

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SÍŤOVÁ HRA: HON NA PONORKU

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

PETR JAŠEK

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SÍŤOVÁ HRA: HON NA PONORKU

NETWORK GAME: DESTROY THE SUBMARINE

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

PETR JAŠEK

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. IGOR SZÖKE, Ph.D.

BRNO 2012

Abstrakt

Tato bakalářská práce se zabývá vývojem jednoduché síťové počítačové hry. Práce popisuje základní schéma vývoje počítačové hry. Je v ní také provedena analýza různých možností síťové implementace. Výstupem práce je počítačový program psaný v jazyce C++. K implementaci síťového rozhraní byla použita knihovna **Boost 1.50.0**. Síťová komunikace byla implementována jako architektura klient-server s pomocí protokolu UDP. K implementaci grafického rozhraní a zvuku byla použita knihovna **Allegro 5.0.7**.

Abstract

This bachelor's thesis describes development of a simple network game. This thesis describes basic scheme of computer game development. Also an analysis of various implementation of network is made in this thesis. The output of this work is a computer program written in the C++ language. For implementation of the network the **Boost 1.50.0** library was used. Network communication was implemented as client-server architecture with usage of the UDP protocol. For implementation of graphics and sound, the **Allegro 5.0.7** library was used.

Klíčová slova

Allegro, C++, klient, ponorky, server, síťová hra, síťová komunikace

Keywords

Allegro, C++, client, network communication, network game, server, submarines

Citace

JASEK, Petr. *Síťová hra: Hon na ponorku*. Brno, 2012. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií.

Sítová hra: Hon na ponorku

Čestné prohlášení

Tímto prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Igora Szökeho, Ph.D. a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Petr Jašek

30. července 2012

Poděkování

Chtěl bych poděkovat Ing. Igorovi Szökemu, Ph.D. za vedení a rady, které mi poskytl při tvorbě této bakalářské práce. Dále bych chtěl poděkovat všem svým kamarádům, kteří se podíleli na testování a zajišťovali mi zpětnou vazbu. Poděkování patří také mým rodičům, kteří mi poskytovali všeobecnou podporu během celého mého studia.

© Petr Jašek, 2012.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod	3
2 Popis hry a uživatelského rozhraní	4
2.1 Vlastnosti hry a její „pravidla“	4
2.1.1 Ponorka	4
2.1.2 Zbraně	4
2.1.3 Mapy	5
2.1.4 Herní módy	5
2.2 Uživatelské rozhraní hry	6
3 Síťová komunikace	8
3.1 IPv4 nebo IPv6?	8
3.2 UDP nebo TCP?	8
3.2.1 TCP	9
3.2.2 UDP	9
3.2.3 Jiné možnosti	10
3.2.4 Zvolený protokol	10
3.3 Typ sítě z pohledu komunikace hráčů	10
3.3.1 Peer-to-Peer	10
3.3.2 Klient-Server	11
3.3.3 Zvolená architektura	11
3.4 Problémy architektury klient-server	12
3.4.1 Predikce na straně klienta	12
3.4.2 Extrapolace a interpolace	14
3.4.3 Synchronizace klientů se serverem	16
4 Implementace hry a testování	17
4.1 Použité knihovny	17
4.1.1 Boost	17
4.1.2 Allegro	17
4.2 Herní objekty	17
4.2.1 Grafická reprezentace objektů	17
4.2.2 Fyzická reprezentace objektů	19
4.2.3 Hlavní objekty	20
4.3 Detekce kolizí	21
4.3.1 CollisionShape	21
4.3.2 Dynamické zjišťování typu objektu	22
4.4 Implementace síťové komunikace	23

4.4.1	Průběh komunikace	23
4.4.2	Třídy používané pro zasilání dat	24
4.5	Implementace serveru	24
4.5.1	Zjišťování globální IP adresy	25
4.5.2	Logika hry	25
4.6	Implementace klienta	27
4.6.1	SoundManager	27
4.6.2	ResourceManager	27
4.6.3	GraphicManager	28
4.6.4	InputManager	28
4.6.5	NetworkManager	28
4.6.6	GameObjectManager	29
4.6.7	GameManager	30
4.7	Testování	30
5	Závěr	32
	Literatura	34
	Přílohy	34
A	Tabulka příkazů pro server	35
B	Typy zbraní	36
C	Obsah CD	37

Kapitola 1

Úvod

Hry za účelem zábavy a vzdělávání vznikají již od nepaměti. Od útlého dětství se pak stávají nedílnou součástí našeho života. V současnosti existuje velké množství různých typů her. Hry, které nevyžadují žádné speciální „pomůcky“, jako je hra „na honěnou“, hra „na schovávnou“ a jiné. Dále existuje nepřehledné množství deskových her, karetních her a také sporty lze zařadit mezi hry. Existují hry ve formě takzvaných hracích knih¹ a takzvané hry na hrdiny², jako je například Dračí doupě. V neposlední řadě mezi hry musíme zařadit i hry počítačové. Právě vývojem jedné takové počítačové hry se zabývá tato práce.

Hra nese název *Hon na ponorku*. Jedná se o jednoduchou arkádovou hru³ pro více hráčů. V kapitole 2 je hra formálně popsána, jsou vysvětlena její pravidla a je popsán návrh grafického vzhledu a uživatelského rozhraní. Kapitola 3 pojednává o teoretické části z pohledu herní sítě komunikace. Jak byla hra nakonec implementována a testována je popsáno v kapitole 4. Závěrečnou kapitolou je kapitola 5, která shrnuje celou práci na hře, věnuje se jejím kladům a záporům, a také nabízí možnosti, jak by se hra dala vylepšit a případně rozšířit.

¹V angličtině *gamebook*.

²V angličtině *role-playing games* nebo zkráceně RPG.

³Arkáda je žánr počítačové hry, založený na jednoduchém a nápaditém konceptu [1].

Kapitola 2

Popis hry a uživatelského rozhraní

Jak již bylo řečeno, *Hon na ponorku* je síťová hra pro dva až šest hráčů. Každý z hráčů ovládá svou ponorku a má za úkol zničit ponorky nepřátelské, ovládané ostatními hráči. Hra je kompletně ve 2D. Ponorky jsou vidět z pohledu shora a hráč nemá možnost pohled změnit. Hra funguje pod operačním systémem Microsoft Windows. Testovány byly verze XP, Vista a Windows 7.

2.1 Vlastnosti hry a její „pravidla“

Jedním z nejdůležitějších faktorů ovlivňujících hratelnost hry jsou herní vlastnosti. Herními vlastnostmi je myšleno to, co nám hra umožní, případně neumožní dělat. Například simulátor letadel, který neumožňuje s letadlem vzlétnout, by se asi mezi hráči příliš neuchytil. Toto je samozřejmě extrémní příklad. Pro hru *Hon na ponorku* jsou důležitými faktory možnost ovládnutí ponorky a schopnost zneškodnit protivníka. Ke hratelnosti hry určitě přispěje větší množství herních módů, kdy si každý hráč může vybrat na co má zrovna chuť. Hra se poté tak rychle neohraje.

2.1.1 Ponorka

Při vytváření hry bylo vycházeno z toho, že ovládání ponorky by mělo být jednoduché a intuitivní. Jedinými možnostmi hráče, jak ponorku ovládat, jsou měnit její směr a rychlost. Hráč může buď zvyšovat nebo snižovat rychlost, přičemž rychlost ponorky může mít i zápornou hodnotu, což znamená, že ponorka couvá. Zrychlení ponorky je v obou směrech konstantní. Směr ponorky může hráč jednoduše měnit zatáčením, kdy ponorka mění svůj směr o konstantní úhel za jednotku času. Pokud hráč nedrží žádnou klávesu, rychlost ani směr ponorky se nemění. Je nutné zmínit, že chování ponorky není ovlivněno fyzikálními zákony. Sice by to mohlo zvýšit reálnost hry, ale zvýšení reálnosti není vždy spojeno se zvýšením hratelnosti. Ponorka může být zničena buď některou z nepřátelských zbraní nebo také tím, že narazí do překážky. Překážkou může být i další ponorka. V takovém případě jsou zničeny obě ponorky.

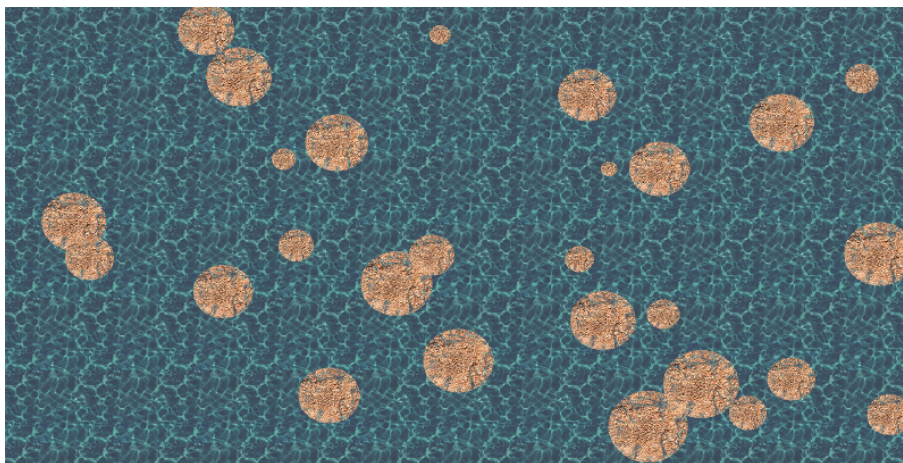
2.1.2 Zbraně

Vzhledem k tomu, že se hra týká ponorek, hlavními zbraněmi jsou v ní torpéda a miny. Každá ponorka může střílet torpéda směrem dopředu i dozadu. V ponorce k tomu slouží přední a zadní komora. Mina může být nabitá do jakékoli komory a je vždy vypuštěna ze

středu ponorky. Hráč si musí vybrat komoru před tím, než začne nabíjet. Změna zbraně či komory po začátku nabíjení je možná pouze za cenu zrušení současného nabíjení. Zbraně mohou být nabity v obou komorách, ale musí tak být učiněno postupně. První musí být dokončeno nabíjení jedné komory, a pak až lze nabít druhou. Na začátku kola nemá hráč nabitou ani jednu komoru. Každá zbraň má jinou dobu nabíjení a jiné vlastnosti. Typy zbraní, které byly ve hře použity, jsou vypsány v příloze B.

2.1.3 Mapy

Pod pojmem mapa je myšleno herní prostředí ve kterém se ponorky pohybují. Mapa obsahuje dvě hlavní části: pevninu a moře. Moře je prostor, ve kterém se ponorka může pohybovat. Pevnina je prostor, kam ponorka vjet nemůže a ani torpédo skrz pevninu neprojde. Dobrá mapa přispívá k hratelnosti a zábavnosti hry a je dobré mít větší množství map, aby se hráčům tak rychle neomrzely. Ve hře je mapa náhodně vygenerována vždy na začátku kola. Skládá se z kruhových oblastí o proměnlivém poloměru, které vyplňují přibližně jednu sedminu mapy. Vygenerovaná mapa je vidět na obrázku 2.1.



Obrázek 2.1: Vygenerovaná mapa

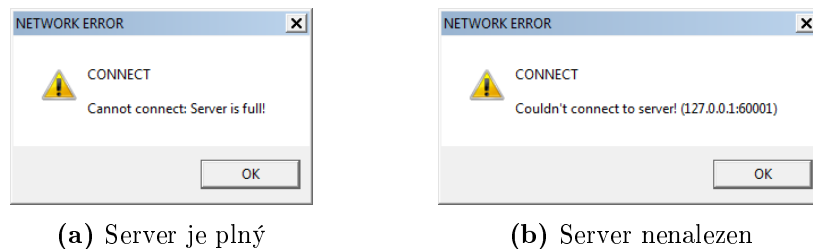
2.1.4 Herní módy

Pro *Hon na ponorku* byly navrženy dva herní módy: Standard a DeathMatch. Hrát je ovšem možné pouze v módu Standard.

- Standard - Hraje se všichni proti všem na kola. Délka jednoho kola je časově omezená (nastavitelná na serveru). Pokud je hráčova ponorka zničena, musí čekat do začátku dalšího kola. Kolo končí pokud přežije pouze jeden hráč, nebo pokud vyprší časový limit kola. Kolo ovšem nekončí okamžitě, ale s určitou prodlevou. Hráči tak mohou být zničeni i po skončení kola. Kdo na konci kola přežije, připsí si výhru.
- DeathMatch - Hraje se všichni proti všem, jako v módu Standard. Rozdíl je, že pokud je hráčova ponorka zničena, objeví se okamžitě znovu ve hře a hráč může pokračovat. Protože hráč po zničení nemá nabito žádné torpédo, je tak v nevýhodě oproti ostatním. Proto je hráč vždy po znovuoobjevení tři vteřiny nesmrtelný.

