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

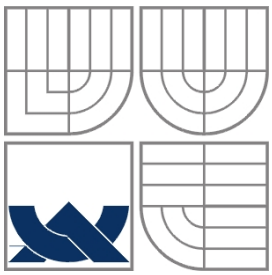
PROJEKT VESMÍRNÉ HRY VYUŽÍVAJÍCÍ SIMULAČNÍ ENGINE DELTA3D

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

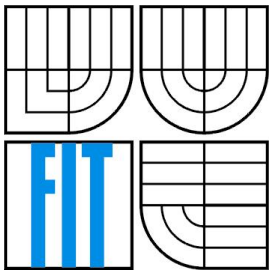
AUTOR PRÁCE
AUTHOR

VÁCLAV SAUER

BRNO 2014



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

PROJEKT VESMÍRNÉ HRY VYUŽÍVAJÍCÍ SIMULAČNÍ ENGINE DELTA3D

SPACE GAME PROJECT USING DELTA3D SIMULATION ENGINE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VÁCLAV SAUER

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN PEČIVA, Ph.D.

BRNO 2014

Abstrakt

Tato práce se zabývá simulačním a herním enginem Delta3D a možností jeho využití pro vývoj her. Poskytuje seznámení s tímto enginem a jeho ideologií. Je zde popsáno, jak tento engine přistupuje k vykreslování, výpočtu fyziky, přehrávání zvuků a jiným záležitostem hojně ve hrách využívaným. Dále zde jsou popsány nástroje poskytované enginem Delta3D, které je možné při vývoji her použít. Těchto informací je poté použito k implementaci samotné jednoduché hry v tomto enginu. Postup při návrhu a implementaci této hry je popsán v hlavní části práce.

Abstract

This bachelor's thesis is about gaming and simulation open-source engine Delta3D and its possibilities in game design and development. Work describes basics and ideology of this engine and how engine handles drawing, physics, audio processing and other things widely used in current games. Furthermore this work describes tools offered by engine, which may be used in the process of creating games. This information is then used to implement simple game created with this engine. Process of design and implementation is described in the main part of this thesis.

Klíčová slova

Delta3D, vývoj her, herní engine, herní fyzika, částicové efekty

Keywords

Delta3D, game development, game engine, game physics, particle effects

Citace

Václav Sauer: Projekt vesmírné hry využívající simulační engine Delta3D, bakalářská práce, Brno, FIT VUT v Brně, 2014

Projekt vesmírné hry využívající simulační engine Delta3D

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Pečivy, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Václav Sauer
20.5.2014

Poděkování

Rád bych poděkoval panu Ing. Janu Pečivovi, Ph.D. za jeho vedení a za jeho ochotu a rady a to především v závěru semestru.

Obsah

1 Úvod

Herní engine tvoří jádro každé moderní hry. Mezi funkce, které herní engine může nabídnout patří např. vykreslování scény, řešení fyziky a kolizí, ozvučení, síťová komunikace, animace, správa souborů a dat, umělá inteligence, uživatelské rozhraní, vydávání na různé platformy apod. Co konkrétního a jakým způsobem je nabídnuto pak záleží již na samotném enginu a jeho implementaci.

V dnešní době je již zcela běžné, že studia využívají herní enginy jiných společností, především kvůli úspoře peněz a času. Vývoj herního enginu totiž není jednoduchá záležitost a málokteré studio si ho může dovolit. Díky úsporám spojeným s použitím cizí technologie se pak mohou studia mnohem více věnovat tvorbě samotného obsahu hry namísto vytváření těchto základních funkcionalit. Mezi v dnešní době velmi populární enginy patří například Unity[1], Unreal Engine 4 [2] či CryEngine[3]. Všechny tyto enginy jsou komerční a i přesto, že studia mohou velkou část těchto enginů využívat zcela zdarma, popřípadě za velmi nízký poplatek, najdou se oblasti kde je vhodnější použití nezávislého enginu s otevřeným zdrojovým kódem. Jedním z nich je i simulační a herní engine Delta3D[4].

Tato práce se zabývá procesem tvorby hry v enginu Delta3D. Nejprve je popsán samotný použitý engine Delta3D a jeho součásti. Důraz je přitom kladen především na ty, které jsou ve hře dále použity. Poté je v práci rozebrán proces samotného návrhu hry a poslední část se týká samotné implementace této hry v Deltě3D.

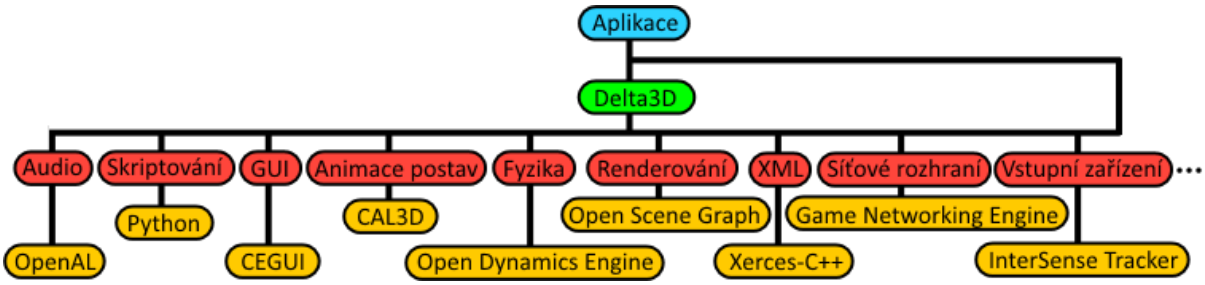
2 Engine Delta3D

Vývoj enginu Delta3D započal v Modelling, Virtual Environments and Simulation institutu (dále MVES), v Naval Postgraduate School v Kalifornii. Engine byl původně zamýšlen pro použití při simulaci armádních projektů, avšak v současnosti je používán mnoha dalšími subjekty[5]. Filozofie projektu okolo enginu Delta3D se skládá ze čtyř bodů. Zachovat otevřenost celého enginu ve všech směrech. Cílem je zabránění fixace enginu na cizí proprietární technologii. Udržovat engine nezávislý na druhu her/simulací které na něm budou postavené. Tím by neměl nastat případ, že bude potřeba vytvořit aplikaci, kterou engine nezvládne. Postavit engine modulárním stylem tak, aby se jakákoliv jeho část dala snadno vyměnit za jinou, lepší či pokročilejší a vybudovat kolem enginu komunitu, jejímiž příspěvky se může engine sám vyvíjet za minimální náklady ze strany institutu. Podrobnější informace o tomto přístupu lze nalézt v článku přímo od institutu MVES[6], z nějž jsou také mimo jiné čerpány následující informace o enginu.

2.1 Struktura enginu Delta3D

Delta3D v sobě integruje několik open-source knihoven a projektů. Mezi tyto projekty patří například:

- OpenSceneGraph pro vykreslování scény pomocí technologie OpenGL
- Open Dynamics Engine pro simulaci fyziky objektů
- Character Animation Library CAL3D pro skeletální animaci postav
- OpenAL pro ozvučení
- Crazy Eddie's GUI pro uživatelské rozhraní
- Game Networking Engine pro podporu síťových aplikací
- Freetype pro renderování textu
- Xerces parser XML souborů a další[7]



Obrázek 1: Struktura knihoven Delty3D

Všechny tyto projekty jsou do Delty3D integrovány tak, aby aplikační rozhraní enginu bylo co nejvíce vysokoúrovňové. To především z toho důvodu, aby tvorba aplikací byla co nejjednodušší. Zá-

roveň však jsou zpřístupněna veškerá nízkoúrovňová rozhraní, což v případě potřeby umožňuje mít nad činností enginu naprostou kontrolu.

Kromě těchto základních knihoven patří k Deltě3D také projekt Delta3D-Extras. Ten obsahuje spoustu dílčích projektů, které se k Deltě3D vztahují. Nejsou však nutně vyvíjeny vývojáři kolem Delty3D, ale spíše samotnou komunitou.

2.2 Vykreslování

K vykreslování je využíváno open-source knihoven OpenSceneGraph (dále OSG). Projekt OSG staví na technologii OpenGL a podpoře velkého množství platforem. Aplikace na něm postavené je možné vydávat jak na systémy pro stolní počítače (Microsoft Windows, Mac OS, Linux aj.), tak na mobilní zařízeních se systémy Android a iOS [8].

OSG má implementováno mnoho funkcí pro dosažení co nejlepšího výkonu. Z nich lze zmínit například podporu pro Level of Detail či vykreslování pouze těch částí scény, které jsou viditelné (view-frustum culling a occlusion culling), OpenGL state sorting či vícevláknové zpracování. Objekty, které mají být vykresleny, jsou v OSG ukládány do stromové struktury a před samotným vykreslením nad nimi OSG provádí optimalizace, které ještě více napomáhají zvyšovat výkon. OSG dále také poskytuje mechanizmus pro načítání modelů a textur, obrázků a mnoha dalších souborů.

Delta3D díky využití OSG z něj tyto vlastnosti přejímá.

2.3 Fyzika objektů

Fyzika objektů je ve hrách velmi důležitý prvek. Míra toho, jak realisticky je fyzika implementována, velmi závisí na typu hry. Ve hrách se můžeme setkat s velmi komplexní fyzikou, kde například v leteckém simulátoru je důležité modelovat pokročilé chování letadla za různých povětrnostních podmínek. V závodních hrách může být velmi pokročilý realistický model beroucí v potaz různé povrchy, po kterých se vozidlo pohybuje, přilnavost jednotlivých kol a síly na ně působící. Zrovna tak se můžeme setkat se hrami, které fyziku implementují velmi jednoduchým způsobem. V tomto případě si můžeme představit jednoduchou 2D hru typu Mario nebo závodní hru, kde ač modely mohou být prostorové, tak vozidlo je stále na zemi, pohybuje se čistě ve dvojrozměrném prostoru a při nárazu se jednoduše zastaví.

Pro fyziku objektů využívá Delta3D v základu Open Dynamics Engine (dále ODE). Díky modularitě enginu je možné pro výpočet fyziky použít také fyzikální engine PhysX či Bullet. ODE nabízí komplexní simulaci fyziky pevných těles včetně simulace jejich pohybu v prostoru v závislosti na silách na ně působících a detekce jejich kolizí[9].

