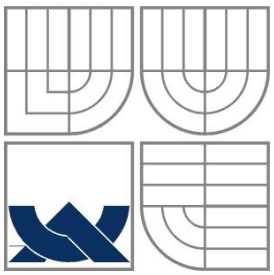
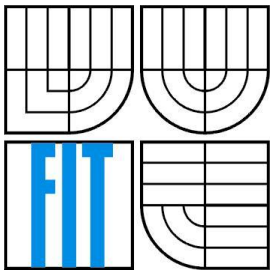


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MULTIPLATFORMNÍ HRA POSTAVENÁ NA XNA

MULTI-PLATFORM GAME BUILT ON XNA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MAREK CHRAMOSIL

VEDOUCÍ PRÁCE
SUPERVISOR

ING. MICHAL ZACHARIÁŠ

BRNO 2012

Abstrakt

Tato bakalářská práce popisuje vytvoření multiplatformní vesmírné 3D hry pro platformy PC a Xbox 360 pomocí XNA Frameworku 4.0. Nejprve představuje technologie použité při vývoji této hry, včetně hardware a schopností konzole Xbox 360, následně se věnuje návrhu, implementaci a optimalizaci této hry. V poslední části popisuje prerekvizity a rozdíly vývoje pro Xbox 360, včetně optimalizací specifických pouze pro tuto konzoli, a nastiňuje možnosti portace této hry pro platformu Windows Phone.

Abstract

This bachelor's thesis describes creation of multi-platform 3D space game written in XNA Framework 4.0 and running on PC and Xbox 360. At first it introduces technologies used in the game, including hardware and capabilities of the Xbox 360 game console. Subsequently it covers design, implementation and optimization of said game. In the last part it describes prerequisities and differences of Xbox 360 development, including console specific optimizations. It also outlines posibility of porting the game to the Windows Phone platform.

Klíčová slova

XNA Framework 4.0, multiplatformní hra, vesmírná hra, Netwars, Netwars Reloaded, Xbox 360, 3D hra, C#

Keywords

XNA Framework 4.0, multi-platform game, space game, Netwars, Netwars Reloaded, Xbox 360, 3D game, C#

Citace

Chramosil Marek: Multiplatformní hra postavená na XNA, bakalářská práce, Brno, FIT VUT v Brně, 2012

Multiplatformní hra postavená na XNA

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Michala Zachariáše. Další informace mi poskytl Ing. Rudolf Kajan. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Marek Chramosil
15. května 2012

Poděkování

Rád bych poděkoval Ing. Michalovi Zachariášovi za ochotné rady a konzultace ohledně této bakalářské práce.

Obsah

1 Úvod

Téměř od začátku počítačového věku používá člověk tento zajímavý stroj kromě práce také ke své zábavě. Co začalo primitivními hrami jako Tennis for Two, které místo displeje používaly osciloskop, dnes pokračuje závody o dosažení fotorealistické grafiky a virtuální reality. Do počítačových her se dostávají nové způsoby ovládání pohybem či myšlenkami a mobilní telefony dosahují výkonu, jaký měly ještě nedávno stolní počítače. Toto dynamické odvětví zábavy se mění tak rychle, že je téměř nemožné předvídat, jak bude vypadat za deset let. Protože ale prudce roste počet zařízení, na kterých je možné hrát hry, vzniká také potřeba existující hry snadno upravit pro spouštění na stále nových strojích a existuje několik názorových proudů a možností, jak takovou přenositelnost zajistit. Od použití internetových standardů a jazyků, které hru zprovozní ve webovém prohlížeči, přes knihovny jako je OpenGL a XNA, až po hotové multiplatformní herní enginy.

Tato bakalářská práce se věnuje knihovně XNA od firmy Microsoft, která je poskytována zcela zdarma a v kombinaci s vývojovým prostředím Visual Studio Express poskytuje jednoduchý a zábavný způsob jak začít vyvíjet multiplatformní počítačové hry. Jejím cílem je implementovat multiplatformní hru běžící na konzoli Xbox 360 a popsat všechna úskalí a specifika, se kterými je nutné se v takovém projektu vypořádat. Kdekoliv bude v dalším textu zmíněn termín „Xbox“, je tímto myšlena herní konzole Xbox 360, ne její předchůdce.

Jednotlivé kapitoly práce se nejprve věnují seznámení použitými technologiemi (včetně Xboxu) a obecnými principy vývoje her. Následně pokrývají implementaci vyvíjené hry, rozdíly mezi PC a Xboxem včetně popisu optimalizace hry pro něj, výkonové testy a popis nutných úprav při případné portaci hry na platformu Windows Phone.

2 Použité technologie

2.1 .NET Framework

.NET Framework byl původně kombinací běhového prostředí CLR (Common language runtime), knihovny tříd BCL (Base class library) a několika dalších knihoven od firmy Microsoft. Ty zajišťovaly vývoj grafického uživatelského rozhraní (WinForms), webových aplikací (ASP.NET) a přístup k datům (ADO.NET). Později se k těmto technologiím ve přidaly například:

- LINQ
- Entity framework
- Windows Communication Foundation
- Windows Presentation Foundation
- Windows Workflow Foundation
- Windows CardSpace

Verze 4 potom přinesla novinky v oblasti paralelního programování a paralelního dotazování nad daty (Parallel LINQ). V současné době je k dispozici developer preview verze .NET Frameworku 4.5, která má přinést podporu vývoje pro grafické uživatelské rozhraní Metro ve Windows 8, lepší podporu asynchronního a paralelního programování a mnoho dalších novinek.
Pro .NET Framework se dá vyvíjet v několika různých jazycích(C#, Visual Basic.NET, Visual C++ a další), které jsou následně přeloženy do mezikódu CIL (Common intermediate language) a jednotně zpracovávány CLR. Kromě výše zmíněných typů aplikací umožňuje .NET Framework také vývoj XML webových služeb, Windows Phone aplikací a dalších typů softwaru, mj. také multiplatformních her pomocí technologie XNA. Současně se snaží se usnadňovat nasazení aplikací (deployment). Protože je .NET Framework relativně velká a komplexní knihovna (v posledních verzích jednotky GB), byly pro zařízení s omezenými zdroji vyvinuty dvě jeho zjednodušené verze. Compact Framework pro mobilní telefony, nebo například Xbox 360, a Micro Framework pro vestavěná zařízení s ještě menším výkonem a velikostí paměti.

2.1.1 CLR

CLR je běhové prostředí zajišťující exekuci kódu, správu paměti a vláken, meziprocesovou komunikaci a další nízkoúrovňové záležitosti. Stará se také o interoperabilitu s existujícím softwarem a knihovnami .DLL a interakci komponent, napsaných v různých jazycích, které splňují Common type system (standard definující mj. reprezentaci a zpracování datových typů v paměti).

2.1.2 BCL

BCL je knihovna znovupoužitelných tříd, které usnadňují programování aplikací implementací často používaných operací a komponent, jako je práce s řetězci, soubory, databázemi a XML, vykreslování obrazu, nebo vstupní a výstupní operace. Ve jmenném prostoru System.Collections také poskytuje velké množství datových kontejnerů a kolekcí jako jsou fronta, zásobník, lineární seznam, nebo hashtabulka.

2.1.3 Automatická správa paměti

Automatická správa paměti má za úkol odstranit úniky paměti a chyby vznikající nesprávným použitím ukazatelů. Také mírně zvyšuje výkon aplikace, protože při každém uvolnění nepoužívané paměti na řízené haldě (managed heap) se garbage collector postará o přeskládání objektů tak, aby ležely v paměti těsně za sebou. Tato „komprimace“ vede k rychlejšímu vytváření nových objektů, protože není nutné hledat dostatečně velké místo souvislé volné paměti, ale použije se ukazatel na konec haldy. Další výhoda je v rychlejším přístupu k už existujícím objektům, protože zmíněná souvislost dat omezí stránkování [1]. Podle verze .NET Frameworku je použitý buď vyspělejší generační garbage collector, nebo jeho jednodušší verze, blíže popsaná v kapitole 6.2.2.

2.2 Jazyk C#

C# je silně objektově orientovaný, vysokoúrovňový programovací jazyk vyvinutý společností Microsoft jako reakce na programovací jazyk Java. Je možné v něm programovat s využitím více programovacích paradigmat, například imperativního, objektově orientovaného, událostmi řízeného, funkcionálního, nebo generického. Od verze 3.0 také obsahuje LINQ (Language integrated query), což je vestavěná podpora unifikovaného dotazování nad různými zdroji dat – například SQL, XML, ale i kolekcemi v operační paměti.
Jeho syntaxe vychází z jazyků C++ a Java, a je dále zjednodušena. Jazyk byl od počátku navrhován pro maximální jednoduchost, bezpečnost (silné typování, kontrola mezí polí, automatická správa paměti garbage collectorem), rychlost vývoje, platformní nezávislost a univerzálnost, díky které je možné jej použít pro široké spektrum programů od velkých enterprise aplikací až po vestavěné systémy. Na druhou stranu, jeho primárním cílem nebylo rychlostí konkurovat jazykům jako je C, nebo jazyk symbolických instrukcí, takže není vhodný pro vysoce náročné výpočty. [2]

V jazyce C# se dá pracovat s pamětí dvěma způsoby. První je používat takzvaný řízený (managed) kód, kde se o veškeré dealokace paměti postará garbage collector. Druhý způsob, neřízený (unmanaged) kód, potom dovoluje programátorovi manuálně přistoupit do paměti pomocí ukazatelů stejně jako v jazycích C/C++. Toto se však až na dobře odůvodněné případy příliš nedoporučuje,

protože je to jedním z hlavních zdrojů chyb v programech, napsaných v nižších programovacích jazycích.
Jazyk C# se (stejně jako všechny ostatní jazyky .NET Frameworku) kompiluje do mezikódu CIL a následně společně s metadaty a dalšími zdroji (texty, ikony) ukládá do takzvaných kompletů (Assembly). Metadata jsou velmi důležitá například pro reflexi – zkoumání a modifikaci programu za jeho běhu. Při spuštění programu se poté přeloží celá aplikace, nebo její části (Just in time kompilace) do nativního kódu dané platformy a vykonají. Pokud je potřeba vyšší výkon, je také možné celou aplikaci pomocí nástroje Ngen přeložit do nativního kódu, nebo z C# pomocí Platform invoke volat funkce z .DLL knihoven, napsaných v neřízeném kódu. [3]

2.3 XNA

XNA je multiplatformní sada řízených knihoven pro vývoj počítačových her a sada procesů pro zpracování herního obsahu při kompilaci. Jeho cílem je usnadnit portaci her z jedné platformy na druhou a zrychlit vývoj her poskytnutím znovupoužitelných tříd pro potřeby společné většině počítačových her. Mezi takové potřeby patří například načítání 3D modelů a textur, vykreslování 2D obrázků (sprite) nebo matematické operace s vektory, maticemi a kvaterniony.

