Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Online hra pro více hráčů Warp Space

Plzeň 2016 Daniel Pavelek

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 23. června 2016

Daniel Pavelek

Poděkování

Tímto bych chtěl poděkoval panu Ing. Tomáši Potužákovi, Ph.D. za jeho trpělivé vedení a cenné rady, díky kterým mohla být tato práce vypracována.

Abstract

The subject of this thesis is to introduce the requirments and problems of computer games programming, to find out the common features and distinctions of several game types, especially MMORPG. Next goal is to describe the tools provided by Java programming language and use them to create the Java MMORPG and test it.

Abstrakt

Cílem této práce je prozkoumat problematiku programování počítačových her, zejména her MMORPG, zjistit jejich klíčové vlastnosti, společné prvky a úskalí. Práce bude zaměřena především na možnosti realizace takové hry pomocí programovacího jazyku Java a bude představeno a otestováno takové řešení.

Obsah

1 Úvod

Úkolem této práce je vytvořit MMORPG (massivelly multiplayer online role playing game) hru v jazyce Java. V rámci tohoto úkolu bude prozkoumána problematika programování počítačových her. Budou uvedeny jednotlivé herní žánry a probrána jejich specifika. Dále budou zkoumány oblasti, kterými se hra jako počítačový program zabývá. Důraz bude kladen především na oblasti herního žánru MMORPG. Bude vysvětlena problematika dotyčných oblastí a dále pak uvedeny možnosti jejich vyřešení.

Dalším bodem práce bude analýza programovacího jazyku Java. Bude zaměřena především na možnosti jeho využití při tvorbě MMORPG her. Budou uvedeny knihovny určené pro implementaci jednotlivých oblastí. Bude probrána vhodnost jejich použití pro daný problém s případnými alternativními řešeními.

Následovat bude popis hry, jejíž snahou bude podle sesbíraných poznatků, demonstrovat takové řešení. Budou vysvětlena její pravidla, specifika a následně také popis její implementace, kde bude vysvětleno naprogramované řešení. V závěru práce pak bude shrnutí dosažených výsledků a jejího testování.

2 Počítačové hry

Než bude rozvedeno hlavní téma této práce, je dobré nejprve uvést základní fakta a nahlédnout do problematiky. Začneme otázkou her samotných, co je vlastně hra jako taková, jak se pozná, čím je charakterizovaná a jakým způsobem se dělí. V dalších kapitolách pak bude rozebráno, jaké z daných herních specifik plynou požadavky na softwarové a hardwarové vybavení.

2.1 Co je hra

To, co je pro všechny hry společné a co je vlastním důvodem jejich existence, popisuje holandský kulturní historik *Johan Huizinga* ve své knize následujícími slovy: *„Hra je dobrovolná činnost, která je vykonávána uvnitř pevně stanovených časových a prostorových hranic, podle dobrovolně přijatých, ale bezpodmínečně závazných pravidel, která má svůj cíl v sobě samé a je doprovázena pocitem napětí a radosti a vědomím „jiného bytí“, než je „všední život“.“* [1]

Lze stručně říci, že její účel je vytvořit prostor pro zábavu. Takový prostor vzniká na základě snahy dosažení určeného cíle za současného dodržování stanovených pravidel.

2.2 Herní žánry

Počítačové hry lze rozdělit do několika skupin. Každá hra obsahuje několik atributů, které ji definují. Zde jsou vybrány tři hlavní. Jsou to *pravidla*, *prostředí* a *ovládání*. Pravidla definují stavový prostor, ve kterém se hráč pohybuje. Určují, co v dané chvíli lze a nelze provést. Prostředí má roli určitých kulis, které hru obohacují a pomáhají se do ní hlouběji ponořit. Ovládání určuje složitost, kterou lze jednotlivé akce provést [2]. Herních žánrů je mnoho a hra samotná může současně spadat do více kategorií, a tak bude uvedeno jen několik hlavních zástupců, jejichž vzájemná odlišnost je patrná. U každého budou uvedeny oblasti, které jsou při programování daného žánru klíčové [3].

2.2.1 FPS

FPS nebo-li *střílečka z pohledu první osoby* z anglického *first person shooter* je hra, kde, jak název napovídá, hráč pozoruje herní prostředí pomyslnýma očima své herní postavy. Za takových podmínek je patrné, že herní prostředí je trojrozměrné.

Typické vlastnosti jsou následující [3]:

- dostatečně frekventované zobrazení 3D scény
- přesné míření a ovládání kamery (hráčovy hlavy)
- věrné vyobrazení těla herní postavy (ruce, zbraň, atd.)
- věrné animace efektů a inteligence herních objektů

2.2.2 Závodní hry

Závodní žánr popisuje takové hry, jejichž hlavním úkolem je řízení vozidla po nějaké trase [3].

Typické vlastnosti jsou následující [3]:

- efektivní zobrazení 3D scény, které, na rozdíl od FPS her, musí počítat s rychlým pohybem hráče. Jednou z technik je například použití 2D „karet“ pro vzdálené, ne tolik pro hru podstatné, objekty jako jsou stromy, kopce a podobné objekty krajiny, které jsou pozorovány jen z jedné strany
- optimalizované datové struktury reprezentující herní prostředí pro efektivní rozhodování umělé inteligence, jako například hledání cesty
- sledování hry z pohledu první nebo třetí osoby a kolize kamery s objekty herního světa, jako například vyřešení problému, kdy hráč projíždí tunelem a kamera by zasahovala do stropu
- dobře vyřešené fyzické vlastnosti pohybu a řízení vozidla

2.2.3 RTS

Pojem RTS lze přeložit jako *strategie reálného času* a pochází z anglického *real-time strategy*. Strategické hry jsou zpravidla zaměřeny na hráčovu schopnost plánovat a taktizovat. Obvyklým cílem je na základě znalostí herní

ekonomiky vytvořit dostatečnou útočnou sílu a na herní ploše porazit počítačového protivníka.

Typické vlastnosti jsou následující [3]:

- herní objekty jsou optimalizované takovým způsobem, aby jich hra byla schopna vykreslit co nejvíce na jednom místě
- hráč obyčejně ovládá hru jediným kliknutím a označováním oblasti pro výběr více jednotek
- hra nabízí velké množství ovládacích prvků GUI jako jsou různá menu, informačními a ovládacími panely

2.2.4 MMOG

MMOG přeloženo jako *online hra pro masivní množství hráčů* z anglického *massively multiplayer online game*. Takový žánr lze popsat jako hru, která umožňuje společné hraní tisícům hráčů současně v *perzistentním* světě. Slovo perzistence je pro tento žánr klíčovou vlastností. Jde o to, že ať se hráč připojí kdykoli, vždy bude jeho herní stav takový, jaký byl při jeho posledním odpojení. Nezáleží, zda-li jde o některý z podkategorií jako například MMORPG (RPG pro více hráčů), MMORTS (strategie pro více hráčů) nebo MMOFPS (střílečka pro více hráčů). Vše, čeho hráč dosáhl v herním světě, mu zůstane nehledě na to, jak dlouho nebyl připojen [4]. Hra zpravidla nabízí hráčům sociální interakci ve formě rozličných hráčských sdruženích, která umožňuje sdílet herní prožitky a stává se nástrojem komplexnější kooperace.

Typické vlastnosti jsou následující [4]:

- dobře vyřešený vzhled a přehledné uživatelské prostředí, obsahující často velké množství ovládacích prvků
- vyřešení latence, tedy doby potřebné pro přenos informace mezi serverem a hráčem
- databáze a správa hráčských dat

3 Programování her

V této kapitole bude popsáno, co vše je nutné promyslet a vědět při programování her. Popis bude zaměřen především na návrh MMORPG her.

3.1 Herní jádro

Pojem *herního jádra* (anglicky *game engine*) vznikl v devadesátých letech 20. století a jeho představitelem byla FPS hra *Doom*. Architektura, kterou byla tato hra programována, zřetelně oddělovala kódy jednotlivých podsystémů [3]. Díky takovému oddělení bylo mnohem snažší kód podsystémů přenést do jiných projektů bez potřeby složitějších úprav. V zásadě takové jádro umožňovalo po několika málo úpravách představovat jinou hru daného žánru a přenositelnost se stala jeho důležitou vlastností. V následujících podkapitolách bude blíže vysvětleno, jakými oblastmi se herní jádro zabývá.

3.2 Síťová komunikace

Přenos dat je klíčový prvek každé hry pro více hráčů. Data mezi jednotlivými počítači a síťovými zařízeními jsou vyměňována na základě domluvených pravidel formou takzvaných protokolů.

3.2.1 Komunikační protokoly

Úkolem protokolů je definovat pravidla, podle kterých komunikující zařízení rozeznají přijatou zprávu a její význam. V rámci sítě Internet je globálně použita sada protokolů *TCP/IP*. Jméno bylo převzato ze dvou základních protokolů *IP* (internet protokol) a *TCP* (transmission control protocol). Pro efektivní komunikaci je třeba správně rozhodnout, který protokol kdy použít [5]. Dále budou rozebrány dva protokoly, které jsou důležité pro praktickou část této práce.

TCP

TCP zajišťuje spolehlivý přenos dat mezi dvěma koncovými zařízeními na Internetu. Přenášená data balí do struktur zvaných *pakety*. Je to *spojovaný* protokol a díky tomu je vhodný všude tam, kde je prioritou úspěšný přenos všech dat. Před začátkem přenosové relace, navazuje a během relace udržuje

a kontroluje, *spojení* s protistranou. Zároveň se stará o správné doručování dat, tedy aby do cílového zařízení dorazila data nezměněná a ve stejném pořadí, ve kterém byla odeslána. To zajišťuje pomocí potvrzovacích paketů, kterými od protistrany získává informace o tom, zda data dorazila a zda jsou v pořádku. V opačném případě je schopen poškozené nebo nedoručené pakety odeslat znovu. Díky této servisní komunikaci je TCP protokol spolehlivý, avšak jeho přenosová rychlost je snížena kvůli většímu objemu dat, která je třeba přenést [5].

UDP

Protokol *UDP* (user datagram protocol) zajišťuje *nespojovaný* přenos dat mezi dvěma koncovými zařízeními na Internetu. Přenášená data balí do struktur zvaných *datagramy*. Nespojovanost znamená, že oproti TCP protokolu neudržuje s protistranou žádné servisní spojení, čili žádným způsobem nezjišťuje, zda a v jakém stavu protistrana data obdržela. Na rozdíl od TCP protokolu dosahuje výrazně vyšší přenosové rychlosti a je to vhodný protokol všude tam, kde občasná nepřesnost nebo ztráta dat není důležitá. Příkladem jsou místa, kde se v reálném čase neustále aktualizují data (například živý přenos videa, pohyb hráče ve hře a tak podobně) [5].

3.2.2 Síťová architektura

Další důležitou otázku návrhu síťové aplikace hraje *centralizace*. Jde o to, jakým způsobem bude vedena komunikace koncových zařízení.

Klient – server

Centralizované řešení, nazývané také *klient – server* znamená, že je zde jedna autorita zvaná *server*, která od připojených zařízení (*klientů*) přijímá požadavky, zpracovává je a odpovídá. Klíčovou výhodou tohoto modelu je schopnost serveru řídit své klienty a provádět důležité výpočty na své straně. Tím lze předejít problémům, kdy by důležité výpočty byly prováděny a zneužívány modifikovanou verzí klientské aplikace. Zároveň se zabraňuje nekonzistenci, pokud klienti potřebují sdílet nějaký společný stav. Slabinou tohoto modelu je fakt, že server je schopen v jednu chvíli obsloužit jen omezený počet požadavků a je tedy možné ho přetížit. Další problém nastává v případě selhání a dočasné nedostupnosti serveru. Tyto nedostatky lze vyřešit zvýšením počtu serverů ve smyslu jejich úplné duplikace nebo rozložení zátěže pro zpracovávání jen určitých požadavků [6].

