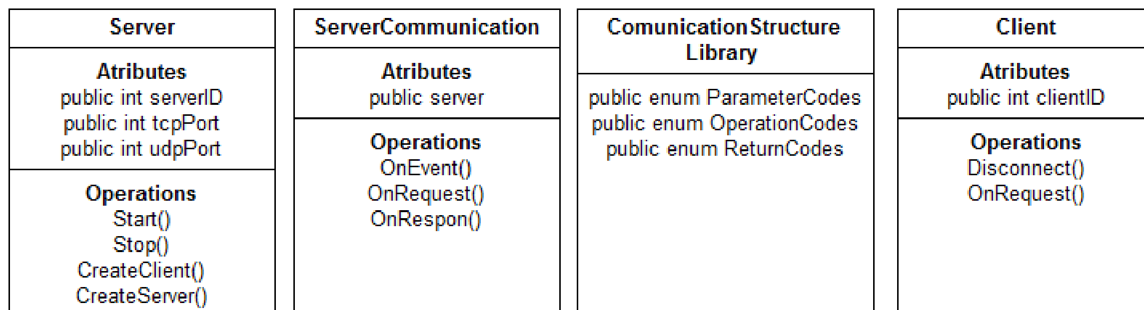
Pokud se jedná o projekt online hry pro více hráčů, jako v tomto případě, do jednotlivých bodů jsou přidány prvky týkající se online části. Přičemž bodem, ve kterém je popsána základní struktura serveru, komunikace a databáze je vývoj. Z bodů, které jsou popsány před vývojem již víme co všechno bude muset server zpracovávat, jaká data by měl klient posílat a přijímat a jak má vypadat základní databázový model. Z těchto požadavků a také ze zkušeností a dovedností vývojového týmu, je potřeba se dohodnout na využití externího frameworku pro tvorbu klienta a serveru nebo na tvorbě vlastního. Po tomto rozhodnutí můžeme přejít k základnímu návrhu.

3.3.1 Příklad návrhu serveru pro gamedesign dokument

V prvním bodě návrhu si upřesníme co vše má serverovská aplikace umět a následně začneme postupně navrhovat a popisovat jednotlivé třídy, přičemž v pomyslné hierarchii, například pro servery začínáme s abstraktní třídou server a k ní postupně přidáváme další třídy, které server potřebuje k běhu.

Příklad:

Návrh serveru v dokumentu můžeme začít s vytvořením abstraktní třídy pro server. Tato abstraktní třída by měla obsahovat veškeré vlastnosti, které každý server potřebuje, například funkce: start, stop, vytvoreniKlienta, proměnné: idServeru, tcpPort, udpPort atd. Při návrhu takovéto aplikace využíváme více serverů zpracovávajících rozdílné požadavky, proto jsou nutné i třídy pro komunikaci mezi servery. Aby server mohl komunikovat s klientem, je nutné vytvořit abstraktní třídu i pro klienta. Dále je nutné upřesnit komunikaci klient server pomocí struktury přijatého nebo odeslaného požadavku.



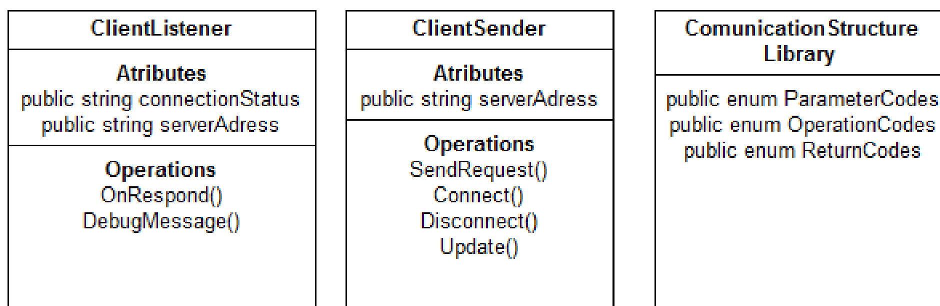
Obr. 2: Ukázka navržených abstraktních tříd pro serverovskou část

3.3.2 Příklad návrhu klienta pro gamedesign dokument

V návrhu klientské části postupujeme stejně jako na straně serveru. Upřesníme si co má klientská aplikace umět z předchozích bodů design dokumentu a začneme vytvářet abstraktní třídy pro klientskou část, opět pomocí pomyslné hierarchie.

Příklad:

Návrh můžeme začít abstraktní třídou, která přijímá odpovědi ze serveru. Třída by měla obsahovat například proměnné: adresaServeru, statusPripojeni, a funkce pro přijetí odpovědí ze serveru. Popřípadě funkce pro chybová hlášení. Další abstraktní třída, kterou potřebujeme pro komunikaci se serverem je třída pro posílání požadavků a samozřejmě využijeme strukturu navrženou na straně serveru pro přijetí odpovědi a odeslání požadavku.



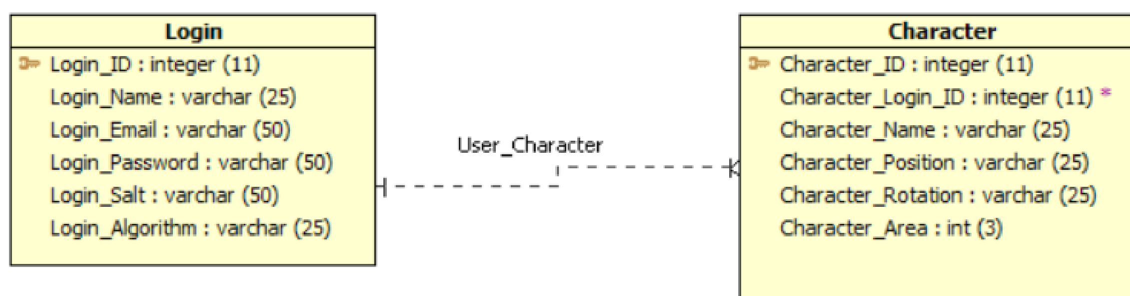
Obr. 3: Ukázka navržených abstraktních tříd pro klientskou část

3.3.3 Příklad návrhu databázového modelu pro gamedesign dokument

V tomto bodě návrhu je důležité rozhodnout, jaká data je potřeba v databázi ukládat. Poté můžeme přistoupit k vytváření databázového modelu, který by měl splňovat požadavky, které jsou předpokládány u tvorby každého databázového modelu.

Příklad:

Začneme tabulkou pro přihlášení, která by měla obsahovat tyto základní atributy: id, přihlašovací jméno, heslo, hashovací algoritmus pro heslo, sůl (konstantní řetězec pro hashování), podle potřeby přidáváme další, například: datum vytvoření účtu atd. Samozřejmě pokud je počet atributů velký, je nutné tabulku rozdělit a spojit příslušnými vazbami. Pokračujeme vytvořením tabulky postava, kterou spojíme s tabulkou pro přihlášení pomocí vazby 1:N (jeden účet může obsahovat N postav). Tabulka postava může obsahovat velký počet atributů, které by jsme měli rozdělit do dalších tabulek a ty podle určitého vazby spojit.



Obr. 4: Ukázka návrhu databázového modelu

3.4 Zhodnocení navržené hry

V téhle fázi vývoje už máme hru navrženou a přecházíme do procesu implementace. V téhle fázi už hru můžeme hodnotit a pokud chceme, aby výsledná hra byla co nejlepší, je zhodnocení i žádoucí.

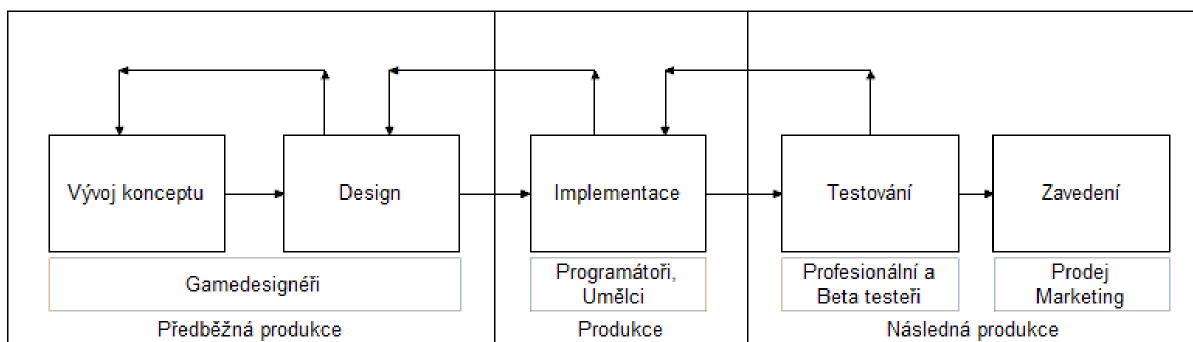
1. Část - zde by si měl gamedesignér položit tři otázky: Je navržená hra z jeho pohledu dobrá? Má z ní dobrý pocit? Bude dobře fungovat? Pokud je spokojen s odpověďmi na tyto otázky, prošel touto částí zhodnocení.
2. Část – otázka pro celý vývojový tým zní: Bude hra dobře udělaná? Aby hra prošla touto částí musí projít kritikou ze strany vývojového týmu a ta závisí hlavně na zkušenostech, které tým s vývojem her má.
3. Část – bude hra pro cílovou skupinu dostatečně atraktivní? Pro úspěšné projití touto části je potřeba rozhodnout zda je design pro cílovou skupinu příjemný, atraktivní a zajímavý.
4. Část – pokryje vytvořená hra náklady na její tvorbu? Tato část by měla být analýzou možnosti prodávat navrženou hru. Měly by být shrnuty všechny náklady, měla by se probrat otázka konkurence, tedy jak podobné jsou konkurenční hry a jak se hře pravděpodobně povede v prodeji.

5. Část – jaký bude pravděpodobně názor hráčů téhle hry? Tuto část pro hodnocení je možné použít až v procesu vývoje, kdy už je možné si hru zahrát. Část se zaměřuje na hráče a to hlavně na: ponoření se do děje, pochopení mechanismu hry a pocit ze hry. V této fázi může hru testovat celý vývojový tým.

Je důležité najít způsob jak těmito částmi pro hodnocení projít. Protože pokud zmíněné hodnocení několikrát využijeme, může nás naučit oddělit špatné a dobré nápady již v rámci vývoje. [6]

3.5 Procesy vývoje

Mezi procesy vývoje se řadí vývoj konceptu, design, implementace a testování. Na vývoji konceptu a designu pracují převážně gamedesignéři. V procesu implementace hlavně zvukaři, grafici a programátoři, popřípadě techničtí poradci. Při procesu testování to jsou pak profesionální testeři a beta testeři. Celkový proces vývoje je stále se opakující cyklus ukončený až zavedením hry neboli prodejem.



Obr. 5: Celý proces vývoje hry

V části předběžné produkce gamedesignéři pracují na konceptu a designu hry, začínají pracovat na gamedesign dokumentu a dávají hře tvar. Dále se snaží nalézt, popřípadě za pomocí týmu vytvořit potřebné zdroje například herní engine. V produkční části programátoři, grafici a zvukaři začínají pracovat na svých odvětvích hry a také na design dokumentu. Práce většinou probíhá v následujícím cyklu :

1. Vytvoření herních mechanismů a designu herní úrovně
2. Grafika: kreslení, modelování, animace
3. Tvorba rozhraní
4. Příběh: události, dialogy, krátké filmové sekvence
5. Zvuky: zvukové efekty, hudba, hlasy

Poslední částí před zavedením hry do prodeje je testování, ve které se profesionální a beta testeři snaží nalézt chyby. V průběhu vytváření hry se stále vracíme do předchozích částí vývoje a upravujeme nalezené chyby nebo nesrovnalosti. [8]

4 HERNÍ ENGINE UNITY3D

Unity3D je multiplatformní herní engine vytvořený společností Unity Technologies. První verze 1.0 byla vydána v roce 2005 a dnes si uživatelé již mohou stáhnout verzi 5.0 vydanou 1.4.2015. Herní engine Unity3D je navržen pro tvorbu her pro PC, konzole a mobilní zařízení. Unity3D podporuje vývoj pro širokou škálu softwarových platforem, které můžeme vidět v tabulce 1.