XNA podporuje vývoj pro následující platformy:

- PC
- Xbox 360
- Windows phone
- Zune / Zune HD
- Microsoft Surface

Základní třídou v XNA je třída Game, která zapouzdřuje celou hru. Na začátku hry se volají metody **Initialize()** a **LoadContent()**, jejichž úkolem je provést inicializaci hry a načíst veškerý herní obsah a grafiku. Poté startuje herní smyčka, reprezentovaná opakovaným voláním metod Update() a Draw(). V metodě **Update()** se zpracovává veškerá herní logika, kolize, zvuky a vstupy od uživatele. Metoda **Draw()** potom zajišťuje vykreslení obrazu. Obě tyto metody jsou (se zapnutou vertikální synchronizací) volány maximálně 60krát (30krát u Windows Phone, kvůli výdrži baterie) za sekundu, aby se grafická karta zbytečně nezatěžovala vykreslováním obrazu, který monitor nedokáže zobrazit.

2.3.1 Matice v XNA

Matice se v XNA používají k transformacím polohy 3D objektů. Typicky modely prochází postupně těmito prostory:

1. **Object/model space** - prostor, ve kterém jsou definované vertexy modelu.
2. **World space** - prostor, který zaujímá model ve 3D světě hry.
3. **View space** – world space transformovaný podle polohy a pohledu kamery.
4. **Screen space** – což je view space transformovaný na 2D zobrazovací zařízení, jako je monitor.

K těmto transformacím slouží 3 typy matic, které se dají vytvořit pomocí statických metod XNA třídy Matrix.

1. **World** – udává, jakou má vykreslovaný objekt polohu, orientaci a jak je zvětšený.
2. **View** – udává polohu a směr pohledu kamery.
3. **Projection** – udává, zda se scéna bude vykreslovat perspektivní nebo paralelní projekcí a vzdálenost near a far clip planes.

2.3.2 HLSL

HLSL, neboli High level shader language je vysokoúrovňový programovací jazyk pro psaní shaderů, který se používá v XNA. Vyvíjely ho společnosti Microsoft a Nvidia až do doby, kdy Nvidia od spolupráce odstoupila a začala vyvíjet svůj vlastní jazyk Cg. [4]
HLSL má podobnou syntaxi, jazykové konstrukce a datové typy jako jazyk C, ale přidává k nim několik dalších prvků. Z hlediska datových typů přibyly např. typy pro reprezentaci barvy pixelu s průhledností (float4) nebo bez (float3), nebo parametrizovatelný typ matice. Vstupní parametry i návratové hodnoty HLSL funkcí mají kromě datového typu definovanou také tzv. sémantiku, která udává, k čemu se daná hodnota bude používat. Možné typy sémantik jsou např. souřadnice na textuře, normála, nebo poloha vertexu. HLSL obsahuje 2 speciální typy funkcí:

1. **Vertex shader** – násobí polohu vertexu postupně world, view a projection maticemi, a tak transformuje vertex z object space do screen space.
2. **Pixel shader** – počítá barvu každého výsledného pixelu na zobrazovacím zařízení.

Uvedené funkce těchto shaderů jsou typické, ale jak uvidíme dále, mohou provádět i jiné operace podle naprogramování. Také je možné definovat více vykreslovacích technik, pomocí různých kombinací vertex a pixel shaderů, včetně opakovaného průchodu grafickou pipeline.

2.3.3 Effect

Effect je XNA třída, která zajišťuje obousměrnou komunikaci C# s HLSL kódem a volbu vykreslovací techniky. Protože platforma Windows Phone v současné době nepodporuje psaní vlastních HLSL shaderů, obsahuje XNA od verze 4.0 pět tříd zděděných od třídy Effect, používaných pro komunikaci s předprogramovanými shadery:

- **BasicEffect** – umožňuje mj. vykreslování otexturovaného modelu nasvíceného až třemi konfigurovatelnými světly.
- **SkinnedEffect**
- **EnvironmentMapEffect –** vykresluje na model odraz okolního prostředí.
- **DualTextureEffect** – umožňuje dvouvrstvý multitexturing.
- **AlphaTestEffect**

Tyto shadery byly navrženy a optimalizovány hlavně pro telefony s Windows Phone, ale dají se použít i na platformách PC a Xbox 360. [5]

2.3.4 XNA profily

Protože XNA funguje na mnoha platformách, jejichž hardware a výkonnost se výrazně liší, byly ve verzi 4.0 zavedeny tzv. profily. Profil definuje schopnosti grafického HW a možnosti použitelné při vývoji hry pod tímto profilem, nedefinuje však požadovaný výkon, takže se může stát, že hra na konkrétním zařízení, splňujícím profil, nebude fungovat dostatečně plynule, přestože na jiném ano. Profil tedy garantuje pouze spustitelnost, nikoliv plynulost. V současné době jsou definovány dva profily – HiDef a Reach. HiDef zahrnuje veškerou funkcionalitu Reach a dále ji rozšiřuje, ale podporují ho pouze nejvýkonnější zařízení. Reach naproti tomu podporuje širší škálu zařízení, ale s omezenou grafickou kvalitou. Srovnání některých vlastností těchto profilů uvádí tabulka 2.1.

	Reach	HiDef
Podporované platformy	Windows Phone, Xbox 360, Windows PC s alespoň DirectX 9 GPU	Xbox 360, Windows PC s alespoň DirectX 10 GPU.
Minimální Shader model grafické karty	Shader Model 2.0	Shader Model 3.0
Maximální rozměr textury	2048	4096
Maximální počet primitiv na jedno volání vykreslení	65 536	1 048 575
Podpora více render targetů	Ne	Ano

Tabulka 2.1 – Srovnání grafických profilů XNA, kompletní verze v AJ dostupná na [6]

2.4 Techniky vývoje her použité ve hře

2.4.1 Obecný princip počítačové 3D grafiky

Nejjednodušší počítačová 3D grafika pracuje s 3D „drátěnými“ modely objektů, které se skládají z úseček spojujících vrcholy (vertexy) v prostoru, reprezentujících tvar objektu, a s texturami, což jsou bitmapy nanášené na drátěný model tak, aby vizuálně připomínal svou reálnou předlohu.
Z těchto informací se procesem zvaným renderování vytvoří dvojrozměrná reprezentace 3D obrazu, kterou už dokáží zobrazit rastrová zobrazovací zařízení, jako například monitory.
Pokročilejší 3D grafika už bere v úvahu a počítá i další vlastnosti obrazu, jako je stínování, průhlednost, hloubka ostrosti, odrazy světla a další fyzikální jevy.

S 3D modely je možné provádět 4 základní geometrické transformace:

1. **Translace** – posunutí objektu v 3D prostoru o daný vektor.
2. **Rotace** – rotace kolem osy otáčení o daný úhel.
3. **Zvětšení** nebo zmenšení objektu.
4. **Zkosení** objektu o daný vektor, který udává míru zkosení ve směrech souřadnicových os.

Tyto transformace jsou postupně aplikovány na model násobením transformačními maticemi v pořadí rotace, translace, zvětšení.

2.4.2 Kvaternion

„Kvaterniony byly navrženy irským matematikem W.R. Hamiltonem v 19. století jako analogie komplexních čísel v prostoru. Kvaternion q je reprezentován čtveřicí

$$q = W + xi + yj + zk$$

kde w, x, y, z jsou reálná čísla a i, j, k jsou kvaternionové jednotky (i odpovídá komplexní jednotce).“ [7]

Kvaterniony se ve hrách používají k reprezentaci rotace objektu ve 3D prostoru, protože narozdíl od Eulerových úhlů při jejich použití nemůže vzniknout gimbal lock. Také je možné kvaterniony jednoduše interpolovat z jedné orientace do druhé a vytvořit tak plynulou animaci. Lineární interpolace kvaternionů je ve hře využito hned na několika místech.

2.4.3 Skybox

Skybox je technika vytváření vzdáleného pozadí, jako je například obloha, nebo v našem případě vesmír, v počítačových hrách. Principielně funguje tak, že je celý herní svět obklopen modelem krychle, na jejíž stěny se mapuje textura pozadí, což může být např. obloha, vesmír, nebo hory v pozadí. Při pohybu hráče (kamery) prostředím se potom skybox může posunovat společně s hráčem tak, aby vytvořil iluzi, že jeho obsah je ve velké vzdálenosti od pozorovatele, a proto se při pohybu zdánlivě nemění. Skybox se vykresluje jako první objekt ve scéně tak, aby byl na pozadí všech ostatních.

2.4.4 Bounding box

Bouding box je pomyslný geometrický objekt (krychle, koule – bounding sphere), obklopující 3D model, který se používá pro zjištění kolizí objektů ve scéně.

3D modely bývají většinou komplexní, skládající se z tisícovek a více trojúhelníků a přesný výpočet jejich kolizí 60krát za sekundu by byl příliš výpočetně náročný. Proto se pro aproximaci kolizí používá bounding box, který modely obklopí jednoduchým geometrickým objektem, a následně počítá kolize pouze těchto objektů.

Možnou nevýhodou tohoto přístupu je snížená přesnost kolizí, takže objekty se budou srážet už ve chvíli, kdy na sebe narazí jejich skyboxy, což může hráč vnímat jako nereálné a rušivé. V této konkrétní hře to ale nebude problém, protože bounding sphere bude reprezentovat energetické štíty lodí ve tvaru koule.