2.2 Uživatelské rozhraní hry

Reálné rozlišení hry je 1440×900 pixelů. Toto rozlišení se ovšem přizpůsobuje současnému rozlišení obrazovky. To může způsobit roztazeni hry v některém směru kvůli jinému poměru stran. Hra nemá žádné menu, a proto se veškeré nastavení musí zadat pomocí konfiguračního souboru. Důležité jsou však pouze tři údaje: IP adresa serveru, port, na kterém hra na serveru běží a jméno hráče. Po spuštění hry se hráč připojí rovnou na server a může hrát. Může se stát, že server je plný a hráč se na něj nemůže připojit (obrázek 2.2a), nebo se na server nedá připojit vůbec (obrázek 2.2b).



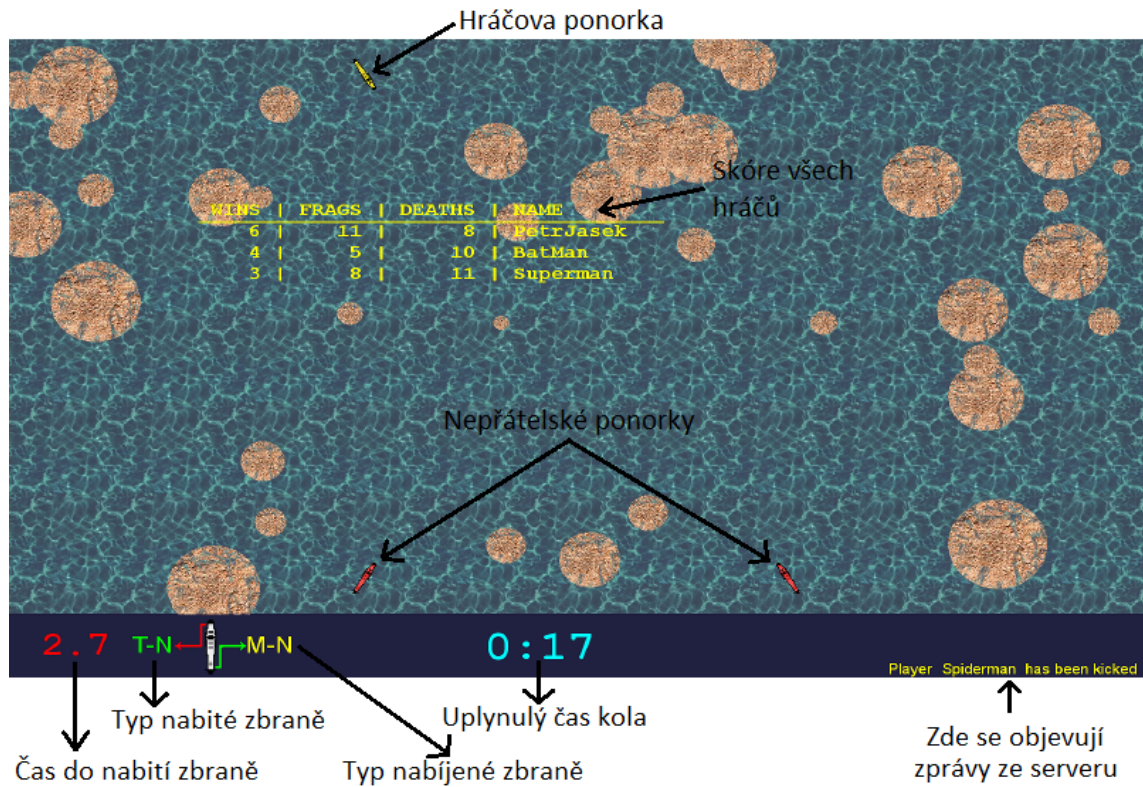
Obrázek 2.2: Chybové zprávy pokud se na server nelze připojit

Po startu nové hry hráč ovládá svou ponorku pomocí klávesnice. Nastavení ovládání je zobrazeno v tabulce 2.1. Během hry si každý hráč může zobrazit tabulku se skóre, která pro každého hráče zobrazuje počet vyhraných kol, počet jeho fragů¹ a počet jeho „smrtí“. Hráčova ponorka je zobrazena žlutě, nepřátelské ponorky jsou červené. Na spodní straně obrazovky je zobrazen informační panel, v jehož levé části je zobrazena nejdůležitější část rozhraní a to, které zbraně má hráč v současnosti nabity, případně nabíjeny, a také kterou komoru (přední/zadní) má hráč zvolenou. Komory jsou zobrazeny jako šipky vedoucí z přední a zadní části ponorky. Vybraná komora je znázorněna tak, že její šipka se zbarví zeleně. Druhá šipka je pak červená. Pokud se nabíjí zbraň, je u komory, do které se nabíjí, zobrazena zkratka daného typu zbraně a tato zkratka je vypsána žlutě. Také se zobrazuje čas do nabití zbraně. Ve chvíli kdy je zbraň nabita, změní se barva zkratky na zelenou. Jak hra celkově vypadá je zobrazeno na obrázku 2.3. Jsou zde i uvedeny popisky k jednotlivým částem grafického uživatelského rozhraní.

Akce	Klávesa
Zrychlení	Šipka nahoru
Zpomalení	Šipka dolů
Zatočení doleva	Šipka doleva
Zatočení doprava	Šipka doprava
Vystřelení torpéda	Mezerník
Výběr přední komory	Q
Výběr zadní komory	A
Výběr zbraně pro nabití	Číslo 1–4
Zobrazení tabulky se skóre	Tab
Ukončení hry	Esc

Tabulka 2.1: Standardní ovládání hry

¹Frag je výraz, který se v počítačové terminologii používá pro bod získaný za zneškodnění protivníka.



Obrázek 2.3: Vzhled hry s popisky

Uživatelské rozhraní serveru Proto, aby hráči mohli hrát, musí být vytvořen server, který zprostředkovává veškerou komunikaci. Server je realizován jako jednoduchá konzolová aplikace a je ho možné ovládat příkazy. Kompletní tabulka příkazů je zobrazena v příloze A. Server má svůj vlastní konfigurační soubor, kterým se dá upravit jeho základní nastavení.

Kapitola 3

Síťová komunikace

V této kapitole je používáno základních síťových termínů bez jejich podrobného vysvětlování. Pokud čtenář nemá velké zkušenosti se sítěmi, může mít s pochopením některých částí této kapitoly problémy.

Síťová komunikace je jednou z nejdůležitějších a nejsložitějších částí hry. Dobrá implementace síťové komunikace může (ale nemusí) znamenat, že hratelnost hry bude dobrá. Na druhou stranu špatná implementace téměř zaručuje, že hratelnost bude špatná a hráč tedy nebude spokojen. Existuje mnoho aspektů, které musí být zváženy předtím, než se začne síťová komunikace implementovat.

3.1 IPv4 nebo IPv6?

Jedním z prvních rozhodnutí, co se síťové komunikace týče, je výběr Internetového protokolu. V praxi se nabízejí pouze dvě možnosti a to buď IPv4 a nebo IPv6.

Přestože bylo dlouho jasné, že adresový prostor IPv4 bude v blízké době vyčerpán¹, podpora IPv6 nerostla zdaleka tak rychle jak se původně předpokládalo. Velká většina síťových her stále používá pouze IPv4, ale některé z novějších her už také podporují nebo se chystají podporovat IPv6, například World of Warcraft. Podpora IPv6 je ale vždy spíše doplňkem IPv4 než aby hra byla vytvořena přímo pro IPv6. Problém IPv6 je také slabá podpora ze strany poskytovatelů Internetového připojení. Ti ještě necítí nutnost investovat do podpory IPv6.

Ačkoli by podpora IPv6 ve hře *Hon na Ponorku* mohla být užitečná, hra IPv6 nepodporuje. Implementace hry však brala v potaz možnost podpory IPv6, a tak nic nebrání jejímu pozdějšímu přidání.

3.2 UDP nebo TCP?

Zvolení správného Internetového protokolu se značně odvíjí od typu hry, pro kterou má být tento protokol použit. Ačkoli existují i jiné protokoly než TCP a UDP, jejich využití pro herní síťovou komunikaci je mizivé. Jak TCP tak i UDP má pro použití ve hrách své výhody a nevýhody.

¹Stalo se tak 3. února 2011, kdy organizace IANA spravující IPv4 adresy, přidělila poslední z nich.

3.2.1 TCP

TCP by se na první pohled mohl jevit jako ideální protokol pro jeho spolehlivost a správné řazení paketů. Jelikož ale funguje nad nespolehlivým protokolem IP, musí být tyto vlastnosti zařízeny speciálními mechanismy. Právě tyto mechanismy a jejich režie způsobují, že je TCP nevhodným protokolem pro některé typy her.

Prvním problémem TCP je, že data jsou přenášena jako tok bajtů. Protože IP přenáší data v paketech, musí TCP samo data do paketů rozdělovat. Interně TCP data shromažďuje, a když je připraveno dostatečné množství dat, odešle je v jednom paketu. To může být problém pokud mají být posílány velmi malé balíčky dat. TCP se může rozhodnout, že vaše data neodešle, dokud se jich neshromáždí určité množství. Poté může vzniknout nežádoucí zpoždění. Tento problém lze vyřešit nastavením možnosti TCP zvané `TCP_NODELAY`, která způsobí, že TCP nečeká na shromáždění dostatečného množství dat, ale odesílá je okamžitě. Bohužel i přesto má TCP další vlastnosti, které jsou pro síťové hry nežádoucí.

K nalezení dalšího problému je nutné si uvědomit, jak TCP zajišťuje spolehlivost přenosu dat. Jak již bylo řečeno, TCP rozděluje tok dat do paketů a tyto pakety jsou poté znovu sestaveny do toku dat na druhé straně přenosového kanálu. Co se ale stane, pokud se paket ztratí? Pakety jsou na druhé straně dále přijímány, ale nemohou být zpracovány dokud není doručen chybějící paket. Odesílatel první musí zjistit, že paket nedorazil a poté musí znovu paket odeslat. Tato procedura opět vytváří nežádoucí zpoždění.

TCP by ovšem nemělo být zatracováno. I přes jeho nedostatky vzhledem k síťovým hrám je i tak často využíván. Například tahové hry (deskové hry, karetní hry, ...) téměř bez výjimky využívají TCP. U těchto her si pravděpodobně ani nevšimnete, že nastala prodleva při komunikaci mezi jednotlivými hráči. Navíc spousta moderních real-time² strategií (RTS) jako je Starcraft, Warcraft a také spousta MMORPG (Massive(ly)-Multiplayer Online Role-Playing Game) jako je World of Warcraft funguje s použitím TCP. Dalo by se polemizovat, je-li TCP pro některé z těchto her správnou volbou (například studie *An Empirical Evaluation of TCP Performance in Online Games* [6]). Nicméně není sporu o tom, že síťová komunikace v těchto hrách funguje natolik dobře, že je hrají milióny hráčů po celém světě.

Z výše uvedeného vyplývá, že i přes občasné větší zpoždění doručení paketů se dá TCP využívat u her, a to dokonce také u real-time her. Velké zpoždění je největším problémem u her, u kterých je očekávána opravdu rychlá odezva. Mezi takovými hry patří FPS³. V FPS hrách je důležité, aby byla hra co nejplynulejší a nejaktuálnější (stav hry jak jí vidí hráč by se neměl příliš lišit od skutečného stavu hry), a pro to se TCP úplně nehodí. Jednoduše řečeno z pohledu hráče vás nezajímá, co se stalo před vteřinou. Chcete mít vždy nejnovější možné informace o stavu hry.

3.2.2 UDP

UDP vytváří pouze tenkou vrstvu nad protokolem IP a oproti TCP nemá prakticky žádné vlastní mechanismy. Jedinou jistotou, kterou UDP poskytuje, je ta, že datagram (UDP paket) je doručen buď celý v pořádku, nebo není doručen vůbec. Pokud je tedy odeslán datagram o velikosti 200 bajtů, nehrozí, že by bylo doručeno například pouze 100 bajtů.

Ačkoli je UDP využíváno právě pro jeho jednoduchost, téměř všechny hry mají vlastní mechanismy pro zajištění aspoň určité spolehlivosti přenosu. Nicméně tyto mechanismy

²Hry, které se odehrávají v reálném čase a reagují okamžitě na akce uživatele.

³Zkratka FPS pochází z anglického First Person Shooter a v češtině se pro tyto hry používá nejčastěji vágní termín „střílečka“. Jedná se o hry kdy hráč vidí hru z pohledu očí jeho herní postavy.

bývají specifické vždy pro danou hru, protože pro každou hru se liší důležitost určitých událostí. Pokud se například ztratí paket z informací, že uživatel opustil hru, je žádoucí, aby byl tento paket poslán znova. Naproti tomu paket s informací o stisknutí klávesy uživatelem nemusí být stěžejní a jeho ztráta může být ignorována.