Těleso v enginu ODE

Pevné těleso má ve fyzikálním enginu ODE několik vlastností, z nichž některé se během simulace mění a jiné zůstávají ve většině případů po celou dobu simulace konstantní. Mezi vlastnosti, které se během simulace mění patří:

- Pozice tělesa, dána třemi souřadnicemi (x, y, z). Tato pozice v enginu ODE odpovídá těžišti objektu

- Rychlost pohybu tělesa v prostoru, dána vektorem (vx, vy, vz)
- Rotace tělesa popsána kvaternionem (qs, qx, qy, qz)
- Rychlost rotace tělesa, dána vektorem (wx, wy, wz)

Vlastnosti, které se během simulace zpravidla nemění jsou například:

- Hmotnost tělesa
- Rozložení hmotnosti kolem těžiště tělesa popsané maticí o velikosti 3x3

Tělesa v enginu ODE tvoří tzv. ostrovy. Ostrov je skupina těles pospojovaných klouby, která nejsou dále rozebíratelná. Každé těleso poté může být v enginu ODE zapnuté či vypnuté. Pokud je těleso vypnuté, tak není při kroku simulace aktualizováno. Tím je možné šetřit výkon potřebný na výpočet fyziky. Engine umí tělesa sám vypínat v případě, že je těleso v klidu po určitou dobu a určitý počet kroků simulace. Programátor také může tělesa vypínat a zapínat manuálně.

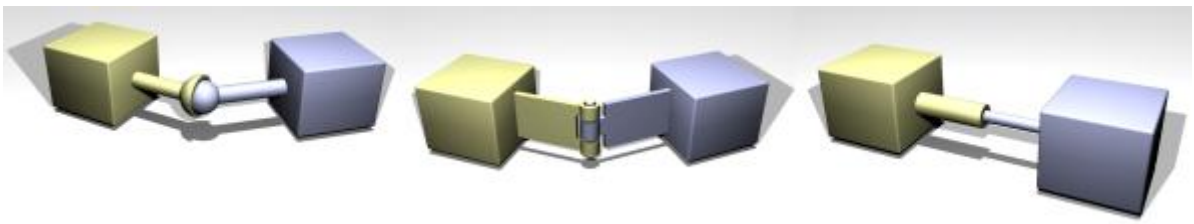
Krok fyzikální simulace

Simulace v enginu ODE probíhá krokově a na starosti jí má integrátor. V jednom kroku simulace integrátor posune čas o daný úsek a následně jsou aktualizovány stavy všech těles. Problémy, které je potřeba během tohoto procesu řešit je stabilita tohoto integrátoru a jeho přesnost. Integrátor v enginu ODE je dle slov autorů velmi stabilní [16] a nemělo by se tedy stát, že zdánlivě stabilní systém například bez zjevného důvodu exploduje. Přesnost integrátoru ODE není dokonalá. Lze ji zlepšit snížením kroku simulace, avšak za cenu větší výpočetní náročnosti. I přesto jsou však tyto nepřesnosti natolik malé, že je bez měření nelze podchytit a pro herní fyziku je engine zcela vhodný.

Pokud je na těleso mezi kroky simulace působeno nějakými silami, jsou tyto síly uloženy do zásobníku. V dalším kroku simulace je na těleso aplikován součet všech sil, které na něj mezi těmito dvěma kroky působily.

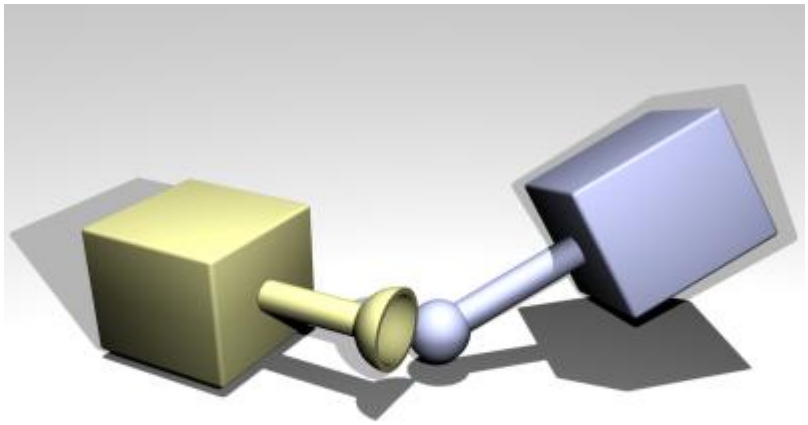
Klouby spojující tělesa

Fyzikální engine podporuje několik typů kloubů, jimiž mohou být tělesa spojena (viz obr. 2). První kloub je dán pouze bodem, ke kterému jsou tělesa vztažena. Tento kloub umožňuje největší volnost pohybu. Druhým kloubem je kloub, kde tělesa jsou vztažena k ose a jejich pohyb je tak možný pouze rotačně kolem ní. Třetím kloubem je kloub, ve kterém se tělesa mohou pohybovat pouze k sobě a od sebe. Pokud jsou dvě tělesa spojena kloubem, tak je jejich pohyb při výpočtu kroku simulace integrátorem vždy spočítán tak, aby odpovídal danému kloubu.



Obrázek 2: Typy kloubů v enginu ODE. Zdroj ODE manuál.

Pokud by se během simulace stalo, že dojde k vykloubení kloubu, například ručním přesunutím jednoho objektu bez přesunutí druhého či chybou za běhu simulace, umožňuje ODE tuto chybu napravit pomocí tzv. *Error correction* parametru. Tento parametr udává, jak hodně se engine snaží vrátit vykloubené klouby nazpět. Pokud je tento parametr nastaven, tak v případě vykloubení jsou na tělesa aplikovány navíc dodatečné síly proti směru vykloubení, které vrátí postupně tělesa zpět.



Obrázek 3: Vykloubený kloub u dvou těles. Zdroj ODE manuál.

Kolize

Pro detekci kolizí je možné využít buď některé ze základních primitivních těles, mezi něž patří koule, kvádr, válec, kapsle, rovina a paprsek nebo lze použít kompletní kolizní model. Detekce kolizí je především při použití složitějšího modelu značně náročná. Je proto vhodné a v praxi i často praktikované u aplikací zjednodušovat kolizní model oproti modelu, který je zobrazen. To musí být činěno s ohledem na následné nasazení aplikace, aby nepřesnou kolizí nemohlo docházet k problémům. Každý kolizní model má v knihovně ODE také určitou hmotnost, která ovlivňuje, jak na něj působí síly. Další užitečnou vlastností knihovny ODE jsou kategorie, do kterých lze objekty přiřazovat. Podle těchto kategorií je možné určovat, mezi kterými objekty může nastat kolize a které objekty spolu nekolidují.

Samotná detekce kolizí se provádí těsně před provedením kroku simulace a probíhají během ní následující události:

1. Nejprve je zavolána funkce, která spočítá body, v nichž se tělesa dotýkají, tzv. kontaktní body. Každý bod určuje souřadnice v prostoru, normálu na povrch v místě kolize a hloubku kolize.
2. V místě kolize je vytvořen speciální kolizní spoj pro každý kontaktní bod. Tomuto spoji jsou doplněny další informace o kontaktu, jako je například působící tření, odrazivost povrchu, minimální rychlost potřebná pro odražení apod.

3. Tyto kolizní spoje jsou vloženy do skupiny, což v pozdější fázi umožňuje rychlé odstranění všech těchto spojů odebráním celé skupiny a tím pomáhá simulaci běžet rychleji.
4. Je proveden krok simulace a po jeho skončení jsou odebrány všechny kontaktní spoje ze systému.

2.4 Částicové efekty

Částicové efekty jsou efekty, u kterých dochází současně k zobrazování velkého množství jednoduchých textur či jiných grafických prvků, takzvaných částic. Chování jednotlivých částic je často závislé na délce života částice. Může tak být odvozen tvar částice, její barva, velikost, rotace, směr a rychlost pohybu a další vlastnosti. Tímto komplexním systémem lze napodobovat prvky v reálném světě, které by jinak byly velmi náročné na vytvoření a vykreslení. Efekty, které lze takto simulovat jsou třeba kouř a jeho variace (vodní tříšť, prach), plameny nebo exploze. Částice mohou také reagovat na kolize s okolním prostředím, vrhat na něj stíny či ho osvětlovat a lze tak docílit velmi realistických napodobenin skutečných situací, jako například jiskry dopadající na podlahu. Výpočty takového chování jsou velmi náročné, v poslední době se však začínají s roustoucím výkonem počítačů v herních enginech objevovat [10].

K vykreslování částicových efektů je v Deltě3D opět využito knihoven OpenSceneGraph. Veškeré částicové efekty jsou drženy ve třídě `osgParticle::ParticleSystem`. Ta má na starosti vytváření, aktualizaci, renderování a destrukci všech částicových efektů. Každý jednotlivý částicový efekt, je pak reprezentován třídou `osgParticle::Particle`. Ta obsahuje `osgParticle::ModularEmitter`, který má na starosti chování efektu při vytváření jednotlivých částic. Toto chování řídí tří subkontrolery:

- `osgParticle::Placer` – Určuje, kde budou vytvářeny nové částice v efektu.
- `osgParticle::Shooter` – Nastavuje počáteční rychlost a směr pohybu jednotlivých částic
- `osgParticle::Counter` – Řídí množství produkovaných částic.

Poslední důležitou částí částicového efektu je `osgParticle::Program`, který popisuje chování částic během jejich existence. Částice tak mohou například zrychlovat, jako by byly ve větru či zpomalovat jako kdyby se nacházely ve velmi hustém prostředí[11].

2.5 Ozvučení

K ozvučení využívá engine knihovnu Open Audio Library, zkráceně OpenAL. Tato knihovna umožňuje vytvářet komplexní 3D ozvučení ve scéně kolem uživatele. Umožňuje přehrávání mono i stereo zvuků, lineární a exponenciální změnu jejich hlasitosti vzhledem ke vzdálenosti od uživatele, Dopplerův efekt či změnu výšky zvuku (pitch efekt). Lze pomocí ní vytvořit kompletní ozvučení hry, ať už se jedná o efekty ve hře, hudbu na pozdí či rozhovory.