U složitějších modelů, jako je lidská postava, může být někdy potřebné detekovat nejen zda ke kolizi došlo, ale také na kterém místě modelu. Pro tyto účely se modely skládají z více dílčích modelů (dlaň, předloktí, trup...), z nichž každý má vlastní bounding box.

2.4.5 Billboard

Jako billboard se v počítačových hrách označuje otexturovaný čtverec, tvořený dvěma trojúhelníky a natočený kolmo k vektoru pohledu kamery. Tato technika šetří výpočetní zdroje při vykreslování velkého množství stejných objektů.

3 Herní konzole Xbox 360

Obr. 3.1 – Xbox 360 Slim se pohybovým ovládáním Kinect v popředí.

Xbox 360 je druhá generace herní konzole od společnosti Microsoft. Jako zobrazovací zařízení využívá televizi nebo monitor s připojením před HDMI, nebo komponentní kabel. Pro připojení periferií je k dispozici 5 USB 2.0 portů. Konzole podporuje současný lokální multiplayer až 4 hráčů, nebo hru přes internet.

Operační systém Xboxu nejsou Windows. Sice používá některé Windows API, ale na druhou stranu je ořezaný o části, které nejsou potřebné (hardware abstraction layer), nebo je Xbox implementuje jinak [8].

Od svého uvedení na trh v roce 2005 už konzole podstoupila několik modernizací a die-shrinků výpočetních jednotek z 90 na 45nm výrobní technologii, díky čemuž výrazně klesla její spotřeba elektrické energie. První verze se prodávala se zdrojem schopným dodávat 203 wattů a kvůli velké spotřebě energie a špatnému návrhu chlazení občas trpěla přehříváním a následným selháním, známým jako Red Ring of Death, které se projevovalo rozsvícením třech červených diod okolo vypínacího tlačítka [9]. Současná verze s kódovým jménem Corona, vydaná v roce 2011, už si vystačí se 115 watty a má CPU, GPU i eDRAM integrované v jednom pouzdře.

Vývoj her pro tuto konzoli je možný buď pomocí XNA po zakoupení App hub účtu za 99 $, nebo po zakoupení C++ development kitu XDK, který obsahuje i upravenou konzoli se zvýšeným množstvím operační paměti a emulátorem optických disků pro ladění her [10]. Cena tohoto kitu není oficiálně

zveřejněná, z neoficiálních informací na internetu se pohybuje okolo 10 tisíc $ [11] a vyžaduje podepsání dohody o mlčenlivosti s firmou Microsoft, což tuto možnost automaticky vyřazuje pro většinu nadšenců a nezávislých vývojářů.

3.1 Hardware konzole

Xbox 360 obsahuje tříjádrový procesor RISC architektury PowerPC, schopný zpracovávat dvě vlákna jedním jádrem (simultánní multithreading) a taktovaný na frekvenci 3,2 GHz. Kvůli jednoduchosti a ceně používá, narozdíl od moderních out-of-order procesorů, pouze méně výkonné in-order zpracování instrukcí. Každé hardwarové vlákno má k dispozici 128 VMX-128 registrů a všechna jádra sdílí 1MB L2 cache.
Výpočet grafických operací provádí grafický čip Ati Xenos. Ten se skládá ze 48 unifikovaných (schopných zpracovávat buď pixel, nebo vertex shadery) shader procesorů taktovaných na 500 MHz. Také používá 10MB rychlé vestavěné RAM (eDRAM) pro urychlení operací jako je antialiasing (vyhlazování hran), nebo alpha blending.
Celá konzole má k dispozici 512 MB GDDR3 RAM, která je sdílená mezi procesorem a grafickým čipem. Tato hodnota může být limitující pro velikost dat a textur u velmi náročných her.
Hry se načítají z DVD disků, nebo vestavěného pevného disku a konzole může také sloužit jako multimediální přehrávač s podporou mnoha hudebních a filmových formátů.

3.2 Ovládání

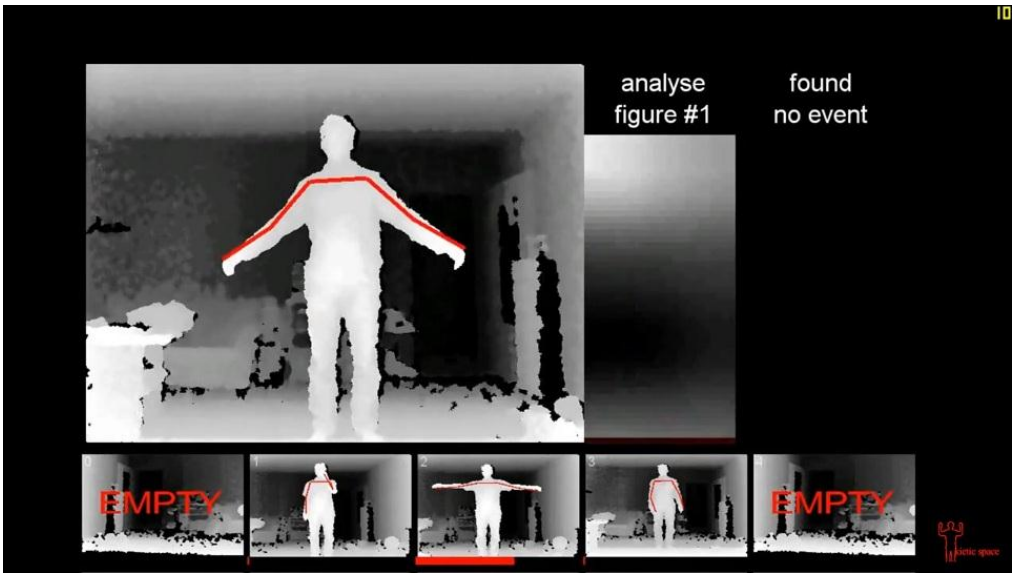
Hry na Xbox 360 se primárně ovládají bezdrátovým obouručním ovladačem (gamepadem) se dvěma analogovými joysticky, jedním směrovým ovladačem a omezeným počtem akčních tlačítek (8), což je nutné brát v úvahu při návrhu ovládání hry. Alternativně je možné připojit i klávesnici.

3.2.1 Kinect

Pro určité typy aplikací a her je také možné použít pohybové ovládání Kinect. To pracuje tak, že uživatel stojí před Kinectem, a pohyby rukou, celého těla, nebo hlasem může konzoli ovládat.
Kinect tvoří vestavěný systém na čipu PrimeSense PS1080, dvě kamery (RGB a infračervená) a infračervený projektor, pokrývající prostor před ním sítí specificky rozmístěných bodů (viz obr. 3.2). [12]

Obr. 3.2 – Obraz promítaný Kinectem a snímaný IR kamerou, převzato z [13]

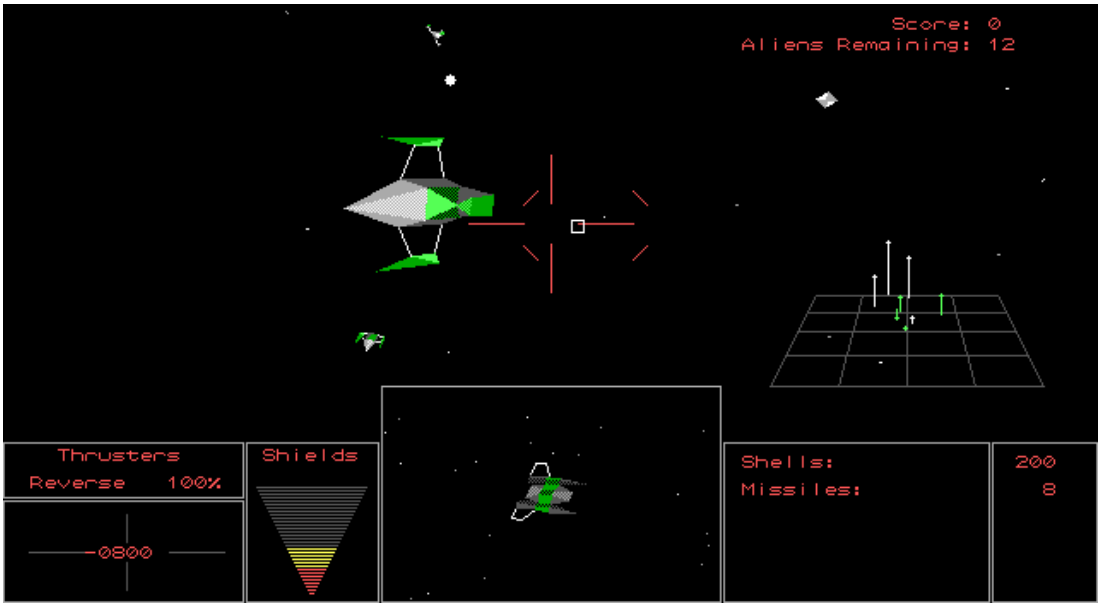
Podle deformace obrazců, tvořených těmito body potom systém vypočítá hloubkovou mapu (jakýsi z-buffer) scény před Kinectem a pomocí série pravidel v ní rozezná osobu a části jejího těla. Z těchto informací vytvoří kloubový model postavy a s využitím předem vytvořené sady dvou set možných postojů v něm doplní chybějící informace, pokud si uživatel část těla náhodou zakrývá. [14]



Obr. 3.3 – Hloubková mapa a kloubový model postavy vytvořené Kinectem, převzato z videa na [15]

4 Specifikace hry

Rozhodl jsem se naprogramovat hru, volně inspirovanou starší hrou Netwars a jejím pokračováním Advanced Netwars (viz obr. 4.1.) od společnosti Caldera. Obě tyto hry používaly jednoduchou grafiku a fungovaly v prostředí operačního systému MS-DOS.



Obr. 4.1. – Screenshot hry Advanced netwars.

Moje hra je nazvaná **Netwars Reloaded** a přestože přebírá některé herní prvky jako například systém soubojů, herní logika je a cíle hry jsou odlišné.

Stručný seznam specifikací:

Hra bude mít 3 herní módy:

- **Kill 'em All** – cílem hry je zničit všechny nepřátelské lodě.
- **Survival** – hra generuje stále nové a nové nepřátele ve zkracujícím se intervalu, cílem je přežít co nejdéle.
- **Epic Battle** – hra vytvoří dva týmy o počtu několika desítek lodí, odlišené typem lodí, které bojují proti sobě a hráč se stane členem jednoho z nich. Vyhrává tým, který zničí všechny lodě soupeřů.