Poměrně velkým nedostatkem UDP ovšem je to, že postrádá kontrolu toku dat (flow control) a kontrolu zahlcení (congestion control). Kontrola toku dat se stará o to, aby nebylo přetíženo cílové zařízení. Pokud je jedno ze zařízení rychlejší, nemusí druhé zařízení stíhat zpracovávat všechna data, která první zařízení odešle. Kontrola zahlcení zase zabraňuje přetížení sítě. K zahlcení může například dojít, pokud více zařízení sdílí jeden uzel (například router) a posílají takové množství dat, které router není schopen zpracovávat. Ve chvíli kdy je router přetížen, dojde k obrovskému úpadku kvality připojení. Proto TCP podporuje kontrolu zahlcení a snaží se takovýmito situacím předcházet. Je vždy dobré alespoň nějakou kontrolu zahlcení a kontrolu toku implementovat i pod UDP.

3.2.3 Jiné možnosti

Jako jedna z dalších možností se nabízí kombinace UDP a TCP, kdy data, která mají být spolehlivě doručena, jsou posílána pomocí TCP a data citlivá na okamžité doručení pomocí UDP. Toto je samozřejmě možné a v praxi se u některých her i používá. Nicméně tato kombinace má některá úskalí. Jedno z nich je zmíněno v článku „Characteristics of UDP Packet Loss Effect of TCP Traffic“ [10]. Článek zmiňuje problém, který by se zjednodušeně dal vysvětlit tak, že TCP protokol si pro sebe zabírá prostor a dochází pak k častější ztrátě UDP paketů. Dále u kombinace TCP a UDP může vzrůst složitost zpracování přichozích dat. Jak již ale bylo zmíněno, tyto problémy neznamenají, že nelze vytvořit kvalitní síťovou komunikaci využívající kombinace TCP s UDP.

Další možností pak je využít jiný protokol transportní vrstvy. Existuje řada protokolů, které by se daly pro síťovou komunikaci využít: DCCP (Datagram Congestion Control Protocol) [RFC 4340], RTP (Real-time Transport Protocol) [RFC 3550] nebo SCTP (Stream Control Transmission Protocol) [RFC 4960]. Na jejich podrobnou analýzu bohužel není v této publikaci místo.

3.2.4 Zvolený protokol

Jelikož je hra *Hon na Ponorku* hrou, od které se očekávají rychlé reakce, a také po zvážení všech výhod a nevýhod jednotlivých protokolů, byl pro implementaci zvolen protokol UDP. Protokol TCP by se však dal také použít, protože dnešní sítě a Internet jsou natolik spolehlivé, že by jeho nevýhody byly z velké většiny potlačeny. Jedinou opravdovou nevýhodou pro TCP by pro tuto hru mohlo být větší množství zasláných dat, kvůli jeho větší hlavičce.

3.3 Typ sítě z pohledu komunikace hráčů

Síťová komunikace mezi hráči probíhá ve velké většině na základě dvou architektur.

3.3.1 Peer-to-Peer

Peer-to-Peer (P2P) architektura je postavena na vzájemné komunikaci hráčů, kdy jeden hráč přímo posílá data ostatním hráčům. Základní idea je hru abstrahovat do série tahů (lockstep). Všechny příkazy uživatele jsou pak zpracovány na začátku každého tahu [5].

Jako příklad uživatelského příkazu může sloužit například přesunutí jednotky nebo začátek konstrukce budovy. Tento typ komunikace trpí z herního pohledu hned několika nedostatky.

Za prvé hráči musí být nějak synchronizováni, aby hra každého z nich byla ve stejném stavu. Toto je možné zařídit u tahových her, u kterých jsou stavy hry jasně dány. Problém nastává u složitějších her. U těch je nutné zařídit, aby byly deterministické do té míry, že akce uživatele se promítne stejně jak do jeho hry, tak do hry všech ostatních hráčů. Tohoto chování je velmi složité dosáhnout u komplikovanějších her. I malý rozdíl s postupem času může přerůst v rozdíl značný (efekt motýlích křídel). Stačí vzít v potaz počítání čísel s pohyblivou řádovou čárkou, které na různých počítačových architekturách může dávat rozdílné výsledky.

Dalším problémem P2P architektury nastává ve chvíli, kdy se hráč chce připojit do již probíhající hry. Aby byl zachován determinismus, hráč musí získat kompletní stav hry, včetně všech možných náhodných proměnných, které mohly být vygenerovány na začátku hry. Ačkoli to je technicky možné, není to příliš časté právě kvůli složitosti zachytit a zaslat kompletně deterministickou počáteční pozici uprostřed již probíhající hry.

Posledním z problémů, které budou zmíněny je podvádění (cheating). V P2P architektuře může být pro některé typy her složité odhalit, zda některý z hráčů nezasílá ostatním upravená data, a tím nevylepší svou pozici.

3.3.2 Klient-Server

Architektura klient-server používá jeden počítač, který je označován jako server. Ten spravuje celou hru včetně komunikace hráčů, kteří jsou v této architektuře označováni jako klienti. Jelikož celá hra doopravdy existuje pouze na serveru, není nutné, aby hra byla dokonale deterministická. Klienti jsou víceméně pouze terminály, které zobrazují stav hry na serveru a interagují s ním.

V čistém klient-server modelu klient nesimuluje žádnou logiku hry. Jediné co dělá je, že zasílá uživatelské akce na server. Server tyto akce provádí a zasílá klientovi nový stav hry. Problémem tohoto modelu je, že klient se nový stav hry dozví až po určité době. Ta je závislá na kvalitě připojení k serveru a už i poměrně malé zpoždění se hráči promítne pomalou reaktivitou hry. Tento problém je řešen takzvanou klientskou predikcí⁴. Problém bude popsán dále v této kapitole (sekce 3.4.1).

Architektura klient-server má ještě jednu skrytou výhodu. Velké množství uživatelů se v Internetu nachází za překladačem síťových adres (NAT), jako je například router. Protože nemají veřejnou adresu, není možné se k těmto uživatelům připojit přímo (pokud nemají router speciálně nastaven). Tím, že je použit server, tento problém odpadá. Klient se připojuje na server a vytvoří ve svém NAT kanál pro komunikaci a příchozí pakety pak mohou být směrovány ke klientovi. Tato technologie se nazývá *hole punching*. Server sám ovšem musí mít veřejnou IP adresu.

3.3.3 Zvolená architektura

Jelikož hra *Hon na Ponorku* není hrou tahovou a není ji dost dobře možné rozdělit na posloupnost tahů, byla pro ni zvolena architektura klient-server. Tím se hra vyhne problémům spojeným s P2P. Na druhou stranu si přinese problémy spojené s architekturou klient-server. Jako typ serveru byl zvolen takzvaný dedikovaný server.

⁴V angličtině se používá termín *client-side prediction*.

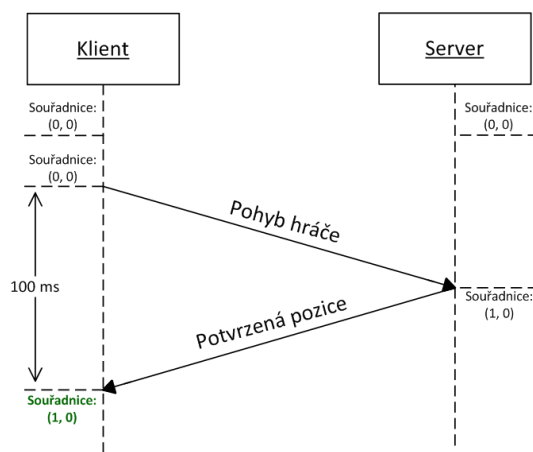
Dedikovaný server je server, který běží jako samostatný program oddělen od klienta. U některých her je možné hru spustit a nechat ostatní uživatele připojit se k vám do vaší hry. To dedikovaný server neumožňuje právě proto, že běží mimo klienta. Všichni klienti se musí k serveru připojit. Výhodou dedikovaného serveru je, že může být spuštěn na odděleném počítači a má tak výpočetní sílu celého počítače. Případně, pokud hra není výpočetně náročná, může na jednom počítači být spuštěno více serverů. Nenastane také situace, že hráč chce ukončit hru a tím ukončí i server. Klienti se jednoduše od serveru mohou odpojit, ale hra na něm zůstane pořád funkční.

3.4 Problémy architektury klient-server

Jak již bylo nastíněno, architektura klient-server není dokonalá. Tato sekce se věnuje jejím největším problémům a možnostem jak tyto problémy řešit. V této sekci bylo mimo jiné vycházeno ze článků *Source Multiplayer Networking* [2] a *Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization* [4].

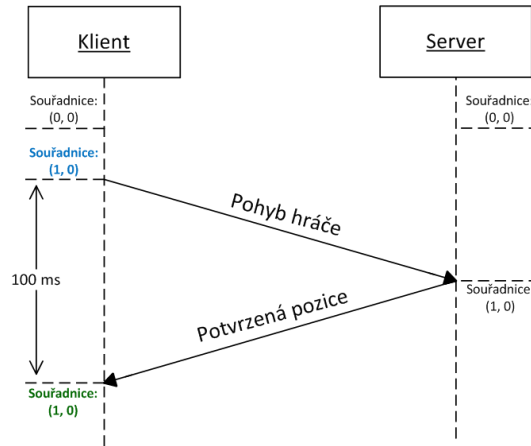
3.4.1 Predikce na straně klienta

Prvním problémem vzniká ze samotné podstaty jak architektura klient-server funguje. Vezměme například situaci, kdy klient stiskne klávesu pro pohyb. Tato akce se odehraje v čase $t = 0 \text{ ms}$, kdy hráč stojí na souřadnicích $(0, 0)$. Akce je odeslána na server. Pokud počítáme s dobrým připojením, doba cesty paketu mezi klientem a serverem může být 50 ms . Budeme předpokládat, že zpracování zprávy serverem a její promítnutí do hry nezabere žádný čas. Pohyb hráče se promítne tak, že se přesune na pozici $(1, 0)$. Klient poté obdrží novou pozici od serveru v čase $t = 100 \text{ ms}$. Hráč tedy musí čekat 100 ms než se pohne (obrázek 3.1). Toto zpoždění se samozřejmě zásadně promítne do hratelnosti.



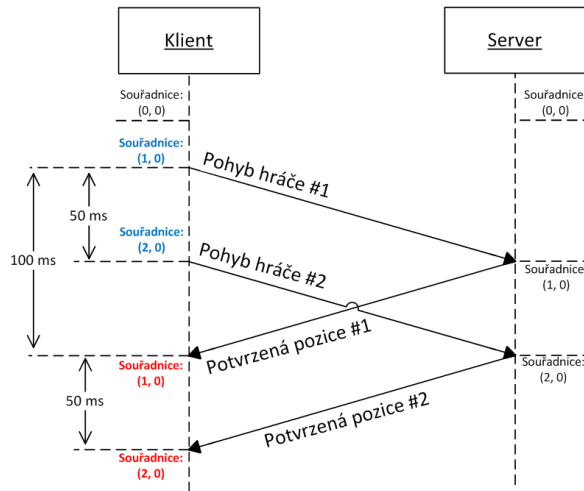
Obrázek 3.1: Ilustrace problému způsobeného zpožděním mezi klientem a serverem

Problém zpoždění mezi klientem a serverem řeší predikce na straně klienta. Její podstata je v tom, že klient jednoduše předpokládá, že zpráva bude serveru doručena. Za předpokladu, že hra je deterministická (aspoň do určité míry), může poté klient simulovat svůj pohyb stejně jako kdyby dostal okamžitě potvrzení od serveru (obrázek 3.2). Tohoto mechanismu je využíváno velice často, aniž by si hráči uvědomovali, že se něco děje. Ve chvíli kdy hráč stiskne klávesu pro pohyb začne se okamžitě hýbat. Tento mechanismus funguje dobře, protože většina paketů je opravdu serveru doručena.



Obrázek 3.2: Predikce na straně klienta pro jednu akci

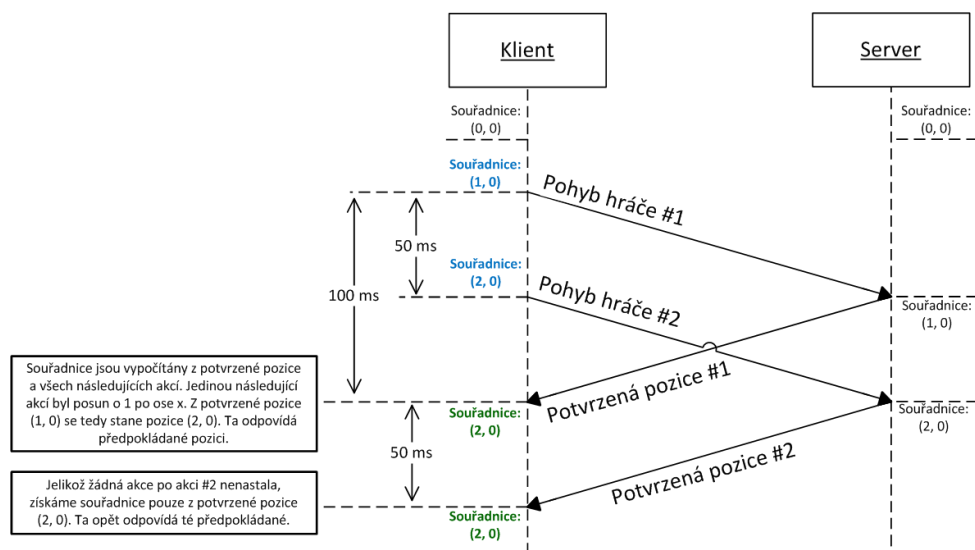
Co se ale stane, pokud by se hráč pohnul podruhé ještě předtím, než by bylo od serveru potvrzeno první posunutí? Tento problém ilustruje obrázek 3.3. V této situaci by se souřadnice, které klient obdrží od serveru lišily od těch předpokládaných, a protože je server autoritativní, klient by musel své souřadnice nastavit podle serveru. Vzápětí ovšem přijde potvrzení druhého pohybu a klient se musí opět posunout. Ve hře by to znamenalo, že se hráč z jeho pohledu přesune, pak se přesune o krok zpět a následně opět vpřed. To je samozřejmě nepřijatelné, ale řešení tohoto problému není nějak složité.