2.6 Uživatelské rozhraní

Jako uživatelské rozhraní využívá Delta3D především projektu Crazy Eddie's GUI, dále jen CEGUI [12]. Tato knihovna je implementována v jazyce C++ a podporuje platformy Windows, Linux a Mac OS ve 32 bitové i 64 bitové verzi.

Tvorba uživatelského rozhraní je možná dvěma způsoby. Prvním z nich je vytvoření uživatelského rozhraní programově ve zdrojovém kódu. Druhým způsobem je nadefinování uživatelského rozhraní pomocí XML dokumentu, který se následně do aplikace načte. Uživatelské rozhraní definované pomocí XML dokumentu obsahuje několik druhů souborů.

Prvním typem jsou soubory s koncovkou *imageset*. Ty poskytují definici bitmap použitých pro jednotlivé prvky uživatelského rozhraní. Ke každému imageset souboru proto patří i obrázek, zpravidla ve formátu *tga*, obsahující tyto prvky.

Soubory s koncovkou *looknfeel* obsahují popis jednotlivých prvků uživatelského rozhraní jako je tlačítko, checkbox, okno apod. Je v nich nadefinováno z jakých částí imagesetu se daný prvek skládá a často je zde uvedeno několik variant daného prvku (například s rámem a bez rámu).

Soubory s koncovkou *font* popisují jeden konkrétní font, který je v daném uživatelském rozhraní využit a to jak typ fontu, tak i jeho velikost.

Dalším ze souborů je typ s koncovkou *scheme*. Tento soubor popisuje jedno konkrétní schéma. Jsou v něm uvedeny použité imagesety, fonty a použitý looknfeel. Je proto možné mít například vytvořeno několik různě vybarvených imageset souborů a pouze změnou toho, který se aktuálně používá, měnit vzhled celého uživatelského rozhraní.

Posledním souborem je soubor *layout*. Tento soubor popisuje samotné uživatelské rozhraní. Je v něm popsáno rozložení prvků na obrazovce, velikost jednotlivých prvků uživatelského rozhraní, jejich popisky, reakce na uživatelský vstup apod.

Výhodou uživatelského rozhraní definovaného pomocí XML dokumentu je možnost úpravy tohoto rozhraní aniž by bylo potřeba zasahovat do kódu samotného programu.

Dobře vytvořené uživatelské rozhraní je jedna z nejdůležitějších věcí pro každou aplikaci. Uživatel se v něm musí snadno a rychle orientovat a mít možnost lehce pomocí něj udělat to, co opravdu potřebuje.

2.7 Nástroje pro práci s enginem

Delta3D poskytuje pro usnadnění vývojářům několik nástrojů:

- AIUtility – Nástroj pro vytváření a editaci wapointů a navigačních map pro umělou inteligenci.
- Animation Viewer – Nástroj pro správu animací skeletálních modelů.
- Object Viewer – Nástroj pro prohlížení map, modelů a pixel shaderů.
- Particle Editor – Nástroj pro vytváření a úpravu částicových efektů pro engine Delta3D.
- STAGE – Nástroj pro komplexní tvorbu map pro engine Delta3D.

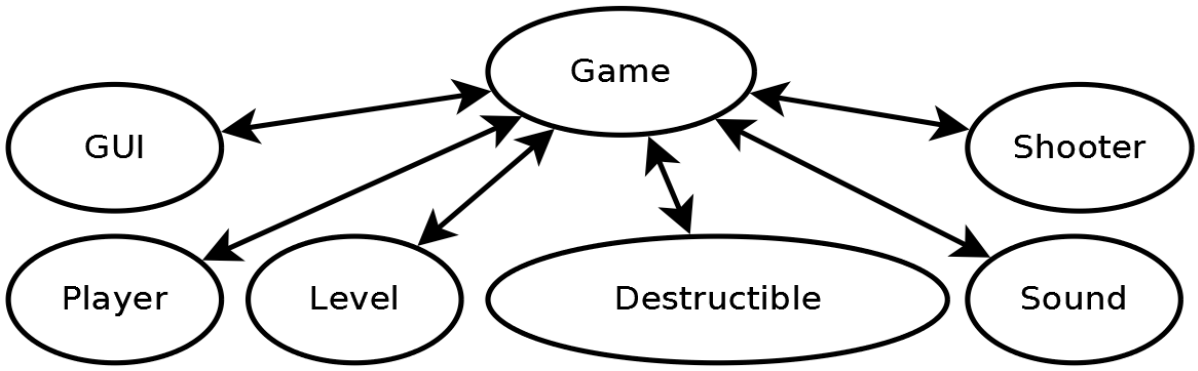
Z těchto programů jsem použil především Particle Editor a editor STAGE. Samotné využití je popsáno dále.

3 Návrh hry

Jako základní žánr hry byla zvolena jednoduchá 3D simulace ve vesmíru. V této simulaci hráč ovládá jednomístnou bojovou loď vybavenou štítem a laserovým dělem. Ve hře hráč plní jednoduchou příběhovou linii, během níž narazí na další prvky z herního světa.

3.1 Struktura hry

Hru utváří jeden centrální prvek, pojmenovaný Game. Ten tvoří samotné jádro hry. Na sobě obsahuje komponenty, jež mají na starosti samotnou funkcionalitu jednotlivých prvků hry. Jádro hry má na starosti komunikaci mezi jednotlivými komponenty a obsluhu základních událostí, které předává jednotlivým komponentům dle potřeby.



Obrázek 4: Struktura hry

Mezi komponenty, které toto jádro obsluhuje, patří:

- GUI – řídí a obsluhuje uživatelské rozhraní
- Player – obsahuje veškeré informace o hráči a řídí ovládání lodě hráče
- Level – obstarává nahrávání úrovní a získávání informací o objektech z úrovní
- Destructible – hlídá stav zničitelných herních objektů a ničí je pokud jsou sestřeleny a jejich zdraví klesne na nulu
- Sound – slouží pro přehrávání zvuků ze hry
- Shooter – vytváří a střílí projektily a následně je destruuje, když už nejsou potřeba

4 Implementace

V této kapitole je rozebrán postup při vytváření hry. Nejprve je zde popsáno jakým způsobem je možné v enginu vytvářet obsah a nahrávat ho do něj a poté je zde také popsána samotná implementace jak jádra hry, tak jednotlivých komponent.

4.1 Použitá verze enginu Delta3D

Tento projekt byl původně vyvíjen na enginu Delta3D ve verzi 2.7.5, v průběhu jeho tvorby pak byla uvolněna novější verze Delta3D 2.8.0-beta, na kterou byl projekt aktualizován. Ta přinesla především spolu s aktuálním SDK balíčkem potřebným pro vývoj také runtime balíček pro uživatele aplikací postavených na Deltě3D. Dále je zde také přidána podpora pro Microsoft Visual Studio ve verzi 2010 a 2012.

4.2 Příprava obsahu hry

Engine Delta3D podporuje širokou škálu formátů souborů a to jak pro modely objektů, tak pro textury, obrázky či zvuky. Množství podporovaných formátů je dáno především knihovnami, které jsou v Deltě3D použity. Tvůrce obsahu tak má často na výběr z několika programů, ve kterých může obsah tvořit, ať už to je komerční software jako například 3D Studio Max, tak volně dostupné programy typu Blender.

4.2.1 Modely objektů

Modely objektů pro vesmírnou hru pocházejí z několika zdrojů. Některé jsou vytvořené přímo pro potřeby tohoto projektu a některé pocházejí z různých online zdrojů.

Složité modely

Složité a komplexní modely jako například model základny či modely lodí pocházejí z online archivu turbosquid.com. Tento zdroj byl vybrán, protože nabízí velké množství vysoce kvalitních modelů dostupných zdarma, chráněných pod Royality Free licencí [15]. Tyto modely byly následně pomocí pluginu OSGExp pro 3D Studio Max vyexportovány do binárního formátu `osgb`. Použití tohoto formátu, který pochází přímo z OpenSceneGraph zaručuje dobrou kompatibilitu a vysoký výkon při jeho zpracování knihovnou. Další výhodou tohoto formátu je možnost uložit model spolu s materiály, texturami, animacemi atd.

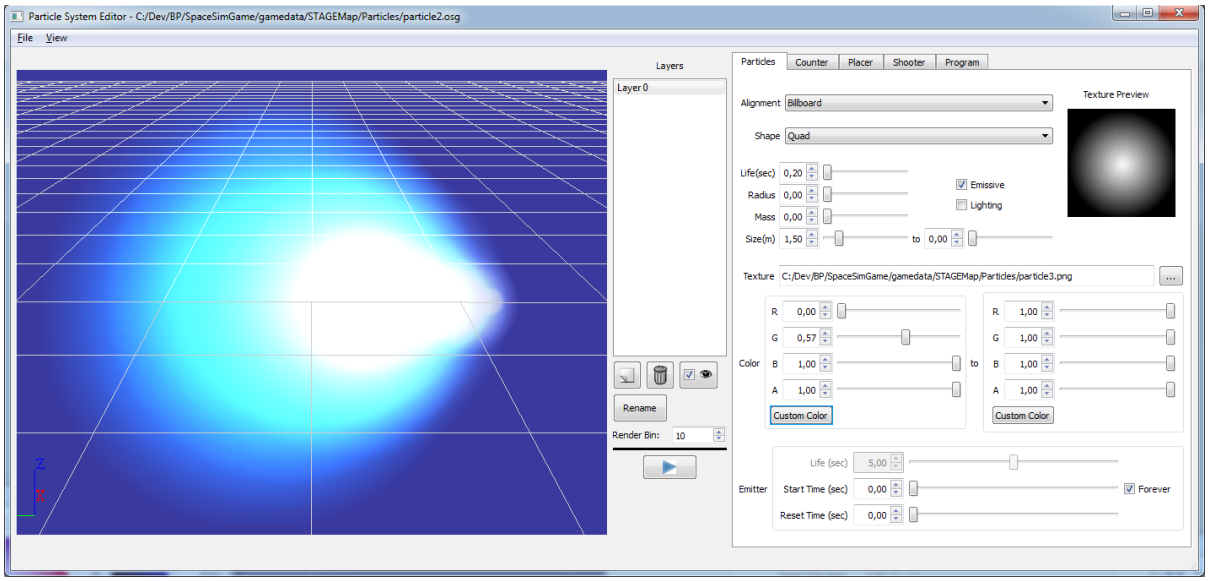
Jednoduché modely

Jednoduché modely jako například model laseru či model teleportační bóje jsou tvořeny přímo pro účely tohoto projektu. Proces jejich tvorby se skládal z několika kroků. Nejprve byl navrhnut základní tvar v programu SketchUp od společnosti Trimble Navigations Limited. Tento program je velice jednoduchý a nabízí tak možnost rychlého prototypování a zkoušení různých verzí modelu. V případě prototypování byl následně model exportován do formátu 3D Studio Mesh (`3ds`), který Delta3D umí zpracovat a rovnou použit. Pro finální verzi byl model vyexportován do formátu `fbx` od společnosti

Autodesk, který se nejlépe osvědčil a nezpůsoboval žádné problémy při konverzi textur apod. Následně bylo potřeba model importovat do programu 3D Studio Max pro doladění materiálů a dalších detailů a na závěr opět vyexportovat `osgb` model.