- Hra bude obsahovat 3 lodě, ze kterých si hráč bude moci vybrat.
- Lodě budou mít energetické štíty.
- Zbraně budou zahrnovat jednu energetickou zbraň s neomezeným počtem střel a samonaváděcí rakety, kterých bude mít hráč omezený počet, korelující s počtem nepřátelských lodí.

- Hra bude mít menu s možností výběru herního módu, nastavení grafiky a hry, s nápovědou ovládání.
- Mezi hrou samotnou a menu bude možné opakovaně přepínat.
- Hra se automaticky přizpůsobí rozlišení obrazovky a zapne v celoobrazovkovém režimu, současně ale poskytne možnost zvolit libovolné rozlišení a vypnout celoobrazovkový režim.
- Hra bude mít možnost zapnout a vypnout vertikální synchronizaci.
- Hra bude mít možnost spuštění v Retro módu, který bude vykreslovat pouze barevné drátěné modely.
- Hráč bude mít k dispozici radar, na kterém bude viditelná poloha nepřátelských lodí, radar bude automaticky škálovat měřítko podle nejvzdálenější nepřátelské lodě.
- Hra bude obsahovat částicové efekty při výbuchu nebo kolizích.

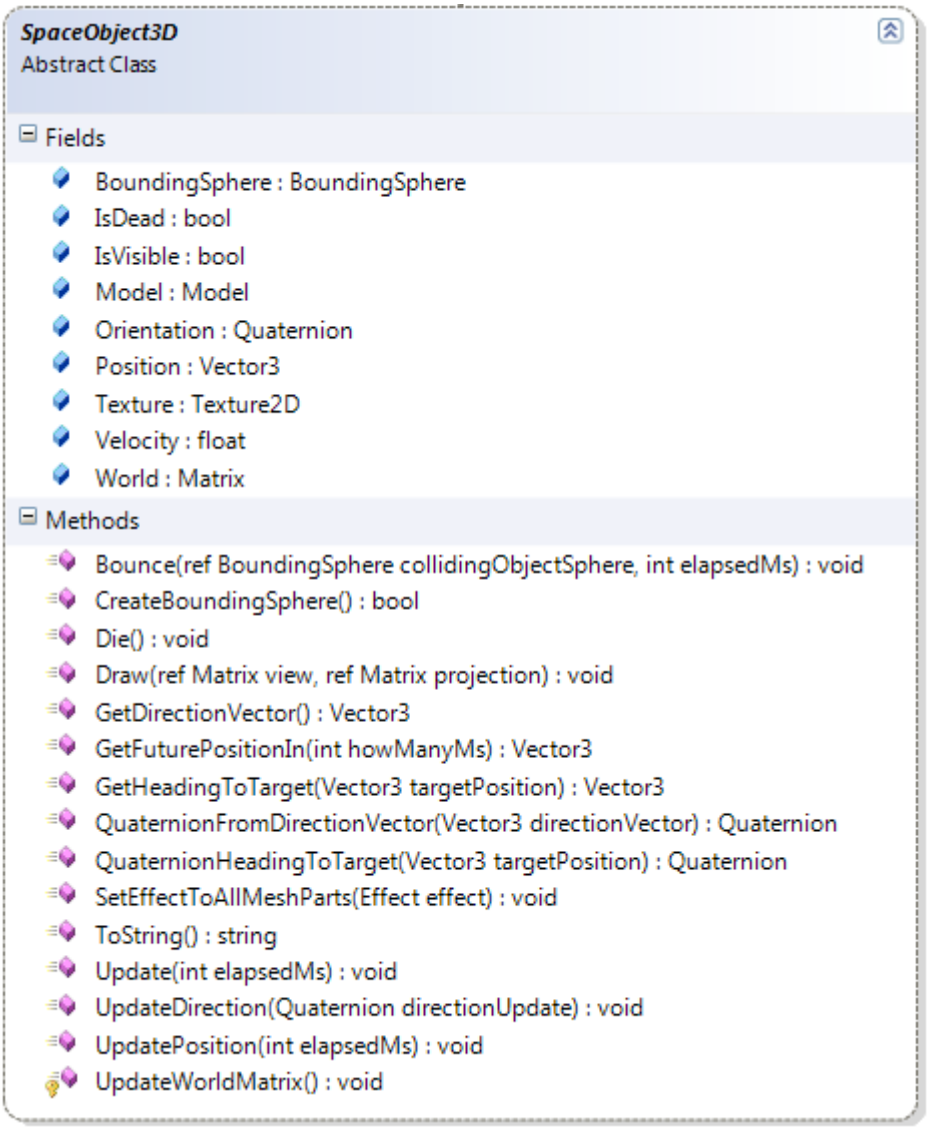
5 Implementace hry

Zdrojový kód hry je napsaný v angličtině včetně komentářů. Celá hra je rozdělena do 4 jmenných prostorů, které spojují logicky související části hry:

- `SpaceObjects` – obsahuje všechny vesmírné objekty.
- `ControlClasses` – obsahuje třídy, které ovládají a spojují jiné třídy do funkčních celků.
- `GameLogic` – obsahuje herní logiku.
- `RenderClasses` – obsahuje třídy, jejichž hlavním účelem je vykreslování grafiky.

Základní třídy, zapouzdřující celou hru, jsou `Menu`, `Engine` a `NetwarsReloaded`, která dědí od XNA třídy `Game` a obsahuje implementaci konečného automatu, jenž přepíná mezi Menu a Enginem. Dále přepíná grafické volby jako je antialiasing a další, a automaticky detekuje a nastavuje rozlišení podle počítače, na kterém je hra spuštěna. Třída také načítá a obsahuje objekty sdílené mezi Menu a Enginem (modely a textury lodí). Třída `NetwarsReloaded` je implementovaná jako Singleton. Nastavení hry popisuje třída `Settings`, která se pomocí XML serializace ukládá při vypnutí hry a načítá při opětovném spuštění. Menu je konečný automat, který registruje stisk kláves nebo tlačítek gamepadu a podle stavu, ve kterém se právě nachází, provede příslušnou akci. Také upravuje jedinou instanci třídy `Settings` v hlavní třídě hry podle nastavení, která si uživatel zvolil, a spouští hru samotnou.
Všechny třídy vykreslující na obrazovku mají veřejnou metodu `Initialize`(), kterou je nutné volat při každé změně rozlišení, protože počítá polohy jednotlivých prvků GUI na obrazovce.

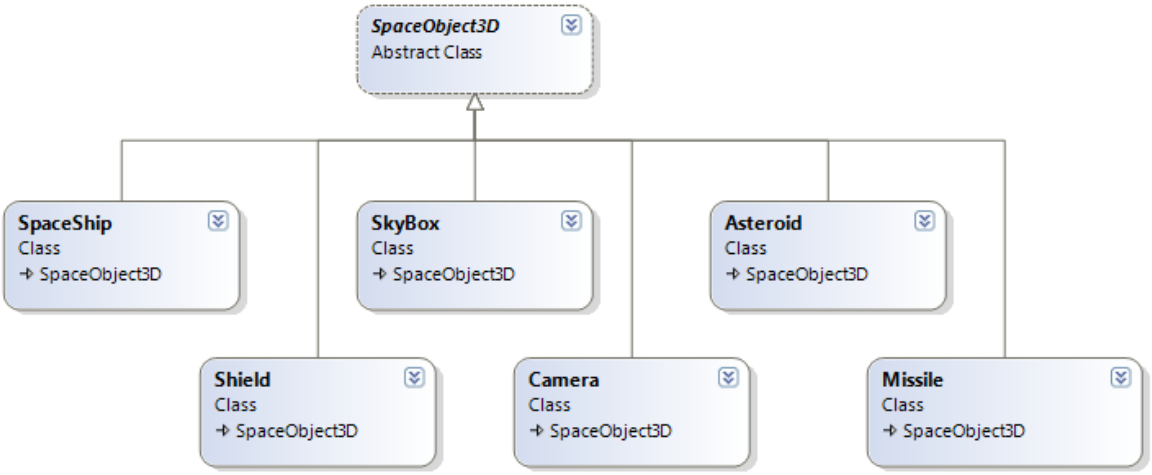
5.1 Vesmírné objekty



Obr. 5.1 – Abstraktní třída SpaceObject3D

Základní datovou třídou, od které jsou odvozeny všechny vesmírné objekty je abstraktní třída `SpaceObject3D` (viz obr. 5.1), která reprezentuje návrhový vzor Template. Obsahuje informace o poloze objektu v 3D prostoru, jeho orientaci, rychlost, model, texturu, world matici a další vlastnosti. Také implementuje metody pro výpočet nové polohy objektu v každém snímku, zjištění očekávané polohy objektu v budoucnosti, vytvoření bounding sphere podle modelu, odraz v případě kolize a další. Zajímavá je metoda `QuaternionFromDirectionVector`, která převede směrový vektor na kvaternion orientovaný stejným směrem tak, že nejprve vytvoří rotační matici a tu potom zkonvertuje na quaternion. Tohoto využívá navigace počítačového hráče, o které bude zmínka dále.

Všechny vesmírné objekty podporují vykreslování pomocí tříd `BasicEffect` a `EnvironmentMapEffect`. Všechny třídy dědící od třídy SpaceObject3D ukazuje obrázek 5.2.



Obr. 5.2 – Třídy odvozené od abstraktní třídy `SpaceObject3D`.

5.1.1 Kamera

Hra používá kameru typu Spring chase, což znamená, že kamera následuje určitý objekt ve scéně (konkrétně hráčovu loď) a zároveň plynule interpoluje svou orientaci do orientace sledované lodi. Takto vytvořená kamera působí přirozeněji, než kdyby se chovala jako pevně připojená k lodi. Kamera je napozicovaná vždy jednu jednotku za a půl jednotky nad loď a míří na cíl posunutý o vektor `CameraTargetToShipOffset` před loď. Také je možné využít funkce zoom a kameru přiblížit a oddálit, přičemž při maximálním přiblížení se přepne ze spring chase na kameru z pohledu první osoby. Kamera dále poskytuje view a projection matice a dokáže vytvořit bounding frustum pro účely view frustum cullingu.