Desktopové a webové aplikace	Chytré telefony a tablety	Konzolové aplikace	Chytré televizory
Windows	Android	PS3, PS4	Android TV
Linux	iOS	PS Vita	Samsung Smart TV
Mac	Windows Phone	Xbox One	Tizen
Web player	BlackBerry 10	Xbox 360	-
WebGL	Tizen	Wii U	-

Tabulka 1: Podporované softwarové platformy

Unity3D je ke stažení ve 2 verzích: Personal a Professional. Personal edice je zdarma a pro práci na menších projektech je dostačující. Hry vytvořené v Personal edici je možné prodávat, avšak hranice hrubého příjmu nesmí překročit částku 100 000,- dolarů. Professional edice je s vydáním verze 5.0 také zdarma, ovšem stejně jako u Personal verze, pokud vydanou hru prodáváme, hranice hrubých příjmů nesmí překročit uvedenou částku. Pokud ano, je nutné si zakoupit licenci, přičemž licence se vydává na uživatele.

4.1 Unity3D editor

Unity3D editor se skládá z několika částí, přičemž každá slouží ke specifickému účelu. Všechny části jsou pohyblivé, je tedy možné poskládat si je podle svých představ. Hlavní části unity3D editoru se skládají z: Project, Hirearchy, Inspector, Scene, Game, Animator a Console. [10]

Project

V této části vidíme strukturu projektu. Každý projekt obsahuje složku Assets, ve které jsou uložena veškerá data projektu. Při tvorbě projektu je vhodné dodržovat přehlednou strukturu a ve složce assets data ukládat do příslušných složek.

Jelikož v této práci bylo unity3D použito hlavně pro tvorbu klienta, struktura objektu obsahovala pouze složky:

Scenes – složka obsahovala potřebné scény pro testovacího klienta

Plugin – tuto složku je nutné v projektu mít, pokud využíváme externí knihovny. Složka obsahuje knihovny potřebné pro komunikaci se serverem.

Scripts – složka obsahuje všechny použité scripty

Hierarchy

Tato část obsahuje všechny objekty vložené ve scéně. Objekty jsou poskládány do přehledné hierarchie předků a jejich potomků.

Hierarchie testovacího klienta obsahovala pouze prázdný herní objekt, který obsahoval script testovacího klienta.

Inspector

Inspector obsahuje podrobné informace o vybraném herním objektu. Především tedy připojené komponenty, které můžeme objektu přidávat, odebírat a upravovat jejich vlastnosti.

Inspector byl využit především pro přidání scriptu danému hernímu objektu.

Scene

V okně scene nastavujeme pozice objektů. Tvoříme zde tedy areál, ve kterém se bude daná scéna odehrávat. Objekty do scény můžeme přidávat i pomocí scriptu. Zde si tedy můžeme vytvořit základní strukturu a pomocí scriptu do scény vložíme okrasné nebo mnohočetné prvky, například stromy, keře atd.

Okno scene při tvorbě klienta bylo využito jen pro přidání prázdného herního objektu.

Game

V této části vidíme scénu z pohledu hlavní kamery. Při spouštění projektu se nám toto okno automaticky otevře a my můžeme otestovat aplikaci. Jakmile je scéna spuštěna jsme v takzvaném play módu, kde můžeme objekty ve scéně upravovat a okamžitě vidět změny bez nutnosti jejich uložení.

V testovacím klientovi se zobrazovaly GUI prvky sloužící pro simulaci klienta. Byla to hlavně tlačítka pro posílání požadavků, textová okna pro posílání zpráv a přihlašovacích údajů a poté prvky zobrazující text pro přijaté odpovědi.

Animator

V této části vážeme unity3D animační systém na herní objekt.

Tato část v testovacím klientovi nebyla nijak využita.

Console

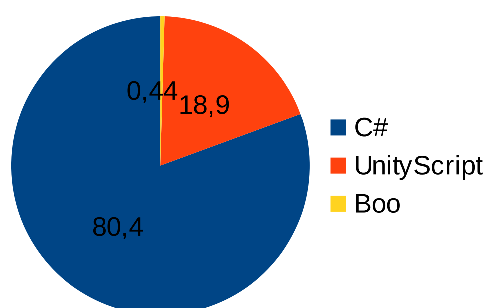
V této části se vypisují chyby. Jak při běžící aplikaci, tak i při načtení nového nebo upraveného scriptu. Do console se taky dají vypisovat zprávy při běžící aplikaci, což je velmi nápomocné při hledání chyb v aplikaci.

Při testování aplikace bylo toto okno hojně využíváno a to hlavně pro vypisování přijatých dat, hledání chyb nebo oznámení o proběhnutí nějaké funkce.

Pro tvorbu testovacího klienta bylo zvoleno standardní rozvržení projektu a to hlavně z důvodu využití pouze oken Console, Game a Project.

4.2 Unity3D scripty

Chování herních objektů je kontrolováno pomocí komponent, které jsou objektu přiřazeny. Unity3D obsahuje spoustu univerzálních komponent už v editoru, ale při vytváření hry programátor potřebuje využít svých komponent přiřazujících hernímu objektu nové chování a vlastnosti, k tomu slouží scripty. Unity3D dovoluje použít 3 typy jazyků pro scriptování: C#, který podle statistiky využívá přes 80% uživatelů, UnityScript neboli JavaScript pro Unity, využívá necelých 20% uživatelů a nakonec Boo script, který využívá 0,44% uživatelů. Proto bylo v nejnovější verzi 5.0 odstraněno tlačítko pro vytvoření Boo scriptu a tento jazyk už nebude nadále podporován v dokumentaci. Ovšem pokud v projektu už nějaký Boo script existuje, bude stále pracovat stejně jako v předchozích verzích. [9]



Obr. 6: Graf využití scriptovacích jazyků v unity

Při realizaci testovacího klienta byl využit jazyk C#, z důvodu předchozích zkušeností s tímto jazykem.

Struktura Unity3D scriptu

Při vytvoření scriptu v editoru nám unity vygeneruje předdefinovanou strukturu scriptu, kterou můžeme vidět na ukázce níže.

```
using UnityEngine;
using System.Collections;

public class FirstClass : MonoBehaviour {
//inicializace
void Start () {
}
//volána jednou za snímek
void Update () {
}}
```

Na ukázce můžeme vidět, že struktura obsahuje dvě reference na Unity3D knihovny a třídu pojmenovanou podle názvu vytvořeného scriptu (souboru), v tomto případě *FirstClass*. Tato třída dědí ze třídy *MonoBehaviour*, která propojuje námi vytvořený script s vnitřním fungováním unity enginu. Můžeme si ji představit jako třídu dovolující vytvořit nový typ komponenty, kterou dále můžeme přiřadit hernímu objektu. Ve třídě *FirstClass* byly vygenerovány dvě metody:

Start() - je volána automaticky unity enginem při prvním využití scriptu, ještě před prvním zavoláním funkce *Update()*. Metoda slouží především pro inicializaci proměnných.

Update() - je volána automaticky unity enginem při každém snímku. Tato metoda je nejčastěji využívána pro implementaci chování objektů ve hře, například chůze, aktivace nějaké činnosti nebo odpovědi na vstup uživatele.

Unity3D script dále může obsahovat spoustu jiných funkcí, které obsahuje třída *MonoBehaviour*. Tyto funkce jsou volány například při vstupu uživatele, při vypnutí aplikace nebo při povolení/zakázání objektu. Při tvorbě testovacího klienta však byly využity pouze základní funkce *Start()* a *Update()*, funkce *OnApplicationQuit()* volána před vypnutím aplikace a funkce *OnGUI()*, která je popsána v následující kapitole.

4.3 Unity3D GUI

V nejnovější verzi Unity3D 5.0 vydané 1.4.2015 je GUI systém nahrazen novým UI systémem. GUI systém je však stále funkční, ovšem již jej není doporučeno používat. Testovací klient byl vytvořen ještě před vydáním nové verze, proto pracuje ještě na starém GUI systému. Uživatelské prostředí je v testovacím klientovi vytvořeno pomocí scriptu, kde se vytvořené grafické prvky vytvářejí stejným způsobem ve starém i novém systému. [10]

Funkce OnGUI()

Uživatelské ovládací prvky, které chceme do našeho projektu v unity přidat, využívají funkci *OnGUI()*, která je stejně jako funkce *Update()* volána každý snímek. Vytvoření GUI prvků ze scriptu je velmi jednoduché, jak můžeme vidět na ukázce níže.

```
public class FirstGUITest : MonoBehaviour {
void OnGUI () {
     GUI.Box(new Rect(10,10,100,90), "Výběr levelu");
     if(GUI.Button(new Rect(20,40,80,20), "Level 1"))
          Application.LoadLevel("Nazev sceny");
     if(GUI.Button(new Rect(20,70,80,20), "Level 2"))
          Application.LoadLevel("Nazev sceny");
}
}
```

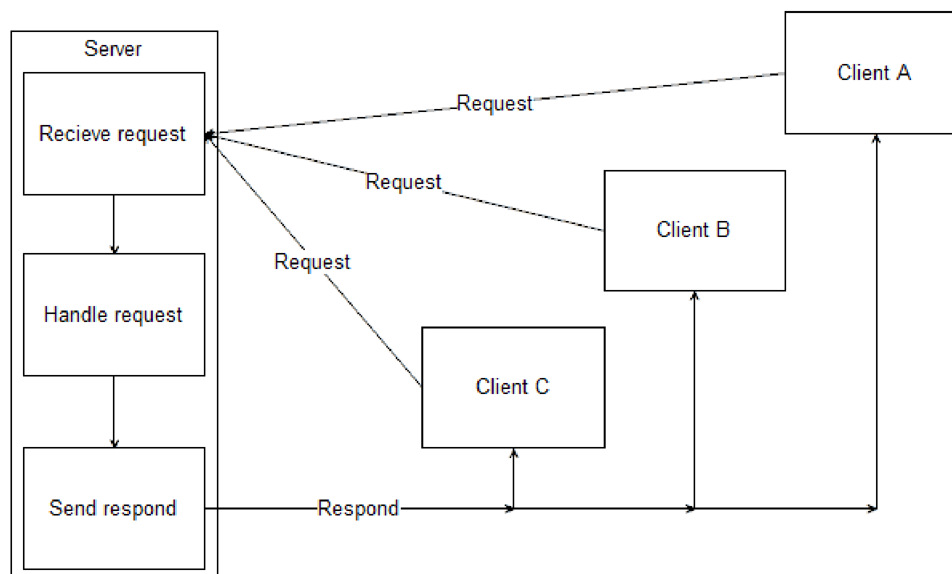
Ukázka na předchozí straně zobrazuje vytvoření dvou tlačítek a jednoho boxu, který zastřešuje obě tlačítka. Pro většinu GUI prvků musíme v prvním parametru nastavit pozici a velikost prvku, tu nastavujeme pomocí struktury *Rect(float zleva, float zvrchu, float šířka, float výška)*. Dalším parametrem GUI prvků je pak stringová hodnota určující text na prvku. V ukázce můžeme dále vidět funkci *LoadLevel()* spadající do třídy *Application*, která načte scénu podle zadaného parametru.