4.2.2 Tvorba částicových efektů

Pro tvorbu částicových efektů využívá Delta3D technologií OSG, nad nimiž poskytuje vlastní rozhraní. Samotné částicové efekty je možné tvořit přímo programově, avšak mnohem lepší je využít program Particle System Editor, jenž je součástí enginu Delta3D.



Obrázek 5: Rozhraní programu Particle System Editor.

V tomto programu je možné nastavit podstatnou část vlastností částicového efektu. Mezi nejdůležitější patří především výběr bitmapy, způsob vystřelování jednotlivých částic a jejich chování během života samotné částice. Pro komplexnější efekty je pak potřeba ovládat chování částicového efektu přímo za běhu programu ať už přes rozhraní Delty3D nebo detailněji přes rozhraní samotného OSG.

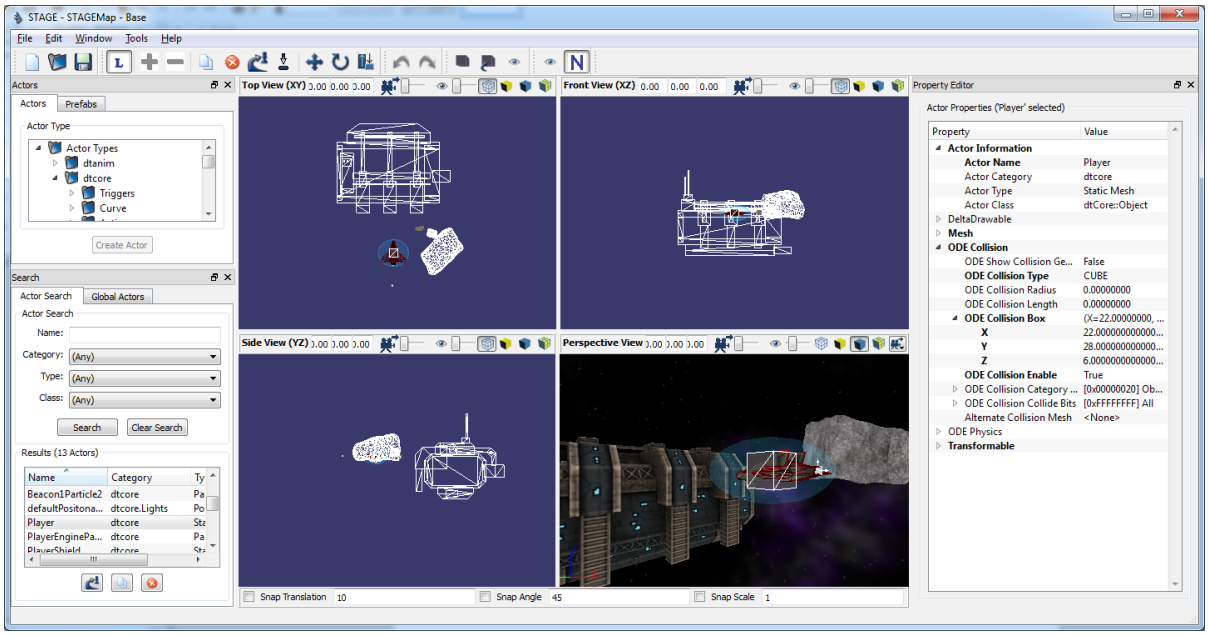
4.2.3 Tvorba uživatelského rozhraní

Pro tvorbu uživatelského rozhraní do hry neposkytuje engine Delta3D žádný nástroj, pouze ukázkové rozhraní s potřebnými soubory typu scheme, imageset, atd. Projekt CEGUI nabízí dva nástroje pro ulehčení tvorby uživatelského rozhraní a to CELayoutEditor pro tvorbu a úpravu rozložení prvků na obrazovce a CEImagesetEditor pro snadnou tvorbu imagesetů. Detailnější editace uživatelského rozhraní jako je doplnění událostí pro akce volané tlačítky je třeba provést přímo v XML dokumentech. Uživatelské rozhraní ve hře vychází z ukázkového uživatelského rozhraní distribuovaného s Deltou3D. Je však značně pozměněno, aby jeho styl odpovídal hře a jsou do něj doplněny prvky pro zobrazení stavu štítů a zdraví hráče a pro zobrazení výkonu motoru. Ukázka vytvoření jednoduchého tlačítka je uvedena níže.

```
<Window Type="WindowsLook/Button" Name="NewGame" >
   <Property Name="Font" Value="MASTOD_L" />
   <Property Name="Text" Value="New" />
   <Property Name="UnifiedAreaRect" Value="{{0.03,0},{0.6,0},{0.2,0},{0.65,0}}" />
   <Event Name="Clicked" Function="NewHandler" />
</Window>
```

4.2.4 Tvorba herních úrovní

Jednotlivé úrovně hry jsou vytvořeny v editoru STAGE poskytovaném enginem Delta3D. Tvorba úrovně se skládá z několika kroků. Je nutné navrhnout základní rozložení úrovně, umístění hráče a počítačem ovládaných objektů, promyslet změnu úrovně během hraní samotné hry a další [13]. Rozhraní editoru STAGE je vidět na následujícím obrázku.

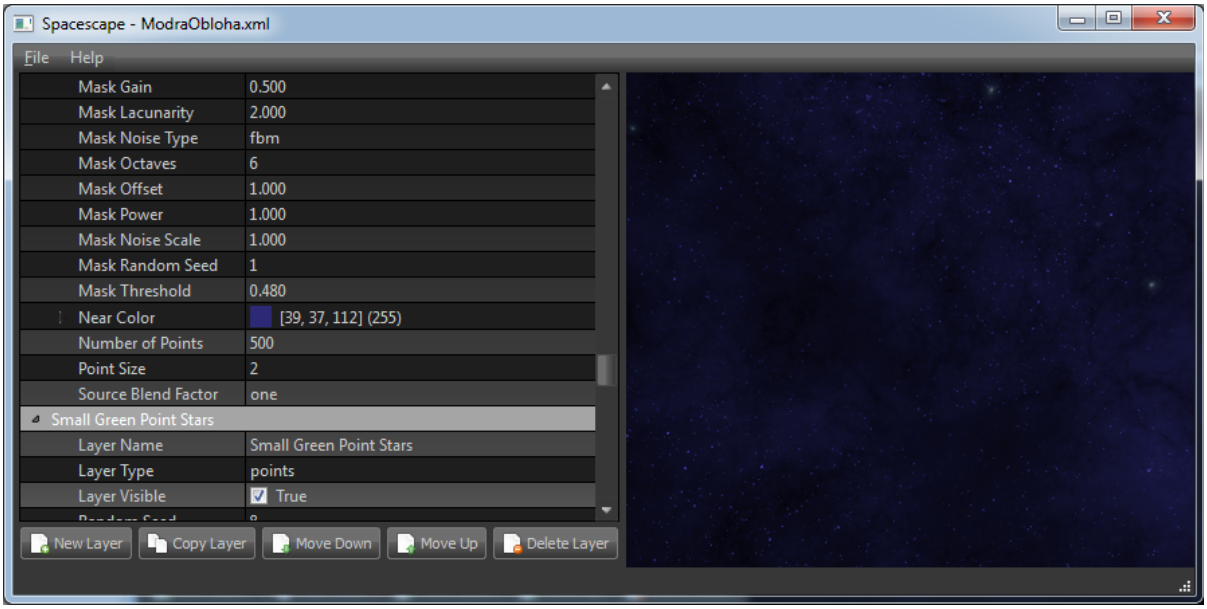


Obrázek 6: Rozhraní editoru STAGE.

Při vytváření úrovně je poté potřeba nejprve nahrát samotné modely, částicové efekty a další potřebná data do samotného editoru. Následně jsou tyto objekty umístěny do scény. U každého z nich je potřeba nastavit jeho základní vlastnosti jako je velikost, rotace a umístění. Důležité je také nastavení pokročilých vlastností jako kolizní model objektu a zda objekt reaguje dle fyzických zákonů daných enginem či je ve scéně statický, nepohyblivý.

Neméně důležité je umístit do scény osvětlení a nadefinovat skybox. Skybox je komponenta, určující pozadí v těch místech, kde není vykreslen žádný objekt. Jedná se o krychli obalující celou scénu, na jejíž vnitřní stranu jsou namapovány textury, odpovídající požadovanému pozadí. Tyto textury musí na hranách kostky na sebe navazovat, aby nebyl dojem pokažen nesouvislým přechodem. Při vykreslování je poté na prázdná místa, kde není žádný objekt, vykresleno toto pozadí.

Pro tvorbu textur skyboxu bylo využito volně dostupného programu Spacescape. Tento program procedurálně dle zadaných parametrů umožňuje vytvořit libovolné hvězdné pozadí. Výhodou je i možnost nadefinovat si barvy hvězd a tím sjednotit barevnou náladu skyboxu se scénou.



Obrázek 7: Editor textur skyboxů, program Scyscape.

4.2.5 Ozvučení

Ozvučení hry se skládá z krátkých efektů, které doprovázejí střelbu a interakci hráče v menu. Veškeré zvuky pocházejí z online archivu freesound.org.