5.1.2 Skybox

Skybox je blíže popsán v předchozích kapitolách, konkrétní implementace pouze nastaví svou polohu na polohu kamery, vypne použití depth bufferu příkazem `GraphicsDevice.DepthStencilState = DepthStencilState.None;`. Tento příkaz způsobí, že vzdálenost vykreslených bodů skyboxu se nebude zapisovat do depth bufferu a skybox tak bude zobrazený na pozadí všech ostatních objektů. Následně depth buffer opět zapne, aby se ostatní objekty vykreslovaly správně. Skybox ke své funkčnosti vyžaduje instanci třídy `Camera`.

5.1.3 Raketa

Samonaváděcí raketa, tvořená třídou `Missile`, je jednou z hráčových zbraní. Při vytvoření očekává její konstruktor instanci třídy `SpaceShip`, na kterou se má raketa navigovat. Pokud ji nedostane, letí rovně. Po odpálení raketa rotuje okolo souřadnicové osy Z, postupně zrychluje a navádí se na cílovou loď. Pokud ji nedostihne do určitého času, definovaného vlastností `TimeToLive`, pak se raketa sama zničí. Navádění probíhá lineární interpolací kvaternionu rakety do kvaternionu, jehož vektorová část část byla vytvořena ze směrového vektoru k cíli. To dává raketě zajímavější a nepředvídatelné chování, protože neletí přímo, ale často i několika oblouky.

5.1.4 Štít

Každá loď má energetický štít, který reprezentuje třída `Shield`. Ta má dva režimy, štít může být viditelný buď stále, nebo se vykreslí pouze po zásahu a postupně vyhasíná. Jeho vykreslování probíhá pomocí 3D modelu koule a vlastního shaderu, který podle stavu štítu mění jeho barvu a intenzitu v pořadí modrá, zelená, žlutá a červená, čímž vizuálně indikuje stav dané lodi.

5.1.5 Vesmírná loď

Třída `SpaceShip` popisuje vesmírnou loď, která má určité zdraví a štít, je schopná střílet rakety nebo projektily, zrychlovat, zpomalovat i pohybovat se bokem a má cílovou loď, kterou se snaží zničit. Také je možné ji ovládat libovolnou třídou tvořící počítačového hráče, stačí aby implementovala rozhraní `IComputerPlayer`:

```
public interface IComputerPlayer
{
    SpaceShip ControlledShip { get; set; }
    void Play(int elapsedMs);
}
```

Důležitá je veřejná metoda `AimsAtTarget()`, která vrací logickou hodnotu podle toho, jestli loď míří na svůj cíl nebo ne. Funguje tak, že je vytvořen zaměřovací paprsek mířící před loď a kontroluje se, jestli protíná bounding sphere cílové lodi.

5.2 Částicové efekty

Střely a částicové efekty ve hře implementují třídy `Billboard` a `BillboardSystem`. `Billboard` je třída obsahující polohu, směr, čas života billboardu a indikátor označující, zda je daný billboard aktivní. Toho se využívá při recyklaci těchto objektů tak, aby nebyla zbytečně alokována stále nová paměť. `BillboardSystem` zapouzdřuje skupinu billboardů se stejnou texturou, rychlostí a rozměry. Hra využívá 3 různé billboard systémy, jeden pro střely, druhý pro exploze a třetí pro částice vznikající při zásahu nebo kolizích.

5.2.1 BillboardSystem

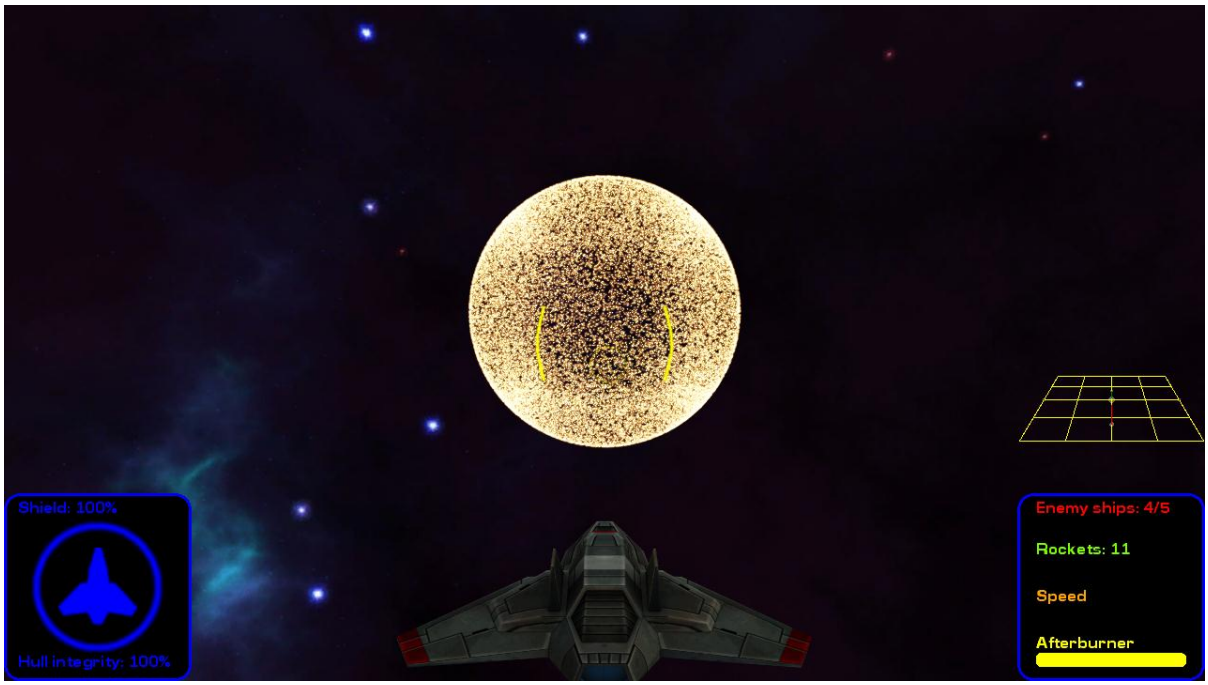
Třída `BillboardSystem` vychází ze stejnojmenné třídy popsané v knize 3D Graphics with XNA Game Studio 4.0 [16], ale je modifikována pro potřeby hry a vylepšena z hlediska efektivity. Protože vytváření velkého množství billboardů, je pro procesor příliš výpočetně náročné a musí se provádět každý snímek, třída `BillboardSystem` k tomu využívá vertex shader a grafickou kartu, která tyto operace dokáže provádět paralelně a výrazně efektivněji - ve vertex shaderu na základě texturové souřadnice posune každý vertex ze středu do odpovídajícího rohu billboardu podle vektoru pohledu kamery.

Třída obsahuje generický seznam billboardů, metody pro update jejich polohy, pole indexů a struktur `VertexPositionTexture` a dynamické index a vertex buffery. V konstruktoru jsou výše zmíněná pole a buffery inicializovány na výchozí velikost a následně se realokují pouze v případě, že je potřeba uložit více prvků než v předešlém snímku.

Důležitá je metoda `CreateBuffers`, která provádí následující:

- Pro každý billboard vytvoří a přidá do příslušného pole:
  - 4 struktury VertexPositionTexture, každou definovanou stejným bodem v prostoru (středem billboardu) a jinou texturovou souřadnicí, reprezentující jeden roh textury.
  - 6 indexů, které se odkazují na tyto vertexy a tvoří dohromady dva trojúhelníky.
- Jsou vytvořeny dynamický index buffer a dynamický vertex buffer typu `VertexPositionTexture`, a data z polí jsou zkopírována do nich.

Jako poslední stojí za zmínku metoda `CreateExplosion()`, která podle zadaných parametrů vytvoří explozi (viz obr 5.3).



Obr. 5.3 - Ukázka částicové exploze.

5.2.2 Vykreslení billboardů

Kvůli možnému překrývání se billboardy kreslí dvěma průchody s využitím alpha testingu. První průchod vykreslí pouze pixely s alpha složkou větší než je hodnota HLSL parametru `AlphaTestValue` (neprůhledné pixely), druhý vypne depth buffer a vykreslí průhledné pixely.

5.3 Kolize

Systém kolizí ve hře zajišťuje třída `CollisionHandler`, která obsahuje metody pro detekci kolizí různých herních objektů a reakci na ně. Taková reakce může zahrnovat poškození objektů, jejich zničení, vytvoření exploze, nebo odraz objektů. V konstruktoru přijímá tato třída reference na všechny objekty, jejichž kolize kontroluje, a pokud místo některého dostane prázdnou referenci, vyvolá výjimku `ArgumentNullException`.

Zjišťování kolizí objektů probíhá „naivním“ algoritmem „každý s každým“, který má asymptotickou časovou složitost $O(N^2)$. Toto řešení výkonově zcela dostačuje při běžných počtech objektů a má výhodu ve snadné čitelnosti a udržovatelnosti. Nad určitý počet objektů (zhruba několik stovek až jeden tisíc) však přestává výkonově dostačovat a bylo by nutné ho nahradit jiným.

5.3.1 Paralelní detekce kolizí

Ve hře je zabudována experimentální podpora paralelní detekce kolizí s využitím Task parallel library .NET Frameworku a metody Parallel.For(). Zajišťuje ji třída `ParallelCollisionHandler`, která dědí od třídy `CollisionHandler` a přepisuje implementaci jejích nejnáročnějších metod. Díky této optimalizaci je možné na čtyřjádrovém procesoru při šesti stech lodí dosáhnout zvýšení snímkové frekvence z 80 na 150fps. Ve výchozím stavu je ale tato vlastnost vypnutá, protože na jednojádrových procesorech může díky režii při vytváření paralelní smyčky snížit výkon až o padesát procent, není kompatibilní s konzolí Xbox 360, a není řádně otestovaná. Zapnout se dá pomocí možnosti Parallel optimizations v nastavení hry.