4.4 Unity3D Networking

Jde o robustně a flexibilně navržený systém pro vytváření sítí. To znamená, že zodpovědnost za věci, které by se mohly provádět automaticky je převedena na vývojáře. Vývoj v systému probíhá velmi rychle, z důvodu velké podpory ze strany komunity pomocí tutoriálů nebo i dokumentace na stránkách Unity3D. Při vývoji jakékoliv online hry je vhodné vědět o používaném nástroji co nejvíce, vyvarujeme se tak možným problémům, které mohou vzniknout, například při použití tutoriálů. [11]

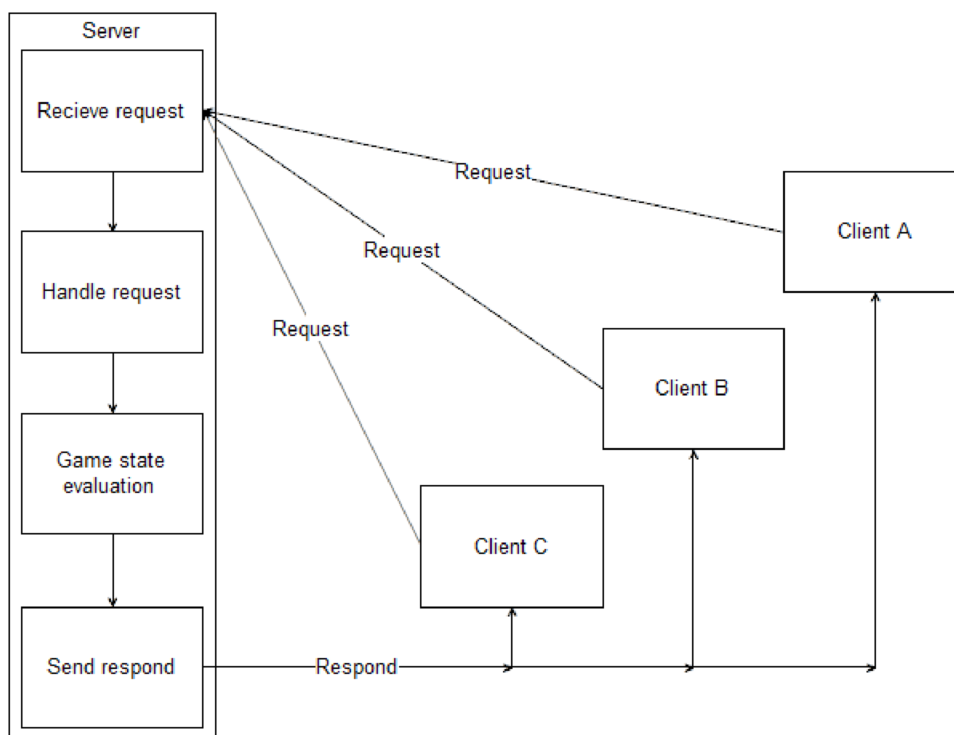
4.4.1 Servery

Non-Authoritative Server – tento přístup je jednoduchý na vývoj, protože nekontroluje každý vstup uživatele. Klient si vstupy a akce uživatelů, herní logiku a fyziku zpracovává sám. Server jen přeposílá požadavky klientů s případnými menšími úpravami. Nevýhodou tohoto přístupu je možná nesynchronizace u klientů. Například malé rozdíly mezi pohybem hráčů u různých klientů mohou vést k nesynchronizaci.



Obr. 7: Jednoduchá ukázka ne-autoritativního herního serveru

Authoritative Server – tento přístup vyžaduje, aby server zpracovával veškeré vstupy a akce jednotlivých hráčů, pravidla, které hra má a také simulaci celého světa. Každý klient posílá vstup nebo akci uživatele na server a server mu odpoví aktuálním stavem hry. To znamená, že žádný klient nemůže změnit stav hry sám, což dovoluje serveru naslouchat požadavkům klientů a podle nich rozhodnout o stavu hry. Výhodou tohoto přístupu je mnohem složitější cheatování ze strany hráčů. Další výhodou je, že zpracování fyziky hry probíhá na straně serveru, tudíž nedojde k možné nesynchronizaci. Potenciální nevýhodou může být čas, za který je přijata odpověď ze serveru. Tedy pokud hráč provede nějakou akci a odpověď ze serveru přijde se zpožděním, bude toto zpoždění na hráči vidět. Možným řešením tohoto problému je tak zvaný *client-side prediction.* Tato technika dovoluje klientovi lokálně měnit stav hry, ale v případě potřeby musí být schopen přijmout opravu ze strany serveru.

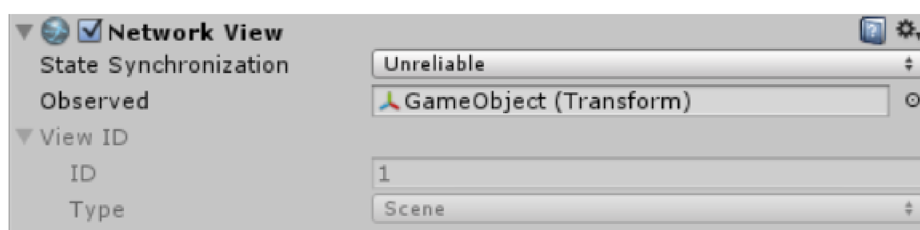


Obr. 8: Jednoduchá ukázka autoritativního herního serveru

Unity3D dovoluje použít oba přístupy, záleží tedy na vývojáři jaký přístup bude pro jeho hru vhodnější. Autoritativní server má mnoho výhod, ale oproti ne-autoritativnímu serveru vykonává mnohem více operací.

4.4.2 Komunikace

Network Views – hlavní unity3D komponenta dovolující sdílení dat v síti. Tato komponenta dovoluje použít dva typy technologií ke komunikaci: remote procedure call a state synchronization.



Obr. 9: Ukázka komponenty Network View

Na obr. 9 vidíme komponentu Network View obsahující proměnné:

State synchronization – parametr určující jak se bude stav posílat

- Unreliable – posílá se kompletní stav. Je posíláno vetší množství dat, ale je minimalizována možnost ztráty paketu.
- Reliable Delta Compressed – bude zaslán pouze rozdíl mezi předchozím a následujícím stavem. Pokud se nic nezmění, nic se neposílá. V případě ztráty paketu je poslán znovu.
- Off – technologie state synchronization je vypnuta. Nastavíme pokud chceme používat pouze technologii remote procedure call.

Observer – data, která budou odeslána

View ID – unikátní identifikátor pro Network View

- ID – id Network View v konkrétní scéně
- Type – buď uložený ve scéně nebo přiřazen za běhu

Remote procedure call – dále jíž RPC, je technologie dovolující programu vyvolat funkci na jiném počítači pomocí sítě, kde síť je i spojení mezi klientem a serverem, které jsou na jednom počítači. Klient může poslat RPC serveru i klientům a server může RPC poslat jednomu nebo více klientům.

V Unity3D se posílání informací pomocí RPC nejčastěji používá při provádění nebo řízení individuálních událostí. RPC funkce jsou deklarovány ve scriptech, které jsou jako komponenty připojeny k danému hernímu objektu. Tento objekt musí obsahovat komponentu network view, která odkazuje na script obsahující RPC funkci. RPC funkce pak může být volána z jakéhokoliv scriptu uvnitř daného herního objektu.

Na ukázce níže můžeme vidět použití RPC funkce. Na začátku si deklarujeme proměnou *networkView,* kterou ve funkci *Start()* inicializujeme. Ve funkci *Update()* poté kontrolujeme, zda bylo stisknuto tlačítko pro otevření dveří, pokud ano pošleme pomocí proměnné *networkView* a funkce RPC zprávu o otevření dveří. Parametry RPC funkce jsou string názvu funkce, která se má zpracovat a enum RPCMode, udávající komu je zpráva určena. Funkci, která má být vyvolána označíme třídou RPC, jak můžeme vidět v dolní části ukázky.

```
public class FirstClass : MonoBehaviour
{
	NetworkView networkView;

	void Start()
	{
		networkView = new NetworkView();
	}

	void Update()
	{
		if (Input.GetButtonDown("OpenDore"))
			networkView.RPC("OpenDore", RPCMode.All);
	}

	[RPC]
	void OpenDore()
	{
		//kód pro otevření dveří
	}
}
```

State Synchronization – používá se pro zasílání dat, která se konstantně mění, například pohyb, běh, skok atd. Tento typ dat je po síti posílán pravidelně, proto je nutné co nejvíce zredukovat množství těchto dat.

V Unity3D technologie state synchronization sdílí data neustále se všemi klienty. Pro synchronizaci objektu je nutné přidat danému objektu komponentu network view a nastavit jí vlastnost, kterou má pozorovat. Tato vlastnost je pak synchronizována se všemi klienty ve hře.

V ukázce na následující straně můžeme vidět posílání proměnné *healt* pomocí state synchronization. Pro sledování proměnné *healt*, která je datového typu int, používáme metodu *OnSerializeNetworkView()*, která je automaticky volána při odesílání nebo přijímání této proměnné. Proměnná je předávána pomocí třídy *BitStream*. Tato třída obsahuje bool proměnné *isWriting*, *isReading* definující, zda je proměnná zapisována nebo čtena. Pokud chceme využít tuto ukázku, musíme ji přiřadit danému hernímu objektu. Poté tomuto objektu přidat komponentu network view, u které nastavíme parametr State Synchronization a parametru Observed přiřadíme proměnnou *healt*.

```
public class FirstClass : MonoBehaviour
{
	public int healt = 100;
	void OnSerializeNetworkView(BitStream stream,
	NetworkMessageInfo info)
	{
		if (stream.isWriting)
		{
			int sendHealt = healt;
			stream.Serialize(ref sendHealt);
		}
		else
		{
			int recievedHealt = 0;
			stream.Serialize(ref recievedHealt);
			healt =  recievedHealt;
		}
	}
}
```

Za účelem vytvoření serverovské aplikace byla technologie Unity3D networking také zvažována. Ovšem všeobecným názorem komunity je, že se tato technologie nehodí pro MMO hry a využitím nezávislé sady knihoven pro vývoj serveru nám umožňuje mnohem větší volnost při realizaci komunikace.

Unity3D je velmi efektivní nástroj pro vývoj her. Jeho používání nám může usnadnit práci v mnoha směrech, ať už grafickým uživatelským rozhraním, předdefinovanými komponentami, enginem nebo obrovskou komunitou ochotnou pomoci s jakýmkoli problémem. Od verze 5.0 je volně dostupná i professional verze, což je pro nezávislé vývojáře velká výhoda. Enginy jako Unity3D a UnrealEngine umožňují díky uživatelskému rozhraní a obrovské komunitě vyvíjet hry jak profesionálům, tak i začátečníkům bez předchozí znalosti programování, proto je v těchto projektech velký potenciál.

5 NÁVRH APLIKACE

Tato kapitola se zabývá celým procesem návrhu: serveru, databáze, klientských knihoven pro Unity3D a také popisu použitých nástrojů. První část kapitoly se věnuje výběru vhodných nástrojů pro vývoj. Druhá část vybrané nástroje popisuje a ve třetí části se věnuje návrhu serverovské aplikace a knihoven pro její využití v Unity3D klientovi.

5.1 Výběr vhodných nástrojů

5.1.1 Výběr serveru

Prvním bodem při návrhu serveru bylo nutné vytvořit seznam kritérií, podle kterých vybereme vhodnou sadu knihoven pro samotnou tvorbu serveru. Hlavním kritériem pro výběr byla kompatibilita knihoven s vývojovým prostředím Unity3D, tedy možnost použití zvolených knihoven při vývoji v Unity3D. Druhým kritériem byla vhodnost použití sady knihoven pro tvorbu online her, nejlépe her typu MMO. Třetím kritériem byla licence dovolující použít server pro nekomerční účely a s možností mít ve stejný čas připojených alespoň 20 hráčů. Mezi hlavní kritéria patřila samozřejmě i stabilita. Seznam dále obsahoval několik podbodů, které byly při výběru vítány.

Seznam zvolených kritérií:

- Kompatibilita s Unity3d
- Vhodné pro hry typu MMO
- Licence
- Stabilita
- Podbody:
  - Jazyk sady knihoven (C#)
  - Osvědčení
  - Velká komunita vývojářů

Při širším výběru byla využita hlavně komunitní fóra zabývající se vývojem her, kde vláken týkajících se výběru vhodného serveru bylo nespočet. Většina vývojářů přispívajících do těchto vláken už daný problém řešila nebo několik serverů využila ve vlastním projektu. Do užšího výběru pak byli zvoleni tito kandidáti: Photon Server [1], SmartFoxServer 2x [2] a Lidgren [3]. U těchto kandidátů pak bylo nutné projít si dokumentaci na webových stránkách a podle toho vybrat. Pomocí informací nalezených na webových stránkách byla vytvořena tabulka podle zvolených kritérií:

	Photon Server	SmartFoxServer	Lidgren
Kompatibilita s Unity3d	ANO	ANO	ANO
Vhodnost pro MMO hry	ANO	ANO	ANO
Free licence	100 hráčů	100 hráčů	neomezeno
Stabilita	Stabilní	Stabilní	Stabilní
Jazyk sady knihoven	C#	Java	C#
Osvědčení v projektech	Využito v mnoha komerčních i nekomerčních projektech	Využito v mnoha komerčních i nekomerčních projektech	doplnit
Komunita	velká	velká	střední

Tabulka 2: Kritéria pro výběr vhodné sady knihoven

Jako nejvhodnější kandidát byl nakonec zvolen Photon Server, který splňoval všechna zvolená kritéria. Navíc podporuje širokou škálu klientských platforem jako: Unity3D, .NET, Android, iOS, Linux, Mac atd.