4.3 Jádro hry

Jádro hry je ve hře implementované třídou Game a dědí přímo z `dtABC::Application`. To je základní třída pro aplikace v enginu Delta3D. Díky tomu je v něm možné odchytávat veškeré události spojené jak s obsluhou zařízení jako je myš nebo klávesnice, tak události spojené s vykreslováním.

Standardní postup při vytváření objektu této třídy, tedy při spouštění hry, je zavolání konstruktoru, kterému se jako parametr předá cesta ke konfiguračnímu souboru enginu Delta3D. Následuje zavolání metody `Config()` pro konfiguraci aplikace a spuštění samotné aplikace metodou `Run()`.

```
dtCore::RefPtr<Game> game = new Game("config/config.xml");
game->Config();
game->Run();
```

V konstruktoru třídy game je metodou `Game::SetDefaultWindow()` nastaven nadpis, velikost a pozice okna. Následně dojde k inicializaci veškerých tříd představujících komponenty a tyto komponenty jsou zaregistrovány pro příjem a odesílání zpráv mezi objekty.

V metodě `Config()` dochází k nahrání souboru s konfigurací ze samotné hry a k nastavení této konfigurace. V případě že tento soubor neexistuje, je vytvořen se základní konfigurací.

Po spuštění aplikace se jádro hry stará o zpracování uživatelského vstupu, o akce vykonané s vykreslením každého snímku, správné počítání kolizí a předávání zpráv mezi jednotlivými komponenty hry.

4.4 Implementace jednotlivých součástí hry

V této kapitole je popsána implementace jednotlivých součástí hry. Jsou zde zmíněny především ty věci, které jsou považovány za důležité, či se u nich narazilo při implementaci na problémy.

4.4.1 Zpracování uživatelského vstupu

Pro zpracování uživatelského vstupu jsou využity metody `KeyPressed`, `KeyReleased` pro klávesnici a `MouseButtonPressed`, `MouseScrolled`, `MouseMoved` a `MouseDragged` pro myš. Tyto metody jsou volány enginem Delta3D v případě stisknuté klávesy během každého vykresleného snímku aplikace.

Metoda `KeyPressed` je volána při stisku jakékoliv klávesy na klávesnici. Podle typu klávesy předané parametrem do této metody je možné zapauzovat/odpauzovat hru stiskem klávesy `Escape` a zobrazit text s nápovědou stiskem klávesy `F1`.

Do komponenty hráče jsou z metod `KeyPressed`, `KeyReleased` a `MouseButtonPressed` předávány stisknuté klávesy a tím je ovládán pohyb a střelba lodi hráče ve scéně. Dále je do komponenty hráče z metod `MouseMoved` a `MouseDragged` při uvolněném a stisknutém tlačítku předávána vzdálenost ujetá myší, čímž je ovládána změna směru, kterým letí loď hráče. Protože je při stisknutém tlačítku střelby událost stisknutí poslána pouze jednou, bylo nutné do události `MouseDragged` přidat vytváření falešné události o stisknutém tlačítku v případě, že je opravdu stisknuté. Tím není pro opakovanou střelbu nutné tlačítko znovu mačkat, ale stačí ho pouze držet.

```
if(mouse->GetButtonState(dtCore::Mouse::MouseButton::LeftButton) == true)
{
	m_Player->MousePressed(dtCore::Mouse::MouseButton::LeftButton);
}
```

Další falešné události jsou vytvářeny v metodě `MouseScrolled`, která za každou událost otočení kolečka myši vytvoří událost stisku tlačítka pro přidání/odebrání energie motorů na komponentě hráče.

4.4.2 Hráč

Veškeré záležitosti týkající se hráče ve hře obstarává třída `Player`. Řídí jeho pohyb, udržuje v sobě reference na model lodi, model štítu a částicový efekt motoru. Dále obsahuje informace o stavu hráčova štítu a zdraví a také obstarává pohyb kamery.

Pohyb hráče

Loď hráče je ve scéně fyzikální těleso s danou hmotností a tvarem, proto veškerý pohyb lodi je určen aplikováním sil na tento objekt. Aby vůbec k nějakému pohybu došlo, musí nejprve hráč obdržet událost o zmáčknuté klávese či o pohybu myši. Tyto události jsou do něj předávány ze třídy Game.

```
void KeyChanged(const int key, const bool value);
void MouseMoved(float horizontal, float vertical);
void MousePressed(dtCore::Mouse::MouseButton button);
```

Informace o zmáčknutých klávesách jsou ve třídě hráče uloženy a podle toho, které jsou zmáčknuty, je při dalším výpočtu fyziky na hráče aplikován impulz síly. Velikost tohoto impulzu bývá odvozená od velikosti síly a doby působení síly. Zde je impulz síly pokaždé stejný. Důvodem je to, že síly působící na hráče jsou ve hře aplikovány když je přepočítávána fyzika a toto se děje v pravidelných časových intervalech. Tím pádem jsou čas i síla určující tento vektor pokaždé stejné.

Drobnou výjimkou je vektor síly způsobující dopředný pohyb hráče. Tento vektor je počítán podle aktuálně nastaveného výkonu motoru. Před aplikováním je tedy tento vektor vynásoben aktuálním výkonem motoru (v rozsahu od 0.0 do 1.0) a tím je korigována rychlost lodi.

Otáčení lodi je počítáno odlišně. Při každém vypočítaném a zobrazeném snímku hry je ve třídě hráče přičtena vzdálenost ujetá myší. Tato celková vzdálenost se zde přičítá, dokud nedojde k výpočtu fyziky. K němu dochází v jiných časových intervalech než k výpočtu snímků hry. Velikost rotační síly je určena celkovou vzdáleností, kterou urazila myš mezi těmito dvěma výpočty fyziky. Tím je docíleno přesného a zároveň velmi jemného ovládání směru, kterým loď letí.

Úprava kamery, štítu a efektu motoru

Model štítu, kamery a částicový efekt motoru jsou za hráče umístěny v každém jeho updatu. Tento proces zahrnuje načtení souřadnic a rotace z modelu lodi a umístění kamery, štítu a částicového efektu správně relativně k této pozici a rotaci. Zároveň je podle nastavení výkonu motoru upraven částicový efekt motoru, aby odpovídal tomuto výkonu.

Poškození hráče

Zda je hráč živý či mrtvý určují dva ukazatele. Prvním z nich je štít, ze kterého se v případě poškození hráče odečítá nejdříve a který se pomalu za čas doplňuje. To umožňuje, aby hráči byly odpuštěny drobné chyby a přispívá k dobré hratelnosti hry. Druhým ukazatelem je samotné zdraví či poškození lodě. To se neobnovuje a odebírá se, až když je hráčův štít vyčerpán. V případě, že i to klesne na nulu, nastává konec hry.

Ve hře jsou dva způsoby, jak se hráč může zranit a oba druhy poškození vypočítává metoda `void Player::Damage(double value, Player::DamageType damageType).` První způsob poškození je kolize hráče s jiným objektem. V tomto případě je míra poškození závislá na rychlosti střetu a v případě velmi malých rychlostí není hráč poškozen vůbec. Tím je možné opatrně manévrovat v těsných prostorech, aniž by loď byla příliš poškozována a hráč zbytečně frustrován. Druhým způsobem zranění je zásah laserem. Ten ubírá vždy stejné množství ze štítu, popřípadě zdraví hráče.

Při smrti hráče se zobrazí hlavní menu se zprávou o konci hry. Hra přejde do nehratelného režimu a v menu je na pozadí místo letící lodě zobrazena scéna ze hry. Opětovně lze hru spustit stisknutím tlačítka s nápisem *New Game*.

4.4.3 Uživatelské rozhraní a hlavní menu

Uživatelské rozhraní obstarává třída/komponenta Gui. V jejím konstruktoru dochází k inicializaci vlastního uživatelského rozhraní v enginu Delta3D. Je zde nahráno schéma, a layout a nastavena obsluha tlačítek. Také jsou zde načteny reference na ty prvky uživatelského rozhraní, ke kterým se často, tedy až každý snímek, přistupuje. Vyhledání komponenty v uživatelském rozhraní nějakou chvíli trvá a u desítek komponent vyhledaných během každého vykresleného snímku by byl negativně ovlivněn výkon hry. Tímto dojde k jejich vyhledání jen jednou při načítání hry, poté se k nim již přistupuje přes tyto reference a celý proces je znatelně rychlejší.

```
//Vytvoření uživatelského rozhraní
  m_GUI = new dtGUI::GUI(cam, keyboard, mouse);
//Nahrání schématu
  m_GUI->LoadScheme("WindowsLook.scheme");
//Nastavení obsluhy tlačítek
  dtGUI::ScriptModule* sm = new dtGUI::ScriptModule();
    sm->AddCallback("NewHandler" , CEGUI::SubscriberSlot(&Gui::NewGameHandler, this));
    sm->AddCallback("QuitHandler", CEGUI::SubscriberSlot(&Gui::QuitHandler, this));
    //atd.
  CEGUI::System::getSingleton().setScriptingModule(sm);
//Nahrání layoutu
  m_GUI->LoadLayout("game.layout");
```

Důležité je nastavit uživatelskému rozhraní, aby propouštělo události z klávesnice a myší dále pod sebe, jinak není možné ovládat hru.

```
m_GUI->GetWidget("Root")->setMousePassThroughEnabled(true);
```

Uživatelské rozhraní se skládá z pěti oken. První dvě jsou okno s menu hry a okno s nastavením. Ta obsahují pouze tlačítka a zaškrtávací políčka, umožňující snadnou úpravu nastavení či spouštění a ukončení hry.

Obrázek 8: Menu hry a nastavení hry

Dalšími okny je vyskakovací okno pro zobrazení příběhového textu a okno pro zobrazení nápovědy. Posledním je okno zobrazující herní ukazatele jako je zdraví hráče, výkon motoru a stav štítu.

Toto poslední okno je aktualizováno při každém vykreslování snímku a tím má hráč velmi rychlou odezvu na své akce.