5.4 Ovládání

Hra se dá ovládat pomocí klávesnice a myši, nebo pomocí gamepadu. Na platformě PC fungují obě možnosti, na Xboxu pouze gamepad. Ovládání hry zajišťují třídy `KeyboardInputHandler`, `MouseInputHandler a GamePadHandler`. Všechny tři obsahují metodu `ProcessInput`, která se volá v každé iteraci herní smyčky a zpracovává veškerý vstup od uživatele. Z něj vytvoří kvaternion, reprezentující změnu směru a ta se, společně s dalšími akcemi, aplikuje na ovládanou loď, jejíž referenci si tyto třídy ukládají při vytvoření.

5.4.1 Myš

Pomocí myši hráč ovládá směr letu a střelbu z obou zbraní, nebo také zoom kamery. Třída `MouseInputHandler` omezuje polohu kurzoru na obrazovce do čtverce o straně výšky obrazovky, aby omezila maximální možnou změnu směru na stejnou hodnotu jak ve vertikální, tak horizontální ose. Z vektoru polohy kurzoru od středu obrazovky potom vytvoří změnu směru lodi.

5.4.1.1 Zaměřování

Hra obsahuje dva typy zaměřování střel, mezi kterými může hráč zvolit: fixní střelbu dopředu před loď a zaměrování kurzorem. Zaměřování střel probíhá následujícím způsobem:

1. Metoda `CreateTargetingRay()` vytvoří dva body se souřadnicemi polohy kurzoru na obrazovce, ale rozdílnou Z souřadnicí (udává hloubku v depth bufferu). Tyto dva body poté promítne na near a far clip plane a jejich spojením vytvoří zaměřovací paprsek, protínající celý pohledový jehlan (view frustum).

2. Metoda `GetTargetHeading()` projde seznam všech nepřátelských lodí a zkontroluje, zda jejich bounding sphere protíná zaměřovací paprsek. Pokud ano, vrátí jejich polohu, pokud ne, vrátí zaměřený bod na far clip plane.

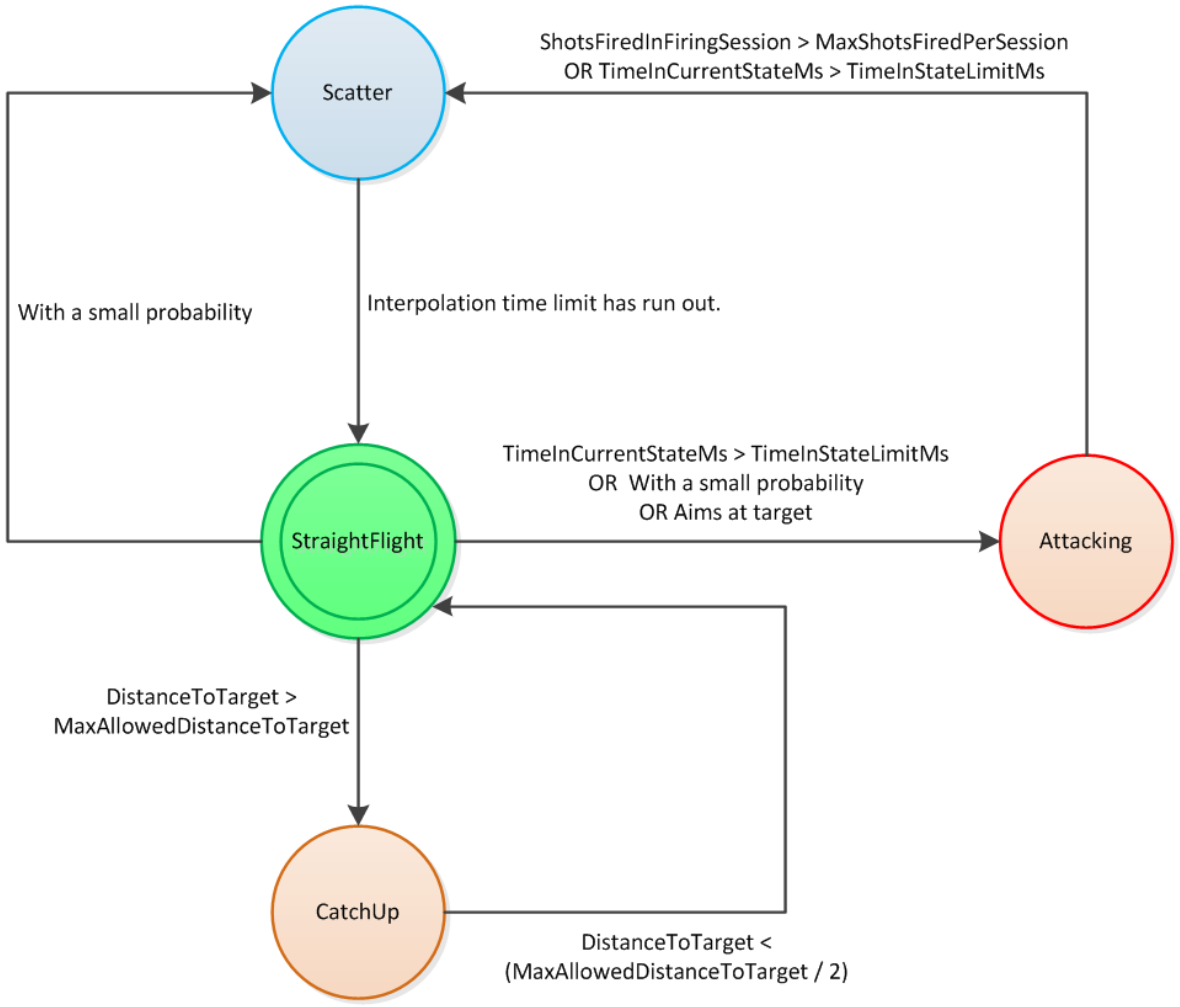
Zaměřování raket probíhá analogicky, pouze je použita jiná metoda `GetTargetedShip()`, která vrací referenci zaměřené lodi nebo null.

5.4.2 Gamepad

Třída `GamePadHandler` dědí od `MouseInputHandler` a rozšiřuje její funkčnost o ovládání dalších akcí přepsáním metody `ProcessInput`. Navíc obsahuje pouze metodu, která převádí stav analogového joysticku gamepadu na polohu kurzoru na obrazovce, k ostatním činnostem využívá zděděných metod.

5.5 Umělá inteligence

Chování počítačového protivníka, obsažené ve třídě `AiPlayer`, je volně inspirováno chováním duchů ve hře Pacman [17]. Popisuje ho konečný automat na obrázku 5.4.



Obr. 5.4 – Konečný automat popisující chování nepřátel

Každý stav má časový limit o délce několika sekund, který se generuje při každém přechodu jako náhodné číslo mezi minimální a maximální hodnotou, aby se nepřátelé chovali nepředvídatelněji a neměnili chování všichni ve stejnou chvíli, což by negativně ovlivnilo obtížnost a výkon hry.

Popis jednotlivých stavů a přechodů mezi nimi:

- **StraightFlight** je výchozí stav, kdy loď letí rovně. Opouští ho po uplynutí časového limitu, pokud před ni vletí cílová loď, nebo náhodně s nízkou pravděpodobností.
- **Scatter** je stav, kdy se loď postupně otáčí do předem vygenerovaného (většinou náhodného) směru. Po uplynutí časového limitu se přepíná do stavu StraightFlight.

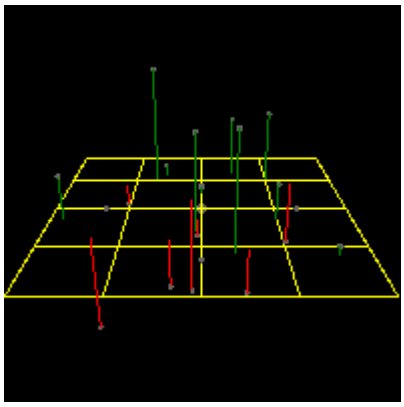
- **Attacking** je stav útočící lodi, která se snaží namířit na nepřítele a střílet. V okamžiku, kdy se příliš přiblíží cílové lodi, si vypočítá polohu nad sebou a přepne se do stavu Scatter, aby se vyhnula kolizi. Během jednoho trvání tohoto stavu má k dispozici limitovaný počet střel a také omezenou přesnost střelby, aby měl lidský hráč šanci vyhrát. Ze stejného důvodu je také (statickou proměnnou MaxAttackingEnemies) definován maximální počet současně útočících nepřátel. Pokud by mělo dojít k jeho překročení, všechny další lodě přechází do stavu Scatter.
- **CatchUp** je stav, kdy loď zrychlí a snaží se dostat blíže k cíli, aby hráč neztratil vizuální kontakt s nepřáteli.

Některé přechody nejsou v konečném automatu zakresleny kvůli přehlednosti – například do stavu CatchUp se loď může přepnout z jakéhokoliv jiného, pokud se příliš vzdálí od cíle. Jakékoliv manévrování lodi probíhá pomocí lineární interpolace kvaternionu a metody `SpaceObject3D.QuaternionHeadingToTarget()`.

5.6 Uživatelské rozhraní

Uživatelské rozhraní vykresluje třída HeadsUpDisplay pomocí bitmap a XNA třídy Spritebatch. Každý prvek uživatelského rozhraní má definovanou texturu, polohu svého levého horního rohu na obrazovce a obdélník udávající v jaké velikosti se bude vykreslovat. Zvláštní část uživatelského rozhraní tvoří Radar.

5.6.1 Radar



Obr. 5.5 - Radar

Radar ukazuje pozice nepřátel relativně k lodi hráče. Nejprve vypočítá vektory vzdálenosti nepřátelských lodí. Poté vytvoří rovinu hráčovy lodi tak, že transformuje rovinu, definovanou souřadnicovými osami X a Z, podle orientace lodi lidského hráče. Následně s využitím

extension metody `Vector3.GetDistanceToPlane()` vypočítá vzdálenost a průmět polohy každé nepřátelské lodi do zmíněné transformované roviny. Největší průmět si také uloží jako rozsah, aby mohl automaticky upravovat měřítko.
Vykreslení radaru probíhá na texturu v paměti (tzv. RenderTarget) a ve dvou fázích. V první se vykreslí mřížka radaru okolo počátku soustavy souřadnic z pohledu shora a zezadu. Poté se kamera přesune nad a za hráčovu loď a vykreslí nepřátele pomocí jednoduchého modelu koule. Pro dodání informace o třetím rozměru vykresluje radar také barevné úsečky vedoucí od každé lodi do jejího průmětu na rovinu – zelené u lodí nad rovinou, červené pod (viz obr. 5.5).