5.1.2 Výběr databázového systému

Při výběru vhodného databázového systému nebyl zvolen žádný postup. Byl vybrán MySQL systém a to hlavně z důvodu předchozích zkušeností s tímto systémem. Také kvůli jeho bezplatné licenci a obrovské podpory ze strany komunity.

5.2 Photon Server

Jedná se o multiplatformní vývojový framework určený pro vývoj her, navržený německou společností ExitGames. Obsahuje knihovny pro vývoj serverovské části, ale i pro vývoj klientské části, u které podporuje velké množství platforem. Komunikace mezi serverem a klienty probíhá synchronně v reálném čase, kdy klienti mohou při komunikaci využívat rozdílné platformy. Komunikace je založena na UDP protokolu, u kterého je možné si zvolit spolehlivý nebo nespolehlivý přenos a je využita technologie RPC. Vývoj v tomto frameworku probíhá v jakémkoliv .NET jazyce například C# nebo Managed C++.

Tabulka 3. zobrazuje klientské platformy a vývojové softwary, ve kterých je možno při vývoji použít Photon framework. [1]

Podporované platformy	Frameworky pro vývoj v daných platformách
Unity3D	Po exportu ve vývojovém prostředí: Všechny podporované
.NET	.NET, Windows 8 Phone, Windows 8 RT
Cocos2d-x	Android NDK, iOS, Marmalade, Windows
Samostatné platformy	Android, Andriod NDK, Flash, iOS, JavaScript, Linux, Mac OS X, Playstation Mobile, Windows
Unreal Engine	Android NDK, Windows, Mac OS X, iOS
Marmalade	Po exportu ve vývojovém prostředí: Všechny podporované
Xamarin	Po exportu ve vývojovém prostředí: iOS, Android, Windows, Mac OS
Corona	Po exportu ve vývojovém prostředí: iOS, Android, Windows Phone, Kindle

Tabulka 3: Podporované platformy a vývojové prostředí pro Photon klienty

5.2.1 Architektura Photon serveru

Architektura Photon Serveru je zobrazena na obr. 10 a skládá se z jádra a kontrolní aplikace zastřešující Photon framework a aplikace vytvořené pomocí frameworku. V jádře Photon serveru se vývojáři snažili využít co nejefektivněji technologie, které umožňují komunikaci s klienty na různých platformách. Kontrolní aplikace se vykonává pomocí .NET CLR (Common Language Runtime) a ten je provozován Photon jádrem. Technologie, které jsou uvedeny v architektuře Photon Serveru jsou popsány níže.

Obr. 10: Architektura Photon Serveru

Jádro

Jádro Photon Serveru je z důvodu výkonu napsáno v jazyce native C++. Pro co nejefektivnější manipulaci se sockety využívá technologii IOCP a podporuje 4 typy protokolů: spolehlivý UDP, TCP, Webové sockety a HTTP.

Technologie IOCP – I/O Completion ports, je technlologie poskytující efektivní vláknový model, sloužící pro zpracování více asynchronních vstupních/výstupních požadavků ve víceprocesorovém systému. Když proces vytvoří I/O Completion port, systém vytvoří přidruženou frontu pro požadavky s účelem obsluhovat tyto požadavky. Proces, který obsluhuje mnoho souběžných asynchronních požadavků může pracovat efektivněji se spojením I/O Completion ports a předem přidělenou skupinou vytvořených vláken, které čekají na přidělení práce (thread pool). [12]

Protokoly -

- **UDP** - Protokol UDP (User Datagram Protocol) je standard sady protokolů TCP/IP definovaný ve specifikaci RFC 768, User Datagram Protocol (UDP), patřící do transportní vrstvy. Protokol UDP se využívá pro rychlý a nenáročný přenos dat bez zajištění spolehlivosti mezi hostiteli TCP/IP. Protokol UDP poskytuje nespojované datagramové služby, snažící se doručit data všemi dostupnými prostředky. Nezaručuje však doručení datagramů ani správné pořadí přenosu. Pokud je požadována spolehlivá komunikace, je nutné použít TCP protokol nebo spolehlivý UDP protokol. [13]
- **RUDP** – Protokol RUDP (Reliable User Datagram Protocol) je protokol transportní vrstvy zaměřený na řešení, pro která je UDP protokol moc jednoduchý (je požadováno přijetí paketů ve správném pořadí) a TCP protokol je na dané řešení příliš složitý nebo nákladný na přenos. Protokol RUDP implementuje funkce podobné TCP protokolu, ale s menší režií. RUDP tedy rozšiřuje UDP o tyto funkce: potvrzení přijetí paketu, přeposílání ztracených paketů, protokol s posuvným okénkem (Sliding window protocol) a bufferování s kontrolou přetečení paměti (Buffer overflow), které je vhodnější při posílání menšího množství dat v jednu chvíli, než streamování v reálném čase. [14]
- **TCP** – Protokol TCP (Transmission Control Protocol) je standard sady protokolů TCP/IP definovaný ve specifikaci RFC 793, Transmission Control Protocol (TCP), patřící do transportní vrstvy. TCP protokol se používá pro spolehlivý přenos dat, zaručuje tedy přijetí dat nebo detekci chyby a přijetí dat ve správném pořadí. Při využití TCP mezi sebou hostitelé vytvoří spojení, přes které obousměrně posílají data. [15]

- **Webové sockety –** Web socket protokol je standard definovaný ve specifikaci RCF 6455, patřící do transportní vrstvy. Jedná se o nezávislý protokol založený na TCP poskytující plně duplexní komunikační kanály přes jedno TCP spojení. Protokol je navržen pro použití ve webových prohlížečích a serverech, ale je možné jej použít v jakékoliv klientské aplikaci či serveru. [16]
- **HTTP** – Protokol HTTP (Hypertext Transfer Protocol) je standard definovaný ve specifikaci RCF 2616, Hypertext Transfer Protocol (HTTP), patřící do aplikační vrstvy. Protokol je určený pro výměnu hypertextových dokumentů ve formátu HTML. Jedná se o protokol pracují na principu požadavek/odpověď v komunikaci klient server.

Kontrolní aplikace

Kontrolní aplikace se vykonává pomocí .NET CLR (Common Language Runtime) a ten je provozován Photon jádrem. Kontrolní aplikace obsahuje námi vytvořené serverovské aplikace běžící na Photonu. Námi vytvořené aplikace mohou být napsány v jakémkoliv .NET jazyce.

CLR (Common Language Runtime) [17] – jedná se o virtuální stroj řídící správné vykonávání .NET programů a je součástí .NET frameworku. Mezi nejdůležitější část CLR patří služba just-in-time-compilation. Jedná se o kompilaci prováděnou při vykonávání programu, v tomto případě se převádí zkompilovaný kód do strojového kódu, který CPU počítače rovnou provede. Mezi další služby, které poskytuje CLR patří: správa paměti, zpracovávání vyjímek, garbage collection, bezpečnost a zpráva vláken a bezpečné typování.

5.2.2 Základní pojmy

Aplikace

Aplikace je serverovská část logiky hry. Všechny funkce hry zahrnující například technologii RPC nebo ukládání dat jsou realizovány v této části. Zkompilovaný kód v podobě knihoven je následně načten pomocí nativního jádra Photonu. Aplikace, které mají být spuštěny při spuštění jádra a počáteční parametry spouštěných aplikací jsou definovány v konfiguračním XML souboru. Photon umožňuje běh více aplikací najednou, přičemž každá může mít jiný účel.

Herní logika

Herní logika definuje, jak bude klient komunikovat se serverem. Realizuje operace, eventy a vše ostatní co bude server zpracovávat. Výchozím bodem pro realizaci herní logiky je třída *Application* definovaná v knihovně *Photon.SocketServer.dll*.

Operace

Operace v Photonu jsou ekvivalentem RPC technologie. Klienti volají operace kdykoliv potřebují poslat data na server. Operace, které pracují s daty jsou realizovány na straně serveru. Aby server poznal, kterou operaci má vykonat, přikládá klient k posílaným datům také klíč operace. Operace probíhají mezi jedním klientem a serverem, ostatní klienti o nich nevědí. O přenos dat a serializaci dat na straně klienta a serveru se stará Photon framework. Serializace je automatická pro datové typy zobrazené v tabulce 4.

datový typ	velikost v bytech
byte	2
long	9
int	5
float	5
double	9
string	3 + vellikost(UTF8.GetBytes(string)
short	3
boolean	2
byte-array	5+1*velikost
int-array	5+4*velikost
hashtable	3 + velikost(klíčů) + velikost(hodnoty)
dictonary	3 + velikost(klíčů) + velikost(hodnoty)
Array of <typ>	4 + velikost(položek) - počet(položek)

Tabulka 4: Automaticky serializované datové typy

Pokud je potřeba, Photon dovoluje vytvořit si na straně klienta vlastní datový typ, který je možné následně posílat. Pro vytvoření je nutné implementovat metody pro serializaci a deserializaci tohoto datového typu a následně ho zaregistrovat pomocí funkce z frameworku. Tato implementace musí být provedena u každého klienta. Na straně serveru tento typ registrovat nemusíme (pokud server s datovým typem nebude pracovat), protože Photon Server umožňuje přeposílat neznáme datové typy.

Eventy

Photon eventy jsou navrženy pro předávání zpráv klientům. Každý event si nese byte hodnotu neboli klíč, podle kterého klient zpracuje přijatá data. Na rozdíl od operací, eventy mohou přijít kdykoliv, například, když se hráč přihlásí do hry.

Připojení a Timeouty

Photon RUDP protokol vytváří spojení mezi serverem a klientem, kdy pro připojení jsou posílány příkazy v rámci UDP balíčku mající pořadová čísla a příznaky, pokud jsou spolehlivé. Jestliže ano, přijatý konec je potvrzen příkazem. Spolehlivé příkazy jsou posílány v krátkých intervalech, dokud nepřijde potvrzení. Pokud potvrzení nedorazí, čas připojení vyprší. Toto připojení kontrolují obě strany nezávisle na sobě a obě strany mají svá pravidla na vyhodnocení připojení. Pokud je zjištěn timeout, provede se odpojení, ale pouze na té straně, kde byl timeout zjištěn. Jakmile druhá strana zjistí, že první neodpovídá nejsou jí posílány žádné další zprávy. Odpojení tedy nejsou synchronní. [18]

5.2.3 Požadavky

Operační systém

Doporučený operační systém pro vývoj, je Windows 8.1 x64, ale je možné použít i systémy Windows 7, 8. Na systémech Windows XP, Vista by měl být vývoj možný, ale tyto systémy nejsou nadále podporovány. Doporučený operační systém pro hostování je Windows server 2012 R2 x64, ale je možno využít i Windows Server 2008 R2.