Obrázek 3.3: Predikce na straně klienta pro dvě akce

Každá zpráva musí mít sekvenční číslo a klient si musí zprávy pamatovat, dokud nejsou potvrzeny ze strany serveru. Jakmile přijde ze serveru potvrzení nějaké zprávy, klient může tuto a všechny zprávy s nižším sekvenčním číslem vymazat ze své paměti. Poté musí znovu provést klientskou predikci pro všechny akce, které provedl po akci, která byla právě potvrzena. Tento systém ilustruje obrázek 3.4. Užitečnou vlastností tohoto způsobu predikce je, že se vypořádá i s výpadky paketů. Pokud vypadne potvrzující paket, nenastává vůbec žádný problém. Pokud vypadne paket s klientskou akcí, z pohledu hráče dojde k posunu, který zařídí predikce. Server ovšem nic nepotvrdí a ztráta se projeví až při potvrzování další akce. Server totiž nedostal zprávu o první akci a tak provede pouze akci druhou. Z pohledu

hráče dojde k skokové změně pozice, ale dále již vše bude fungovat v pořádku. Při použití nespolehlivého přenosu mezi klientem a serverem tomuto bohužel nejde zabránit.



Obrázek 3.4: Predikce na straně klienta pro dvě akce s ukládáním zpráv

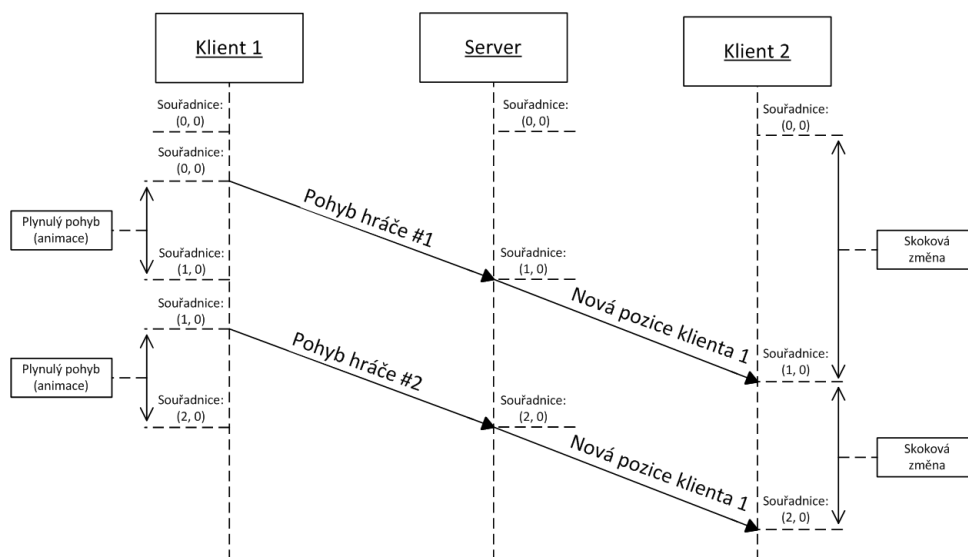
Predikce z pohledu klienta je vynikající věc, která značně zlepšuje pocit z hraní, ale má i svá úskalí a může způsobovat situace, které se hráčům neznalým síťové implementace hry mohou zdát zvláštní. Pokud se například v FPS hře hráč snaží schovat za roh před druhým hráčem, který po něm střílí, může se mu stát, že bude zabít ze svého pohledu až za rohem. Z jeho pohledu se totiž už za roh dostal, ale to je jen klientská predikce. Jeho protihráč po něm vystřelil ještě před rohem. Prodleva mezi klienty a serverem pak umožnila se hráči pohybovat, i když byl z pohledu serveru vlastně už trefen. Tento fenomén, který je proklínán mnoha hráči FPS her, má ovšem ještě jednoho viníka, který se na něm podílí větší měrou než klientská predikce. Tímto viníkem je takzvaná interpolace.

3.4.2 Extrapolace a interpolace

Než budou vysvětleny termíny extrapolace a interpolace, musí být ukázáno, proč jsou vůbec potřeba. Model komunikace, který byl popsán v předchozí sekci, má ale trhliny. Ty nejsou patrné pokud zkoumáme komunikaci klient-server z pohledu jediného klienta. Problém se ukáže až s přidáním dalšího klienta. Pro ilustraci použijeme předchozí příklad s pohybem klienta. Nyní ale předpokládáme, že změna pozice nenastává okamžitě, ale jedná se o plynulý přesun z jedné pozice na druhou, který trvá 50 ms. Na klientovi 1, který pohyb vykonává, je možné pohyb provádět plynule, protože klient o něm ví. Ostatním klientům ovšem ze serveru přijde až nová pozice klienta 1. Problém je zobrazen na obrázku 3.5. V zásadě ho lze řešit dvěma variantami: extrapolací⁵ nebo interpolací.

Extrapolace se snaží odhadnout pohyb hráče na základě jeho předchozího pohybu. Nutným požadavkem pro její fungování je, aby objekty hry, které mají být extrapolovány, nemohly nárazově měnit svůj stav. Takovou hrou mohou být například závody aut. Auta obvykle nemohou měnit nárazově svou rychlost ani směr. Je tedy možné předpokládat,

⁵V literatuře je také používám anglický termín *Dead reckoning*.



Obrázek 3.5: Ukázka jak je klient viděn z pohledu jiného klienta

že pokud se auto pohybuje určitou rychlostí, bude se přibližně touto rychlostí pohybovat i nadále a je tedy možné odhadnout jeho budoucí pozici.

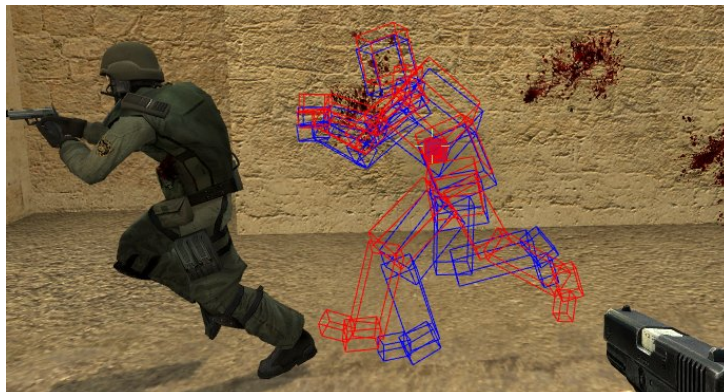
Interpolace čerpá z již známých informací o stavu herních objektů. Nesnaží se zjistit jak se budou objekty pohybovat v budoucnosti. Místo toho pouze aproximuje jejich pohyb mezi známými kroky v minulosti. Ve chvíli, kdy klient obdrží nový stav objektu, nepřesune objekt hned na toto místo. Místo toho vychází ze známého stavu objektu v minulosti a pouze simuluje přesun z tohoto stavu do obdrženého stavu. Tento způsob má jednu obrovskou nevýhodu a to tu, že hráč pak vidí stavy ostatních hráčů (objekty) v minulosti. Při interpolaci je důležitý parametr, o jak velký časový úsek mají být ostatní objekty zpožděny. Toto číslo závisí na tom, jak často server posílá nové stavy hry a musí být větší, než časový úsek mezi jednotlivými aktualizacemi od serveru. Kdyby čas interpolace byl menší, nepřišel by nový stav od serveru včas. Pokud tedy server posílá nové stavy každých 100 ms, čas interpolace by měl být větší než 100 ms. Pokud se navíc chceme vyhnout problémům při ztrátě paketu, měl by tento čas být více než dvojnásobný ($> 200 \text{ ms}$). Pokud by došlo k většímu výpadku mezi klientem a serverem, klientovi by „vypršel“ čas pro interpolaci a nevěděl by, jak se bude stav objektů měnit dále. V takovou chvíli lze buď použít extrapolaci nebo jednoduše objekty nechat v posledním známém stavu.

Největším problémem interpolace tedy je, že hráč vidí ostatní v minulosti. To může způsobit značné problémy například u FPS her, kdy se snažíte trefit jiného hráče, ale střílíte vlastně po jeho stínu z minulosti. Tento problém řeší takzvaná kompenzace zpoždění⁶.

Kompenzace zpoždění je technika, která částečně smazává problémy interpolace. Nejlépe ji lze vysvětlit na příkladu FPS hry. Jak již bylo řečeno, hráč ve skutečnosti nevidí ostatní hráče v přítomnosti, ale vidí jejich pozici v minulosti („stín“). Pokud na takovýto stín vystřelí, nemusí se tedy trefit, protože hráč se již mohl přemístit. Tento problém kompenzace zpoždění řeší tak, že pokud server zjistí, že hráč po někom střílí, vypočítá si z historie stavů hry, kde byly všechny objekty v době této střelby pro střílejícího hráče. Poté se až rozhodne,

⁶ V angličtině se používá termín *Lag compensation*.

jestli hráč druhého hráče trefil nebo ne a případně tuto skutečnost promítne do současného stavu. Jak funguje interpolace a kompenzace zpoždění je ukázáno na obrázku 3.6.



Obrázek 3.6: Obrázek pochází ze hry Counter-Strike: Source a byl pořízen na serveru se zpožděním 200 ms. Červený obrys postavy ukazuje pozici postavy v době, kdy po ní hráč střílel. Modrý obrys pak ukazuje pozici jak ji aproximoval server z dostupných informací. Lze vidět, že v době kdy hráč registruje zásahy je postava daleko za místem, kde byla zasažena.

3.4.3 Synchronizace klientů se serverem

Protože mezi klienty a serverem existuje různé a navíc proměnlivé zpoždění, je prakticky nemožné klienty se serverem zcela synchronizovat avšak částečná synchronizace možná je. Naivní pohled na fungování serveru by mohl být ten, že server při každém obdržení zprávy od klienta vypočítá nový stav hry a pošle jej okamžitě zpět klientovi. To by v praxi ale znamenalo velkou zátěž pro procesor serveru a také síť. Místo toho server simuluje hru v diskretních časových krocích. V každém kroku pak zpracovává všechny zprávy od uživatelů, simuluje logiku hry a na konci každého kroku posílá nový stav uživatelům. Takto dostanou všichni uživatelé stejný stav hry.

Kapitola 4

Implementace hry a testování

V této kapitole bude popsáno, jak byl implementován server i samotná hra. K implementaci byl využit jazyk C++ a vývojové prostředí Microsoft Visual Studio 2010.

4.1 Použité knihovny

Při vývoji byly použity knihovny Boost 1.50.0 a Allegro 5.0.7.

4.1.1 Boost

Knihovna Boost je vlastně souborem knihoven, které pokrývají širokou škálu problémů. Některé z těchto knihoven dokonce pronikly do standardu C++. Ve hře byly nejvíce využity knihovny Boost.Asio pro práci se sítí, Boost.Thread pro práci s vlákny a pro serializaci dat posílaných přes síť Boost.Serialization.