5.7 Engine

Engine je třída zapouzdřující všechny objekty ve hře a reference na zdroje jako jsou modely, textury a další. Po jeho vytvoření je nutné zavolat metodu Initialize, která přijímá zdroje sdílené s Menu a instanci třídy Settings, podle které vytvoří objekty pro daný typ hry.
Obsahuje dvě hlavní metody `Loop` a `Draw`.

5.7.1 Herní smyčka

Herní smyčku obsahuje metoda Loop, která opakovaně provádí následující:

- Update polohy billboardů
    - Projektily
    - Částice explozí
    - Částice vznikající při kolizích
- Update kamery a view matice
- Výpočet pohledového jehlanu
- Update hráčovy lodi
- Zpracování vstupu od uživatele
- Update dalších objektů:
    - Lodě ovládané počítačem
    - Rakety
    - Asteroidy
- Update uživatelského rozhraní
- Nastavení indikátoru viditelnosti u všech herních objektů (kvůli view frustum cullingu)
- Odstranění zničených objektů
- Kontrola stavu hry a případné vypsání výsledku

- Vrácení stavu hry.

5.7.2 Vykreslování

Obr 5.6 – Ukázka Retro módu

Vykreslování hry probíhá v metodě `Engine.Draw` v přesně daném pořadí, protože použité shadery, stejně jako třída `SpriteBatch`, mění stavové objekty grafické karty. Na začátku vykreslování se nastaví tzv. `RasterizerState`, což je objekt, který udává, jak se budou polygony a textury rasterizovat. Pokud si hráč zapnul Retro mód (viz obr. 5.6), vypne se vykreslování skyboxu a vlastnost `RasterizerState.FillMode` se nastaví na hodnotu `FillMode.WireFrame`. Toto nastavení má silně negativní vliv na výkon, protože grafická karta nemůže provádět backface culling.

Jako první jsou vykresleny všechny neprůhledné objekty, poté se vykreslí průhledné štíty, billboardy a uživatelské rozhraní. Nakonec se všechny stavové objekty uvedou do výchozího stavu tak, aby nenarušovaly rendering dalšího snímku, tímto kódem:

```
GraphicsDevice.DepthStencilState = DepthStencilState.Default;
GraphicsDevice.BlendState = BlendState.Opaque;
GraphicsDevice.RasterizerState = RasterizerState.CullCounterClockwise;
GraphicsDevice.SamplerStates[0] = SamplerState.LinearWrap;
```

6 Vývoj pro konzoli Xbox 360

Pro vývoj a ladění na Xboxu 360 je nutné splnit několik počátečních podmínek. První z nich je Windows Live účet, propojený s Microsoft App Hub účtem. Live účet je možné získat zdarma po registraci, App hub účet vyžaduje registraci a zaplacení poplatku 99$, nebo ověření statusu studenta přes Dreamspark. Při registraci těchto účtů je nezbytné vyplnit do kolonky „Země“ jiný stát než je Česká republika, například Velkou Británii, jinak není možné stáhnout potřebnou aplikaci XNA Game Studio Connect. Zatímco placený účet dovoluje publikaci a prodej až 10 vytvořených her v sekci Indie games na Xbox marketplace, studentský účet je omezen na ladění a spouštění hry na lokálním Xboxu.

Dalším krokem je vytvoření kopie projektu pro Xbox 360 z kontextového menu projektu v programu Visual studio. Poslední nezbytnou prerekvizitou je konzole Xbox 360 Premium s pevným diskem připojená ke stejné lokální síti jako počítač, na kterém se vyvíjí. Verze Standard s několika gigabajty flashové paměti vývoj v XNA nepodporuje.

6.1 Vlákna

Xbox 360 nabízí 6 hardwarových vláken (číslovaných od nuly), z nichž každá dvě odpovídají jednomu fyzickému jádru procesoru. Vlákna 0 a 2 jsou rezervována pro XNA Framework, takže aplikace musí využít zbylá 4. Protože simultánní multithreading funguje na principu sdílení výpočetních jednotek a zdrojů (např. cache) jádra procesoru, maximálního výkonu dosáhneme použitím posledního, volného, jádra. Afinitu vlákna k procesoru musí programátor nastavit manuálně, což se ve hře také děje, tímto řádkem:

```
Thread.CurrentThread.SetProcessorAffinity(new int[] { 4 });
```

6.2 .NET Compact Framework

Compact framework je zmenšenou verzí standardního .NET FW pro potřeby méně výkonných a přenosných zařízení. S tím jde bohužel ruku v ruce i snížený výkon jeho základních komponent, které popisuje tato podkapitola.

6.2.1 JIT kompilátor

Just-in-time kompilátor compact frameworku má značně omezené možnosti optimalizace kódu při překladu. Podporuje inlining metod pouze do velikosti 16 bajtů IL kódu, přičemž taková metoda nesmí obsahovat větvení kódu (podmíněné příkazy), parametry typu float, nebo lokální proměnné. Také nedokáže inlinovat virtuální metody [18]. Tato omezení se však dají relativně snadno řešit manuálním inlinováním nejvíce kritických metod.

6.2.2 Garbage collector

Compact framework obsahuje pouze jednoduchý garbage collector typu „Mark and sweep“ [19], který, narozdíl od generačního GC, při každém spuštění prochází všechny vytvořené objekty. Tato operace může trvat i několik desítek milisekund, a tak způsobovat výrazné zasekávání hry. Doba strávená sběrem (latence garbage collectoru) roste přibližně lineárně s rostoucím počtem existujících objektů, ovšem vliv na ni má i celková velikost haldy, kterou musí GC přeskládávat, a její fragmentace v paměti [20]. Garbage collector compact frameworku se spouští vždy po alokaci jednoho megabajtu paměti, takže aby bylo možné zabránit nebo alespoň omezit spouštění kolekcí, je nutné zvolit jeden ze dvou následujících postupů [21]:

1) Nevytvářet během hry žádné nové referenční objekty, nebo jich vytvářet tak malé množství, aby součet jejich velikostí během jedné hry nepřesáhl 1MB. Tento přístup efektivně zabrání spouštění garbage collectoru, ale je relativně obtížné ho implementovat, obzvláště pokud se s ním nepočítá od raného návrhu aplikace. Referenční typy se samozřejmě používat dají, pouze je nutné je vytvořit při inicializaci hry a následně recyklovat už vytvořené, namísto alokace dalších. Hodnotové typy jako jsou XNA Vectory, Quaterniony, nebo matice, se vytváří na zásobníku a garbage collector nespouští.
2) Udržovat haldu co nejjednodušší, což znamená uchovávat na ní co nejméně objektů referenčních typů proto, aby se sběry sice spouštěly, ale trvaly krátkou dobu a nezpůsobovaly cukání hry. Tento přístup je založen hlavně na použití velkých kolekcí (polí nebo generických seznamů) hodnotových typů, které se z hlediska garbage collectoru tváří jako jediná reference, a proto jsou, i přes svou značnou velikost, zkontrolovány velmi rychle.

Je důležité zmínit, že „garbage“ vytvářejí i operace, od kterých to programátor nemusí čekat. Například datový typ string je referenční a neměnný, takže každý pokus o jeho změnu či konkatenace způsobí vytvoření nového objektu. Dalšími zdroji nových alokací může být například cyklus Foreach, který vytváří iterátor, nebo operace boxing, což je zabalení hodnotového typu na referenční.

6.3 Nasazení hry

Před prvním připojením je nutné na konzoli vyhledat a stáhnout aplikaci XNA Connect, která zajišťuje spojení s Visual studiem a umožňuje spouštění a ladění našeho projektu. Po jejím spuštění se vygeneruje párovací klíč (viz obr 6.1), který je třeba zadat do dialogu „Add device“ v XNA Game Studio Device Center. Tuto proceduru je nutné absolvovat pouze jednou, poté už se zařízení připojují automaticky.

Obr. 6.1 – Aplikace XNA Connect a vygenerovaný párovací klíč, převzato z [22]

Připojení ke konzoli, zkopírování zkompilovaného projektu a jeho spuštění je zcela v režii Visual studia a probíhá automaticky po zvolení možnosti „Deploy“ nebo po stisknutí klávesy F5. Nasazení aplikace probíhá inkrementálně, takže při opakovaném spouštění se kopírují pouze změněné nebo přidané soubory, což má dramatický vliv na rychlost ladění. Visual studio také podporuje tvorbu screenshotů ze hry, nebo vzdálené ladění, takže při vložení breakpointu do kódu se aplikace zastaví a je možné zkoumat její stav stejně jako na PC.

6.4 Zobrazení hry

Při návrhu hry běžící na Xboxu je nutné brát v úvahu několik specifických vlastností monitorů a televizních přijímačů:

- **Bezpečná oblast obrazovky (Title safe region):** Staré CRT obrazovky nemusí zobrazovat celý zdrojový obraz, ale jeho okraje mohou ořezávat nebo deformovat. Proto je vhodné použít pro zobrazení uživatelského rozhraní a dalších důležitých herních informací bezpečnou oblast obrazovky, která obsahuje vnitřních 80-90% obrazu. XNA poskytuje objekt typu Rectangle definující tuto oblast v závislosti na připojené obrazovce. Je možné k němu přistoupit z hlavní třídy hry kódem `this.Graphics.GraphicsDevice.Viewport.TitleSafeArea`, a v prostředí Windows vrací obdélník, definující rozlišení obrazovky bez jakéhokoliv ořezu. Při testování bylo zjištěno, že tento problém se nevyhýbá ani novým Full HD LCD televizím, takže hra pozicuje prvky uživatelského rozhraní podle vlastnosti `TitleSafeArea`.
- **Velikost** - prvky uživatelského rozhraní a texty by měly být dostatečně velké, aby si zachovaly čitelnost i z větší vzdálenosti od televize.