.NET Framework

Doporučená verze je .NET SDK 4.5, ale je možné použít Microsoft .NET SDK 3.5 SP1 nebo .NET SDK 4.0. [18]

5.2.4 Konfigurační soubor

Hlavní konfigurační soubor je PhotonServer.config. Tento soubor je použit pro nastavení aplikací, které budou načteny při spuštění kontrolní aplikace Photonu. Pro tyto aplikace je možné dále nastavit IP adresy, porty nebo přednastavení hodnot, pro výkon serveru.[18]

Applications

Element Applications definuje, které aplikace budou spuštěny při startu instance. Její parameter default udává, která aplikace bude spuštěna, pokud není element Application nastaven. Element Application pak definuje aplikaci, která má být spuštěna. Element Application obsahuje tyto parametry:

- Name – název aplikace
- BaseDirectory – jméno hlavní složky, která musí obsahovat složku bin, ve které jsou soubory pro serverovskou aplikaci, složka je ve složce deploy
- Assembly – název knihovny, ve které je daná aplikace
- Type – namespace, ve kterém je obsažena daná serverovská aplikace
- ForceAutoRestart – pokud je hodnota nastavena na true, je aplikace automaticky restartována po změně souboru náležícího dané aplikaci, přičemž všechna existující připojení jsou zrušena, bez jejich náležitého odpojení

- WatchFiles - sleduje soubory, které následně způsobují restart
- ExcludeFiles – vyloučené soubory ze sledování

Na ukázce níže můžeme vidět nastavení aplikace pro spuštění.

```
<Applications Default="Master">
     <Application
          Name="Master"
          BaseDirectory="MasterServerIntro"
          Assembly="ServerToServer"
          Type="ServerToServer.MasterServer.MasterServer"
          ForceAutoRestart="true"
          WatchFiles="dll;config"
          ExcludeFiles="log4net.config">
     </Application>
</Applications>
```

UDPListeners a TCPListeners

Elementy UDPListeners slouží pro nastavení IP adresy a portu pro danou aplikaci. Na ukázce níže můžeme vidět nastavení těchto atributů pro danou aplikaci, kdy vytvoříme element UDPListener a nastavíme mu atributy IPAdress, Port, OverrideApplication. Atribut OverrideAplication znamená, že pokud se klient připojí k danému portu, skončí v dané aplikaci.

```
<UDPListeners>
     <UDPListener
          IPAddress="0.0.0.0"
          Port="5055"
          OverrideApplication="Master">
     </UDPListener>
</UDPListeners>
```

Pomocí elementu TCPListeners nastavujeme podobně jako v elementu UDPLiseners atributy IP adresu a port pro danou aplikaci. Dále však nastavujeme atributy PolicyFile a InactivityTimeout. Atribut InactivityTimeout znamená dobu v milisekundách pro přijímání dat. Pokud je tato doba překročena, je spojení přerušeno. Atribut PolicyFile nastavuje soubor určený pro povolení přístupu k daným portům.

```
<TCPListeners><TCPListener
          IPAddress="0.0.0.0"
          Port="4530"
          OverrideApplication="Master"
          PolicyFile="Policy\assets\socket-policy.xml"
          InactivityTimeout="10000">
</TCPListener></TCPListeners>
```

Dále je možné nastavit elementy PolicyFileListeners, WebSocketListeners.

Instance

Instanci definujeme pod elementem Configuration. V instanci definujeme všechny výše uvedené elementy, tedy UDPListeners, TCPListeners, Applications. V instanci dále definujeme atributy pro připojení a přenos dat mezi klientem a serverem. Instance jsou pak zobrazeny v kontrolní aplikaci jako tlačítka, pomocí kterých spouštíme tyto instance. Na ukázce níže můžeme vidět nastavení atributů pro instanci.

```
<Configuration>
     <Default
          MaxMessageSize="512000"
          MaxQueuedDataPerPeer="512000"
          PerPeerMaxReliableDataInTransit="51200"
          PerPeerTransmitRateLimitKBSec="256"
          PerPeerTransmitRatePeriodMilliseconds="200"
          MinimumTimeout="5000"
          MaximumTimeout="30000"
          DisplayName="VirtualWorldServer">

          <Applications>...</Applications>
          <TCPListeners>...</TCPListeners>
          <UDPListeners>...</UDPListeners>
     </Default>
<Configuration>
```

5.2.5 Ukázka využití Frameworku

Tato kapitola popisuje nejdůležitější třídy Photon frameworku pro vytvoření serverovské aplikace, třídy pro komunikaci s klientem na straně serveru a poté třídy pro komunikaci se serverem na straně klienta. Ukázky jsou zobrazeny v jazyce C#.

Serverovská část

Na ukázce níže můžeme vidět základní strukturu třídy pro vytvoření serverovské aplikace. Třída *SimpleServer* musí dědit ze třídy *Photon.SocketServer.ApplicationBase*, která je základní třídou pro serverovskou aplikaci v photon frameworku. Jedná se o abstraktní třídu, tudíž můžeme implementovat abstraktní metody *CreatePeer()*, *Setup()*, *TearDown()*. Metody *Setup()* a *TearDown()* jsou volány automaticky při spuštění a vypnutí serveru. Metoda *CreatePeer()* vrací třídu nebo potomka třídy *PeerBase*. *PeerBase* je základní třída pro připojení a komunikaci se serverem. Po vytvoření připojení se tedy zavolá metoda *CreatePeer*. Vstupní parametr *InitRequest* metody *CreatePeer(InitRequest initRequest)* je třída poskytující inicializační požadavek.

```
using Photon.SocketServer;
public class Simple : ApplicationBase
{
    protected override PeerBase CreatePeer(InitRequest
initRequest)
    {
        //volána při připojení klienta
    }

    protected override void Setup()
    {
        //volána při startu serveru
    }

    protected override void TearDown()
    {
        //volána při vypnutí serveru
    }
}
```

Aby klient mohl se serverem komunikovat, je nutné vytvořit třídu dědící z třídy *PeerBase*. Strukturu této třídy můžeme vidět na ukázce níže. PeerBase je abstraktní třída, můžeme tedy implementovat metody *OnDisconnect()* a *OnOperationRequest()*. Metoda *OnDisconnect()* je volána při odpojení klienta. Metoda *OnOperationRequest()* je volána při požadavku klienta, vstupní parametry metody jsou *OperationRequest* a *SendParameters*. Třída *OperationRequest* obsahuje požadavek klienta a struktura *SendParameters*, která obsahuje informace o požadavku, například zda je zakódován nebo po jakém kanálu je posílán. Konstruktor třídy PeerBase obsahuje 2 vstupní parametry, *IrpcProtokol* a *IphotonPeer*. Interface *IrpcProtokol* specifikuje protokol, kterým bude klient se serverem komunikovat. Interface *IphotonPeer* obsahuje informace, proměnné a funkce o připojeném klientovi.

```
using PhotonHostRuntimeInterfaces;
using Photon.SocketServer;
class ClientPeer : PeerBase
{
    public ClientPeer(IRpcProtocol protocol, IPhotonPeer
unmanagedPeer)
        : base(protocol, unmanagedPeer)
    {

    }
```

```
          protected override void
OnDisconnect(PhotonHostRuntimeInterfaces.DisconnectReason
reasonCode, string reasonDetail)
          {
            //volána při odpojení klienta
          }

          protected override void
OnOperationRequest(OperationRequest operationRequest,
SendParameters sendParameters)
          {
            //volána při požadavku klienta
          }
     }
```

Po přidání řádku vracejícího třídu *ClientPeer* do metody *CreatePeer()* ve třídě *SimpleServer*, server může komunikovat s klientem.

```
     return new ClientPeer(initRequest.Protocol,
initRequest.PhotonPeer);
```

Clientská část

Na ukázce níže můžeme vidět základní strukturu třídy sloužící pro přijetí dat ze strany serveru. Tato třída dědí po interface *IphotonPeerListener*, který slouží pro zpětné volání klientské strany. Metody implementované v ukázce níže jsou metody zděděného interface a stručný popis je uveden v komentářích těchto metod. Parametry *StatusCode* a *DebugLevel* jsou výčty obsahující hodnoty pro status připojení a pro informace o chybě. Parametr *EventData* obsahuje odpověď ze serveru v podobě klíče a dat. Parametr *OperationRespond* je třída obsahující odpověď v podobě kódu operace, dat a návratového kódu.

```
     using ExitGames.Client.Photon;
     class ClientPhotonPeer : IPhotonPeerListener{
          public void DebugReturn(DebugLevel level, string
message)
          {
               //zavolá se při chybě v aplikaci
          }
          public void OnEvent(EventData eventData)
          {
               //zavolá se při přijetí eventu
          }
```

```
        public void OnOperationResponse(OperationResponse
operationResponse)
        {
            //zavolá se při přijetí odpovědi
        }
        public void OnStatusChanged(StatusCode statusCode)
        {
            //zavolá se při změně statusu klienta
        }
    }
```

Data posílající se ve třídách *EventData*, *OperationResponse,* ale i *OperationRequest* jsou uložena v abstraktním datovém typu *Dictionary<klíč, hodnota>* v podobě *Dictionary<byte, object>*. *Dictonary* je kolekce jejíž hodnoty nejsou indexovány pomocí číselného indexu, ale klíče.

Pro připojení a odesílání dat na server se používá třída PhotonPeer. Pro inicializaci je nutné této třídě jako vstupní parametr předat třídu dědící po interface IphotonPeerListener. Deklaraci a inicializaci můžeme vidět na ukázce níže.

```
    PhotonPeer peer = new PhotonPeer(new ClienPhotontPeer());
```

Pro připojení k serveru je nutné zavolat metodu Conect(), třídy PhotonPeer. Na ukázce níže můžeme vidět volání metody Connect(), která obsahuje 2 vstupní parametry typu string adresa serveru a název aplikace, ke které se klient připojuje.

```
    peer.Connect("Adresa serveru", "název aplikace");
```

Nejdůležitější metoda na straně klienta je metoda Service(). Tato metoda řídí, kontroluje a organizuje všechny zbývající odpovědí a eventy ve vyrovnávací paměti a odesílá požadavky z fronty. Tato metoda by měla být pravidelně volána minimálně 2 krát za sekundu. Na ukázce níže můžeme vidět volání metody.

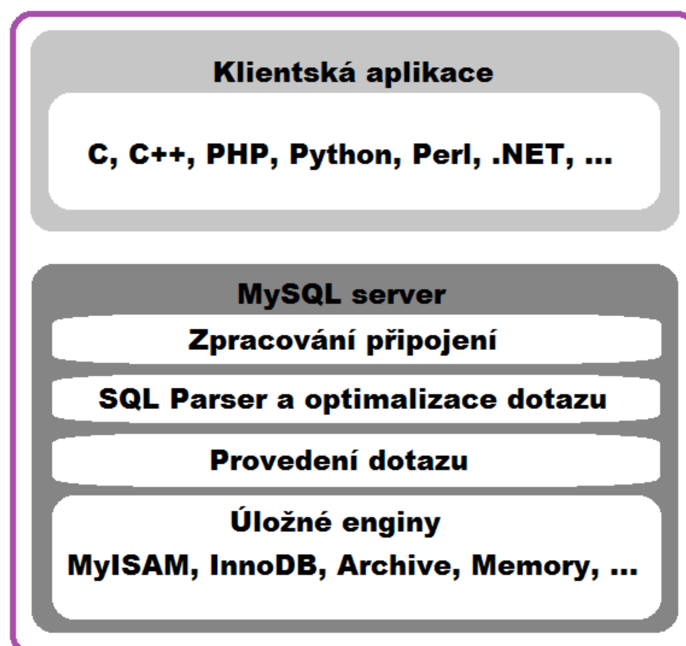
```
    peer.Service();
```

Požadavek na server je odesílán pomocí metody OpCustom(). Tato metoda obsahuje 4 parametry, první parametr je datového typu byte a určuje operaci, která se má na serveru provést. Druhým parametrem jsou data uložená v kolekci Dictonary. Třetím parametrem je bool proměnná nastavující spolehlivou nebo nespolehlivou komunikaci a posledním parametrem je byte určující kanál, po kterém budou data odeslána. Ukázka níže ukazuje strukturu metody.

```
    peer.OpCustom(byte operacniKod, Dictionary<byte,
object>() data, bool poslatSpolehlive, byte kanál);
```

Kanály jsou používány pro určení priority zpráv, ale také pro určení závislosti na jiné operaci. Závislost na jiné operaci znamená, že pokud má být některá operace znovu přeposlána, provoz v jiných kanálech nebude pozastaven, ale v kanálu se stejnou hodnotou ano, dokud nebude operace k přeposlání obdržena.