P2P

Decentralizované řešení popisované zkratkou *P2P* (z anglického *peer to peer* lze přeložit jako topologie *rovný s rovným*). Jde o to, že zde není žádný server a na požadavky jednoho uzlu může odpovídat více jiných. Hlavní výhoda a vlastnost toho modelu je fakt, že nedostupnost nebo zrušení jednoho uzlu nemá zásadní vliv na zbytek systému. Nevýhodou je nesnadná správa celého systému právě kvůli absenci centrální autority. Ke komunikaci v takových sítích se často využívá *multicasting* s UDP pakety, který umožňuje poslání jedné zprávy více příjemcům, aniž by musel posílat každému z nich danou zprávu zvlášť [6].

Multicasting

Multicasting je nástroj umožňující poslat jednu informaci více klientům jedinou zprávou. Jsou k tomu zapotřebí síťová zařízení podporující multicasting a jejich účel je snížit množství dat posílaných po síti. Zamezuje potenciálním cyklům a duplikacím paketů a to tak, že mezi členy multicastové skupiny vytváří stromový komunikační vzor. Používají se pro něj speciální IP adresy, zvané multicastové IP adresy třídy D, jejíž rozsah se pohybuje od 224.0.0.0 do 239.255.255.255 [6].

3.2.3 I/O strategie serveru

Standardní způsob, jakým spolu server a klient komunikují je takový, že každý z nich požádá operační systém o přidělení a otevření komunikačního primitiva, tzv. *soketu*. Skrze ten pak mohou posílat a přijímat data [7]. Problém je v tom, že u TCP spojení musí server každému klientu otevřít jeho vlastní soket, na kterém budou komunikovat a udržovat relaci. U UDP takový problém nenastává, protože relaci udržovat nepotřebuje a serveru pro všechny klienty stačí otevřít jediný soket. Úkol, který je třeba vyřešit, je příjem a čtení dat ze soketu. Není totiž možné určit, kdy data přijdou, a tak je čtení takzvaná *blokující operace*, kdy je vlákno, které chce číst data, uspáno, dokud data nedorazí. To může způsobit problémy, protože server by v takovém případě musel čekat na data od prvního klienta, pak na data od druhého a tak dále. Všichni ostatní musí čekat, než přijdou na řadu a je nutné, aby všichni data posílali, jinak se server zastaví nad klientem, který neodpovídá [8].

Multithreading

Základní myšlenka řešení zmíněného problému je taková, že každému klientu k soketu přidělíme čtecí vlákno, které ho bude obsluhovat. Nebudou tím zasaženi ostatní klienti a server může běžet nezávisle na tom, kdy který klient vysílá. Problém tohoto řešení je ale právě v množství vláken, která pro svůj paralelní běh potřebují dodatečné systémové prostředky. Z těchto výkonnostních důvodů je možné vytvořit jen omezené množství vláken. V případě, kdy by vysílali všichni klienti zároveň v krátkých časových intervalech, narostla by výrazným způsobem spotřeba výkonu procesoru a mohlo by dojít k přetížení serveru, který by nestíhal všechny instrukce zpracovávat [8].

Select

Nedostatky předchozího řešení lze elegantně vyřešit použitím systémového volání *select*, které umožňuje nechat si zaregistrovat a monitorovat vybrané sokety. Dále pak stačí jediné vlákno, které tímto systémovým voláním získá seznam všech soketů, které obsahují data k přečtení a ta zpracovat [8].

3.3 Vizualizace a interakce

Dalším důležitým úkolem hry je *plynulé* zobrazování herního stavu a zpracování uživatelových akcí. Slovo plynulé značí schopnost dostatečně často překreslovat obraz tak, aby se změny herního stavu uživateli jevily co nejvíce přirozené.

3.3.1 FPS

Lidské oko je schopno rozeznat zhruba deset až dvanáct různých, po sobě jdoucích, snímků. Jakmile se jejich frekvence zvýší, začne je oko vnímat jako celek a vzniká iluze pohybu. Tato frekvence tedy znamená počet snímků za vteřinu a značí se anglickou zkratkou FPS neboli *frames per second*. Ve filmovém průmyslu je tato frekvence zpravidla 24 FPS. Nelze říci, že by při takové frekvenci byl obraz plynulý, ale díky efektu zvaném *motion blur* (rozmazání pohybu), způsobeném při natáčení v kamerovém objektivu, tak jsou vyplněny pomyslené mezery mezi snímky a v mozku vniká iluze plynulosti [9].

V oboru počítačových her je ale obraz zpravidla vykreslován ostře a vykreslování při 24 snímcích stále může působit trhaně. Pro dostatečné vykreslování

počítačových her se tedy využívá 60 i 120 FPS, při kterých je obraz vnímán plynuleji a pevněji [9].

3.3.2 Renderování

Když je již znám požadavek na frekvenci vykreslování, bude rozebráno vykreslování samotné. Z angličtiny převzaté slovo *rendering* označuje proces, při kterém je podle určitých pravidel vytvořen obraz z počítačových dat. Zpravidla jde o vytvoření obrazu z dat trojrozměrného modelu, kde pravidla určují osvětlení, materiál objektu, odlesk a stínování. Jde o častější a výpočetně náročnější úlohu, ke které se využívá grafický procesor grafické karty, který je pro operace tohoto typu optimalizován a šetří výkon hlavního procesoru počítače [10].

3.3.3 Grafická API

Informace o tom, co má monitor zobrazit, mu předává grafická karta. Pro využívání funkcí grafické karty se používají takzvaná grafická *API* (application programming interface), které poskytuje programátorům jednotný přístup ke grafickým funkcím nezávisle na typu používané grafické karty [11]. V následujících podkapitolách budou stručně popsána běžně používaná API.

Direct3D

Jde o nízkoúrovňové grafické API společnosti Microsoft a je specifické pro operační systémy Windows. Patří do sady knihoven *DirectX*, které vytvářejí široké rozhraní pro ovládání moderního hardwaru. Slouží k programování aplikací zabývajících se 2D a 3D grafikou. Je navrženo tak, aby odděloval práci CPU a GPU a každý z nich využíval pro operace, ke kterým je lépe navržen [12].

OpenGL

OpenGL (open graphics library) je další světově rozšířený soubor nízkoúrovňových knihoven pro tvorbu programů pracujících s 2D a 3D grafikou. Jeho silnou stránkou je multiplatformnost, přenositelnost a spolehlivost. Společně s mnoha společnostmi zabývajícími se 3D grafikou a výrobou grafického hardwaru definují standard *The OpenGL Architecture Review Board*, jakým toto API pracuje [13]. Díky tomu je ho možné použít pro práci na mnoha operačních systémech pro široké množství grafického hardwaru.

3.4 Herní logika

Nyní již přecházíme k tématu, které utváří samotnou povahu hry, tedy způsobu chování všeho, co hra obsahuje. Zaměřím se na popis obecného návrhu datových struktur, jejich chování a správu.

3.4.1 Herní objekt

Základní entita hry se nazývá *herní objekt*. Jeho úkolem je uchovávat a manipulovat s informacemi o svém stavu ve hře. Při návrhu herních objektů je dobré co nejlépe znát veškeré požadavky na vlastnosti a chování objektů ve finální podobě hry, aby bylo možné co nejlépe určit, jakým způsobem budou tyto vlastnosti v programu reprezentovány a jaké programátorské techniky použít pro co nejefektivnější implementaci. Jde o zásadní úlohu, aby v pozdější fázi vývoje bylo možné, v případě potřeby, nějaké vlastnosti do modelu přidat. Pokud bude návrh dobrý, nepůjde o složitý úkol [14].

Herním objektem může být například nějaký statický objekt bez zvláštní funkce, prezentující určitý prvek herního světa; pohybující se střela mířící na cíl; bonus, který může být hráčem sebrán; objekt představující herní postavu s určitým chováním, zvaný anglickou zkratkou NPC (non-person character) a také objekt prezentující samotného hráče. Každý z těchto objektů má své specifické vlastnosti a chování a také některé vlastnosti a chování naopak společné. Společným prvkem všech objektů může být například jejich pozice v herním světě a také informace o tom, jakým způsobem mají být vykresleny na obrazovku. Naopak strom stojící u cesty nepotřebuje mít definován atribut rychlosti ani funkce pro pohyb. Ty jsou důležité pro pohybující se projektil [14].

3.4.2 Správa objektů

Když jsou již připraveny vhodné struktury pro herní objekty, je dalším úkolem definovat způsob jejich evidence ve hře. Způsob, jakým budou drženy pohromadě a skrze který bude možno nad nimi efektivně provádět různé operace, především *vkládání*, *odebírání*, *procházení* a *přístup podle identifikátoru*. Je třeba určit nebo navrhnout datovou strukturu, která bude nejlépe odpovídat potřebám konkrétní hry.

3.4.3 Komunikace objektů

Důležitou součástí správy objektů je schopnost poskytovat objektům informace o ostatních. Například NPC se rozhoduje, zda je v okolí bezpečno, zda ho některý jiný objekt ohrožuje nebo naopak jiný objekt je možné napadnout. Dalším příkladem je, že hráč vstoupí do hry a potřebuje získat herní stav, tedy objekty ve svém okolí. Je zde dobré si uvědomit, že těmto objektům, které se dotazují, není třeba říkat úplně vše. Vystřelenému projektilu stačí k testování kolize poskytnout jen seznam objektů z jeho bezprostředního okolí. Není třeba ho testovat s objekty na druhé straně mapy, kde je evidentní, že se neprotínají. Proto je při návrhu struktury dobré uvažovat i nad tímto faktem a sdružovat objekty uložené ve strukturách podle jejich *oblastí zájmu*, což výrazným způsobem sníží počet kroků, které pro danou operaci nejsou relevantní.

Dalším bodem komunikace objektů je způsob implementace této komunikace. Jsou dva hlavní způsoby, kterými to lze provést [14].

Přímý přístup

Jde o nejjednodušší způsob předání informace, kdy objekt, který chce vykonat nějakou akci s jiným objektem, potřebuje získat jeho referenci a přímo zavolá požadované funkce. Výhodou tohoto řešení je jeho jednoduchost. Nevýhodou je, že s rostoucí komplexností programu, provázanost událostí v kódu se může stát nepřehlednou [14].

Události

Druhý způsob slouží k větší dekompozici kódu. Každá vzniklá akce se jako zpráva předá správci událostí a ten ji rozešle všem, kdo jsou pro její typ registrováni jako příjemci. Objekt, který akci vyvolá tedy neví, kdo všechno tuto událost dostane, což je svým způsobem cíl tohoto řešení, ale ne vždy to musí být potřeba. V určitých situacích je vhodné, aby zdrojový objekt mohl rozhodnout o příjemcích a jejich případném pořadí a předešel tím například nekonzistenci dat [14].

3.4.4 Hlavní smyčka

Každá hra se skládá ze sady operací, které se neustále znovu a znovu opakují. Slouží k tomu, aby byl pravidelně aktualizován herní stav a byl hráči vykreslován na obrazovku. Toto opakování se nazývá *herní smyčka* nebo také

hlavní smyčka. Smyčka každé hry je ve své podstatě programována na míru konkrétní aplikaci, ale i přes mnoho možností její implementace, lze obecně shrnout několik bodů, které popisují její činnost.

Typické body herní smyčky:

- zpracování uživatelských vstupů (stisky kláves atp.)
- aktualizace herního stavu, volání aktualizačních procedur všech podsystému a jejich objektů
- vykreslení herní scény, přehrání zvukových efektů
- čekání na konec intervalu pro dodržení frekvence

Herní smyčku lze implementovat několika způsoby. Prvním je implementace v *jednom vlákně*. Jde o postup, kdy se neustále dokola opakují všechny zmíněné body. Jde o velice jednoduché řešení. Jeho nedostatky se ale projeví ve chvíli, kdy je potřeba některé úkoly aktualizovat při jiné frekvenci. Problémem může být, že vykreslování čeká na krok aktualizace. Cílem ve hrách bývá snaha dosáhnout co nejvyššího FPS. Slabina, která se také může projevit je, že na nevýkonném počítači může trvat výpočet ve smyčce tak dlouho, že přesáhne dobu intervalu a ihned se spustí znovu. Tím pádem nebude stíhat a zároveň vytíží činnost procesoru na maximální hodnotu.