Obrázek 9: Ukazatele stavu lodě

V případě, kdy ve hře hráč stiskne tlačítko či provede nějakou jinou akci, kterou Gui neumí obsloužit, je informace o tomto zaslána zprávou pomocí systému, který nabízí samotný engine Delta3D. Každá zpráva v tomto systému je identifikována jménem a nese ukazatel na objekt typu `void*`, jímž je možné předat libovolná data. Níže jsou ukázány typy zpráv zasílaných z Gui a příklad zaslání zprávy.

```
enum ButtonEvent
{
   NONE,             //Žádná nebo nedefinovaná akce
   NEW_GAME,         //Hráč zmáčkl tlačítko nové hry
   SETTINGS_OK,      //Hráč potvrdil nastavení ve hře
   STORY_ADVANCE,    //Hráč zavřel okno s příběhovým textem
};
...
//Vytvoření objektu s informacemi, který zpráva ponese
Event * event = new Event(Gui::ButtonEvent::NEW_GAME);
//Zaslání zprávy s názvem "GUI" všem objektům, které jsou k objektu Gui zaregistrováni
//jako posluchači
SendMessage("GUI", event);
```

Hlavní menu bývá ve hrách implementováno různými způsoby. Od jednoduchých obrazovek s několika tlačítky a obrázkem na pozadí po komplexní prostorové menu s různými vizuálními efekty. Pro tuto hru byl vybrán kompromis spojující co nejjednodušší ovládání spolu s prostorovým grafickým pozadím, aby byl hráč nalákán na hraní hry, jak je ukázáno na následujícím obrázku.

Obrázek 10: Celá obrazovka menu hry.

Téměř celý prostor hry vyplňuje vesmírná loď na hvězdném pozadí, kterou ve hře hráč ovládá. Tato loď se na obrazovce lehce pohybuje po horizontální a vertikální ose a také mírně rotuje, čímž je imitováno, že loď letí. Zároveň kolem lodi prolétávají bílé částicové efekty představující drobné vesmírné smetí, které tento efekt spolu s efektem proudící energie z motoru ještě umocňují. Mírně rotuje také kamera, čímž je docíleno pohybujícího se hvězdného pozadí. Tím je hráči předem ukázáno, co může od hry čekat a motivuje ho to ji hrát.

Aby byl tento pohyb plynulý a nepůsobil rušivě, nestačí pouze jednoduché lineární posouvání. Lepšího efektu je dosaženo posouváním podle funkce sinus či kosinus. Parametrem této funkce je aktuální čas simulace.

Dále je loď také posouvána pohybem kurzoru po obrazovce, což zvyšuje interaktivitu celého menu se snahou hráče u hry ještě více udržet. Ukázka výpočtu tohoto posuvu je vidět níže.

```
//Proměnné posX a posY obsahují polohu myši v okně v intervalu od 0.0 do 1.0.
//Proměnná time obsahuje aktuální čas.
  //Nastavení pozice lodě
  position.SetTranslation(  posX / 5 + sin(time * 0.5) / 2,
                            0,
                            posY / 5 + cos(time * 0.4) / 2 );
  //Nastavení rotace lodě
  position.SetRotation(sin(time * 0.2), cos(time * 0.3), 0);
  //Nastavení pozice a rotace
  object->SetTransform(position);
```

4.4.4 Ozvučení

O ozvučení ve hře se stará třída `Sound`. Tato třída je implementována podle návrhového vzoru Singleton. To znamená, že třída Sound má pouze jednu instanci objektu a poskytuje globální možnosti přístupu k této instanci[14]. Tento návrhový vzor byl zvolen z důvodu potřeby k této třídě přistupovat z mnoha míst ve hře, kdy zároveň stačí jediná instance tohoto objektu pro možnost přehrávání zvuků. Instance této třídy je přístupná pouze přes metodu `static Sound * Sound::GetInstance()`. V této metodě je také vytvořena instance třídy při jejím prvním zavolání. Jedná se o tzv. odloženou inicializaci. V případě, že by ve hře nebyl přehrán žádný zvuk, nebude instance třídy nikdy vytvořena. Veškeré použité zvuky jsou při konstrukci instance třídy Sound nahrány z disku a drženy v privátních proměnných. Tím jsou tyto zvuky enginem Delta3D drženy v paměti a není potřeba je při každém přehrání znovu nahrávat. Toto by bylo především u častých efektů jako je střelba velmi nežádoucí a mohlo kvůli častým přístupům na disk způsobovat zpomalení celé hry.

Přehrání zvukového efektu pak probíhá zavoláním příslušné metody z této třídy. Každý zvuk je přehráván svou unikátní metodou.

```
const char* Sound::LaserShotPath  = "gamedata/Sounds/laser.wav";

void Sound::PlayLaserShot()
{
   //Vytvoření zvuku
   dtAudio::Sound* sound = dtAudio::AudioManager::GetInstance().NewSound();
   //Nahrání zvukového souboru (proběhne z mezipaměti)
   sound->LoadFile(LaserShotPath);
   //Nastavení akce vykonané po dohrání zvuku
   sound->SetStopCallback(OnSoundStopped, m_Instance);
   //Spuštění samotného přehrávání
   sound->Play();
}
```

Po dohrání zvuku je enginem zavolána funkce `OnSoundStopped`, ve které dojde k uvolnění prostředků potřebných během přehrání samotného zvuku a které již nejsou potřeba.

K uvolnění zvuků, které jsou nahrány v mezipaměti samotné instance třídy Sound a dalších prostředků, které třída využívá, dochází při zavolání destruktoru na instanci této třídy. Tento destruktor je zavolán až při ukončování hry a je důležité volat ho až v případě, kdy je jisté, že již nebude k této třídě znovu přistupováno. Pokud by totiž byla zavolána metoda GetInstance() po zavolání destruktoru, došlo by k opětovnému vytvoření instance této třídy.

4.4.5 Herní fyzika

Veškerý pohyb objektů ve hře je řízen fyzikálním enginem uvnitř Delty3D. Pro chování, které odpovídá hře z prostředí vesmíru je potřeba tento engine správně nastavit. Nejprve je potřeba správně nastavit vlastnosti samotného prostředí, ve kterém se objekty pohybují. To znamená především vypnout ve scéně gravitaci působící na objekty, která je v základním nastavení zapnutá. Následně při nahrávání scény vytvořené předem v editoru STAGE se pro každý objekt, který se má ve scéně pohybovat podle zákonitostí fyziky, musí fyzika povolit.

```
object->EnableDynamics(true);
```

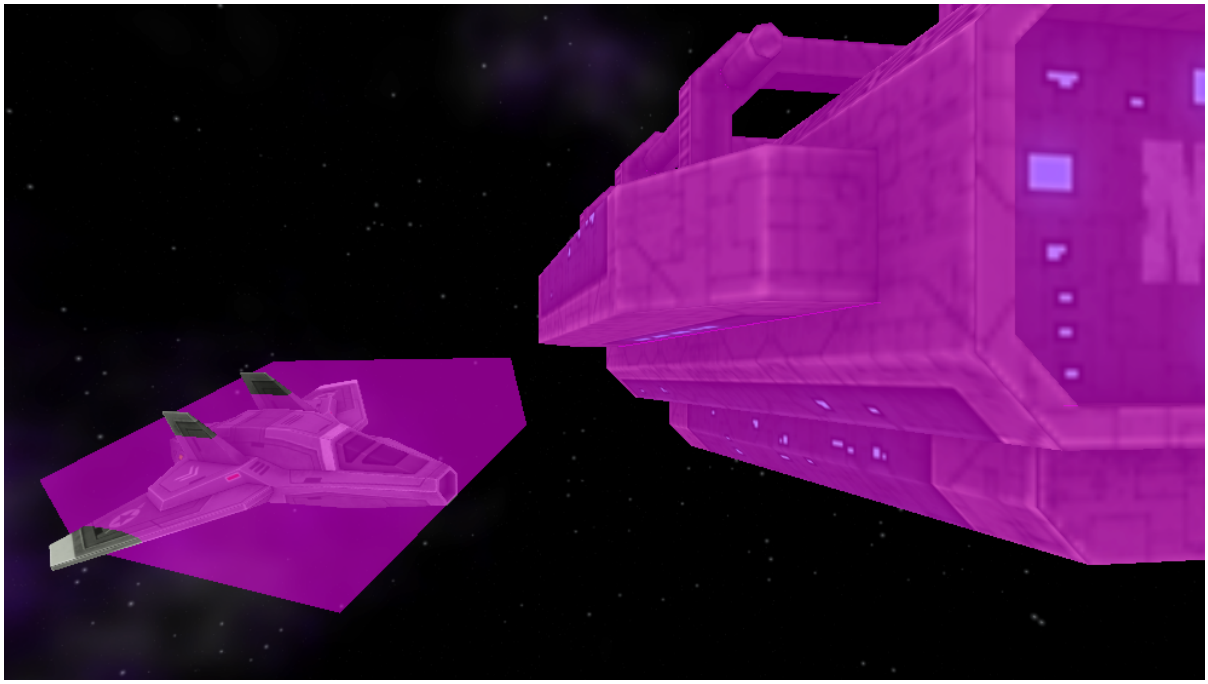
Fyzika lze pro objekty povolit již v editoru STAGE. Pro případ vesmírné hry však nešlo tento postup použít, protože v prostředí editoru STAGE nejde vypnout gravitace a objekty ihned po povolení fyziky začnou padat vesmírem dolů.