4.1.2 Allegro

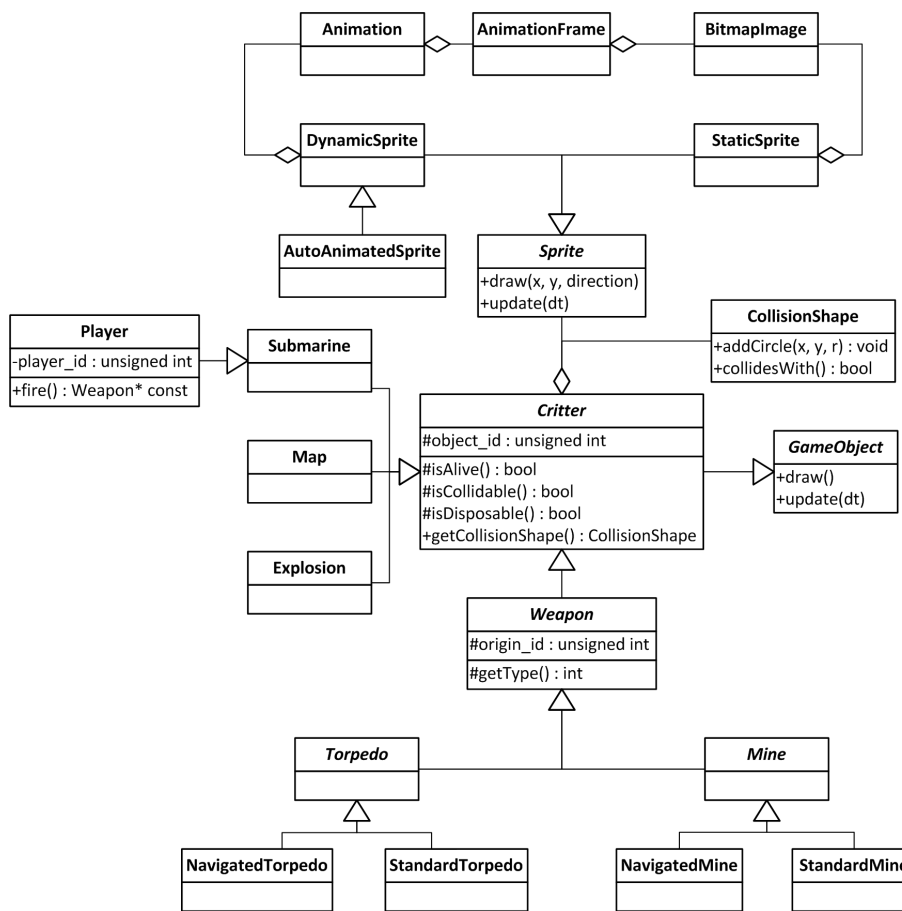
Knihovna Allegro je knihovna specializovaná na programování her v jazycích C a C++. Poskytuje prostředky často využívané v programování her. Je to multiplatformní knihovna a ve hře byla využita například pro práci s uživatelským vstupem, grafikou, zvukovými efekty nebo časovači.

4.2 Herní objekty

Herní objekt je každý objekt, na který můžeme ve hře narazit. Hlavními objekty jsou hráči, zbraně a mapa. Všechny herní objekty mají stejné základní rozhraní, aby s nimi mohlo být jednoduše manipulováno. Diagram tříd týkající se herních objektů je zobrazen na obrázku [4.1](#).

4.2.1 Grafická reprezentace objektů

Každý objekt, má-li být vidět, musí mít svou grafickou reprezentaci. Pro tento účel byla vytvořena třída Sprite.



Obrázek 4.1: Diagram tříd herních objektů

Sprite Tato abstraktní třída představuje grafickou reprezentaci objektu a deklaruje dvě hlavní metody:

```

void draw(double x, double y, double direction);
void update(double dt);

```

Metoda **draw** vykresluje objekt na dané pozici a v daném směru. Metoda **update** pak pouze aktualizuje stav objektu, protože grafická reprezentace objektu se může v čase měnit. Tyto metody jsou volány z odpovídajících metod třídy **Critter**, případně z jejích potomků.

StaticSprite, DynamicSprite, AutoAnimatedSprite Tyto třídy definují typ grafické reprezentace objektu. Zatímco **StaticSprite** je neměnný obrázek, **DynamicSprite** se může měnit v čase nebo na základě stavu objektu. **AutoAnimatedSprite** je pak předem časově nastavená animace, která se využívá například u animací explozí (třída **Explosion**). Všechny třídy přímo nebo nepřímo využívají třídy **BitmapImage**.

Animation Tato třída je využívána ve třídě **DynamicSprite** a představuje třídu pro animaci. Obsahuje pole snímků (třída **AnimationFrame**), kdy každý snímek má určitou časovou délku. Poté na základě uplynulého času mohou být snímky procházeny. Je ale také možné nebrat na čas ohled a snímky vybírat na základě jiných předpokladů, např. stavu herního objektu. Každý snímek využívá třídy **BitmapImage**.

BitmapImage Třída `BitmapImage` je abstrakcí nad strukturou `ALLEGRO_BITMAP`, kterou poskytuje knihovna `Allegro`. Poskytuje funkce pro snadnější práci s bitmapy a je využívána pro vykreslování všech herních objektů.

4.2.2 Fyzická reprezentace objektů

Fyzickou reprezentací objektu jsou myšleny jeho fyzikální vlastnosti a veškerá logika jeho chování.

GameObject Základní třídou je abstraktní třída¹ `GameObject`. Tato třída deklaruje dvě metody:

```
void draw(void);  
void update(double dt);
```

Jak již jejich názvy napovídají, metoda `draw` je určena pro vykreslení objektu a metoda `update` pro jeho aktualizaci. Metoda `update` pak přijímá jako parametr desetinné číslo, které značí množství času, uplynulého od poslední aktualizace v sekundách.

Critter Z třídy `GameObject` dědí třída `Critter`, která představuje základní reprezentaci pro všechny herní objekty. Tato třída obsahuje atributy pro polohu objektu, jeho směr, rychlost, id objektu a také ukazatel na grafickou reprezentaci (třída `Sprite`) tohoto objektu. Jelikož server nepotřebuje grafickou reprezentaci, je pomocí direktiv preprocesoru atribut pro grafickou reprezentaci zahrnut pouze u klienta. Metoda `draw`, která využívá grafické reprezentace je pak u serveru pomocí direktiv preprocesoru upravena tak, aby nedělala nic. Metoda `update` provádí aktualizaci pozice pomocí Eulerovi metody, kdy délka kroku odpovídá uplynulému času, který se předává jako argument metodě `update`. Pozice je poté vypočítána jako:

$$x = x + v \times \cos \alpha \times \delta t$$

$$y = y + v \times \sin \alpha \times \delta t$$

kde x a y jsou souřadnice objektu, v je jeho rychlost, α je jeho směr v radiánech a δt je uplynulý čas.

Explosion Třída `Explosion` reprezentuje explozi. Její grafická reprezentace je automatická animace (třída `AutoAnimatedSprite`). Třída je využívána při explozích zbraní a ponorek. Ačkoli by exploze mohla teoreticky být pouze grafickou animací, není tomu tak. Exploze byla implementována jako objekt, kvůli možnosti, že bude interagovat s dalšími objekty. Například by exploze sama o sobě mohla zničit ponorku v jejím okolí.

Weapon, Torpedo, Mine `Weapon` je abstraktní třída reprezentující zbraň. Nejdůležitějšími atributy jsou id hráče, který tuto zbraň vypustil, a také typ zbraně. V této třídě jsou deklarovány dvě metody, které umožňují navigování zbraní:

```
void actualizeTarget(Player * const player);  
void navigate(void);
```

¹Fakticky je `GameObject` rozhraní, protože pouze deklaruje metody, ale v C++ lze vytvářet pouze třídy nikoli rozhraní.

Metoda `actualizeTarget` přijímá jako argument ukazatel na objekt hráče a ukládá si informace o tom, jak navigovat zbraň k tomuto objektu. Tyto informace jsou využívány při opětovném volání metody k tomu, aby bylo vyhodnoceno, zda je nový cíl lepším cílem než cíl starý. Metoda `navigate` poté tyto informace používá pro upravení stavu zbraně. Abstraktní třídy `Torpedo` a `Mine` reprezentují torpédo respektive minu a obě dědí z třídy `Weapon`.

Submarine Třída `Submarine` reprezentuje ponorku. Jejími nejdůležitějšími atributy jsou atributy pro změnu stavu ponorky, jako je na například zrychlování, zpomalování a zatáčení. Dalšími atributy jsou atributy, které určují, která komora ponorky je v současnosti vybrána, která je nabíjena a která zbraň je nabíjena, případně nabitá. Ponorka má definovanou maximální rychlost vpřed a vzad, kdy rychlost vzad může dosáhnout maximálně 80% rychlosti vpřed. Zrychlení a rychlost otáčení jsou definovány jako konstanty. Třída má tyto důležité metody:

```
Weapon * const fire(void);
Explosion * const destroy(void);
```

Metoda `fire` na základě vybrané komory a zbraně v této komoře vytvoří nový objekt, který reprezentuje zbraň vystřelenou z ponorky. Ukazatel na tento objekt je vrácen jako návratová hodnota této funkce. Metoda `destroy` slouží ke zničení ponorky. Metoda patřičně změní stav ponorky a jako návratovou hodnotu vrací ukazatel na objekt exploze.

4.2.3 Hlavní objekty

Hlavní objekty jsou nejdůležitější objekty použité ve hře.

Player Třída `Player` dědí ze třídy `Submarine` a reprezentuje hráče. Každý hráč má své unikátní id, které je mu přiděleno serverem.

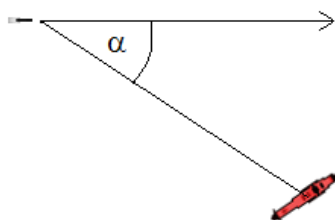
Map Tato třída reprezentuje mapu. Mapa je implementována jako pole náhodně vygenerovaných kruhů. Mapa se generuje ve třech krocích:

```
void startGenerating(void);
void addProtectedArea(int x, int y, int r);
void generateNew(void);
```

Prvním krokem je metoda `startGenerating`, která mapu inicializuje. Druhým krokem je přidávání chráněných území pomocí metody `addProtectedArea`. Ta jako argumenty přebírá souřadnice a poloměr kruhu s nímž nesmí kolidovat žádná vygenerovaná část mapy. Tyto informace ukládá do pole chráněných území. Tímto je zajištěno, že mapa neblokuje místo, kde se objeví hráčova ponorka. Třetím a posledním krokem je vlastní generování mapy pomocí metody `generateNew`. Tato metoda generuje kruhy s náhodnou pozicí a poloměrem². Každý vygenerovaný kruh je zkontrolován, zda nekoliduje s některým chráněným územím. Pokud ne, je tento kruh přidán do pole kruhů dané mapy. Pokud ano, je vygenerován nový kruh. Plocha všech validních kruhů je sčítána a kruhy jsou generovány do té doby, než celková plocha vygenerovaných kruhů překročí $\frac{1}{7}$ celkové plochy mapy. Vykreslování mapy pomocí metody `draw` funguje tak, že se bitmap pro jeden „kruh mapy“ vykreslí pro každý „kruh mapy“ podle jeho velikosti a pozice.

²Poloměr má maximum a minimum definované jako konstanty ve třídě `Map`.

StandardTorpedo, NavigatedTorpedo, StandardMine, NavigatedMine Tyto třídy reprezentují veškeré zbraně implementované ve hře. **StandardTorpedo** je torpédo (dědí ze třídy **Torpedo**), které má konstantní rychlost a směr. **NavigatedTorpedo** je torpédo, které je pomalejší než **StandardTorpedo**, ale má automatické navádění. To funguje tak, že se pro každý z možných cílů spočítá úhel mezi směrem torpéda a pozicí cíle (obrázek 4.2). Torpédo se poté navádí na cíl, který je pod nejmenším úhlem. **StandardMine** je mina (dědí ze třídy **Mine**), která zůstává na místě, kam byla hráčem umístěna. **NavigatedMine** je mina, která zůstává na svém místě, dokud se k ní nepřiblíží ponorka na určitou vzdálenost. Tato vzdálenost je definována jako konstanta a pokud ji ponorka překročí, mina se konstantní rychlostí začne pohybovat směrem k ponorce a začne ji „honit“.



Obrázek 4.2: Výpočet úhlu pro navigované torpédo

4.3 Detekce kolizí

Pro detekci kolizí slouží třída **CollisionShape**, jejíž instanci obsahuje v sobě každý objekt.

4.3.1 CollisionShape

Tato třída tvoří „kolizní tvar“ objektu. Tento tvar je tvořen skupinou kruhů, které jsou uspořádány vždy tak, že pokrývají co nejlépe daný objekt. Pro snížení výpočetní náročnosti, existuje ještě jeden kruh, který byl pojmenován *aproximační* a ten pokrývá celý objekt.

Instance třídy **CollisionShape** je vygenerována po každé aktualizaci objektu, protože při každé aktualizaci může objekt změnit svou pozici či směr. Pro tuto činnost existuje metoda **generateCollisionShape**. Každá třída, jejíž instance mají specifický „kolizní tvar“, obsahuje svou vlastní definici této metody. Vzhled „kolizního tvaru“ pro ponorku je zobrazen na obrázku 4.3.



Obrázek 4.3: „Kolizní tvar“ pro ponorku

Metoda `collidesWith`, která přijímá jako argument instanci třídy `CollisionShape`, vypočítává, zda spolu dvě instance kolidují. První zjistí, zda spolu kolidují aproximační kruhy obou instancí. Pokud ano, testuje každou dvojici kruhů, které tvoří „kolizní tvary“ daných instancí. Kolize mezi kruhy nastane, pokud vzdálenost jejich středů je větší než součet jejich poloměrů (rovnice 4.1).

$$r_1 + r_2 > \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (4.1)$$

4.3.2 Dynamické zjišťování typu objektu

Jelikož každý typ objektu může mít při kolizi s ostatními typy různé chování, je nutné během kolize vědět, jaké typy objektů spolu kolidují. Poté může být zavolána příslušná funkce. Pro vyřešení tohoto problému byla použita metoda *double dispatch*.