Problém různých frekvencí lze vyřešit použitím časovače, který na základě znalosti intervalu hlavní smyčky a požadovaného intervalu spouštění, aktualizuje daný podsystém jen tak, jak je potřeba. Problém slabších počítačů lze vyřešit úpravou hlavní smyčky tak, aby byla schopna v případě potřeby některé kroky vynechat.

Druhý způsob implementace je použití více vláken, kdy jsou jednotlivé subsystémy uzavřeny do svých vlastních běhových smyček a výkon je rozložen na více jader procesoru, případně i GPU grafické karty. Slabou stránkou tohoto řešení je přirozeně synchronizace vláken [14].

3.5 Herní data

Každá hra, pokud nemá vše uloženě pevně v kódu, potřebuje zdrojová data pro inicializaci a nastavení svých objektů. Také multimédia jsou již v dnešní době nedílnou součástí her. Herní jádro tak musí být schopno načíst nejrůznější typy dat jako jsou bitmapové obrázky, 3D modely objektů, animace,

zvukové stopy, data pro kolize a herní fyziku, data herního světa a tak dále [3]. Navíc paměť počítače není omezená a zpravidla není možné mít v jednu chvíli načtena všechna, protože by se tam nevešla. Proto je nutné kromě schopnosti data načíst a zpracovat, také správně rozhodovat o tom, která data uchovávat, která přidat a která odebírat.

3.5.1 Souborové API

Na herní jádro jsou pro práci s daty a soubory z pravidla kladeny následující požadavky:

- manipulace s názvem souboru a jeho umístěním v adresáři
- vytvoření, odstranění, otevření, zavření, čtení a zápis souboru
- zjištění obsahu složky
- správa asynchronních požadavků (streamování)

Zároveň je třeba umět pracovat se souborovou cestou, protože jednotlivé operační systémy používají různé oddělovače a také různá mapování diskových jednotek. Operační systémy Windows používají na začátku absolutní cesty název diskové jednotky označované písmenem abecedy a pro síťovou jednotku cestu uvádí dvě zpětná lomítka a název zařízení. Oddělení složek cesty pak používají jedno zpětné lomítko. Absolutní cesta v operačních systémech UNIX začíná dopředným lomítkem a to se také užívá k oddělování jednotlivých složek cesty. Připojené disky jsou pak takzvaně *připojovány* (anglicky *mounted*) a stávají se součástí hlavní cesty [3].

Souborové API herního jádra může k I/O operacím využívat nativní API operačního systému nebo knihovny některého nízkoúrovňového jazyka. Výhodnějším řešením je ale obalení těchto funkcí do vlastních I/O funkcí. To má několik výhod. Umožní to programátorům stejné chování na všech platformách, ačkoli nativní knihovny budou na konkrétních platformách rozdílné. Druhou výhodou je, že široké API systému je zjednodušeno jen na omezené množství funkcí požadovaných herním jádrem [3].

Asynchronní I/O

Díky tomu, že načítání a ukládání je časově náročná operace, je dobré, pokud to jde, provádět tyto operace na pozadí. Pro ten úkol je třeba rozšířit o danou funkcionalitu I/O knihovny jádra. V zásadě jde o předání potřebných úkolů nějakému vláknu. Naštěstí není vždy nutné psát vše od základu,

protože některé operační systémy již tyto asynchronní I/O operace poskytují. Vždy je nezbytné ošetřit synchronizaci vláken, pokud načítající vlákno nedoběhne včas nebo z nějaké důvodu skončí chybou a hlavní na něj musí čekat. Některé asynchronní knihovny umožňují programátorovi určitým způsobem předvídat délku operace a určit *deadline*, aby operace hlavní vlákno nezdržela. Také nabízí možnost určit, co se má stát, pokud by tento deadline nebyl dodržen [3].

3.5.2 Typy dat

Během vývoje hry je důležité myslet na způsob, kterým bude hra po dokončení distribuována. Zdrojový herní kód zpravidla zabírá oproti herním datům jen zlomek místa, a tak je třeba promyslet i otázku, jaká přenosová média lze použít a jakým způsobem ukládat data tak efektivně, aby zabrala co nejméně místa a zároveň byla zachována jejich kvalita.

Data objektů

Jde o data, která popisují herní objekt. Zpravidla se jedná o popis *těla* herního objektu. Tělem je míněn popis bodů a jejich propojení, které tvoří jeho geometrický tvar. Jsou důležitá pro výpočet kolizí s jinými objekty a pro aplikaci textur. Tyto body jsou dvojice či trojice reálných čísel. Existuje mnoho standardních formátů pro efektivní uchovávání těchto dat. Jejich výhodou je přenositelnost i mezi různými modelovacími a grafickými nástroji [14].

Animace

Animací se v kontextu počítačových her myslí předem připravený pohyb objektů. Příkladem může být objekt vlajky vlající na stěžni. Bylo by neefektivní neustále vypočítávat její vlnění ve větru pomocí knihovny pro fyziku, a tak postačuje takové vlnění vypočítat jednou a uložit si v závislosti na čase pozice jednotlivých bodů, a pak jen podle těchto dat pohybovat všemi objekty, kterých se daná animace týká [14].

Data herního prostředí

Dalším typem dat jsou ta, která specifikují instanci objektu hry nebo herního světa. Zpravidla jde o hodnoty, které se skrze I/O knihovnu jádra přiřadí odpovídajícím atributům objektu. Formát těchto dat je specifikován tím, jak je hra naprogramována, ale lze použít některé univerzální formáty jako například XML. To umožňuje přehlednou a snadnou editaci a je vhodné zvláště

při vývoji hry kvůli ladění. Pro nasazení do produkce se ale doporučuje, kvůli velikosti dat a zvláště bezpečnosti (znemožní jednoduše upravit data), použít vlastní binární formát [14].

Grafické soubory

Grafické soubory oproti zdrojovým kódům tvoří převažující část dat hry. Jejich vlastností je, že čím vyšší kvalitu požadujeme, tím více paměti obrázek potřebuje. Je třeba vybrat vhodný formát grafického souboru a potřebnou kvalitu obrazu a zvážit dobu potřebnou k nahrání souboru z disku. Formát hraje roli díky kompresním technikám, kvalita se skládá z *rozlišení* a *barevné hloubky*. Nalezení požadované rovnováhy může významně ovlivnit celkovou velikost hry a rychlost jejího načítání [14].

Zvukové soubory

Zvukové soubory hrají podobnou roli jako grafické a platí pro ně stejná omezení. Maximální kvalitu určuje *vzorkovací frekvence* a počet *zvukových stop* [14]. Téma bude rozebráno v kapitole 3.6.

Video sekvence

Příběh her bývá často doplněn videosekvencemi. Jedná se obohacující prvek, který hráče více vtahuje do děje. Z programového hlediska je nutné promyslet způsob řešení, jak takové videosekvence přehrávat. Nabízí se dvě možnosti. První je přehrání skutečného videa ve hře. Problém je zřejmý a tou je velikost videa, které zásadně ovlivní velikost celé hry. Výhodou je ale, že se takto dá zobrazit prakticky cokoliv a není to výpočetně nijak náročné. Druhé řešení vychází z již popsaných animací. Jde o to, že video je renderováno přímo herním jádrem podle připraveného scénáře. Výhodou je zásadně větší množství ušetřeného místa na médiu a také fakt, že video se svým vzhledem neliší od hry, čímž nedojde k možnému rušivému efektu, kdy by kvalita nahraného videa byla odlišná od kvality zobrazení herní scény herním jádrem [14].

3.5.3 Správa herních dat

Díky tomu, že hra může obsahovat velké množství dat, je zapotřebí nějaký nástroj, který umožňuje jejich efektivní správu, editaci a aktualitu. Základním řešením může být nějaké *sdílené síťové uložiště*. To má však nevýhodu v udržování aktuálních verzí v případě, kdy dva lidé upravují stejný obsah.

Dále je možné použít některý *verzovací systém*. Takové systémy jsou ale často navrženy na práci s dokumenty a pracují s lokálními kopiemi a v případě velkého množství datových souborů nemusí být jejich použití vhodné.

S herními daty jsou často spjatá určitá *metadata* popisující, jakým způsobem mají být data použita a zpracována. Například u dat animace je nutné sdělit frekvenci snímků. Pro efektivní správu souborů s jejich externími metadaty, které jsou navrženy pro potřeby konkrétního jádra, je tedy vhodné vytvořit vlastní systém, zvaný *databáze zdrojů* . Vlastnosti takové databáze by měly být následující [3]:

- schopnost zpracovat nejrůznější typy souborů
- umožňovat vytváření nové soubory, editovat a také je mazat
- přesouvat soubory do jiných umístění a uložišť
- udržovat *referenční integritu* vazeb na sobě závislých souborů (zajistit, že při přesunu nebo smazání souboru nezůstane jiný soubor na něm závislý)
- udržovat konzistentní historii verzí souborů
- umožnit vyhledávání (vývojář například chce zjistit, které textury jsou použity v nějaké úrovni hry a tak podobně)

Dále bude uvedeno několik těchto nástrojů, které lze v praxi použít:

UnrealEd

Jde o databázi zdrojů herního jádra *Unreal Engine 3* firmy *Epic Games*. Je součástí tohoto jádra a díky tomu umožňuje spravované soubory ihned zobrazit v herním prostředí a sledovat a ladit, jakým způsobem se budou chovat ve hře. Jeho silnou stránkou je také *UnrealEd's generic Browser* umožňující velmi efektně vyhledávat a prohlížet jakýkoli soubor databáze. Toto univerzální rozhraní je jeho velkou výhodou oproti jiným herním jádrům, která spravují data jinými, často složitými způsoby [3].

Ogre3D

Ogre3D je grafický engine, který disponuje velmi dobře zpracovaným správcem souborů. Tento správce představuje rozhraní pro nahrání jakéhokoli typu souboru a je navržen tak, aby ho bylo možné rozšiřovat. Díky tomu je možné snadno implementovat novou část , která bude schopna zpracovat požadovaný typ herních dat [3].

XNA

XNA je nástroj pro vývoj her od společnosti Microsoft. Jeho souborový správce je navržen jako plugin do vývojového prostředí *Visual Studia* [3].

3.6 Audio

Další oblastí herního jádra je schopnost přehrání zvuků. Nejde o prvek tak zásadní, že by bez něj nemohla hra fungovat, ale je to důležitý nástroj pro umocnění herního zážitku. Dává hře k dispozici další rozměr. V této části bude vysvětleno jakým způsobem se zvukem herní jádro pracuje.

3.6.1 Vzorkování

Reálný nebo-li analogový zvuk je v prostoru přenášen vibracemi molekul materiálu, kterým prochází. Pro práci se zvukem v počítači je nutné nějakým způsobem umět tento analogový zvuk vyjádřit v digitální podobě. To lze provést jen tak, že analogový signál budeme snímat po částech v nějaké zadané frekvenci. Těmto částem se říká *vzorky*. Každý vzorek reprezentuje hodnotu akustického signálu v daném časovém okamžiku. Kvalita výsledného digitálního signálu závisí na *vzorkovací frekvenci* vyjadřované v jednotce Hertz a na rozsahu hodnot (počtu bitů), kterými je zvuk popisován. Kvalita CD je například vzorkovaná na 44,1 kHz a každý vzorek používá 16 bitů pro zakódování 65 536 možných hodnot [6].

3.6.2 Zvukové soubory

Zvukové soubory existují v mnoha různých formátech [14].

- WAV – formát, který ukládá data bez komprese, čímž se urychluje ukládání a načítání, protože odpadá nutnost dekódování. Díky tomu ale také potřebuje více místa na disku.
- MP3, OGG – oba tyto formáty využívají ztrátovou kompresi, čímž oproti formátu WAV mohou dosáhnout až cca desetinásobné úspory dat bez výrazné ztráty kvality. Naproti tomu ale jsou třeba pro přehrání dekódovat procesorem [14].
- MIDI – formát, který by se svým vztahem k předchozím dal přirovnat ke vztahu rastrové a vektorové grafiky. Neobsahuje hudební stopu, ale jen informace o notách, tempu a nástrojích. Výsledný zvuk závisí

na implementaci v koncovém zařízení. Výhoda tohoto formátu je jeho zanedbatelná velikost oproti předchozím.