Protože se pohybujeme ve vesmíru, kde na objekty nepůsobí téměř žádné tlumivé síly jako je odpor prostředí, dochází při působení vnější síly na objekt k jeho trvalému pohybu či rotaci. Ač je toto chování fyzikálně správné, v prostředí her tohoto typu je dle zkušeností lepší, když se objekt chová, jako kdyby na něj odpor prostředí alespoň trochu působil. Tím je loď hráče částečně stabilizována a v případě nekontrolovatelné rotace po srážce s jiným objektem stačí přestat loď ovládat a ona se sama uklidní. Toto chování se nastavuje přiřazením příslušného směrového a rotačního tlumení na samotném objektu.

```
object->GetBodyWrapper()->SetLinearDamping(0.01f);
object->GetBodyWrapper()->SetAngularDamping(0.04f);
```

4.4.6 Kolize objektů

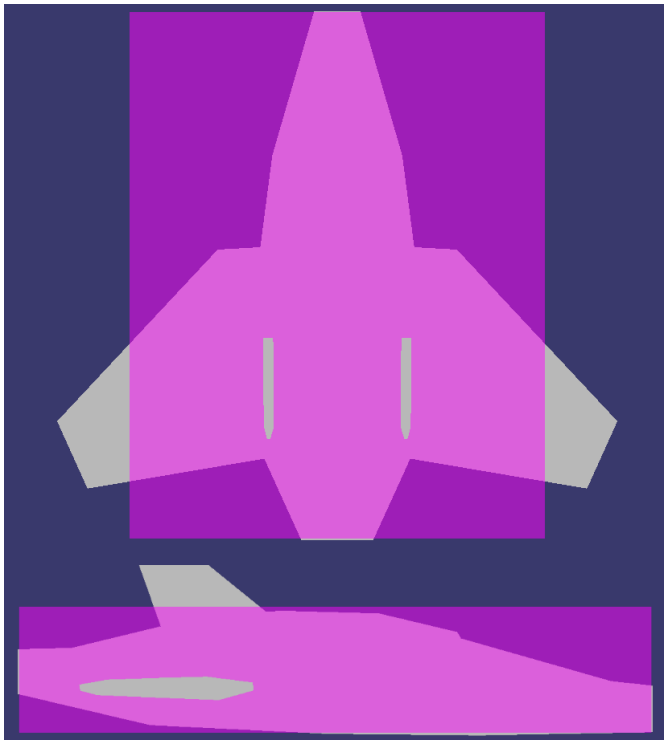
Open Dynamics Engine řeší v základu kolize sám. Aby mohlo ke kolizi nastat, musí mít objekt přiřazen kolizní model. Tento kolizní model může odpovídat přesně modelu objektu ve hře, avšak z důvodu vysoké náročnosti výpočtu kolizí u velmi detailních objektů je často zjednodušován. Na obrázku číslo 11 vidíme dva různé objekty, objekt lodě hráče a objekt základny. Model, který je ve hře vyobrazen má standardní texturu a barvy, zatímco kolizní model je v růžové barvě.



Obrázek 11: Ukázka dvou různých kolizních modelů.

Základna má různé výstupky a je poměrně členitá, proto je u ní vhodné aby byl kolizní model věrný modelu ve hře. Zde odpovídá 1:1 což vzhledem k nízké složitosti samotného modelu nečiní z hlediska výkonu veliký problém. Model lodi hráče oproti tomu má jako kolizní model nastaven

jednoduchý kvádr, který téměř vůbec tvar lodi nekopíruje. Při samotném hraní si však hráč tohoto problému téměř nevšimne. To proto, že tento kvádr je na loď velmi přesně usazen a končí zhruba na stejném místě jako loď (viz obrázek 12).



Obrázek 12: Detail posazení kolizního modelu na loď.

Když fyzikální engine detekuje, že by mohly dva objekty spolu kolidovat, zavolá funkci řešící tuto kolizi. Tato funkce spočítá body, ve kterých dané objekty kolidují a pokud opravdu kolidují, tak poté vypočítá kontaktní spoj těchto dvou objektů a zašle hře zprávu o detekované kolizi. Bohužel toto řešení není pro potřeby hry dostatečné. Problém nastává v případě, kdy hráč střílí po nějakém objektu a střela do tohoto objektu narazí. Při nárazu je sice střela ihned vymazána ze scény, ale díky již vygenerovanému kontaktnímu spoji trefený objekt i tak dostane impulz síly, jako kdyby se se střelou srazil a začne rotovat a pohybovat se v prostoru.

Naštěstí ODE umožňuje nadefinovat vlastní funkci, která tento problém řeší. Její definice je uvedena níže.

```
void NearCollisionCallback (void *data, dGeomID o1, dGeomID o2);
```

Tato funkce se chová podobně jako originální funkce a také z ní vychází. V parametrech do funkce přichází objekty, u kterých se testuje kolize (o1 a o2) a ukazatel na objekt třídy Game. Při detekci kolize je ověřeno o jaké objekty se jedná a v případě, že se jedná o střely, tak nedojde k vygenerování kontaktu. Tím se na objekt zasažený střelou neaplikuje žádná síla a ten zůstává nehybný.

Další změnou je odpadnutí potřeby zasílat tuto kolizi jako zprávu, protože se na objektu třídy Game mohou přímo zavolat metody oznamující kolizi. První z těchto metod je metoda `void Game::ProcessShot(dtCore::Transformable* shot, dtCore::Transformable* target).`

Tato metoda má na starosti vytváření efektu zásahu laserem a udělení poškození zasaženému objektu. Druhou je metoda `void Game::ProcessCollision(dtCore::Scene::CollisionData cd)`. Ta je volána v případě jakékoliv jiné kolize než se střelou laseru. Obstarává vykonání akce, pokud hráč vlétl do triggeru spouštějícího nějakou událost, či poškození hráče pokud narazil do pevného objektu. Velikost poškození je poté přímo úměrná rychlosti, jakou hráč letí.

Poslední důležitou změnou v této funkci je úprava parametrů samotného kolizního spoje. V něm je snížen koeficient odrazivosti a také podstatně snížen koeficient tření. Tím je chování objektů v beztížném stavu vůči sobě mnohem realističtější.

4.4.7 Částicové efekty a střelba

Částicové efekty

Vytváření nových částicových efektů je ve hře možné dvěma způsoby. První a jednoduší z nich je vložení částicového efektu přímo do scény přes editor STAGE, kde můžeme efektu nadefinovat jeho polohu a rotaci. Tento způsob je využit například na efekt motoru lodě. Druhým způsobem vytváření efektů je s pomocí třídy `dtCore::EffectManager`. Tato třída poskytuje jednoduché rozhraní, ve kterém se do EffectManageru nejprve nahraje a namapuje částicový efekt podle jména. V momentě, kdy chceme samotný efekt vytvořit, zavoláme metodu pro přidání efektu s daným jménem do scény na danou pozici. Při tomto přidávání je důležitý parametr `timeToLive`, kterým se definuje doba, po kterou má být efekt ve scéně. Tento parametr by neměl být kratší než je doba trvání samotného efektu, protože poté by došlo ke zmizení efektu ještě před jeho skončením. Tento systém je použit pro efekt odletujících jisker při zásahu nějakého objektu laserem a pro efekt výbuchu hráče.

Částicové efekty je možné během jejich existence ovládat. Toho je využíváno pro ovládání intenzity efektu motoru lodi. Prvním parametrem, jenž je ve hře měněn, je doba života vygenerovaných částic. Ta se nastavuje v objektu třídy `osgParticle::Particle` a mění se v závislosti na aktuálním výkonu motoru od nejdelší doby při plném výkonu po nejkratší dobu při nulovém či záporném výkonu (při couvání). Protože při nastavení doby života částice na nulu existuje částice napořád a i při velmi krátké době života částice dochází k jejímu problikávání, je potřeba při vypnutém motoru úplně vypnout generátor částic. Toto se nastavuje v objektu třídy `osgParticle::RandomRateCounter` jenž řídí generování částic, a který je dostupný skrz rozhraní objektu s částicovým efektem. Ukázka kódu ovládajícího toto generování je vidět níže.

```
//Získání vrstvy částicového efektu která generuje efekt motoru
dtCore::ParticleLayer& particleLayer = *EngineParticle->GetSingleLayer("Layer 0");
//Nastavení délky života částice podle výkonu motoru (proměnná EnginePower)
particleLayer.GetParticleSystem().getDefaultParticleTemplate()
       .setLifeTime(std::max(0.0001, (EnginePower) / 3));
//Získání objektu řídícího generování částic
osgParticle::RandomRateCounter *counter
     = static_cast<osgParticle::RandomRateCounter *>(particleLayer.GetModularEmitter()
                                                                  .getCounter());
//Vypnutí generovaní při nulovém výkonu
if(EnginePower == 0.0)
       counter->setRateRange(0, 0);
else
       counter->setRateRange(50, 50);
```

Střelba

Střílení laserů ve hře obstarává třída `Shooter`. Instance této třídy je vytvořena v jádru hry, objektu třídy Game. Tato třída plní dva hlavní účely. Střílení laserů a jejich následné zničení v případě, že již nejsou potřeba.

Lasery lze vystřelit dvěma způsoby. Prvním je zasláním zprávy přes systém enginu Delta3D. Data v této zprávě poté nesou objekt třídy `dtCore::Transform` obsahující pozici a směr odkud má být laser vystřelen. Druhým způsobem pak je samotné zavolání metody `ShootLaser`, které je tento objekt předán jako parametr.

Samotné vystřelení se skládá z několika kroků. Nejprve je potřeba nahrát model laserové střely. Tento model je poté umístěn do scény a předsazen před pozici, která přišla parametrem. To proto, aby nebyl vytvořen uvnitř modelu a ihned se s ním nesrazil. Následně je střele přiřazen kolizní tvar, což je kvádr, který ji obaluje. Poté je střele zapnuta fyzika a je na ní aplikována síla v dopředném směru. Tím je střela vystřelena.

Aby střely nelétaly do nekonečna, jsou při vystřelení vkládány do jednoduchého listu spolu s časem vystřelení. Tento list je poté periodicky kontrolován a pokud střela letí již určitou dobu a s ničím se nesrazila, je předpoklad, že se již nachází mimo oblast obsahující herní prvky a je ze scény odebrána a zničena.

Sestřelení objektu

Aby mohl objekt ve scéně být sestřelen, je potřeba držet k němu informaci o hodnotě jeho zdraví. Tato informace je ke každému zničitelnému objektu držena ve třídě `Destructible`. Při nahrávání scény jsou všechny objekty v ní zkontrolovány a pokud jsou určeny jako zničitelné, je reference na ně spolu s hodnotou jejich zdraví uložena do této třídy. Pokud jádro hry `Game` detekuje kolizi laserové střely s nějakým objektem je zaslána této třídě zpráva spolu s objektem, který byl zasažen. Třída poté u daného objektu sníží hodnotu jeho zdraví a v případě že je nulová, odebere objekt ze scény a na jeho místo vytvoří efekt exploze.