Pro ilustraci jak tato metoda funguje uvažujme následující situaci. Máme abstraktní třídu, která reprezentuje chemickou látku X a pak máme 3 třídy, které představují 3 různé chemické látky A, B a C. My chceme vytvořit funkci, které lze předat jakékoli látky a ona provede jejich reakci:

```
void reaction(X * const x1, X * const x2);
```

Řešení tohoto problému v C++ pak vypadá takto:

```
void AwithA(void) { // Reakce A s A }
void AwithB(void) { // Reakce A s B }
void AwithC(void) { // Reakce A s C }
void BwithB(void) { // Reakce B s B }
void BwithC(void) { // Reakce B s C }
void CwithC(void) { // Reakce C s C }

class X {
    virtual void reactWith(X * const x) = 0;
    virtual void reactWith(A * const a) = 0;
    virtual void reactWith(B * const b) = 0;
    virtual void reactWith(C * const c) = 0;
};

class A : public X {
    virtual void reactWith(X * const x) { x->reactWith(this); }
    virtual void reactWith(A * const a) { AwithA(); }
    virtual void reactWith(B * const b) { AwithB(); }
    virtual void reactWith(C * const c) { AwithC(); }
};

class B : public X {
    virtual void reactWith(X * const x) { x->reactWith(this); }
    virtual void reactWith(A * const a) { AwithB(); }
    virtual void reactWith(B * const b) { BwithB(); }
    virtual void reactWith(C * const c) { BwithC(); }
};

class C : public X {
    virtual void reactWith(X * const x) { x->reactWith(this); }
    virtual void reactWith(A * const a) { AwithC(); }
    virtual void reactWith(B * const b) { BwithC(); }
    virtual void reactWith(C * const c) { CwithC(); }
};
```

```
void reaction(X * const x1, X * const x2) { x1->reactWith(x2); }
```

Takto můžeme i pro dvě látky jejichž typ není znám provést reakci. Uvažujme například situaci:

```
...
A a;
B b;
X * const x1 = &a;
X * const x2 = &b;
reaction(x1, x2);
...
```

Ve funkci `reaction` první dojde k volání metody `A::reactWith(X * const x)`. V této metodě pak dojde k volání metody `B::reactWith(A * const a)`. Tato metoda už pak volá přímo funkci `AwithB()`, což je i to, co bychom očekávali. Stejným způsobem pak funguje implementace detekce kolizí ve hře.

4.4 Implementace síťové komunikace

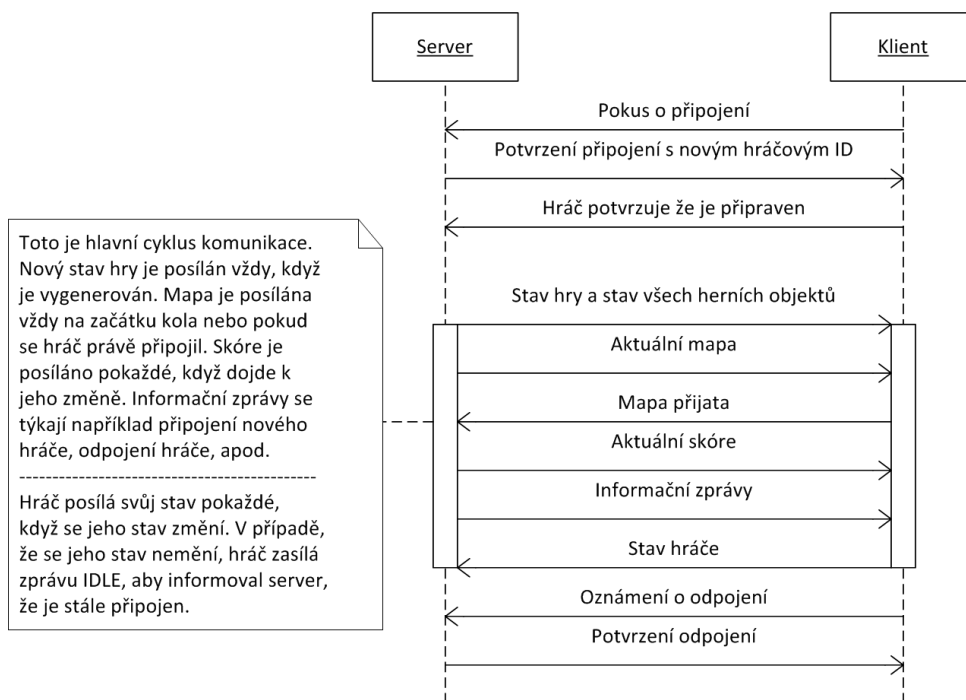
Síťová komunikace byla implementována pomocí knihovny `Boost.Asio`, která poskytuje abstrakci nad sokety a pomocí knihovny `Boost.Serialization`, která poskytuje funkce pro serializaci a deserializaci dat. Jediným problémem této knihovny je, že serializace dat probíhá jejich přepsáním do textové podoby. Tím je sice zaručena dobrá přenositelnost, ale často to také vede k nadměrné velikosti dat. Pro serializaci vlastních tříd musí být vytvořena speciální funkce, která určuje, která data mají být serializována a poté deserializována.

4.4.1 Průběh komunikace

Při posílání dat přes síť musí být vytvořen archiv, do kterého jsou tato data uložena. Pořadí v jakém jsou data uložena je stejné s pořadím, v kterém pak musí být data z archivu načtena. Pro rozpoznávání dat byla implementována třída `MessageHeader`, která představuje hlavičku zprávy. V této hlavičce je uložen typ zprávy a jedna proměnná pro číslo zprávy. Toto číslo slouží různě u různých typů zpráv, ale nejčastěji je využito jako sekvenční číslo zprávy. Toho je poté možné využít pro rozpoznání starší zprávy, která přišla mimo pořadí. Jelikož tato zpráva obsahuje stará data, jsou tato data u některých typů zpráv zahozena. Příjemce zprávy podle typu zprávy uloženého v hlavičce pozná, jaká další data mohou být ze zprávy načteny.

Standardní průběh komunikace je zobrazen na obrázku 4.4. Zobrazuje navázání spojení, posílání zpráv v hlavní smyčce během hry a ukončení komunikace.

Při připojování mohou kromě úspěšného připojení nastat další situace. Pokud server neodpoví, vyprší časovač u klienta a zobrazí se zpráva, že server neodpovídá. Pokud server klienta nemůže přijmout, odpoví mu zprávou o zamítnutí připojení a v takovém případě se klientovi zobrazí zpráva, že server je plný. Další zvláštní situace může nastat, pokud je server ukončen. V takovém případě jsou pomocí speciální zprávy informováni všichni klienti a ve hře se jim vypíše zpráva o ukončení serveru. Hra zůstane nadále spuštěna, ale nebude fungovat logika serveru ani propojení s dalšími hráči.



Obrázek 4.4: Standardní průběh komunikace mezi klientem a serverem

4.4.2 Třídy používané pro zasílání dat

Pro každou ze tříd, jejíž některá data měla být posílána, byla vytvořena třída reprezentující právě tato data. Pro třídu `Player` existuje třída `PlayerState`, která obsahuje informace o stavu hráče. Všechny booleovské atributy byly pro ušetření místa uloženy do jedné proměnné jako pole bitů. Jediná hráčská data, která klient posílá na server, jsou právě stavy hráče doplněné o čas hry, ze kterého daný stav pochází. Další třídou, jejíž instance jsou posílány přes síť, je třída `PlayerInfo`. Ta v sobě obsahuje třídu `PlayerState`, ale kromě stavu hráče obsahuje i jeho id, pozici, rychlost a směr. Vytváří tak kompletní informace o hráči. Třída `Player` pak má metody, které umožňují, aby byla aktualizována ze tříd `PlayerState` a `PlayerInfo`. Protože však server je autoritativní a některé stavy hráče by nemělo být možné přepsat klientem, existují pro aktualizaci stavu hráče dvě funkce, z nichž jedna aktualizuje všechny atributy hráče, ale druhá pouze některé.

Dále pro třídu `Weapon` existuje třída `WeaponInfo` a pro zasílání informací o skóre byla vytvořena třída `GameScore`. Ta zasílá data jako pole skóre jednotlivých hráčů ve formě: jméno hráče, počet vítězství, počet fragů, počet smrtí. Důležitou třídou je také třída `GameState`, která obsahuje všechny informace o aktuálním stavu hry. Ta obsahuje pole instancí třídy `PlayerInfo`, pole instancí třídy `WeaponInfo` a také časové razítko kdy byl tento stav vytvořen. Právě tuto třídu zasílá server všem klientům a je také využívána serverem pro ukládání stavů hry.

4.5 Implementace serveru

Server běží ve třech vláknech. Jedno vlákno slouží pro terminál, ve kterém jsou zpracovávány uživatelské příkazy. Ve druhém vlákne probíhá přijímání dat ze sítě a ve třetím běží odesílání dat a logika hry. Terminál je implementován tak, že čeká na uživatelský vstup a ten se

poté pokusí rozpoznat jako příkaz. Vlákno pro přijímání zpráv po přijetí zprávy extrahuje z této zprávy hlavičku a zjistí její typ. Podle typu pak provede příslušnou činnost. Hlavní a nejsložitější částí serveru je logika hry (sekce 4.5.2).

Server obsahuje mimo jiné seznam klientů. Klient je do seznamu přidán poté, co se serverem úspěšně naváže komunikaci. Je mu poté přiděleno ID, které ho jednoznačně identifikuje. Tuto funkci plní třída `Client`. Ta také obsahuje informace jako je čas poslední přijaté zprávy (pro zjištění, zda klient nepřestal komunikovat), počet výher, fragů a smrtí klienta, jeho jméno, ale také frontu herních akcí, které od něj byly serverem přijaty. Každý klient také má svou instanci třídy `Player`, která reprezentuje jeho ponorku.

4.5.1 Zjišťování globální IP adresy

Jelikož zjišťování globální IP adresy není triviální záležitost, musela být pro něj vytvořena speciální funkce. Pro získání globální IP adresy je využito webového stránky:

```
http://automation.whatismyip.com/n09230945.asp
```

Tato webová stránka vypisuje vaši IP adresu. Stačí tedy zaslat HTML dotaz `GET`³ a z obdržené webové stránky pak pouze vyextrahovat IP adresu. Jediným problémem je, že pokud přestane tento server fungovat, přestane fungovat i funkce pro zjištění globální IP adresy.

4.5.2 Logika hry

Veškerá logika hry se odehrává na serveru. Logiku hry tvoří smyčka, která probíhá v určitém časovém intervalu. Ten je udáván časovačem. V každém časovém intervalu je logice hry předán uplynulý čas od poslední aktualizace. Zde je zevrubně popsání jejího algoritmu:

1. Zjišťování stavu hry
2. Aktualizace kalendáře akcí
3. Nalezení vhodného záznamu stavu hry
4. Aktualizace hry
5. Detekce kolizí
6. Uložení stavu hry
7. Odeslání stavu hry hráčům

Zjišťování stavu hry

Zjišťování stavu hry je popsáno algoritmem 4.1.

Pokud při zjišťování stavu hry dojde algoritmus k příkazu `Ukončit`, znamená to, že logiku hry není nutné v tuto chvíli provádět.

³GET se používá pokud chce klient získat obsah webové stránky.

Algoritmus 4.1 Zjišťování stavu hry

```
if Logika hry je aktivní then
  if Počet hráčů = 0 then
    Zastavit logiku hry
    Ukončit
  else if StavHry ≠ KonecKola then
    Spočítat počet živých a „neživých“ hráčů.
    if Počet živých hráčů ≤ 2 and Počet „neživých“ hráčů ≥ 0 then
      StavHry := KonecKola
      Přidat výhru hráči, který přežil
      ČasKonceKola := ČasKola + TrváníKonceKola
else
  if Počet hráčů = 0 then
    Ukončit
  else
    Spustit logiku hry
    Start kola
    Ukončit
if ČasKola > ČasKonceKola then
  Ukončení kola
  Start kola
  Ukončit
if StavHry = FreezeTime then
  Aktualizovat ČasKola
  if ČasKola < DélkaFreezeTime then
    Ukončit
  else
    Odstranit všechny akce hráčů které nastaly během freeze-timu
    Zrušit freeze-time
    Nastavit správně čas kola
```

Aktualizace kalendáře akcí a nalezení vhodného záznamu stavu hry

Aktualizace probíhá tak, že se všechny akce hráčů, které dorazily na server přidají do kalendáře akcí. Kalendář akcí je seřazený podle herního času, kdy jednotlivé akce nastaly. Zároveň jsou odstraněny všechny akce, které jsou starší než 1 vteřina od aktuálního času hry. Také akce, jejichž čas předbíhá čas hry o více než vteřinu, jsou odstraněny. Toto nastává občas při ukončení kola, kdy hráči posílají ještě akce z minulého kola, ale běží již kolo nové.