3.6.3 Zvuková karta

Klasický úkol ve hrách, je přehrávat více zvuků současně. Například nějakou hudbu na pozadí a zvukové efekty, doplňující nastalé události. Aby bylo možné zvuková data přenést formou zvuků z reproduktorů, je třeba je předat zvukové kartě, která provede jejich převedení na elektrické signály. Předávání těchto dat je třeba velmi přesně načasovat. V případě, že budou předána brzy, přemažou data, která jsou připravena k přehrání před nimi. Pokud pozdě, karta začne znovu přehrávat data, která ji v paměti zůstala po předchozím zvuku [14]. Tento problém bývá řešen v různých zvukových API. Příkladem mohou být *DirectSound* od společnosti Microsoft, *FMod*, *WWise* nebo *Miles Audio* [14].

4 Programovací jazyk Java

Java je vysoko-úrovňový, objektově orientovaný, multiplatformní programovací jazyk. Podle *Tiobe indexu* jde o nejpopulárnější programovací jazyk [15]. Multiplatformnost nebo-li přenositelnost znamená, že program psaný v Javě bude pracovat stejně na jakékoli kombinaci hardwaru a operačního systému, na kterém lze spustit *java virtual machine* (JVM) tohoto jazyka. Je to umožněno tím, že Java svůj kód nepřekládá do sady instrukcí procesoru, ale do takzvaného *byte kódu*, který se dále zpracovává na JVM konkrétního zařízení [16]. Pro běh na rozličných zařízeních bylo vytvořeno několik takzvaných *platforem*, kde každá je realizována pro specifický hardware a účel.

Základní platformy a jejich účel [17]:

- *Java ME* – (micro edition) ve své době určené pro aplikace provozované na mobilních zařízeních (mobilní telefony, PDA, televize, a podobně)
- *Java SE* – (standard edition) aplikace provozované na stolních počítačích
- *Java EE* – (enterprise edition) aplikace pro podnikové a rozsáhlé informační systémy

Dále bude probírána platforma Java SE a možnosti jejího API pro programování MMORPG her.

4.1 Síťové operace

Pro síťová připojení nabízí java *blokující* a *neblokující* API pro TCP i UDP spojení.

4.1.1 Blokující

Blokující síťové operace jsou Javou poskytovány v balíku `java.net`. Jde o nízkoúrovňové síťové API umožňující práci s *IP adresami*, práci se sokety a síťovými rozhraními. K přenosu a zpracování dat využívá `datové proudy` (streamy) z knihovny `java.io`. Pro TCP spojení slouží třída `ServerSocket` a `Socket`. Skrze `Socket` komunikuje klient i server. Serverová část aplikace dále používá `ServerSocket` k naslouchání novým připojením. Pro UDP spojení slouží třída `DatagramSocket`, který je opět použit na straně serveru i klienta [18].

4.1.2 Neblokující

Toto řešení je obsaženo v knihovně `java.nio` a `java.nio.channels`. Jde o alternativní IO API ke knihovnám `java.io` a možňuje provádět IO operace pomocí rozdílných technik. Základním rozdíl je použití *kanálů a bufferů* namísto datových proudů. Druhým je možnost použití *neblokujících IO operací* a *selektorů*. Selektor je prostředek ke sledování stavu kanálů, pokud je použit neblokující režim. Má za úkol vláknu sdělit, zda je daná IO operace připravena. Pro TCP komunikaci se používá třída `SocketChannel`. Na straně serveru se pak k naslouchání nových připojení používá třída `ServerSocketChannel`. Pro UDP pak třída `DatagramChannel` [19].

4.2 Uživatelské prostředí a kreslení

Pro tvorbu grafického prostředí aplikací a kreslení, nabízí Java základní knihovny.

4.2.1 AWT

Grafická knihovna AWT *(Abstract Windows Toolkit)* vznikla jako univerzální rozhraní mezi operačním systémem a aplikací, které tak umožňuje na všech platformách využívat stejné grafické prostředky. Využívá nativní okenní systém operačního systému, díky čemuž aplikace vypadala na každém systému jinak . Zároveň obsahuje propracovaný událostní systém, pro zpracování signálů vstupních zařízení, jako jsou myš a klávesnice [20].

4.2.2 Swing

Knihovny Swing jsou postaveny nad AWT. Rozdíl je ten, že grafické komponenty nejsou závislé na operačním systému, ale utvářeny pouze Javou, díky čemuž vypadá aplikace na všech platformách téměř stejně [20].

4.2.3 Java2D

Pro kreslení 2D grafiky, poskytuje Java vlastní API *Java2D*. Jde o vysokoúrovňového rozhraní umožňující pokročilou práci s grafikou, obrázky, psaním textu, kreslením a geometrií [21]. Oproti nízkoúrovňovým API nedosahuje takového výkonu, ale pro méně náročné úlohy je dostačující a jednoduché.

Vykreslování grafického obsahu na obrazovku je zajištěno pomocí knihovny `java.awt.Graphics` a jejím rozšířením `java.awt.Graphics2D`. Umožňují

vykreslení libovolných geometrických tvarů, jejich okrajů a výplně a provádět s nimi různé transformace. Podporují práci s textem a tiskem a také práci s obrázky.

4.2.4 Java3D

Java3D API umožňuj tvorbu aplikací s trojrozměrnýnou grafikou. Poskytuje vysokoúrovňové možnosti pro tvorbu a manipulaci s 3D geometrickými objekty. Není součástí standardního API, a tak je třeba o k aplikaci přidat explicitně [22].

4.3 Správa objektů

Pro ukládání a práci s více objekty v paměti, nabízí Java v základu celou sadu generických *kolekcí*. Kolekce je rozhraní `java.util.Collection` reprezentující skupinu objektů, nazývané elementy. Implementace kolekcí se liší ve svých vlastnostech, jakou je například *duplikace*, *řazení* a také *složitost* v přístupu a práci s elementy. Nyní bude uvedeno základní dělení kolekcí [23]:

- `Set` – kolekce s unikátními prvky.
- `List` – může obsahovat duplikované elementy. Lze ji nazvat řazenou, protože programátor může určit pozici, na které bude prvek umístěn.
- `Queue` – kolekce reprezentující frontu. Elementy přidává na konec a odebírá zepředu.
- `Deque` – (double ended queue) je kolekce tvořící z elementů frontu a umožňuje přístup na její začátek i konec.
- `Map` – s elementy sdružuje unikátní klíč, skrze který k nim umožňuje přistoupit.
- `SortedSet` – jde o `Set`, který udržuje elementy v pořádí.
- `SortedMap` – jde o kolekci `Map`, která své prvky řadí podle klíče.

4.4 I/O

Se soubory a souborovým systémem umožňuje Java pracovat skrze své souborové API v balíčku `java.io`. Základní třídou je třída `File`, představující

soubor nebo složku umístěnou v rámci cesty souborového systému [24]. Jádrem tohoto API jsou *datové proudy*, reprezentující prostředek pro tok dat ze zdroje k cíli. Dále jsou schopny s daty manipulovat a transformovat podle svých implementovaných vlastností [25]. Datové proudy se dělí na [26]:

- *Bajtové* – IO s binárními daty
- *Znakové* – IO se znakovými daty, podle zadané znakové sady
- *Bufferované* – optimalizované IO snížením počtu volání nativního API JVM
- *Formátované* – umožňuje odesílat text v zadaném formátu a také podle zadaného formátu text číst
- *Standardní* – IO pro práci s konzolí
- *Datové* – binární IO primitivních datových typů a řetězců
- *Objektové* – IO se samotnými objekty

4.5 Audio

Pro práci se zvukem nabízí Java *Sound API* v balíčku `javax.sound.sampled`. Knihovna Sound dokáže přehrávat zvukové formáty s 8 nebo 16 bitovými vzorky se vzorkovací frekvencí od 8000 do 48000 Hz. Nabízí podporu čtení tří samplovaných zvukových formátů souborů: *AIFF*, *AU* a *WAV* [8].

Zvukový soubor lze nahrát pomocí třídy `AudioSystem`. Tato třída zajišťuje několik metod `getAudioStream()` pro otevření zvukového souboru a vrací objekt typu `AudioInputStream`. Dále je zde rozhraní `Line`, které zajišťuje odesílání audia do zvukového systému a jeho příjmu z téhož zdroje. Objekt typu `Line` se získá pomocí metody `getLine()` třídy `AudioSystem`. Dále je zde rozhraní `Clip` implementující `Line`. Objekt tohoto rozhraní nahrává vzorky do paměti `AudioInputStream` a automaticky je předává do zvukového systému. Následuje ukázka kódu pro přehrání zvuku: [8]

```
File file = new File("sound.wav");
AudioInputStream stream = AudioSystem.getAudioInputStream(file);
AudioFormat format = stream.getFormat();
DataLine.Info info = new DataLine.Info(Clip.class, format);
Clip clip = (Clip)AudioSystem.getLine(info);
clip.open(stream);
clip.start();
```

5 Warp Space

V této kapitole budou rozebrány požadavky a specifikace hry a také způsob jakým budou řešeny jednotlivé oblasti.

5.1 Popis hry

WarpSpace je 2D, MMORPG, sci–fi hra. Její prostředí je zasazeno do pomyslného vesmíru, ve kterém se hráč pohybuje ve vesmírné lodi. Kolem se nacházejí různé herní objekty, jako jsou předměty ke sbírání, které hráči zvýší některé atributy, skokové brány, které hráče přemístí na určené cílové souřadnice a počítačem řízení protivníci, kteří na základě vzdálenosti a nastavené míry agrese, se pokoušejí hráče napadat. Každý objekt schopný útoku, je vybaven různými zbraněmi o určitém počtu nábojů a sám obsahuje atributy, které představují jeho zdraví. V případě zničení tohoto objektu bude na jeho souřadnicích vytvořen objekt exploze a zničený objekt bude odstraněn ze hry.

5.2 Síťová komunikace

Pro komunikaci mezi klientem a serverem budou použity přenosové protokoly TCP a UDP. TCP protokol bude použit pro udržování spojení, důležitou a servisní komunikaci. UDP protokol bude použit pro zasílání krátkých, ale častých zpráv, jako je například pohyb objektu nebo střelba.

Pro implementaci síťové komunikaci budou použity prvky knihoven `java.nio` díky možnosti neblokujících operací. Díky tomu bude na straně serveru stačit jedno vlákno pro vyřizování všech TCP požadavků. Pro UDP komunikaci budou tyto knihovny použity také z důvodu používaných bufferů. Pro buffery bude vytvořena recyklační fronta, čímž bude ušetřeno jejich vytváření a odstraňování z paměti.

5.3 Herní jádro

Zde bude popsán způsob, kterým bude ve hře definován, spravován a obnovován herní stav.

5.3.1 Herní objekt

Herní objekt bude ve hře složen ze dvou částí. Své *herní logiky* a *herního těla*. Tyto části budou od sebe v odděleny a skládány podle konfiguračních souborů. Díky tomu bude možné vytvářet objekty se stejným logickým kódem a různými těly a naopak pro stejné tělo bude možné definovat objekty s různou logikou.

Těla objektů

Herním tělem není myšlena jen grafická reprezentace objektu, ale především její rozměr a dodatečné atributy. Dodává svému hernímu objektu informace důležité pro zjištění kolize s jinými objekty, poskytovat data ke správném zobrazení a také určovat souřadnice pro vytváření nových objektů.

Logika

Logický kód herních objektů bude rozdělen podle svých společných vlastností do hierarchické struktury dědění. To znamená, že základní objekt bude obsahovat vlastnosti společné pro všechny herní objekty a jeho potomci ho pak budou rozšiřovat o vlastnosti specifické pro konkrétní třídu objektů.

5.3.2 Správa objektů

Pro práci s objekty bude vytvořen správce, který odliší, které objekty jsou aktivní ve hře, a které ne. Aktivní objekty bude uchovávat pomocí datové struktury, která co nejlépe umožní jejich aktualizace, přidávání, mazání a vyhledávání. Správce bude prostředníkem ostatním komponentám, přistupujícím k objektům, jako například vyřizování událostí ze serveru nebo vykreslování. Neaktivní objekty budou uloženy ve *fondu* (anglicky *pool*). Nově vytvářené objekty budou přejímány z tohoto fondu a naopak odstraňované objekty do něj zpět vraceny. Díky takové recyklaci se zamezí přílišnému vytváření a rušení objektů v paměti.