5.3 MySQL



Obr. 11: Architektura MySQL databázového systému

MySQL je databázový systém, vytvořený švédskou firmou MySQL AB, nyní vlastněný společností Sun Microsystems. Jde o multiplatformní a volně šiřitelný software. Pro jeho snadnou implementaci a výkon je to jeden z nejpoužívanějších systémů. Na obr. 11 můžeme vidět architekturu MySQL. Vrstva, která je úplně nahoře, obsahuje služby, jež obsluhují většinu potřebných nástrojů pro komunikaci s klientem. Ve druhé vrstvě se nachází valná část mozku MySQL, včetně kódu pro rozbor (parsing), analýzu, optimalizaci a pro všechny zabudované funkce. Třetí vrstva obsahuje úložné enginy, ty mají na starosti ukládání a získávání všech dat uložených v MySQL. Server komunikuje s úložnými enginy prostřednictvím API úložných enginů. Což v obrázku vidíme jako provedení dotazu. API obsahuje několik desítek nízkoúrovňových funkcí, které provádějí operace jako "zahájit transakci" nebo "získat řádek, který má tento primární klíč". Úložné enginy nedělají rozbor SQL a nekomunikují mezi sebou, jednoduše pouze odpovídají na požadavky serveru. [19]

5.4 Návrh databáze

Návrh databáze spočíval ve vytvoření ER diagram, který obsahuje 2 tabulky. Tabulku pro správu přihlašování uživatelů a tabulku pro správu postavy ve hře. Tabulky mezi sebou mají vztah 1:1, tedy jeden přihlašovací účet může mít jednu postavu. Na obr. 12 můžeme vidět výsledný ER diagram databáze.

Obr. 12: ER diagram výsledné databáze

Tabulka Login slouží pro správu uživatelských účtů. Obsahuje atributy: Login_ID – id účtu s primárním klíčem, Login_Name – uživatelské přihlašovací jméno, Login_Password – zahashované uživatelské přihlašovací heslo, Login_Improvment – řetězec pro hashování, Login_Algorithm – zkratka hashovacího algoritmu, Login_Created – datum vytvoření účtu.

Tabulka Player slouží pro správu postavy hráče. Obsahuje tyto atributy: Player_ID – id hráče s primárním klíčem, Player_Login_ID – id uživatelského účtu s cizím klíčem, Player_name – jméno hráče, Player_Position – řetězec pozice hráče ve světě, Player_Rotation - řetězec otočení hráče ve světě, Player_Area – areál, ve kterém se hráč nachází, Player_Sex – pohlaví hráče.

Po vytvoření ER diagramu byl v modeleru vygenerován script, pomocí kterého jsme databázi vytvořili na serveru. Pro hostování databáze byl zvolen WampServer 2.5, který využívá MySQL.

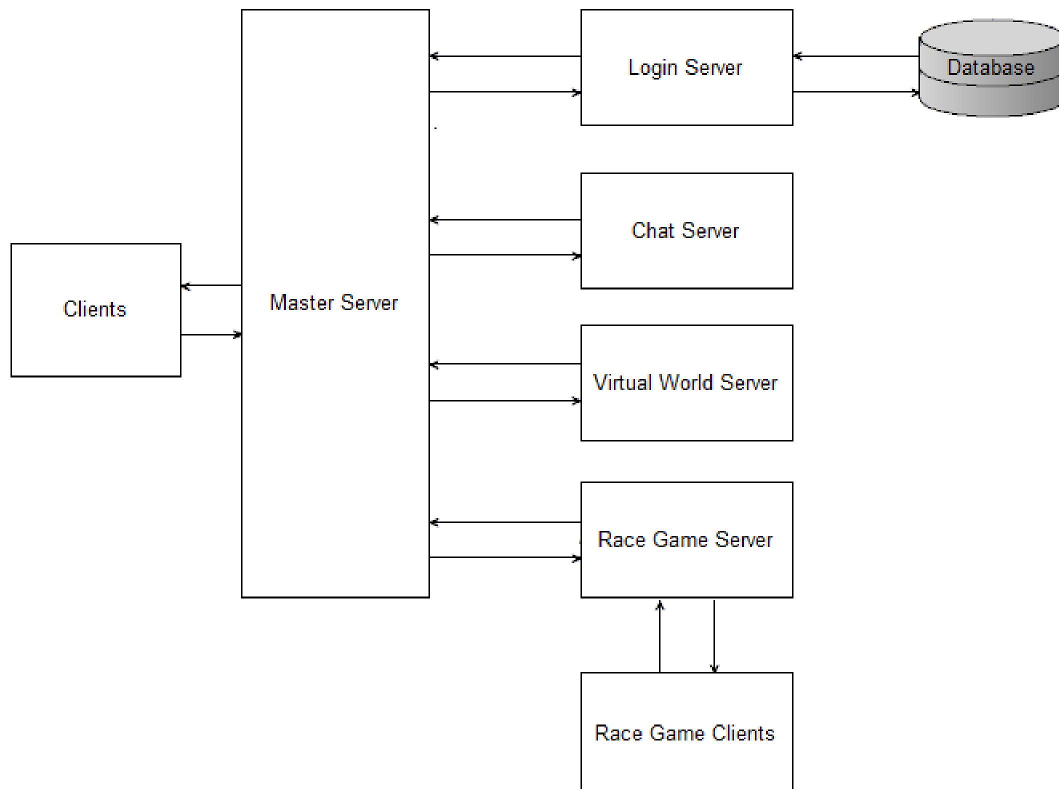
5.5 Návrh Serveru

Kritéria návrhu

Při návrhu serverovské aplikace se vycházelo z myšlenky, že se musí jednat o vícevláknovou aplikaci, kdy jednotlivá vlákna komunikují mezi sebou a s klienty pomocí hlavního vlákna. Hlavní vlákno tedy jen přeposílá požadavky, eventy a odpovědi adresátovi. Ostatní vlákna podle potřeby zpracovávají přijaté požadavky a odesílají odpovědi. Každé vlákno zpracovává jiné typy požadavků, například jen požadavky týkající se chatu a komunikace s databází musí probíhat jen z jednoho vlákna.

Dalšími požadavky týkají se serverovské aplikace, bylo jednoduché vytvoření nového vlákna a jednoduché přidání nové operace, zpracovávající nový požadavek.

Realizace návrhu



Obr. 13: Návrh serverovské aplikace

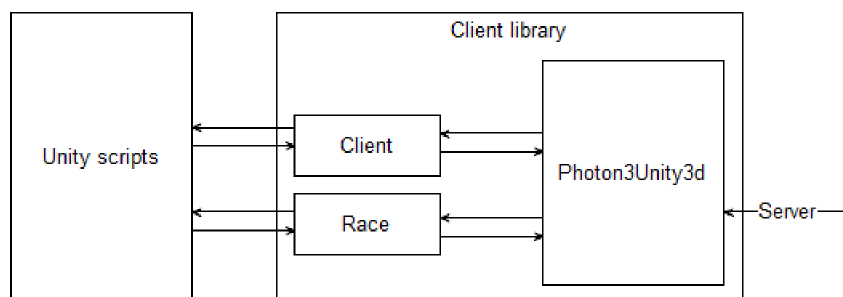
Na obr. 13 můžeme vidět výsledný návrh serverovské aplikace. Serverovská aplikace je navržena jako vícevláknová aplikace, kdy hlavní vlákno Master Server je bod, přes který komunikují klienti s ostatními vlákny a samotná vlákna mezi sebou. Vlákno Login Server komunikuje jako jediné s databází, může tedy vybírat, ukládat a upravovat data v databázi. Dále se Login Server stará o přihlašování a odhlašování hráčů do hry a registraci uživatelů. Vlákno Chat Server zpracovává požadavky týkající se předávání textových zpráv mezi hráči a vlákno VirtualWorld Server zpracovává požadavky týkající se pohybu hráčů ve virtuálním světě. Posledním vláknem je RaceGame Server zpracovávající požadavky týkající se přihlášení, startu a odhlášení ze závodní hry. Vlákno RaceGame Server jako jediné může komunikovat s klientem a to pouze při průběhu závodní hry.

5.6 Návrh knihoven pro Unity3d klienta

Kritéria návrhu

Při návrhu knihoven pro komunikaci se serverem musí klientská část obsahovat třídy pro komunikaci s Master a RaceGame Serverem a odesílání dat musí probíhat synchronně.

Realizace návrhu



Obr. 14: Návrh klientské aplikace

Na obr. 14 můžeme vidět výsledný návrh klientské knihovny. Knihovna využívá pro komunikaci se serverem knihovnu Photon3Unity3d. Tato knihovna je dodávána společně s Photon frameworkem a jedná se o klientskou knihovnu určenou pro Unity3D klienta. Programátor komunikující s navrženou klientskou knihovnou pomocí unity scriptu pracuje s třídami určenými pro komunikaci s Master a RaceGame Serverem, v obr. 14 jsou označeny jako *Client* a *Race*, které potom využívají knihovnu Photon3Unity3d. Soubory tříd *Client* a *Race* tedy zpracují přijatá data z unity scriptu a odešlou na server, a také přijímají data ze serveru, které zpracují a pošlou unity scriptu.

6 REALIZACE APLIKACE

Tato kapitola popisuje nejdůležitější třídy, metody a proměnné, které nejsou standardem při používání Photon frameworku a jsou použity v aplikaci. V první části je popsána serverovská aplikace a v druhé klientské knihovny.

6.1 Serverovská aplikace

MasterServer

Základním kamenem serverovské aplikace je třída *MasterServer*, která dědí ze třídy *ApplicationBase*. Nejdůležitější metodou třídy *MasterServer* je metoda *CreatePeer,* sloužící pro vytvoření třídy komunikující s klientem a se serverem. Dále *MasterServer* obsahuje kolekce, ve kterých jsou uloženi přihlášení uživatelé a podservery.

Na ukázce níže můžeme vidět implementaci metody *CreatePeer*. Pokud se jedná o server vrací tato metoda novou instanci třídy *IncomingSubServerPeer*, pokud ne vrátí instanci třídy *Unity3dPeer* komunikující s klientem. Kontrola se provádí v metodě *IsSubServer()*, kontrolující port. Při vytváření komunikačních tříd *IncomingSubServerPeer, Unity3dPeer* je předávána instance třídy *MasterServer*, se kterou se poté v daných třídách pracuje a vstupní požadavek *InitRequest.* Nad metodou *IsSubServer()* jsou deklarovány kolekce pro připojené uživatele *ConnectedClients* a pro přihlášené servery *SubServers.* Třída SubServerCollection je třída uchovávající a kontrolující přihlášené servery.

```
	public SubServerCollection SubServers { get; protected
set; }
	public Dictionary<string, Unity3dPeer> ConnectedClients;


	protected override PeerBase CreatePeer(InitRequest
initRequest)
	{
		if (IsSubServer(initRequest))
			return new IncomingSubServerPeer(initRequest,
this);
		return new Unity3dPeer(initRequest, this);
	}
```

IncomingSubServerPeer

Tato třída přijímá odpovědi a eventy od podserverů a díky předaní třídy *MasterServer* v parametru konstruktoru, tato třída přeposílá odpovědi a eventy připojeným klientům. Pro větší přehlednost probíhá přeposílání dat klientům ve třídách *Unity3dPeerRespondHandler* a *Unity3dPeerEventHandler*.

Jediný požadavek, který tato třída přijímá je registrace podserveru do třídy *MasterServer.* Metoda zpracovávající požadavek registrace je zobrazena níže.

```
	private OperationResponse
HandleRegisterSubServerRequest(Photon.SocketServer.OperationRe
quest operationRequest)
	{
		var registerRequest = new
RegisterSubServer(Protocol, operationRequest);
		if (!registerRequest.IsValid)
		{
			string msg = registerRequest.GetErrorMessage();
			return new
OperationResponse(operationRequest.OperationCode)
{ DebugMessage = msg, ReturnCode =
(short)ServerToServer.ErrorCode.OperationInvalid };
		}

		_serverID = registerRequest.ServerId;
		SubServerType =
(ServerToServer.SubServerType)registerRequest.ServerType;
		_server.SubServers.OnConnect(this);

		return new
OperationResponse(operationRequest.OperationCode);
	}
```

V metodě *HandleRegisterSubServerRequest*() vytvoříme proměnou *registerRequest* ze třídy *RegisterSubServer* specifikující data contract. Zkontrolujeme, zda je registerRequest validní data contract. Dále nastavíme proměnné specifikující server, pro který bude třída *IncomingSubServerPeer* přijímat data a vrátíme odpověď.