Poté je vypočítán na základě současného času hry a času uplynulého od poslední aktualizace finální čas pro tuto aktualizaci (sečtením těchto časů). Do tohoto času bude simulována logika hry. Jelikož od hráčů přicházejí i akce, které proběhly vzhledem k aktuálnímu času hry na serveru v minulosti, musí být server schopný vrátit se v čase do posledního stavu hry před určitým časem akce. Pokud tedy byly přijaty od hráčů nějaké akce, je nalezena ta z nich, která má nejnižší čas. Poté je v seznamu snímků stavu hry nalezen poslední validní stav, který předcházel dané akci. Hra je přesunuta do tohoto stavu.

Aktualizace hry a detekce kolizí

V kalendáři akcí je nalezena první, která nastává po aktuálním čase hry. Poté se postupně procházejí všechny akce v kalendáři až do doby, než čas akce překročí finální čas pro tuto aktualizaci nebo pokud v kalendáři akcí již žádné akce nejsou. Pro každou z těchto akcí je pak vypočítán čas, který uplynul, a všechny objekty hry jsou aktualizovány o tento čas. Poté je akce provedena. Nakonec se objekty hry aktualizují o čas, který zbývá do finálního času pro tuto aktualizaci.

Detekce kolizí je první provedena pro všechny hráče. První se kontroluje, zda hráč koliduje s některým dalším hráčem. Poté, jestli koliduje s některou zbraní. Pokud ano, je hráči, který zbraň vypustil, přidán jeden frag. Nakonec se kontroluje kolize hráče s mapou. Pokud byl hráč během detekce kolizí zničen, je mu přičtena jedna smrt. Po kolizích hráčů se kontrolují všechny zbraně, zda nekolidují s mapou. V případě že ano, je zbraň zničena.

Uložení stavu hry a odeslání stavu hry hráčům

Stav hry obsahuje informace o všech hráčích, o všech zbraních a čas hry, ve kterém byl tento stav vytvořen. Stav je uložen na konci každé aktualizace, přičemž první stav je vytvořen na začátku kola, před první aktualizací. Po každé aktualizaci je vytvořený stav odeslán postupně všem hráčům. Pokud se nějak během aktualizace změnilo skóre hry, je hráčům tato informace odeslána, stejně jako všechny informační zprávy o událostech, které na serveru nastaly (jako například připojení nebo odpojení hráče).

4.6 Implementace klienta

Klient běží ve dvou vláknech. Jedno vlákno slouží pro přijímání zpráv a jedno pro samotnou hru. K implementaci klienta byla využita série tříd, z nichž každá spravuje některou z částí hry.

4.6.1 SoundManager

Třída `SoundManager`, jak již název napovídá, spravuje zvuky. Stará se o přehrávání hudby, která hraje v průběhu celé hry a také poskytuje funkci pro přehrání zvukových efektů v podobě explozí.

4.6.2 ResourceManager

Třída `ResourceManager` spravuje veškeré prostředky potřebné ve hře, krom zvuků. Je implementována jako jedináček⁴, aby ji mohli využívat všechny ostatní třídy.

`ResourceManager` při svém vytvoření načte z předem definovaných souborů obrázky pro všechny objekty hry, všechny potřebné fonty a vytvoří všechny potřebné animace. Každý z obrázků, fontů a animací je pak uložen jako atribut. Tyto jsou pak dostupné pomocí příslušných funkcí. Tyto funkce vracejí referenci na daný atribut, a tak je možné měnit fonty, obrázky i animace za běhu hry. `ResourceManager` také poskytuje statickou funkci pro načtení bitmapy ze souboru.

⁴Návrhový vzor jedináček je známější pod anglickým pojmem *Singleton* a jedná se o třídu, které lze vytvořit pouze jednou instancí. Tato instance je globální a je tak dostupná z celého programu.

4.6.3 GraphicManager

Třída `GraphicManager` se stará o problematiku spojenou s grafikou. Jejími nejdůležitějšími metodami jsou:

```
void beginDrawing(void);  
void finishDrawing(double current_time);
```

Metoda `beginDrawing` je použita vždy, před začátkem vykreslování na obrazovku. Tato metoda inicializuje vše potřebné pro vykreslování a po jejím zavolání může začít vykreslování objektů. Vykreslování bylo implementováno tak, aby mohlo být použito pro jakékoli rozlišení. Hra má pouze jedno rozlišení a funkce `beginDrawing` ve skutečnosti přepne místo, kam se bude vykreslovat z obrazovky na bitmap, který existuje jako atribut třídy `GraphicManager`. Tento bitmap má velikost stejnou jako je rozlišení hry. Celá hra je tak vykreslena na tento bitmap. Metoda `finishDrawing` je použita pro ukončení vykreslování a až tato metoda ve skutečnosti vykreslí vše na obrazovku. Provádí to tak, že bitmap, na který byla hra vykreslena, roztáhne na aktuální rozlišení obrazovky. To pak může způsobit mírnou deformaci vzhledu hry. Metoda `finishDrawing` přijímá jako argument současný čas a to proto, aby mohla vypočítávat reálný počet snímků vykreslených za sekundu.

4.6.4 InputManager

`InputManager` je třída, která se stará o uživatelský vstup a využívá pro to třídy `GameAction`. Třída `GameAction` představuje herní akci jako je třeba zatočení vlevo, zrychlení, atd. Každá akce má své jméno a také identifikátor aktivity. Ten specifikuje, zda je akce aktivní po celou dobu nebo pouze jednou a poté může být znova aktivní až po deaktivaci. Jako příklad může být uveden rozdíl mezi zrychlením a vystřelením zbraně. Pokud držíme tlačítko pro zrychlení, přejeme si aby bylo aktivní po celou dobu stisknutí. U výstřelu ze zbraně tomu tak není (i když také může být u jiného typu hry). Pokud stiskneme klávesu pro výstřel, přejeme si, aby výstřel proběhl pouze jednou. To, že je klávesa stisknuta delší čas, by na počet výstřelů nemělo mít vliv. Třída `GameAction` pak obsahuje metodu `isActive`, která zjišťuje zda je daná akce hry aktivní.

`InputManager` poté obsahuje pole ukazatelů na akci hry, kdy každý prvek pole je mapován na jednu klávesu klávesnice. Pokud klávesa nic nedělá, má ukazatel hodnotu `nullptr`. Standardně jsou nastaveny základní klávesy, ale třída obsahuje metodu pro nastavení akce na nové klávesy. Důležitou metodou je metoda `processEvent`, která zpracovává událost vyvolanou stiskem klávesy tak, že upravuje patřičné akce hry, pokud se jich daný stisk klávesy týká.

4.6.5 NetworkManager

Třídou pro správu síťové komunikace je `NetworkManager`. K navázání spojení slouží metoda `connect`, která se pokusí připojit na zadaný server a vyvolá výjimku při selhání připojení. Pokud je připojení navázáno, je vytvořeno vlákno, které se stará o příjem zpráv od serveru. Pokud jsou přijata data, jsou uložena a je vyvolána událost (event), která je zachycena v hlavní smyčce hry, a která má typ podle typu přijatých dat.

`NetworkManager` inicializuje po připojení k serveru časovač, který vyvolává události, značící, že by měl být serveru odeslán současný stav hráče. Tyto události jsou opět zachycovány hlavní smyčkou hry, ale pro odeslání dat na server je využíván `NetworkManager`.

Konkrétně jsou používány metody `sendGameEvent`, pokud hráč změnil svůj stav a `sendIdle` pokud hráč svůj stav nezměnil.

4.6.6 GameObjectManager

Třída `GameObjectManager` spravuje veškeré herní objekty hry. Stará se jak o jejich aktualizaci, tak o jejich vykreslování. Hlavními objekty hry jsou pak:

- Informační panel, na kterém jsou zobrazeny všechny informace o hráči
- Pozadí hry (voda)
- Mapa
- Seznam hráčů
- Ostatní herní objekty (zbraně a exploze)

`GameObjectManager` má implementovány metody pro přidávání a vyhledávání hráčů, přidávání zbraní, přidávání ostatních objektů a jejich vyhledávání:

```
void addPlayer(Player * const player);
Player * const getPlayer(unsigned player_id);
void addWeapon(const WeaponInfo & weapon_info);
void addObject(Critter * const object);
Critter * const getObject(unsigned id);
```

Tyto metody se chovají tak, jak napovídají jejich názvy. Důležitými metodami jsou pak také:

```
void startActualization(void);
void finishActualization(void);
void removeDisposable(void);
```

Metoda `startActualization` je volána vždy při začátku aktualizace všech objektů (poté co je od serveru přijat nový stav hry). Způsobuje, že všichni hráči a zbraně které jsou aktualizovány si uloží, že byli aktualizováni. Metoda `finishActualization` poté odstraní všechny objekty, které aktualizovány nebyly, protože tyto objekty již na serveru neexistují. Metoda `removeDisposable` odstraňuje všechny objekty, které mají nastavený příznak `is_disposable`. Tento příznak značí, že mohou být odstraněny. Děje se tak například u explozí poté, co skončí jejich animace.

Metoda pro vykreslování volá u všech objektů funkci `draw`. První vykreslí pozadí, na něj mapu, poté všechny hráče, pak všechny zbraně a nakonec informační panel. Informační panel je vykreslován na základě aktuálního stavu hráče, který je této metodě předán jako argument, ale jsou na něm zobrazeny i informační zprávy ze serveru. Metoda pro aktualizaci postupuje ve stejném pořadí, kdy pro všechny objekty hry je postupně volána metoda `update`. Pokud hráč navíc drží tlačítko pro vypsání aktuálního skóre, `GameObjectManager` vykresluje i toto skóre pomocí funkce `showScore`.

4.6.7 GameManagerer

Hlavní třídou klienta je třída `GameManagerer`, která využívá všech ostatních správců zmíněných výše. Její hlavní metodou je metoda `run`, která spouští hlavní smyčku celého klienta. `GameManagerer` se první pokusí připojit na server. Pokud se to povede, je vytvořen časovač podle nastavených snímků za sekundu. Tento časovač zajišťuje pravidelné vykreslování a aktualizace hry. Poté se spustí hlavní smyčka programu.

V hlavní smyčce se první čeká na nějakou událost. Touto událostí může být vypršení některého z časovačů (časovač pro aktualizace nebo časovač pro zasilání stavu klienta na server), stisknutí klávesy uživatelem nebo událost vyvolaná přijetím zprávy ze serveru. Po přijetí události je spočítán čas, který uplynul od poslední aktualizace (δt). Stisknutí klávesy je předáno instanci třídy `InputManager`, která ji zpracuje. Poté je hra aktualizována pomocí δt a stav hráče je nastaven podle stisknutých kláves. Tento stav je pak odeslán na server a zároveň uložen pro pozdější použití. Pokud událost byla způsobena časovačem pro aktualizace, je hra aktualizována a je nastaven příznak pro vykreslení hry. Pokud je ve frontě událostí již další událost, hra vykreslena není a je vykreslena až ve chvíli, kdy žádná další událost ve frontě událostí nečeká. Tímto je zajištěno, že hra bude vykreslována jen v případě, že je na to dostatek času. Další událostí, která může nastat je vypršení časovače pro posílání stavu hráče serveru. Pokud je hráč naživu je odeslán jeho stav. Jinak je poslána IDLE zpráva, která server pouze informuje, že klient je stále připojen. Dalšími událostmi, které mohou nastat, jsou přijetí zprávy ze serveru a to buď nový stav hry, nová mapa nebo nová informační zpráva. Pokud je přijata nová mapa nebo informační zpráva, je tato předána instanci třídy `GameObjectManager` a ta ji poté spravuje. Zpracování nového stavu hry je složitější.

Zpracování nového stavu hry

Pro správný běh hry je důležitá správná synchronizace se serverem. Pro tuto činnost existuje na klientovi příznak pro synchronizaci. Synchronizace probíhá tak, že veškeré akce klienta jsou zahozeny, čas je nastaven na čas přijatého stavu hry a jsou odstraněny všechny objekty, které mohou být odstraněny. Až poté probíhá vlastní zpracování nového stavu hry. K synchronizaci dochází, pokud příznak synchronizace není nastaven, pokud přijatý čas je menší než čas přijatý naposledy (to nastává při začátku nového kola), nebo pokud přijatý čas je větší než aktuální čas na klientovi.

Všichni hráči jsou aktualizováni na základě přijatých informací. Pokud je přijata informace o hráči, který ve hře není, je tento hráč vytvořen. To samé platí i pro zbraně. Poté je na základě uložených akcí hráče hra simulována mezi časem, který byl přijat ze serveru a současným časem hry na klientovi.

4.7 Testování

Jelikož byla hra vyvíjena iterativně, byla testována postupně. Díky tomu byla odhalena spousta problémů od problémů s rozlišením až po problémy se synchronizací klienta a serveru.