5.3.3 Kolize a geometrie

Při zadání cílových souřadnic potřebuje herní objekt v každém kroku aktualizace svého stavu vypočítat vzdálenost, o kterou se směrem k těmto souřadnicím má posunout a úhel o který se má případně otočit, aby tímto směrem směřoval. Objekt také bude potřebovat zjistit, jaké objekty se nacházejí v jeho okolí a s jakými z nich se překrývá.

Pro tyto potřeby bude nutné naprogramovat geometrickou knihovnu schopnou provést tyto matematické výpočty.

5.3.4 IO

Zdrojové soubory herních objektů a jejich těl budou vytvořeny, odladěny a uloženy *Editorem* herních dat. Kód herních objektů i těl bude obsahovat metody pro uložení a načtených atributů prostřednictvím některé z kolekcí Javy.

V případě přenosu informací o herním objektu po síti, není třeba posílat všechny atributy ani jeho tělo, protože klient tyto informace zná. Důležitý je jen aktuální *stav*, a tak budou herní objekty navíc obsahovat metody pro uložení a načtení herního stavu.

5.4 Editor

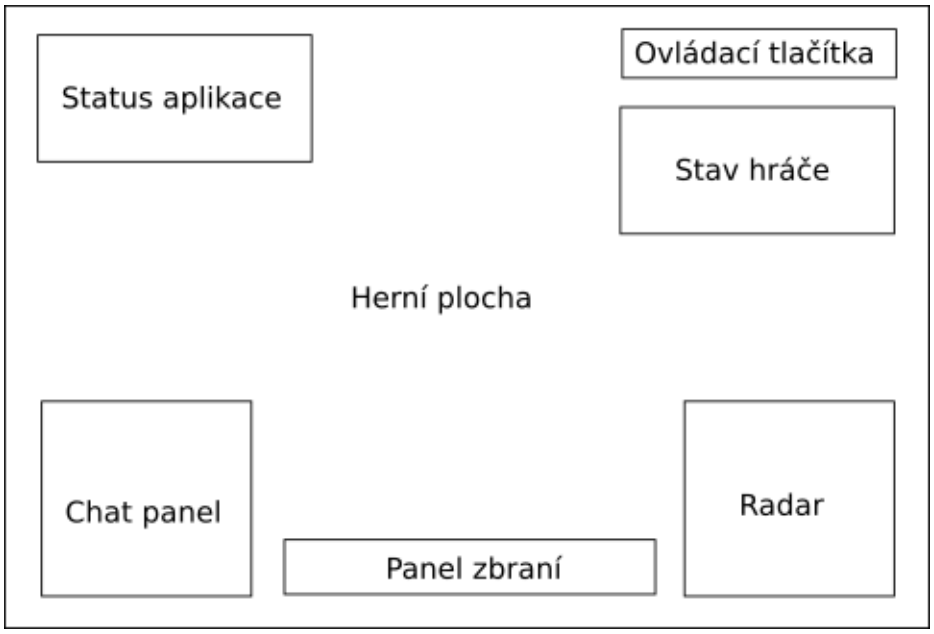
Editor herních dat bude nástroj, který umožní vytvářet herní těla, herní objekty a jejich vzájemné provázání. Bude obsahovat uživatelské rozhraní a vykreslování ladící herní scény. Na tuto scénu bude možné objekty umístit a ladit jejich chování za běhu. Primárním účelem bude správa těchto dat, vedlejším úkolem bude odladit nedostatky v kódu. Díky tomu bude později možné přenést většinu herního kódu do klientské i serverové části aplikace, které na něj navážou síťové operace a herní grafické rozhraní.

5.5 Klient

Klientská aplikace bude uživateli skrze své uživatelské prostředí přístup k serveru. Umožní vytvoření účtu, přihlašování a odhlašování, zobrazení a ovládání hry. Bude se řídit zprávami ze serveru a podle nich provádět požadované akce. Například upravovat herní stav, zobrazovat upozornění a tak podobně. Nebude muset složitě řešit heuristiku při hledání okolí objektů, protože od serveru bude dostávat vždy jen své nejbližší okolí čítající pouhé desítky objektů.

Pro grafické rozhraní a kreslení budou využity knihovny Javy `javax.swing` a `java.awt`, protože jde standardní řešení, které bude pro dané potřeby dostatečné. Bude využit celoobrazovkový režim s možností přepnutí do okenního režimu.

Herní obrazovka se bude skládat z informačních panelů o přípojení, chat panelu, mapy, stavu zbraní a stavu hráče. Návrh rozložení je na obrázku 5.1.



Obrázek 5.1: Dědění vlastností objektů

5.6 Server

Úkolem serveru bude neustále aktualizovat herní stav a vybírat události, které je třeba oznámit konkrétním hráčům. Bude udržovat seznam připojených klientů, jejich stav připojení (připojen, přihlášen, ve hře) a přepínání mezi nimi. Bude muset efektivně vyřešit způsob předávání informací o okolí objektů pro řešení kolizí a událostí.

6 Popis řešení

V této kapitole bude popsána implementace MMORPG hry v jazyce Java. Aplikace je rozdělena do tří částí. *Editoru* herních dat, *klienta* a *serveru*. Nejdříve bude popsán kód herního jádra, který je stejný ve všech třech případech a dále pak rozebrána specifika a řešené problémy každé části zvlášť.

6.1 Herní jádro a logika

V této části budou rozebrány třídy představující herní stav, práce s nimi a jejich vlastnosti.

6.1.1 Herní objekty

Základní prvek této hry je třída `GameObject`. Jde o abstraktní třídu obsahující atributy a základní metody pro každý objekt, který se vyskytuje ve hře.

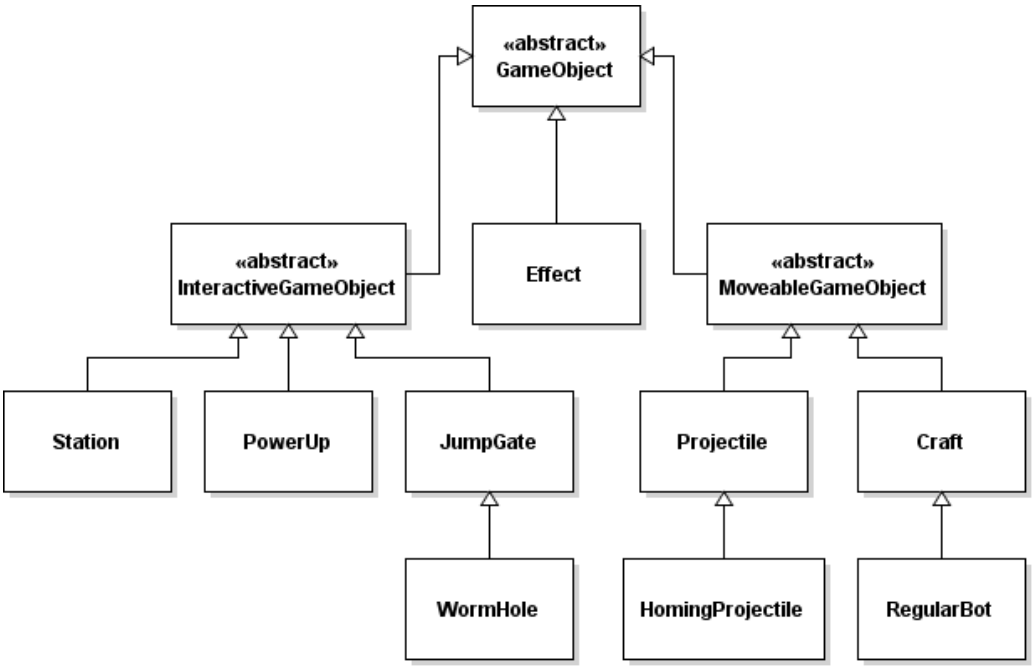
Hlavní atributy třídy `GameObect`:

- *pozice* určující umístění objektu ve světě
- *identifikátor* používaný pro synchronizaci objektů mezi serverem a klientem
- *systémový název* pro identifikaci typu objektu
- *status* označující v jakém stádiu svého života se objekt nachází (*bežící*, *zastavený*, *zničený*)
- *herní tělo*, které určuje grafickou reprezentaci objektu

Důležitou metodou je veřejná metoda `update()`, ve které objekt definuje své chování v jednom kroku herního cyklu. Pokud potomek chce své chování změnit, musí tuto metodu překrýt. Další metody pak slouží pro práci s herním stavem objektu.

Herní objekt dále rozšiřují další abstraktní třídy `InteractiveGameObject` a `MoveableGameObject`. První tvoří základ pro objekty, kde je nějaký způsob interakce. Například skok branou nebo sebrání předmětu. Druhá třída potom přidává atribut `cílové pozice` a metody pro pohyb a je základem pro

všechny objekty, které se pohybují. Za zmínku stojí její potomci `Craft`, který představuje objekt hráče a `Projectile` tvořící základ pro objekty zbraní ve hře. Důležitou třídou je také `RegularBot`, která je potomkem třídy `Craft` a rozšiřuje její logiku o automatické rozhodování. Jejími instancemi jsou pak NPC. Způsob dědění objektů je znázorněn na obrázku 6.1.



Obrázek 6.1: Dědění vlastností objektů

6.1.2 Herní těla

To jakým způsobem je objekt vykreslován a jakou má velikost je určeno v instanci herního těla, které je součástí každého herního objektu. Jde o třídu `GameObjectBody` a jejího potomka `CraftBody`.

Atributy třídy `GameObjectBody`:

- *název* pro jednoznačnou identifikaci
- *centrální bod*, označující střed obrázku, kolem kterého je možné ho otáčet
- *pole obrázků* je použito pro případnou animaci
- *meta informace* jako je výška, šířka a data pro případné animování

`CraftBody` pak první třídu rozšiřuje o umístění zbraní a motorů. Tyto body jsou pak ve hře použity pro počáteční umístění projektilu nebo vytvoření efektu stopy za lodí, když se pohybuje.

6.1.3 Vybavení

Součástí je také vybavení hráčovi lodi. Jde o věci, které v nějakém směru rozšiřují jeho základní vlastnosti. Ve hře je jako jediný zástupce vybavení třída `WeaponModul`, která reprezentuje zbraň. Jejím úkolem je umožnit střelbu, což je vytváření objektů třídy `Projectile` s určitou silou a v určité frekvenci. Zde jsou uvedeny hlavní atributy:

- *počet nábojů*
- *aktivace* - logická hodnota sdělující, zda je zbraň aktivovaná
- *poškození* - jaké poškození vznikne zasaženému objektu
- *počet projektilů v dávce*
- *časový rozestup projektilů*
- *časový rozestup dávek*

Dále obsahuje přístupové metody pro aktivaci zbraně, počtu nábojů a metodu `update()` pro obnovení svého stavu.

6.1.4 Serializace a IO

Ukládání a načítání objektů je zajišťováno třídou `SourceObjectIO`. Tato třída pracuje s objekty implementujícími rozhraní `SourceObject`, ve kterém jsou zadány metody `load()` a `save()`. Tyto metody pracují s objektem `java.util.Map`, kde je klíčem a hodnotou řetězec. Skrze tuto mapu jsou pak předávány hodnoty atributů pro uložení do souboru nebo načtení do zdrojového objektu. Třídy, které implementují rozhraní `SourceObject` jsou `GameObject`, `GameObjectBody` a `WeaponModul`. Pro fyzické ukládání a načítání dat používá třída `SourceObjectIO` metody třídy `IOMethods`. Zdrojové soubory pro tříd jsou ukládány ve složce `data/gameData` podle tabulky 6.1

Třída	*Podsložka*	*Přípona*
`GameObject`	`objects`	`wof`
`GameBody`	`bodies`	`wbf`
`WeaponModul`	`equipment`	`wef`

Tabulka 6.1: Složky dat objektů

6.1.5 Správa těl

Třída `GameBodyManager` byla vytvořena, aby od každého herního těla zajistila existenci pouze jedné instance a umožnila její sdílení herním objektům. Třída implementuje návrhový vzor *singleton* a po své prvotní inicializaci načte veškerá data herních těl a dále je pak poskytuje skrze metodu `getBodyFor()`, na základě systémového názvu těla.