Unity3dPeer

Třída přijímá požadavky od klienta a přeposílá je pomocí proměnné *MasterServer* podserverům. Dále detekuje zda se klient neodpojil a obsahuje proměnné o připojeném uživateli. Na ukázce níže můžeme vidět zpracování přijatého požadavku od klienta v metodě *OnOperationRequest()*.

```
	protected override void
OnOperationRequest(OperationRequest operationRequest,
SendParameters sendParameters)
	{
		if(operationRequest.Parameters.
ContainsKey((byte)ServerToServer.ParameterCode.Unity3DUserID))
			operationRequest.Parameters.
Remove((byte)ServerToServer. ParameterCode.Unity3DUserID);
```

```
            operationRequest.Parameters.
Add((byte)ServerToServer.ParameterCode.Unity3DUserID, UserId);

            switch (operationRequest.OperationCode)
            {
                  case (byte)UnityPeerCode.Login:
                        if (!Logged && !
CheckLoggedPeer((string)operationRequest.Parameters[(byte)Unit
yParameterCode.UserName]))
_server.SubServers.LoginServer.SendOperationRequest(operationR
equest, new SendParameters() { ChannelId = 0 });
                        else if
((byte)operationRequest.Parameters[(byte)UnityParameterCode.Su
bOperationCode] ==
(byte)UnitySubOperationCode.RegisterSecurely)
SendFailedLoginRequest();
                        break;
                  case (byte)UnityPeerCode.Chat:
                        //kód pro přeposlání Chat serveru
                        break;
                  case (byte)UnityPeerCode.VirtualWorld:
                        //kód pro přeposlání VirtualWorld serveru
                        break;
                  case (byte)UnityPeerCode.RaceGame:
                        //kód pro přeposlání RaceGame serveru
                        break;
                  default:
                        //nic neprovede
                        break;
            }
      }
```

V první části metody je požadavku přiděleno id uživatele a poté podle operačního kódu v požadavku je provedeno přeposlání podserveru. V ukázce je uveden kód pro přeposlání požadavku třídě *LoginServer*, kde je prvně kontrolováno, zda uživatel už není přihlášen a pokud ano a požadavek operace je registrace, tak je uživateli přeposlána zpráva o neúspěšné registraci pomocí metody *SendFailedLoginRequest()*.Pokud je přihlášen a jedná se o operaci přihlášení neprovede se nic.

SubServer

Jedná se o abstraktní třídu pro podservery, tedy každý podserver, který je vytvořen musí dědit z této třídy. Tato třída obsahuje proměnné informující o podserveru, například UDP adresu, TCP adresu, typ serveru atd. Dále obsahuje metody pro navázání spojení s MasterServerem a abstraktní metodu *AddHandlersToServerPeer()*, která slouží pro přidání tříd, které budou zpracovávat operace daného podserveru.

```
	private void ConnectToMaster()
	{
		if (!ConnectToServer(MasterEndPoint, "Master",
MasterEndPoint))
			return;
	}
```

Metoda *ConnectToMaster()* je volána při spuštění aplikace z metody *Setup()*. *ConnectToMaster()* volá metodu *ConnectToServer(),* což je metoda Photon frameworku pro navázání spojení se serverem. Pokud se navázání spojení nepovede, volá se automaticky metoda frameworku *OnServerConnectionFailed()*, ve které je prováděn reconnect.

Jakmile dojde k navázání spojení zavolá se metoda *CreateServerPeer()* vracející třídu *OutgoingMasterServerPeer,* pomocí které komunikujeme s MasterServerem.

```
	protected override
Photon.SocketServer.ServerToServer.ServerPeerBase
CreateServerPeer(InitResponse initResponse, object state)
	{
		return MasterPeer = CreateMasterPeer(initResponse);
	}
```

V metodě *CreateMasterPeer()* vracíme nově vytvořenou instanci třídy *OutgoingMasterServerPeer*, ale také voláme metodu *AddHandlersToServerPeer()*, která přidává podserveru operace.

```
	protected virtual OutgoingMasterServerPeer
CreateMasterPeer(InitResponse initResponse)
	{
		var serverPeer = new
OutgoingMasterServerPeer(initResponse.Protocol,
initResponse.PhotonPeer, this);
		AddHandlersToServerPeer(serverPeer);
		return serverPeer;
	}
```

OutgoingMasterServerPeer

Třída, přes kterou podserver příjmá požadavky, eventy a odpovědi od MasterServeru a také slouží pro odesílání eventů, odpovědí a požadavků MasterServeru. Třída obsahuje kolekce *RequestHandlers*, *RespondHandlers* a *EventHandlers* obsahující třídy, které provádějí operace daného serveru, kdy při přijetí dat je daná kolekce prohledávána způsobem zobrazeným v ukázce níže.

```
    Handlers.PhotonRequestHandler handler;

    if (operationRequest.Parameters.
ContainsKey((byte)UnityParameterCode.SubOperationCode) &&
RequestHandlers.
TryGetValue(Convert.ToByte(operationRequest.Parameters[(byte)
UnityParameterCode. SubOperationCode]), out handler))
    {
        handler.HandleRequest(operationRequest);
    }
```

Další důležitou metodou této třídy je *Register()*, která je volána v konstruktoru a která posílá požadavek MasterServeru o přidání serveru do kolekce serverů.

```
    protected virtual void Register()
    {
        var request = new
OperationRequest((byte)ServerToServer.Operations.OperationCode
.RegisterSubServer, new RegisterSubServer()
        {
            ServerAdress =
_application.PublicIPAddress.ToString(),
            TcpPort = _application.SubServerTcpPort,
            UdpPort = _application.SubServerUdpPort,
            ServerId = SubServer.ServerID,
            ServerType = (int)_application.SubServerType
        });

        SendOperationRequest(request, new SendParameters());
    }
```

DBUsage

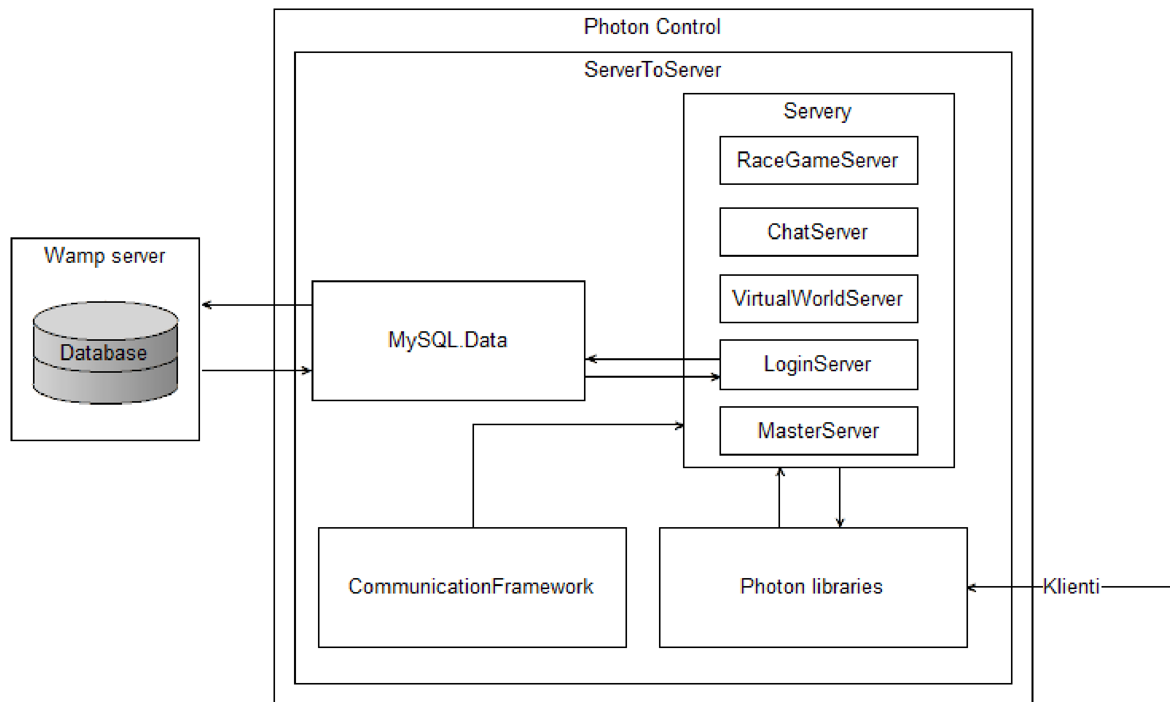
Třída komunikující s databází. Aby byla komunikace možná, je nutné použít knihovnu MySQL.Data. Pro aktualizaci a výběr dat jsou používány třídy MySQLDataAdapter a MySQLCommandBuilder.

Tuto třídu využívá jen třída *LoginServer*, která si při startu databázi stáhne a aktualizuje ji buď při vypnutí serveru nebo co dvě hodiny od zapnutí serveru.

Serverovská aplikace je realizována podle návrhu v předešlé kapitole. Obsahuje tedy jeden hlavní server a 4 podservery. Komunikace probíhá přes hlavní server jak mezi servery, tak s klienty, jen při závodní hře může klient komunikovat s podserverem samostatně. S databází komunikuje jen jeden podserver, kdy data má uložena v paměti a databázi aktualizuje každé 2 hodiny od zapnutí. Při komunikaci je potřeba znát kódy operací jak na

straně klienta, tak na straně serveru, proto byla realizována další knihovna s názvem CommunicationFramework, která potřebná data obsahuje.

Výsledná aplikace



Obr. 15: Hrubá struktura navržené serverovské aplikace

Na obr. 15 můžeme vidět výslednou aplikaci skládající se z knihovny ServerToServer, která využívá Photon knihovny, sloužící pro vytvoření serverů a komunikaci s klientem. Výsledná aplikace dále využívá knihovnu MySQL.Data pro komunikaci s databází a také knihovnu CommunicationFramework. Realizovanou aplikaci hostuje aplikace Photon Control. Vytvořenou databázi hostuje Wamp server.

6.2 Klientské knihovny

MyClientPeer

Hlavní klientská třída. V každé klientské aplikaci musí být tato třída deklarována. Třída slouží pro odesílání a přijímání dat ze serveru. Na ukázce níže můžeme vidět konstruktor, ve kterém jsou inicializovány důležité proměnné. Proměnná *MyPhotonPeer*, je třída *PhotonPeer*, pomocí které se připojujeme a komunikujeme se serverem. Třídě *PhotonPeer* musíme předat do konstruktoru třídu dědící po interface *IphotonPeerListener*, který přijímá data ze serveru. V tomto případě je to třída *ClientListener*. Přijatá data ze serveru musí být předána unity scriptu, k tomuto účelu jsou použity delegáty, které jsou deklarovány v class souboru ClientEventHandlers. Tento soubor obsahuje také třídu *ClientEventHandlers*, která pomocí deklarovaných delegátu vytvoří události, pomocí kterých unity script odchytává

přijatá data ze serveru. Třída HandleRequest zpracovává data, která mají být odeslána na server do podoby požadavku.

```
public MyClientPeer(string serverAddres, string appName)
{
      ClientEvents = new ClientEventHandlers();
      _listener = new ClientListener(this);
      MyPhotonPeer = new PhotonPeer(_listener);
      _handleRequest = new HandleRequest(this);
      _serverAdress = serverAddres;
      _applicationName = appName;
}
```

ClientListener

Tato třída přijímá eventy a odpovědi ze serveru, kontroluje status připojení a případné chyby při přenosu dat. Na ukázce níže můžeme vidět zpracování eventu, kdy podle kódu operace je zavolána metoda třídy *HandleEvent*, která přijatý event zpracuje a pošle unity scriptu.

```
public void OnEvent(EventData eventData)
  {
      switch ((UnityParameterReturnCode)eventData.Code)
      {
          case
UnityParameterReturnCode.OnlinePlayerSuccesfull:
              _handleEvent.OnLogPlayerEvent(eventData);
              break;
          case UnityParameterReturnCode.PlayerMove:
              _handleEvent.OnPlayerMove(eventData);
              break;
      }
  }
```

HandleRequest

Třída zpracovává data pro odeslání na server. Na ukázce níže můžeme vidět metodu posílající veřejnou zprávu. Metoda *PublicMessage()* volá metodu *SendRequest()*, které předává parametry: typ serveru, operace, data a jestli mají být data poslána zakódovaně. Metoda *SendRequest()* pak pomocí třídy *MyClientPeer* (proměnná *_peer*) data odešle serveru pomocí metody *OpCustom()*.