4.4.8 Ukládání a nahrávání stavu a nastavení hry

Ukládání a nahrávání je ve hrách velmi podstatný prvek. Některé hry umožňují uložit kompletní úroveň, s rozmístěním veškerých objektů, nehratelných postav a dalších věcí. Jiné zase poskytují pouze systém záchytných bodů, na které se hráč při pokračování vrací. Co a jak detailně hra ukládá je často také dáno rozhodnutím už při tvorbě konceptu samotné hry. V případě, kdy hráč nemá možnost ukládat si hru v libovolný okamžik, tedy například v momentě kdy očekává, že by se mu mohlo něco stát, prožívá tento moment mnohem intenzivněji.

Ve hře je zvolen systém ukládání pomocí záchytných bodů. K uložení hry dochází v okamžiku, kdy ji hráč ukončuje. Při opětovném spuštění hry je poté ověřeno, zda je ve hře dostupná uložená pozice a nabídnuto pokračování z ní. Ukládáno je zdraví hráče a jeho pozice a poté dokončené záchytné body. Při nahrávání uložené pozice je nahrána scéna se všemi objekty. Ty jsou poté nastaveny jednotlivě tak, aby odpovídaly hráčovu postupu hrou.

Samotné ukládání pozice ve hře a také ukládání nastavení hry je řešeno tvorbou textového dokumentu, který obsahuje tyto informace uložené ve formátu JSON. Pro vytváření a čtení dokumentů ve

tvaru JSON je v projektu využita dodatečná knihovna JsonCpp[17]. Uložená pozice a nastavení hry mají každý svůj soubor.

4.4.9 Vyprávění příběhu

Řízení vyprávění příběhu obstarává ve hře třída `Level`. Příběh je rozdělen na jednotlivé malé segmenty a tato třída poté řeší, které segmenty jsou již dokončeny a které mají být následně provedeny. Vyprávění příběhu je ve hře řešeno zobrazením dialogového okna, ve kterém je napsáno co se právě děje. Označit segment za dokončený a posunout příběh kupředu je možné dvěma způsoby. Prvním z nich je zavolání metody `void Level::Advance()`. Tato metoda slouží pro manuální posun a nejčastěji je volána, když uživatel odsouhlasí okno s dialogem. Druhým způsobem je nastavení časové prodlevy. Tento způsob je vhodný například v případě, kdy vysvětlíme hráči ovládání aplikace a následně mu necháme před zobrazením dalšího dialogu chvilku na vyzkoušení.

Během zobrazení dialogu s příběhovým textem hráč nemůže ovládat loď, protože se čeká na potvrzení tohoto dialogu. Z tohoto důvodu je celý běh hry při zobrazení tohoto dialogu zpomalen na 10% normálního času. Tím je hráči poskytnuto dostatečné množství času na přečtení dialogu, aniž by hrozilo, že nekontrolovaně do něčeho narazí. Díky tomu, že hra není úplně zastavena, nenastává u hráče pocit vypadnutí ze hry.

4.4.10 Nepřátelé

Hlavním nepřítelem ve hře, kromě prostředí samotného, jsou laserové střílny. Tyto střílny brání základu u níž hráč začíná hru, v průběhu hry jsou napadeny virem a obrátí se proti němu.

Chování věží je ovládáno třídou `Tower`. Věž má tři stavy ve kterých se může nacházet. Základní stav je stav nečinnosti. Ten nastává když věž nemá vybraného nepřítele, popřípadě je nepřítel mimo její dosah. Druhým je útočný stav, ve kterém věž střílí lasery na nepřítele. Pohyb nepřítele je přitom věží sledován a ta se za ním otáčí. Poslední stav nastává když je věž zničena. Tehdy přestává být ve scéně vidět a nemůže ji dále ani žádným způsobem ovlivnit.

U těchto věží je důležité, přesouvat ve scéně na jejich pozice veškeré zvuky, které vydávají. V opačném případě by zvuk střelby i zvuk vybuchující věže zněl stejně, jako kdyby pocházel z lodi hráče.

5 Závěr

Engine Delta3D umožňuje vytvářet plnohodnotné hry či simulační aplikace. Díky otevřenému zdrojovému kódu je možné upravit si ho přesně na potřeby daného projektu. Ačkoliv není komunita kolem tohoto enginu tolik aktivní, jako bývala v dřívějších letech, je engine stále aktualizován a to i díky své modularitě a přejímání aktualizovaných knihoven. Výhodou pro uživatele enginu je díky možnosti přímého přístupu k těmto knihovnám předchozí zkušenost s nimi. V tomto ohledu mi chyběla především zkušenost s OpenSceneGraph, Open Dynamics Enginem a CEGUI, jejichž funkcionalitu jsem si musel během vytváření aplikace nastudovat.

Engine poskytuje kromě značně stručné dokumentace samotných zdrojových kódů také několik projektů s příklady. Je však škoda, že neposkytuje těchto příkladů více. Nejlepším přístupem by byl krátký ukázkový kód ke složitějším nebo důležitým metodám přímo v dokumentaci, jako to má například společnost Microsoft ve své Developer Network (MSDN). Projekt Delty3D však na toto má příliš malou základnu aktivních členů.

V této práci byla podrobně prozkoumána možnost použití tohoto enginu pro vývoj her. Byla prozkoumána simulace fyziky a vykreslování různých grafických efektů pomocí enginu Delta3D a i dalších náležitostí s vývojem her souvisejících.

Dle mého názoru je graficky je engine na vysoké úrovni a díky projektu OSG podporuje nejnovější technologie. Velké komerční enginy nabízejí často pokročilejší efekty, mají však za sebou velká herní studia se stovkami zaměstnanců, zatímco do repozitáře Delty3D za poslední rok aktivně přispívají pouze tři autoři. Se stejnou základnou by z OSG, respektive Delty3D, bylo zajisté také možné získat porovnatelný výstup.

Co se fyziky týče, jsou zde možnosti opravdu široké. Už základní modul Open Dynamics Engine poskytuje dostatečně kvalitní a rychlou simulaci fyziky pro většinu potřeb dnešních her. V případě požadavku po pokročilejším fyzikálním enginu je možné využít projekty jako Bullet či PhysX.

Při vývoji další aplikace v enginu Delta3D bych zajisté použil aktory, díky kterým je možné vytvářet vlastní typy objektů použitelných v editoru STAGE. Pro případné rozšíření projektu by bylo možné uvažovat nad implementací pokročilejších typů nepřátel a především v navýšení objemu samotného obsahu hry. Také by bylo možno doplnit hru o další grafické efekty či rozšířit její zvukovou stránku.

6 Literatura

[1] UNITY TECHNOLOGIES. Unity - Game Engine [online]. 2014 [cit. 2014-05-14]. Dostupné z: https://unity3d.com/

[2] EPIC GAMES, Inc. Unreal Engine 4 [online]. 2014 [cit. 2014-05-14]. Dostupné z: https://www.unrealengine.com/blog/welcome-to-unreal-engine-4

[3] CRYTEK GMBH. Cryengine [online]. 2014 [cit. 2014-05-14]. Dostupné z: http://www.cryengine.com/

[4] DELTA3D. Delta3D Simulation and Gaming Engine [online]. 2014 [cit. 2014-05-14]. Dostupné z: http://delta3d.org/

[5] DELTA3D. Delta3D partners [online]. 2014 [cit. 2014-05-14]. Dostupné z: http://delta3d.org/article.php?story=20041110112606610

[6] MCDOWELL, P., R. DARKEN, J. SULLIVAN a E. JOHNSON. Delta3D: A Complete Open Source Game and Simulation Engine for Building Military Training Systems. The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology. 2006-07-01, vol. 3, issue 3, s. 143-154. DOI: 10.1177/154851290600300302. Dostupné z: http://dms.sagepub.com/cgi/doi/10.1177/154851290600300302

[7] Delta3D: Features. Delta3D [online]. 2014 [cit. 2014-05-23]. Dostupné z: http://delta3d.org/article.php?story=20051209133127695&topic=docs

[8] Features. OpenSceneGraph Project Website [online]. 2014 [cit. 2014-05-15]. Dostupné z: http://www.openscenegraph.org/index.php/about/features

[9] ODE Wiki: Manual. In: ODE Wiki [online]. 2012, 10.5.2012 [cit. 2014-05-16]. Dostupné z: http://ode-wiki.org/wiki/index.php?title=Manual

[10] Unreal Engine 4 Documentation: Effects. In: Unreal Engine Technology [online]. 2014 [cit. 2014-05-15]. Dostupné z: https://docs.unrealengine.com/latest/INT/Resources/ContentExamples/EffectsGallery/index.html

[11] WANG, Rui a Xuelei QIAN. OpenSceneGraph 3.0: Beginner's guide : Create high-performance virtual reality applications with OpenSceneGraph, one of the best 3D graphics engines. 1st pub. Mumbai: Packt Publishing, 2010, 385 s. ISBN 978-184-9512-824.

[12] Welcome to CEGUI: Homepage. In: Crazy Eddie's GUI System [online]. 2014 [cit. 2014-05-16]. Dostupné z: http://cegui.org.uk/

[13] ADAMS, Ernest a Andrew ROLLINGS. Fundamentals of game design. Upper Saddle River, N.J.: Pearson Prentice Hall, 2007, s. 416-432. Game design and development. ISBN 0131687476.

[14] ALEXANDRESCU, Andrei. Modern C design: Generic programming and design patterns applied. Vyd. 1. Boston: Addison-Wesley, 2001, s. 119-144. ISBN 0-201-70431-5

[15] TurboSquid: Licensing & Usage. TURBOSQUID. TurboSquid: 3D Models for Professionals [online]. 2014. vyd. 2014 [cit. 2014-05-15].
Dostupné z: http://support.turbosquid.com/entries/31030006-Royalty-Free-License

[16] Manual: Concepts - Integration. In: ODE Wiki [online]. [cit. 2014-05-20].
Dostupné z: http://ode-wiki.org/wiki/index.php?title=Manual:_Concepts#Integration

[17] JsonCpp: Documentation. [online]. 2014 [cit. 2014-05-20].
Dostupné z: http://jsoncpp.sourceforge.net/

Seznam příloh

Příloha 1. Plakát k aplikaci
Příloha 2. CD obsahující:

- Zdrojové kódy aplikace
- Spustitelnou verzi aplikace
- Manuál ke zprovoznění aplikace
- Zdrojový text této práce ve formátu ODT
- PDF dokument této práce
- Zdrojový soubor pro plakát