6.1.6 Správa objektů

Herní objekty jsou sdružovány třídou `GameObjectManager`. Tato třída zajišťuje přístup k jednotlivým instancím a obnovuje jejich stav. Dalším jejím úkolem je zajistit *recyklaci* objektů. Aby se zamezilo neustálému vytváření a odstraňování objektů z paměti, odděluje od sebe objekty *aktivní* a *neaktivní*. Během inicializace při startu aplikace načte určité množství neaktivních objektů, které pak při požadavku na vytvoření objektu, jen objekt označí za aktivní a přesune k aktivním. Neaktivní objekty jsou spravovány třídou `GameObjectPool` a aktivní uloženy v přepravce `GoListsCrate`.

Třída `GameObjectPool` implementuje návrhový vzor *object pool*, jehož úkolem je již zmíněná recyklace objektů. Od každého typu herního objektu uchovává určitý počet neaktivních instancí, které na požádání předává třídě `GameObjectManager`.

6.1.7 Struktury

Herní jádro obsahuje tři vlastní struktury pro ukládání objektů. Každá z nich implementuje iterátor, umožňující snadné procházení prvků v ucelené řadě.

DoubleBuffer

Třídu `DoubleBuffer` lze zjednodušeně popsat jako strukturu obsahující dvě pole. Její životní cyklus se skládá ze dvou, opakujících se fází. Během první jsou do prvního přidávány objekty. Ve chvíli, kdy chce někdo data číst, je první pole poskytnuto ke čtení a druhé vystřídá jeho místo a umožňuje okamžitě znovu zapisovat. Je to vhodný prostředek například pro předávání požadavků mezi vlákny.

GoListCrate

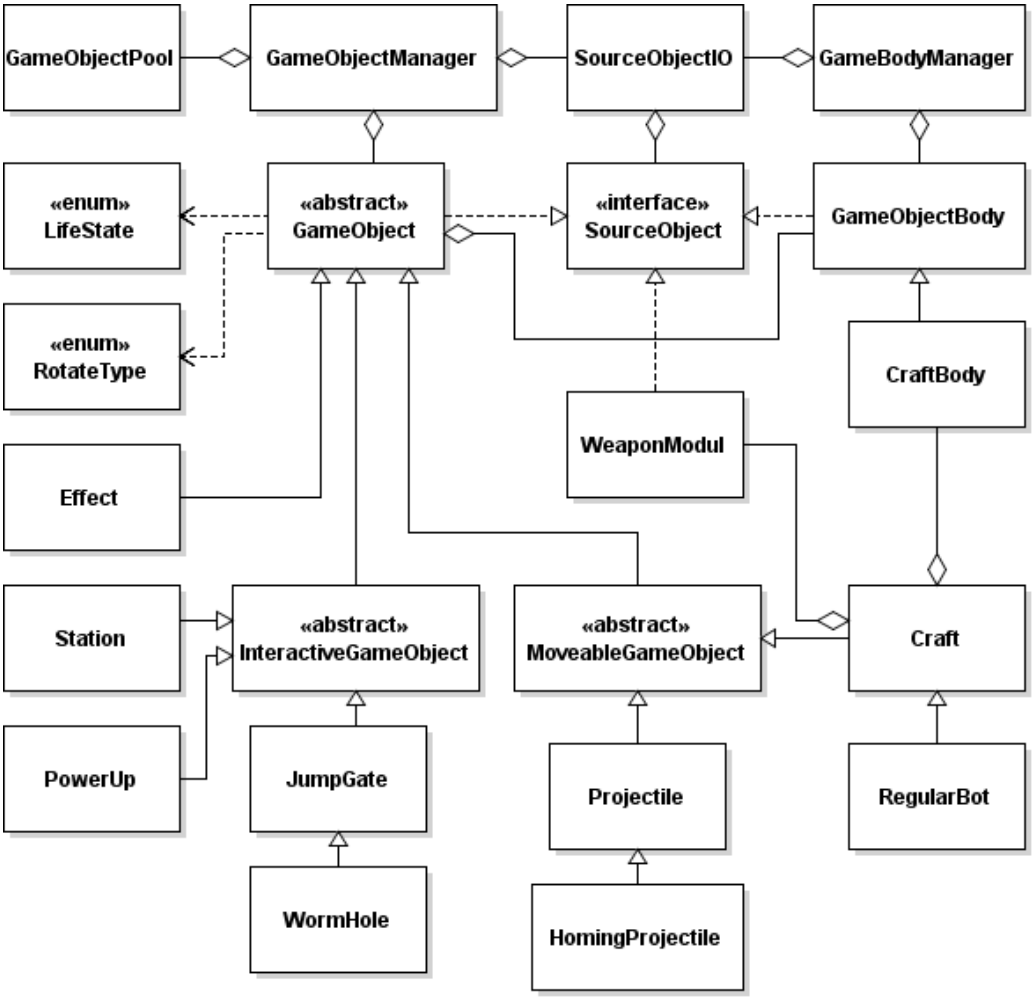
Třída `GoListCrate` slouží k uchování herních objektů ve skupinách podle jejich typu. Usnadňuje tak vyhledání objektu (je znán typ, prohledá se jen jeho seznam) a práci s jen určitým typem objektů. Příkladem může být vyhledání všech statických objektů a jejich vykreslení na pozadí herní plochy.

IndexedArrayList

Třída `IndexedArrayList` slouží k okamžitému vkládání a odebírání objektů. Jde o kombinaci obyčejného pole a fronty. V poli jsou ukládány objekty a fronta uchovává seznam volných indexů. Indexy jsou při vkládání odebírány ze začátků fronty a při odstraňování objektů na její začátek vraceny. Třída je použita například pro ukládání klientů na serveru.

6.1.8 Vztahy tříd

Vztahy základních tříd znázorňuje obrázek 6.2.



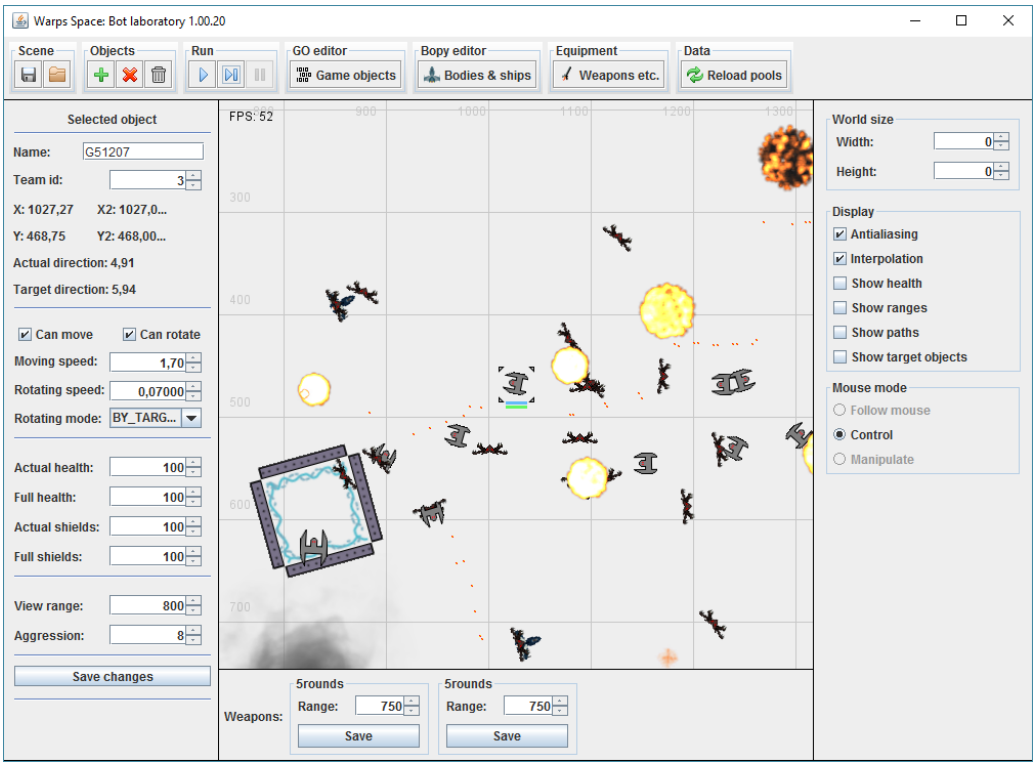
Obrázek 6.2: Vztahy tříd herního jádra

6.2 Editor

Editor herních dat je programován jako okenní aplikace. Skládá se z hlavního okna (třída `MainWindow`) a tří editorů. Hlavní okno obsahuje mnoho ovládacích prvků pro práci s objekty a skládá se z několika panelů. Hlavní panel představuje třída `DisplayPanel` a slouží k vykreslování herní scény. Mimo jiné zajišťuje zpracování uživatelských vstupů. Myší lze označit libovolný objekt a přesunovat ho. Druhým důležitým panel je představován třídou `DataPanel`, který nabízí online zobrazení atributů označeného ob-

jektu. Tyto atributy zároveň umožňuje za běhu měnit a ukládat. Náhled okna editoru je na obrázku 6.3.

Dále obsahuje nástroje pro tvorbu a editaci herních objektů, těl a zbraní. Editor herních objektů je popsán třídou `GameObjectEditor`. Jeho hlavním úkolem je k editovanému hernímu objektu přiřadit existující herní tělo a určit třídu implementace některé z tříd herního objektu. Editor herních těl je ve třídě `BodyEditor`. Jde o interaktivní editor, umožňující vybrat obrázek a přiřadit k němu bod otáčení, umístění zbraní a motorů. Zároveň nabízí práci se sprity pro vytvoření animace. Třída `WeaponEditor` pak představuje editor zbraní. Především slouží k odladění rychlosti a frekvence střelby pro kterou nabízí živý náhled.



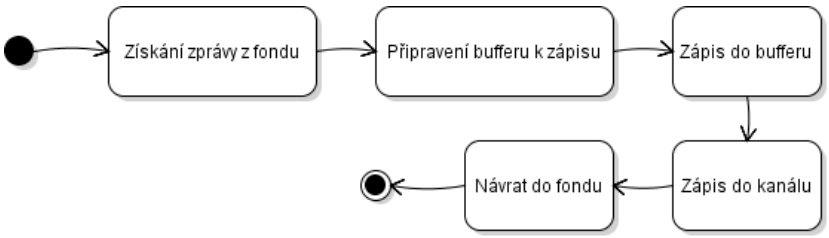
Obrázek 6.3: Náhled hlavního okna editoru

6.3 Protokol

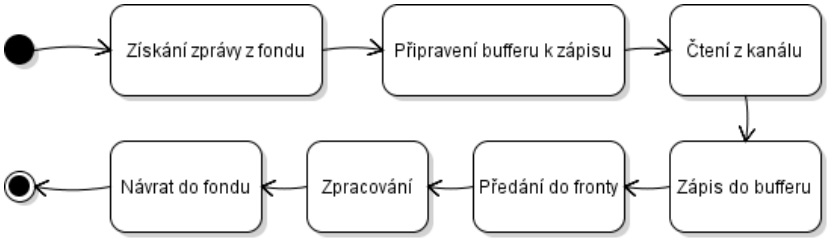
Nyní bude probrána společná síťová část serverové i klientské aplikace. Základním objektem je třída s názvem `MessageProcessor`. Jde o návrhový vzor `singleton` a obsahuje metody pro vyřizování veškeré komunikace. Dru-

hým základním objektem je třída `Message`, která slouží jako prostředek pro výměnu dat. `MessageProcessor` obsahuje frontu těchto zpráv, které při zavolání metody `processMessages()` všechny zpracuje. Do fronty se zprávy dostávají metodou `addMessage()` prostřednictvím vláken, která obsluhují čtení z TCP a UDP spojení.