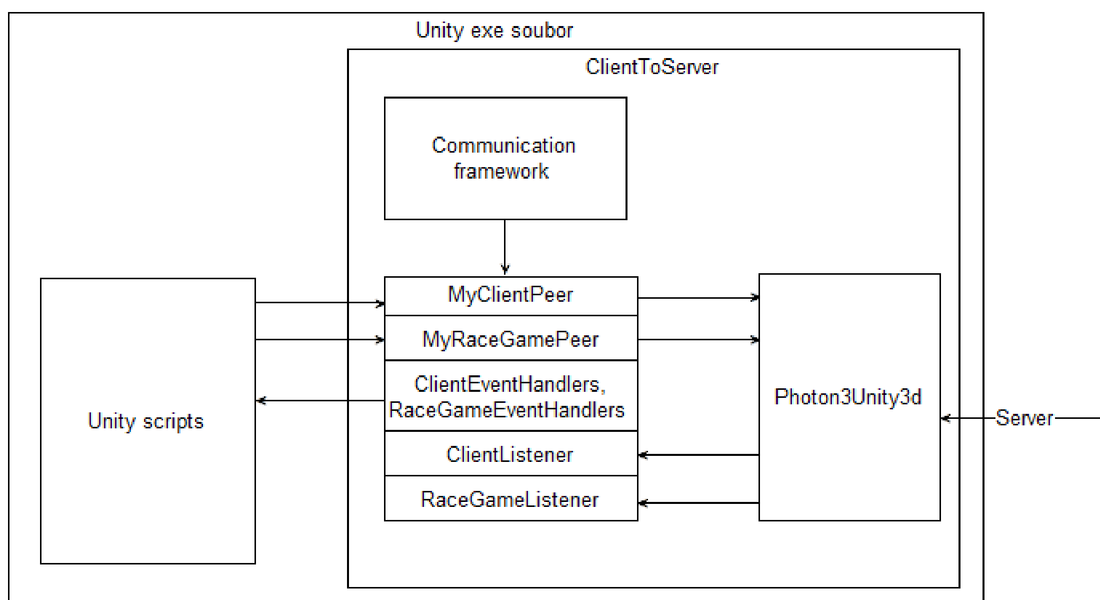
```
     public void PublicMessage(string startPlayer, string
message)
     {
          SendRequest(UnityPeerCode.Chat,
UnitySubOperationCode.ChatText, 1,
new Dictionary<byte, object>
          {
{ (byte)UnityMessageParameterCode.StartPlayer, startPlayer },
{ (byte)UnityMessageParameterCode.Message, message },
{ (byte)UnityParameterCode.Area, UnityArea.FirstArea},
          }, false);
}

     private void SendRequest(UnityPeerCode upc,
UnitySubOperationCode usoc, byte channelID, Dictionary<byte,
object> parameters, bool encrypt)
     {
          parameters.Add((byte)UnityParameterCode.
SubOperationCode, usoc);
          _peer.MyPhotonPeer.OpCustom((byte)upc, parameters,
true, channelID, encrypt);
     }
```

Komunikace se závodní hrou

Komunikace se závodní hrou probíhá podobně, jak bylo uvedeno v předchozích popisech tříd, přičemž nejdůležitější třídy, které používá jsou třídy *RaceGameListener*, *MyRaceGamePeer* a *RaceGameEventHandlers.*

Výsledná knihovna



Obr. 16: Struktura realizované klientské knihovny

Na obr. 16 můžeme vidět hrubou strukturu realizované aplikace. Navržená klientská knihovna jménem ClientToServer používá ke komunikaci Photon3Unity3d knihovnu, pro specifikaci operací využívá knihovnu CommunicationFramework a jména tříd, které jsou v obrázku zobrazeny jsou nejdůležitější třídy pro komunikaci se serverem a pro předávání dat unity srciptu.

Serverovská aplikace a klientská knihovna byly napsány v programovacím jazyce C# a pro vývoj bylo použito vývojové prostředí Microsoft Visual Studio 2012. Oba projekty byly napsány s pomocí objektově orientovaného programování a byl kladen důraz na to, aby další vývoj byl co nejjednodušší, hlavně v serverovské části. Výsledná serverovská aplikace je vícevláknový non-authoritative server využívající MySQL databázi pro uchování dat o uživatelích a hráčích. Realizovaná klientská knihovna je napsána primárně pro Unity3D klienta, ale mělo by být možné ji použít pro jakéhokoliv podporovaného klienta Photon frameworkem, bylo by nutné pouze změnit referenci na Photon klientskou knihovnu a popřípadě udělat malé úpravy v kódu.

7 ZÁVĚR

Cílem této diplomové práce bylo seznámit se s procesem tvorby počítačových her a herním enginem Unity3D a následně navrhnout a poté realizovat herní server, včetně aplikačního programového rozhraní a knihoven pro jeho využívání pro Unity3D klienta.

V první části se diplomová práce věnuje historii žánrů počítačových her a procesem jejich tvorby s důrazem na návrh, ve kterém jsou popsány důležité vlastnosti gamedesignéra, game design dokument a zhodnocení navržené hry. Následně jsou stručně popsány ostatní procesy vývoje.

Ve druhé části byly popsány základy herního enginu Unity3D a poté se kapitola věnuje Unity3D nástroji pro tvorbu multiplayerových her, nazvaného Unity networking. Nástroj je popisován hlavně z důvodu, že hlavní část práce byla navrhnout a realizovat herní server a knihovny pro jeho využívání, pro Unity3D klienta a Unity networking byl jeden z možných kandidátu jako nástroj pro vývoj.

Ve třetí části je popsán návrh a vybrané nástroje pro tvorbu serveru a klientských knihoven. Jako nástroj pro tvorbu serveru a klientských knihoven byl vybrán Photon server, což je multiplatformní framework, určený pro vývoj online her pro více hráčů. Při popisu Photon serveru byl kladen důraz na části frameworku, sloužící pro vytvoření serveru a komunikaci s klientem, a taktéž byly popsány části, sloužící pro vytvoření klientských knihoven. V návrhu jsou uvedena kritéria a výsledný popis navržených nástrojů s ilustracemi. Pro ukládání dat o uživatelích byl zvolen MySQL databázový systém, který byl stručně popsán a následně byl uveden návrh databáze pomocí v ER diagramu.

V poslední kapitole diplomové práce jsou popsány nejdůležitější třídy a metody realizovaných nástrojů.

Realizovaný herní server a klientské knihovny slouží pro komunikaci v rámci online hry pro více hráčů a závodní hru. Nástroje, které byly realizovány v této diplomové práci, jsou součástí projektu online hry pro více hráčů s možností hraní závodní hry, na kterém jsem pracoval se svými kolegy Jakubem Hamalem a Robertem Kováčem.

Základní myšlenkou tohoto projektu je propagace ÚAI FSI VUT v Brně, a také použití vytvořeného projektu s teoretickými poznatky diplomových prací pro další závěrečné práce na ÚAI, které by se měly zaměřit na rozvoj projektu.

Seznámil jsem se s problematikou tvorby počítačových her a herním enginem Unity3D. Navrhnul a realizoval herní server včetně aplikačního programového rozhraní a knihoven pro jeho využívání. Použitá technologie byla úspěšně zvládnuta a výsledná sada knihoven je navržena a realizována tak, aby byl její další vývoj co nejsnadnější. Vzhledem k úspěšnému dokončení projektu může být tato práce oporou při tvorbě serverovské aplikace pro online hry pro více hráčů.

8 SEZNAM POUŽITÉ LITERATURY

[1] Photon Server. *Exit Games.* [online]. ©2003 – 2015 [cit. 2015-04-30]. Dostupné z: *https://www.exitgames.com/en/OnPremise/*

[2] SmartFoxServer 2X. *SmartFoxServer.* [online]. ©2004-2012 [cit. 2015-04-30].Dostupné z: *http://www.smartfoxserver.com/products/sfs2x#p=intro*

[3] Lidgren-network-gen3. *Lidgren-network-gen3.* [online]. ©2010-2013 [cit. 2015-04-30]. Dostupné z: *https://code.google.com/p/lidgren-network-gen3*

[4] Historie vývoje počítačových her. *Root.cz.* [online]. ©1998-2015 [cit. 2015-04-30]. Dostupné z: *http://www.root.cz/clanky/historie-vyvoje-pocitacovych-her-1-cast-prvni-milniky/*

[5] What a Game Designer Needs to Know. *Scirra.* [online]. ©2015 [cit. 2015-04-30]. Dostupné z: *https://www.scirra.com/tutorials/249/what-a-game-designer-needs-to-know*

[6] The Five and a Half Filters of Game Creation. *Scirra.* [online]. ©2015 [cit. 2015-04-30]. Dostupné z: *https://www.scirra.com/tutorials/341/the-five-and-a-half-filters-of-game-creation*

[7] File:OXO emulated screenshot.png. *Wikipedia.* [online]. poslední aktualizace 27.6.2013[cit.2015-05-06].Dostupné z :*http://en.wikipedia.org/wiki/File:OXO_emulated_screenshot.png*

[8] The Process of Game Creation & the Game Design Document. *Digital Worlds – Interactive Media and Game Design.* [online]. 10.4.2008 [cit. 2015-05-07]. Dostupné z: *https://digitalworlds.wordpress.com/2008/04/10/the-process-of-game-creation-the-game-design-document*

[9] DOCUMENTATION, UNITY SCRIPTING LANGUAGES AND YOU. *Unity Blog.* [online]. Copyright © 2015 Unity Technologie [cit. 2015-05-09]. Dostupné z:*http://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you*

[10] Unity Manual. *Unity – Manual.* [online]. Copyright © 2015 Unity Technologies [cit. 2015-05-09]. Dostupné z: *http://docs.unity3d.com/Manual/index.html*

[11] Network Reference Guide. Unity – Manual. [online]. Copyright © 2015 Unity Technologies[cit.2015-05-10]. Dostupné z:*http://docs.unity3d.com/Manual/NetworkReferenceGuide.html*

[12] I/O Completion Ports. *Windows Dev Center.* [online]. © 2015 Microsoft [cit. 2015-05-10]. Dostupné z: *https://msdn.microsoft.com/en-us/library/windows/desktop/aa365198%28v=vs.85%29.aspx*

[13] User Datagram Protocol. *Wikipedia.* [online]. 29. 4. 2015 [cit. 2015-05-10]. Dostupné z: *http://en.wikipedia.org/wiki/User_Datagram_Protocol*

[14] Reliable User Datagram Protocol. *Wikipedia.* [online]. 17. 3. 2015 [cit. 2015-05-10]. Dostupné z: *http://en.wikipedia.org/wiki/Reliable_User_Datagram_Protocol*

[15] Transmission Control Protocol. *Wikipedia.* [online]. 7. 5. 2015 [cit. 2015-05-10]. Dostupné z: *http://en.wikipedia.org/wiki/Transmission_Control_Protocol*

[16] WebSocket. *Wikipedia.* [online]. 12. 5. 2015 [cit. 2015-05-10]. Dostupné z: *http://en.wikipedia.org/wiki/WebSocket*

[17] Common Language Runtime. *Wikipedia.* [online]. 8.4. 2015 [cit. 2015-05-10]. Dostupné z: *http://en.wikipedia.org/wiki/Common_Language_Runtime*

[18] Photon Server Intro | Exit Games. *Exit games.* [online]. ©2003 – 2015 [cit. 2015-05-16]. Dostupné z: *http://doc.exitgames.com/en/onpremise/current/getting-started/photon-server-intro*

[19] MySQL. *Wikipedia.* [online]. 3. 1. 2015 [cit. 2015-05-16]. Dostupné z: *http://cs.wikipedia.org/wiki/MySQL*

[20] HALL, Rick a Jeannie NOVAK. *Game development essentials*. Clifton Park, NY: Delmar/Cengage Learning, 2008, xxii, 271 p. ISBN 978-141-8052-676.