V první fázi byl testován základní vzhled hry. Bylo to pouze pozadí s ponorkou, kterou uživatel mohl pomocí šipek ovládat. Takto byla otestována reaktivnost ponorky a možnosti jejího ovládání. Také se přišlo na to, že se hra při jiném rozlišení obrazovky vykresluje špatně. Proto byl navržen systém, který prošel až do finální verze hry. Hra se nevykresluje

na obrazovku, ale na bitmap který je na všech počítačích stejný a až ten se potom vykresluje na celou obrazovku.

Ve druhé fázi byla testována komunikace mezi klientem a serverem. Zde bylo zjištěno, že navržený model hlavní smyčky síťové komunikace bude muset být kompletně předělán. Byl zbytečně komplexní pro tak jednoduchou hru a navíc, a to je asi hlavní, nefungoval. Nový navržený model je již použit ve finální hře a funguje poměrně spolehlivě.

Třetí fáze testování byla zaměřena na více hráčů na serveru. Bylo již možné střílet torpéda, ale nebyla ještě vytvořena detekce kolizí. Šlo jen o ověření, zda se server při větším náporu nezhroutí, nebo zda klienti nedostávají nesmyslná data. Testování probíhalo jak na lokální síti, tak na síti, kdy odezvy hráčů byly cca od 10 ms do 100 ms. Testování neodhalilo žádné závažnější problémy. Navíc bylo ověřeno, že všichni hráči vidí hru ve stejném stavu.

Ve čtvrté fázi testování byla testována detekce kolizí a také všechny zbraně. Bylo zjištěno, že detekce kolizí funguje dobře, ale u zbraní byly mírně pozměněny některé vlastnosti, jako rychlost a schopnost navigace (navigované miny reagovali na velké vzdálenosti a byly moc rychlé). Byla také navýšena rychlost ponorek.

V páté fázi bylo do hry přidáno generování mapy a časování pro zbraně. Generování mapy fungovalo dobře nicméně časy nabíjení zbraní byly sníženy, protože se ukázalo, že zasáhnout protivníka je poměrně složité a čekat pak dlouhou dobu na nabití zbraně je nezábavné. Také byly odhaleny a opraveny drobné chyby při aktualizaci nabití zbraně.

V poslední fázi byl přidán informační panel a zvuky. Odezva na toto uživatelské rozhraní byla dobrá. Informační panel určitě usnadnil hraní a přehled ve hře. Přestože tato fáze neměnila nic na síti, byly až v ní odhaleny některé chyby. Některé menší, ale jedna poměrně závažná. Tuto chybu se nepodařilo odstranit. Hráči se při ní mohou pohnout pouze o kousek a server je neustále vrací na jejich startovní pozici. Tato chyba zmizí po začátku nového kola, ale většinou se zase za několik kol objeví.

Kapitola 5

Závěr

To, že ve hře zůstaly chyby je poměrně velkou kaňkou a to zejména ona chyba (zminěná v sekci 4.7), kdy je hra pro hráče na celé jedno kolo nehratelná. Nicméně testování a hledání chyb u dvou vícevláknových programů, které navíc běží na různých počítačích se ukázalo jako velice náročné. Byl k tomuto účelu, jak u klienta, tak u serveru implementován logovací systém, který zapisoval záznamy o různých událostech do souboru. Počet řádků v těchto souborech byl však v řádech deseti-tisíců a najít v něm moment, kdy došlo k chybě a co ji vlastně způsobilo vyžadovalo značné úsilí a také hodně času.

Hratelnost byla ve finální fázi uživateli zhodnocena jako *dobrá* a hra jim přišla i zábavná. To by se dalo považovat za dílčí úspěch stejně jako fakt, že hra až na zmiňovanou chybu funguje celkem bezproblémově.

Během vývoje a testování vznikla spousta nápadů na rozšíření či vylepšení hry. Zmíním zde jen ty, které považuji pro hru za nejužitečnější.

1. V první řadě nejde ani o vylepšení, ani o rozšíření hry, ale o nalezení a opravu oné výše zmíněné chyby. Tato chyba by určitě hře zabránila v uchycení i v malém okruhu hráčů.
2. Vytvoření menu pro hru. Ačkoli menu bylo pro hru navrženo a některé jeho části byli i naprogramovány, nebylo ve finální verzi implementováno. Není to sice zásadní problém, ale je určitě dobré mít ve hře menu a nemuset řešit nastavení pomocí konfiguračních souborů.
3. Přidat možnost na serveru měnit vlastnosti zbraní tak, aby si každý mohl zbraně nastavit podle svého gusta. Dále také možnost měnit nastavení pro generování mapy a vylepšit generování mapy přidáním jiných tvarů než kruhů. Také možnost přidat například podmořské sopky, které by čas od času vybuchly a zničily ponorky v jejich blízkosti. Chvilí před výbuchem by se na hladině mohly objevit bublinky, které by hráče na sopku upozornily.
4. Implementovat další herní módy. Například navržený *DeathMatch* nebo mód, kdy budou hráči hrát v týmech.
5. Zvětšit velikost mapy za hranice obrazovky, kdy hráč by byl v centru obrazovky a mapa by se pohybovala. Hráč by také nemusel vidět celou obrazovku, ale třeba jen určité okolí kolem své ponorky. Maximální množství hráčů by se pak mohlo zvýšit a hra by mohla být zajímavější.

6. Vylepšit grafický vzhled hry. Ačkoli grafický vzhled ponorky je animace, má tato animace pouze jeden snímek. Stačí vytvořit obrázky ponorky například pro zatáčení, zrychlování, apod. a poté nastavit, aby se animace měnila podle současného stavu ponorky. Takto to mělo podle návrhu být, ovšem nezbyl čas na kreslení ponorek.
7. Přidat více zbraní a změnit množství životů ponorky. Každá zbraň by pak mohla ubírat jiné množství životů.
8. Redukovat množství dat posílaných přes síť a použít kompresi dat. Jelikož jsou data posílána jako text, komprese by mohla být celkem účinná.
9. Přidat do hry chat, přes který by si hráči mohli posílat zprávy a přes který by mohl být ovládán i server. Hráč by se například musel první přihlásit jako administrátor speciální zprávou a heslem a poté by mohl měnit nastavení serveru jako by zadával příkazy přímo do terminálu serveru.
10. Bylo by vhodné dodělat implementaci pro adaptaci hry na různá rozlišení obrazovky.
11. Měl by se otestovat klient i server, jaké množství dat oba posílají a jak velký výkon procesoru spotřebovávají. Toto bylo sice provedeno, ale jen orientačně. Bylo by dobré vědět, jak velkým zatížením hra i server jsou a zda by se nemělo zapracovat na nějakých optimalizacích.

Samozřejmě se dá najít ještě spousta věcí, jak by se hra dala vylepšit. Já osobně bych hru přepracoval kompletně od začátku, když už teď vím a mám zkušenosti, jak řešit některé problémy. Velká část návrhu byla tvořena s minimem praxe. To se ve výsledku dost projevilo a to i přes iterativní vývoj hry, kdy některé části byly kompletně přeprogramovány, někdy i vícekrát.

Zadání práce dá se říct bylo splněno. Hra byla formálně popsána a byla jasně dána její pravidla. Uživatelské rozhraní bylo navrženo a implementováno. Možnosti síťové komunikace byly analyzovány a vhodně implementovány stejně jako komunikační protokol. Celá hra byla otestována jak co se týče hratelnosti, tak co se týče funkčnosti, i když funkčnost nebyla stoprocentní. Výše byly zmíněny i možnosti dalšího rozšíření.

Literatura

- [1] Wikipedie: Otevřená encyklopedie. *Arkáda (žánr počítačových her)*. Listopad 2011, [online] [citováno 19. 01. 2012]. Dostupné z: [http://cs.wikipedia.org/w/index.php?title=Ark%C3%A1da_\(%C5%BE%C3%A1nr_po%C4%8D%C3%ADta%C4%8Dov%C3%BDch_her\)&oldid=7583285](http://cs.wikipedia.org/w/index.php?title=Ark%C3%A1da_(%C5%BE%C3%A1nr_po%C4%8D%C3%ADta%C4%8Dov%C3%BDch_her)&oldid=7583285)
- [2] VALVE: Developer Comunity. *Source Multiplayer Networking*. Říjen 2011, [online] [citováno 25. 06. 2012]. Dostupné z: https://developer.valvesoftware.com/w/index.php?title=Source_Multiplayer_Networking&oldid=161213
- [3] ARMITAGE, Grenville, CLAYPOOL, Mark, BRANCH, Philip. *Networking and Online Games: Understanding and Engineering Multiplayer Internet Games*. John Wiley & Sons Ltd, 2006. ISBN 978-0-470-01857-6. .
- [4] BERNIER, Yahn W.. VALVE. *Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization*. 2001, [online] [citováno 25. 06. 2012]. Dostupné z: <http://web.cs.wpi.edu/~claypool/courses/4513-B03/papers/games/bernier.pdf>
- [5] BETTNER, Paul, TERRANO, Mark. 1500 archers on a 28.8: Network programming in Age of Empires and beyond. V: *Game Developer Conference 2001*, San Jose USA, Březen 2001. Dostupné z: http://www.gamasutra.com/view/feature/3094/1500_archers_on_a_288_network_.php
- [6] CHEN, Kuan-Ta, HUANG, Chun-Ying, HUANG, Polly, et al. An Empirical Evaluation of TCP Performance in Online Games. V: *Proceedings of ACM SIGCHI ACE 06*, Los Angeles USA, Červen 2006.
- [7] HARBOUR, Jonathan S.. *Game Programming All in One*. Third edition vydání. Thomson Course Technology PTR, 2007. ISBN 978-1-59863-289-7. .
- [8] ŽÁRA, Jiří, BENEŠ, Bedřich, SOCHOR, Jiří, et al. *Moderní počítačová grafika*. Druhé, přepracované a rozšířené vydání. Brno: Computer Press, 2004. ISBN 80-251-0454-0. .
- [9] RUCKER, Rudy. *Software Engineering and Computer Games*. Addison-Wesley, 2011. ISBN 0201-767910. .
- [10] SAWASHIMA, Hidenari, HORI, Yoshiaki, SUNAHARA, Hideki, et al. *Characteristics of UDP Packet Loss Effect of TCP Traffic*. 1997. Dostupné z: https://www.isoc.org/inet97/proceedings/F3/F3_1.HTM

Příloha A

Tabulka příkazů pro server

Příkazy se zadávají do terminálu serveru po jeho spuštění. Pokud se v tabulce za nějakým příkazem vyskytuje hodnota v hranatých závorkách, znamená to, že tento příkaz lze zadat bez hodnoty. V takovém případě příkaz pouze vypisuje stávající hodnotu.

Příkaz	Význam
<code>help</code>	vytiskne nápovědu
<code>load_cfg FILE</code>	načte nastavení ze souboru <code>FILE</code> , který musí být ve složce <code>cfg</code>
<code>start</code>	spustí běh serveru
<code>stop</code>	zastaví běh serveru
<code>restart</code>	restartuje běh serveru
<code>exit</code>	ukončí běh serveru
<code>ip</code>	vypíše globální IP adresu serveru (pokud je dostupná)
<code>loacl_ip</code>	vypíše lokální IP adresu serveru
<code>port [PORT]</code>	vypíše nebo nastaví port serveru na hodnotu <code>PORT</code> (ke změně portu musí být server zastaven)
<code>max_players [X]</code>	vypíše nebo nastaví maximální počet hráčů na hodnotu <code>X</code>
<code>players</code>	vypíše počet aktuálně připojených hráčů a informace o nich
<code>kick ID</code>	vyhodí ze serveru hráče s id <code>ID</code>
<code>time_per_round [X]</code>	vypíše nebo nastaví délku kola na hodnotu <code>X</code> (v sekundách)
<code>freeze_time [X]</code>	vypíše nebo nastaví délku freeze-time (začátek kola, kdy se hráči nemohou hýbat) na hodnotu <code>X</code> (v sekundách)

Tabulka A.1: Tabulka příkazů pro server

Příloha B

Typy zbraní

- Standardní torpédo
 - Nabíjení - 2,5 s
 - Přímý směr
- Naváděné torpédo
 - Nabíjení - 4,5 s
 - Automatické navádění
- Standardní mina
 - Nabíjení - 1,5 s
 - Po vypuštění zůstává na místě
- Naváděná mina
 - Nabíjení - 4,0 s
 - Po vypuštění zůstává na místě, ale pokud se k ní přiblíží nepřátelská ponorka, začne ji pronásledovat dokud se ponorka dostatečně nevzdálí

Příloha C

Obsah CD

CD obsahuje sedm složek:

- **Documentation:** Dokumentace ke zdrojovým kódům, která byla vygenerována programem *doxygen*.
- **Game:** Spustitelné soubory se hrou.
- **Latex_Code:** Latexové soubory, z nichž byl vygenerován text technické zprávy (bakalářské práce).
- **Literature:** Literatura použitá v technické zprávě, kterou je možné volně distribuovat.
- **Other:** Další soubory k bakalářské práci (např. použité knihovny).
- **Report:** PDF soubor s technickou zprávou.
- **Source_Code:** Zdrojové soubory pro hru.