Třída `Message` obsahuje objekt `ByteBuffer` a poskytuje další metody pro práci s ním. Příkladem může být zápis a čtení řetězců. V serveru i klientu jsou tyto zprávy zachovány ve fondu a jejich životní cyklus je popsán obrázky 6.4 a 6.5.



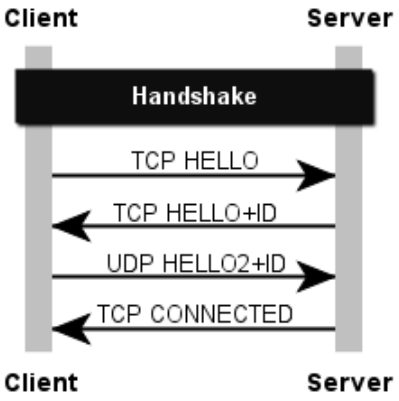
Obrázek 6.4: Životní cyklus zprávy při zápisu



Obrázek 6.5: Životní cyklus zprávy při čtení

6.3.1 Herní protokol

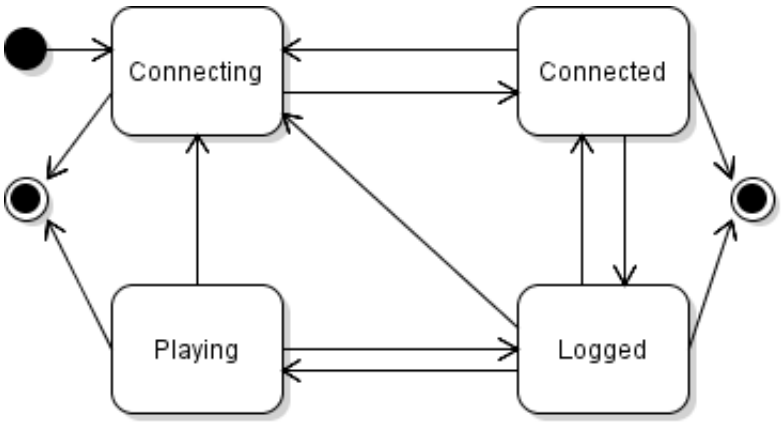
Zprávy posílané mezi serverem a klienty jsou z důvodu úspory velikosti, v binárním formátu. První 4 byty určují délku zprávy, další 2 potom typ zprávy. Typy jsou definovány třídou `MessageTypes` a dodatečné atributy pak ve třídě `MessageParams`. Další pokračování zprávy určuje třída `MessageProcessor`. Ukázku komunikace popisuje obrázek 6.6.



Obrázek 6.6: Schéma navázání spojení

6.4 Klient

Hlavní třídou klientské aplikace je třída `WarpSpaceClientMain`, která po svém spuštění zinicializuje ostatní komponenty (`GameObjectManager`, `MessageProcessor`). Klíčovými třídami této aplikace jsou `MainLoop` a enum `AppState`. Aplikace během svého životního cyklu projde několika stavy definovanými v `AppState`. Třída `MainLoop` je určena k tomu, aby podle stavu aplikace, prováděla příslušné operace. Stavy a přechody mezi nim jsou znázorněny na obrázku 6.7.

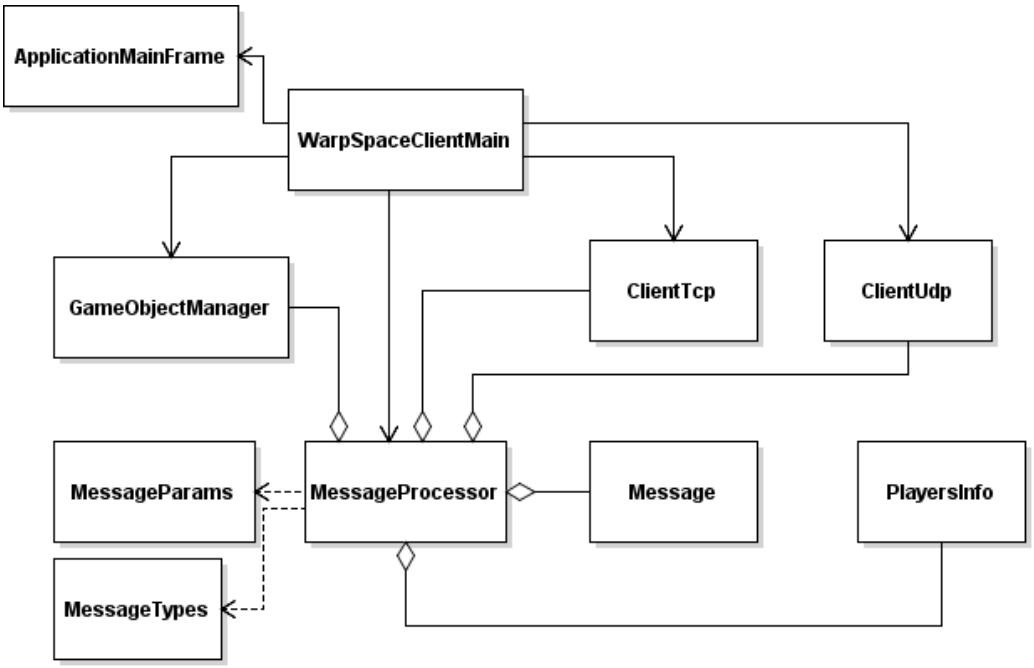


Obrázek 6.7: Stavový automat klientské aplikace

Grafické uživatelské rozhraní se skládá ze třídy `ApplicationMainFrame` a několika dalších, reprezentujících různé panely (například pro přihlášení, registraci, vykreslování herní scény). `ApplicationMainFrame` poskytuje metody pro jejich přepínání.

6.4.1 Vztahy tříd

Vztahy základních tříd znázorňuje obrázek 6.8.



Obrázek 6.8: Vztahy tříd na straně klienta

6.5 Server

Hlavní třídou serverové aplikace je třída `WarpSpaceServerMain`, která stejně jako klientská, zodpovídá za inicializaci herního jádra a síťových připojení. Dále používá třídu `PlayersDatabase`, která obsahuje informace o všech hráčích. Poskytuje metodu `addNewPlayer()` pro jejich přidávání a metodu `getPlayerById()` pro jejich vyhledávání. Tato třída je postavena nad objektem `java.util.Map`, ve kterém uchovává instance hráčů po klíčem jejich číselného identifikátoru. Objekt hráče je reprezentován třídou `Player`.

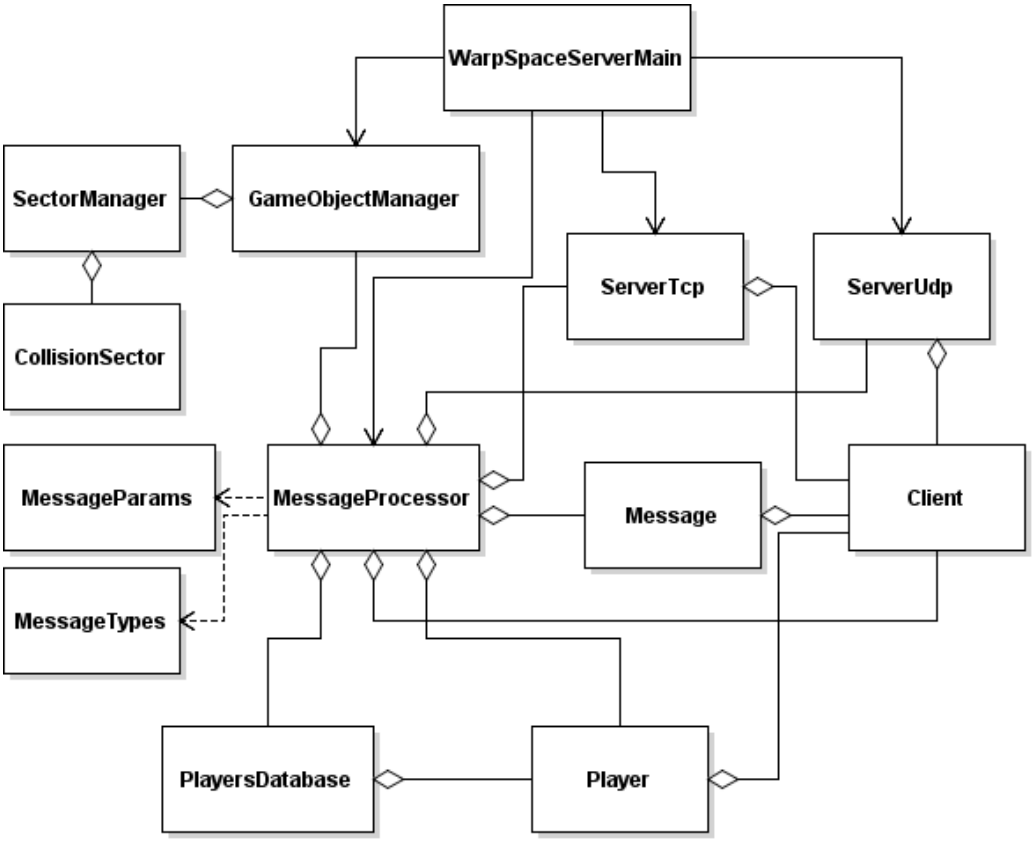
Server dále pracuje s třídou `Client`, která představuje objekt připojeného klienta. Jako atributy obsahuje instance TCP a UDP soketů pro komunikaci a také instanci třídy `Player`, která je mu přidělena, pokud je přihlášen. Seznam připojených klientů je uchováván ve třídě `MessageProcessor`.

Také je třeba vysvětlit, jakým způsobem server řeší posílání herních událostí. Většina z nich se týká jen svého blízkého okolí, a tak o nich stačí

informovat jen okolní hráče. To je zajišťováno třídou `SectorManager`, do které je možné objekty vkládat a ona je interně třídí a sdružuje do struktur podle oblastí, ve kterých se vkládaný objekt nachází. Následně umožňuje vyhledávání objektů kolem zadaných souřadnic a jednoduchým způsobem předá data konkrétní oblasti.

6.5.1 Vztahy tříd

Vztahy základních tříd serveru znázorňuje obrázek 6.9.



Obrázek 6.9: Vztahy tříd na straně serveru

7 Testování

V této kapitole bude uvedeno, jakým způsobem byla aplikace testována a laděna v průběhu celého vývoje.

7.1 Vlastní nástroj pro ladění

Během první poloviny vývoje, byla práce vyvíjena spolu s vizualizačním nástrojem (později editorem herních dat) a sloužila jako jeho logická a datová vrstva. Díky tomu bylo možné relativně jednoduše odladit knihovny geometrických operací pro výpočty pohybu, rotací a kolizí, IO knihovny pro práci se soubory a také správce těchto souborů.

7.2 Jednotkové testy

V druhé polovině vývoje bylo třeba řešit přenos dat mezi serverem a klienty. Byla snaha vytvořit pro ten úkol sadu jednotkových testů, které by jednoznačně určovaly správnost řešení. Později od toho ale bylo upuštěno z časových důvodů a důvodu nevhodné architektury aplikace, která by se musela pro takové testování předělat a ohrozit celou předchozí práci.

7.3 Logování

V předposlední fázi vývoje byl vytvořen úzký okruh lidí, kteří jako první podali autorovi zpětnou vazbu a pomáhali testovat chování aplikací při větším počtu uživatelů v reálném prostředí. Bylo objeveno a opraveno několik chyb. Server i klient byl vybaven logovacím systémem, využívající knihoven Javy `java.util.logging`, který umožňoval pochopit bližší souvislosti s popisovanými problémy.

7.4 Závěrečné testování

V poslední fázi byla aplikace uvolněna na internet s instrukcemi pro testování. Do toho se zapojilo 16 osob z různých věkových a technicky zdatných skupin. Postupovali podle připraveného scénáře, který je provedl přihlášením k serveru, vstupem do hry a jejím ovládáním, konflikty s ostatními hráči a

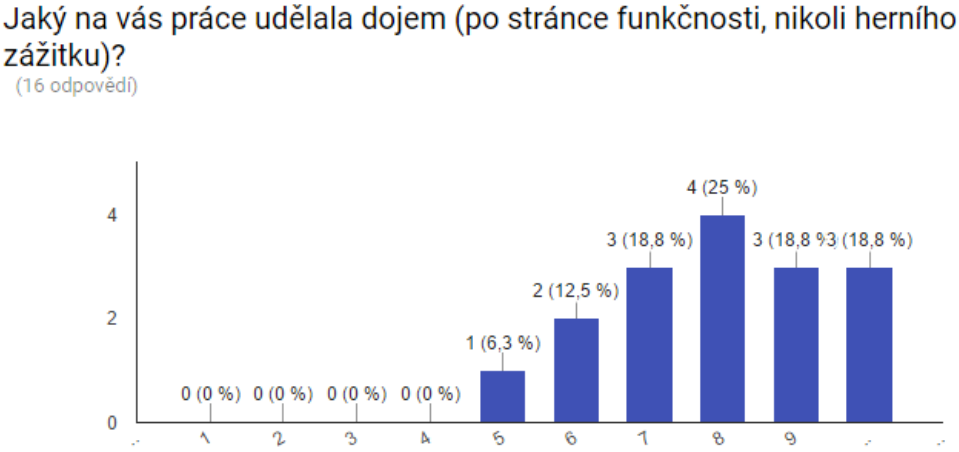
NPC. Dále testovali výpadky spojení a chyby při přihlašování a vlastní možnosti pro narušení provozu. V tabulce 7.1 je shrnutí jejich hodnocení:

Oblast	*V pořádku*	*S problémy*	*Popis problému*
Založení účtu	100 %	0 %	-
Vstup do hry	87,5%	12,5%	Nespecifikováno
Plynulost pohybu	25%	75%	Občasné, ale znatelné cukání.
Průchod branou	87,5%	12,5%	Nejasné
Střelba	68,8%	31,2%	Projektily nebyly vidět
Zachování herního stavu	100 %	0 %	-

Tabulka 7.1: Výsledky testování

V závěru měli testeři možnost popsat další objevené chyby. Několikrát se vyskytl problém, že při střelbě nebyl cílový objekt nijak poškozen. Druhou zmiňovanou chybou bylo, že hráč sebral bonus, ale ten se nepřičetl. Při cíleném pokusu chyby zopakovat však vše fungovalo dle očekávání.

Poslední otázka směřovala na celkový dojem z funkčnosti. Odpověď je na obrázku 7.1.



Obrázek 7.1: Závěrečné hodnocení testerů

8 Závěr

Cílem této práce bylo prozkoumat problematiku programování MMORPG her a jejich programování v Javě a takovou hru vytvořit. Během analýzy problematiky nebyly nalezeny žádné závažné nejasnosti. Při implementaci byl nejdříve vytvořen editor, ve kterém byla postupně odlazena herní logika a vytvořena herní data. Práce v tomto prostředí ukázala, že pro zadané požadavky je navržená implementace postačující. Chování objektů bylo plynulé a chovaly se očekávaným způsobem.

V druhé fázi vývoje bylo stávající herní jádro přeneseno do nově vytvořeného serveru a klienta a provázáno se síťovou částí obou aplikací. Bylo třeba udělat několik úprav jádra, neboť zvolenou kompozici nebylo možné takto používat na síti. Zpravidla šlo o cyklickou tvorbu událostí, které pak zahlcovaly spojení aplikací.

Během závěrečného testování pak bylo testery odhaleno několik nedostatků souvisejících se synchronizací serverového a klientského herního stavu. Nicméně chyby se zdály být spíše nahodilého charakteru a v omezeném čase a nedostatku lidských prostředků nebylo možné hru odladit k úplné spokojenosti autora. Přes to vše lze říci, že odevzdaná práce splňuje všechny body zadání.

Hru je možné dále rozvinout například o funkce správy hráčských účtů a herních atributů, tvorbu úkolů, herních spolků, obchodu a herní ekonomiky.

Přehled zkratek

MMORPG	Massivelly multiplayer online role playging game
FPS	First person shooter
RTS	Real time strategy
MMOG	Massively multiplayer online game
MMORTS	Massivelly multiplayer online real time strategy
MMOFPS	Massivelly multiplayer online first person shooter
TCP	Transmission control protocol
UDP	User datagram protocol
P2P	Peer to peer - rovný s rovným
FPS	Frames per second - počet snímků za vteřinu
GUI	Graphic user interface - grafické uživatelské rozhraní
API	Application programming interface
CPU	Central processor unit
GPU	Graphical processor unit
JVM	Java virtual machine
IO	Input, output
NPC	Non player character

Literatura

[1] Huizinga, J. *Homo ludens a study of the play-element in culture.* Routledge and Kegan Paul, 1998. ISBN 0-7100-0578-4.

[2] Kramer, W. *What is a Game?* [online]. The Games Journal. [cit. 2016/05/30]. Dostupné z: `http://www.thegamesjournal.com/articles/WhatIsaGame.shtml`.

[3] Gregory, J. *Game engine architecture.* A K Peters, 2009. ISBN 978-1-56881-413-1.

[4] Rick Hall, J. N. *Game development essentials.* Delmar/Cengage Learning, 2008. ISBN 978-1-4180-5267-6.

[5] *TCP/IP Frequently Asked Question* [online]. Mike Oliver. [cit. 2016/05/30]. Dostupné z: `http://www.itprc.com/tcpipfaq/faq-1.htm`.

[6] Davison, A. *Programování dokonalých her v Javě.* Computer Press, 2006. ISBN 80-7226-944-5.

[7] *What is a Socket?* [online]. www.tutorialspoint.com. [cit. 2016/05/30]. Dostupné z: `http://www.tutorialspoint.com/unix_sockets/what_is_socket.htm`.

[8] David Brackeen, L. V. B. B. *Vývoj her v jazyku Java.* Grada, 2004. ISBN 80-247-0874-4.

[9] Vevera, P. *30 FPS vs. 60 FPS - shrnutí velké internetové "debaty"* [online]. CDR server s.r.o., 2015. [cit. 2016/05/30]. Dostupné z: `http://cdr.cz/blog/30-fps-vs-60-fps-shrnuti-velke-internetove-debaty`.

[10] *What is GPU rendering?* [online]. Art And Animation studio. [cit. 2016/05/30]. Dostupné z: `http://furryball.aaa-studio.eu/aboutFurryBall/whyGpu.html`.

[11] *API - application program interface* [online]. Quinstreet Enterprise. [cit. 2016/05/30]. Dostupné z: `http://www.webopedia.com/TERM/A/API.html`.

[12] *Getting Started with Direct3D* [online]. Microsoft. [cit. 2016/05/30]. Dostupné z: `https://msdn.microsoft.com/cs-cz/library/windows/desktop/hh769064(v=vs.85).aspx`.

[13] *OpenGL Overview* [online]. OpenGL. [cit. 2016/05/30]. Dostupné z: https://www.opengl.org/about/.

[14] Mike McShaffry, D. G. *Game coding complete*. Course Technology PTR, 2013. ISBN 978-1-133-77657-4.

[15] *TIOBE Index for June 2016* [online]. TIOBE software BV, 2016. [cit. 2016/06/12]. Dostupné z: http://www.tiobe.com/tiobe_index?page=index.

[16] *JVM - Java Virtual Machine* [online]. Quinstreet Enterprise. [cit. 2016/05/30]. Dostupné z: http://www.webopedia.com/TERM/J/JVM.html.

[17] *Java Platforms* [online]. About.com Tech. [cit. 2016/05/30]. Dostupné z: http://java.about.com/od/gettingstarted/a/javatechnologies.htm.

[18] *Package java.net* [online]. Oracle, 2016. [cit. 2016/06/12]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/net/package-summary.html.

[19] *Java NIO tutorial* [online]. Jenkov Aps, 2016. [cit. 2016/06/12]. Dostupné z: http://tutorials.jenkov.com/java-nio/index.html.

[20] Fletcher, J. *AWT vs Swing* [online]. Embarcadero Technologies, Inc., 2013. [cit. 2016/06/12]. Dostupné z: http://edn.embarcadero.com/article/26970.

[21] *Lesson: Overview of the Java 2D API Concepts* [online]. Oracle, 2016. [cit. 2016/05/30]. Dostupné z: https://docs.oracle.com/javase/tutorial/2d/overview/.

[22] *The Joy of Java 3D* [online]. Java3D.org. [cit. 2016/06/21]. Dostupné z: http://www.java3d.org/introduction.html.

[23] *(The Java^{TM} Tutorials > Collections)* [online]. Oracle, 2016. [cit. 2016/06/12]. Dostupné z: https://docs.oracle.com/javase/tutorial/collections/interfaces/index.html.

[24] *File (Java Platform SE 8)* [online]. Oracle, 2016. [cit. 2016/06/12]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/io/File.html.

[25] *(The Java^{TM} Tutorials > Essential Classes > Basic I/O)* [online]. Oracle, 2016. [cit. 2016/06/12]. Dostupné z: https://docs.oracle.com/javase/tutorial/essential/io/streams.html.

[26] *(The Java^{TM} Tutorials > Essential Classes)* [online]. Oracle, 2016. [cit. 2016/06/12]. Dostupné z: `https://docs.oracle.com/javase/tutorial/essential/io/index.html`.

Příloha A

Uživatelská příručka

Příloha A

Uživatelská příručka

Zde je popsáno jakým způsobem lze aplikace spustit a ovládat.

Instalace

Podmínkou pro běh aplikací je nainstalované běhové prostředí jazyka Java 8. Aplikace není možné spustit přímo z přiloženého CD kvůli nemožnosti zápisu souborů. Zkopírujte prosím složku s nimi na lokální disk.

Editor

Editor pro své spuštění nepotřebuje žádné parametry. Lze ho spustit z příkazové řádky příkazem `java -jar BotLaboratory.jar`. Ukáže se prázdná plocha. Do ní lze vložit libovolný herní objekt a sledovat a měnit v levém panelu jeho atributy. Chcete-li testovat střelbu, je nutné objektům typu `Craft` a `RegularBot` nastavit číslo týmu. Není možné útočit na objekt se stejným číslem týmu a číslem 0, které značí neutralitu.

Server

Server je třeba spustit z příkazové řádky příkazem `java -jar WS Server 1.0.0 pre-alpha.jar`. Je možné zadat právě dva parametry v podobě ip adresy nebo jména serveru a čísla portu. Nebudou-li parametry zadány, bude použit defaultní port a aplikace vyzve k výběru možné adresy.

Serveru je možné za běhu zadávat následující příkazy:

- `help` – vypíše dostupné příkazy
- `status` – vypíše připojené klienty, případně i jejich jména, jsou-li přihlášení
- `players` – vypíše hráče připojené do hry
- `tp` – přemístí hráče na zadané souřadnice
- `tpall` – přemístí všechny hráče na zadané souřadnice

- `sendbc` – pošle všem klientům chat zprávu
- `save` – uloží stav hráčů
- `close`, `exit` – uloží stav hráčů a ukončí aplikaci

Klient

Klientskou aplikaci lze spustit z příkazové řádky příkazem `java -jar WS 1.00.3 pre-alpha.jar`. Stejně jako u serveru je možné zadat právě dva parametry v podobě ip adresy nebo jména serveru a čísla portu. Nejsou-li zadány, klient se automaticky pokusí připojit na adresu testovacího serveru na defaultním portu.

Vstup do hry

Po spuštění je zobrazen dialog pro přihlášení nebo založení účtu. Po přihlášení se zobrazí obrazovka hangáru (v této verzi ještě prázdná). Do cca 2 vteřin by se mělo aktivovat tlačítko „Enter the game“.

Ovládání

- levé tlačíko myši – označení objektů
- pravé tlačítko myši – směr letu
- čísla 1,2,3 (+,é,š) – zapnout / vypnout zbraň

Střelba

Červené tečky na mapě značí nepřítele. Po jeho označení levým tlačítkem a stiskem některé z číselných kláves na levé straně klávesnice 1,2,3 (+,é,š) se aktivuje příslušná zbraň. 1+3 jsou kanony, které nepřítele zraní jen pokud se od výstřelu projektilu nepřesune jinam. 2 odpaluje řízené rakety s delší dobou přebíjení a delší operační oblastí.