Další z aspektů zobrazování hry, se kterým je ale nutné počítat na všech platformách, je poměr stran. Hra může být vykreslována na zobrazovači s libovolným poměrem stran a měla by být schopná se s tímto vypořádat.

6.5 Ovládání hry

Gamepad používaný k ovládání Xboxu 360 nemusí vyhovovat potřebám všech typů her. Pokud je nutné ve hře rychle a přesně mířit, je možné že hráč bude frustrován ovladatelností s použitím analogových joysticků, zvláště pokud je zvyklý na počítačovou myš.
V tomto případě je možné použít ve hře nějakou formu zaměřovacího asistenta, nebo větší toleranci při míření na cíl. Implementovaná hra používá druhý přístup a při zaměřování proto zvětší poloměr bounding sféry každé lodi na trojnásobek. Pokud hráč míří do takto zvětšené oblasti prostoru, zaměřovací systém to vyhodnotí jako dostatečné a vystřelí přesně na cíl.

6.6 Platformně závislý kód

Multiplatformní hra může, a velmi pravděpodobně také bude obsahovat kód, který je závislý na jedné platformě a neproveditelný na jiné. XNA proto nabízí speciální symboly pro direktivy preprocesoru, kterými stačí ohraničit nekompatibilní kód a ten je potom při překladu pro jiné platformy vypuštěn. Syntaxe těchto direktiv je následující:

```
#if WINDOWS
    // kód zde se přeloží pouze pro platformu Windows
#elif XBOX
    // tento pro Xbox 360
    // všimněte si, že Visual studio opticky zvýrazní kód
    // podle projektu, který je právě otevřen
#elif !XBOX
    // je možné používat i operátor negace pro vyloučení určité platformy
#endif
```

6.7 Optimalizace

Jednou z nejdůležitější vlastností každé počítačové hry jsou hardwarové nároky pro její plynulý běh. Zatímco na PC si hráč může nedostačující část počítače vyměnit za výkonnější, Xbox 360 nic takového neumožňuje. Když k tomu vezmeme v úvahu výkonové nedostatky .NET Compact frameworku, stává se pro něj otázka optimalizací softwaru zcela zásadní. Je ovšem nutné důkladně zvážit, zda nám konkrétní optimalizace vyváží nevýhody v podobě snížení čitelnosti a udržovatelnosti zdrojového kódu, času stráveného její implementací a možnosti zanesení dalších chyb do programu. Nemá smysl optimalizovat činnosti, které se ve hře provedou pouze jednou, nebo například při načítání levelu. Naopak dává velký smysl optimalizovat metodu, která se volá tisíckrát za snímek, nebo v ní hra tráví 80% času. Pomůckou při odhalování takových metod může být například profiler nebo stopování délky provádění jednotlivých akcí.

Tato kapitola popisuje optimalizace využité ve hře. Protože byly implementovány postupně v průběhu celého vývoje hry, je nemožné poskytnout přesné hodnoty snímkové frekvence před a po jejich začlenění do hry. Každá však byla samostatně otestována a přináší měřitelné zvýšení výkonu, přičemž bez některých z nich by byla hra alespoň na Xboxu nehratelná. Například bez recyklace index a vertex bufferů by hra alokovala několik megabajtů paměti za sekundu a následné aktivace garbage collectoru by ji učinily zcela nehratelnou. Některé z popsaných optimalizací mají smysl pouze na Xboxu, jiné se dají úspěšně využít i pro platformu PC.

6.7.1 View frustum culling

Tato optimalizace snižuje počet vykreslovaných objektů tak, že v každém snímku kontroluje, zda je daný objekt právě viditelný, nebo ne. Pokud ne, není nutné ho vykreslovat. Kontrola viditelnosti se provádí vytvořením XNA objektu BoundingFrustum (popisuje pohledový jehlan), a následnou kontrolou, zda je bounding sphere daného objektu v jehlanu obsažena nebo ho protíná. Kontrola průniku těchto dvou útvarů však také spotřebovává určité prostředky, a tak se view frustum culling provádí pouze pro objekty, které mají 3D model, ne například pro billboardy, kterých je ve hře velké množství a jejich vykreslení je relativně nenáročné.

6.7.2 Optimalizace cyklů

Cykly bývají nejnáročnějšími částmi aplikace. Pokud cyklus obsahuje invarianty, je možné je z cyklu vyjmout a provést pouze jednou před ním, což může vyústit v zajímavý výkonový zisk. Tato optimalizace byla provedena např. u view frustum cullingu – BoundingFrustum je invariant a tak se jeho výpočet provádí pouze jednou za snímek a ne pro každý objekt ve hře.
Pokud se za sebou vyskytují dva cykly, které iterují přes stejnou kolekci, nebo mají stejné hodnoty iterační proměnné, je možné je sloučit do jednoho. Popis všech možností optimalizace cyklů by byl velmi rozsáhlý a přesahuje rámec této práce.

6.7.3 Referenční a výstupní parametry funkcí

Při volání metody přijímající parametry hodnotových typů dochází ke kopírování těchto parametrů na zásobník. Toto může v případě velkého množství struktur předávaných hodnotou velmi ublížit výkonu celé aplikace. Další problém je, že přetížené operátory XNA struktur jako je Vector3 či Quaternion jsou ve skutečnosti statické metody, přijímající parametry hodnotou.
Řešením je předávat struktury odkazem a vracet výsledky funkcí pomocí výstupních parametrů, což zabrání zbytečnému kopírování struktur. Zabudované XNA typy jako je Vector3 či Quaternion často implementují přetížené verze svých metod, které používají referenční a výstupní parametry, a jsou proto rychlejší:

```
Vector3 first = Vector3.Zero, second = Vector3.One;
//předávání hodnotou
Vector3 result = first + second;
result = Vector3.Add(first, second);
//přetížená verze metody, která předává parametry a výsledek odkazem
Vector3.Add(ref first, ref second, out result);
```

6.7.4 Využití grafické karty pro FPU výpočty

Kdykoliv je třeba provést mnoho nezávislých operací v plovoucí řádové čárce, je výhodnější přesunout tyto výpočty do grafické karty a zpracovat je paralelně pomocí shaderů. Toto řešení může zajistit zrychlení až o několik řádů, jeho nevýhodou je však výrazně vyšší složitost a horší čitelnost takového kódu. Používá ho třída BillboardSystem.

6.7.5 Manuální inlining metod

U malých a vytížených metod může být výhodné provést manuální inlinovaní do kódu metod, které je volají. Výhodou je odpadnutí režie při skoku a kopírování parametrů metody, nevýhodou může být duplikace kódu a horší udržovatelnost a čitelnost. Obecně se tento postup vyplatí jen u těch nejkritičtějších metod, v celém kódu hry byl použit pouze dvakrát, a to na metody pro update polohy billboardů a kontrolu kolize dvou lodí.

6.7.6 Recyklace objektů

Možná nejvýznamnější optimalizace her v prostředí s jednoduchým garbage collectorem je nějaká forma znovupoužití existujících objektů. Dá se implementovat různým způsobem (např. pomocí kruhové fronty nebo poolingem) a umožňuje zabránit nechtěným spouštěním garbage collectoru a tím narušování zážitku ze hry.

6.8 Testy výkonu

Hra byla testována na dvou konfiguracích počítačů a Xboxu 360.

Sestava 1: AMD Phenom II X4 940 BE (3.0GHz), 4GB ram, Ati Radeon HD 4850, Windows 7
Sestava 2: AMD Athlon X2 3800+ (2.0GHz), 3GB ram, Ati Radeon HD 4550, Windows XP

Na obou těchto strojích poskytuje při běžných počtech lodí (do 200) snímkové frekvence daleko přesahující 60fps. Silnější sestava 1 zvládá přes 60fps i při vykreslování 600 lodí. Vzhledem ke slabší sestavě, která je v dnešní době spíše průměrná až podprůměrná, lze říci, že hra je velmi dobře optimalizovaná a poběží plynule na většině současných počítačů.
Xbox 360 je dle očekávání výrazně slabší. Při rozlišení 1920*1080 vykresluje 60 snímků za sekundu do zhruba 100 lodí ve hře. Při 200 lodích už snímková frekvence chvílemi klesá až ke 30. Všechny testy byly provedeny ve výchozím nastavení hry – tzn. se zapnutým antialiasingem.

6.9 Možnost portace pro Windows Phone

Protože současné telefony s Windows phone jsou hardwarově výrazně slabší než PC nebo Xbox, bylo by v první řadě potřeba snížit velikost textur a detailnost 3D modelů. Jak už bylo zmíněno, tato platforma nepodporuje vlastní shadery, což znamená, že by hra ani nešla zkompilovat kvůli částicovému systému a shaderu pro vykreslování štítu. Částicový systém by bylo nutné přepsat, nebo nahradit jiným, který WP podporuje (například existující Dynamic Particle System Framework) a střely nahradit buď jinou verzí billboardů, nebo velmi jednoduchým 3D modelem. Štíty by musely implementovat vykreslování pomocí BasicEffectu, nebo jiného z předdefinovaných shaderů XNA frameworku.

Ovládání hry by se dalo vyřešit třídou, která vytvoří virtuální joystick a tlačítka na displeji, nebo pomocí akcelerometru. Poslední nutná změna by se týkala úpravy některých konstant umělé inteligence a lodí tak, aby hra mírně zpomalila kvůli horšímu ovládání a menšímu displeji mobilního telefonu.

7 Závěr

Hra byla úspěšně implementována a otestována na Xboxu 360, kde s rezervou dosahuje hratelných hodnot snímkové frekvence. Výsledky výkonových testů také vyvrací jakékoliv obavy o nedostatečnou efektivitu XNA Frameworku a řízeného kódu pro potřeby počítačových her, alespoň na platformě PC. Compact frameworku by výrazně prospěl vyspělejší garbage collector a lépe optimalizující JIT kompilátor. Práce uceleně shrnuje specifika vývoje a optimalizace multiplatformní hry v XNA.

Testováním a profilováním bylo zjištěno, že nejnáročnějšími komponentami hry jsou zjišťování kolizí a částicový systém. Při pokračování vývoje hry by proto bylo vhodné zaměřit se na jejich vylepšení a zrychlení, např. implementací kolizního modelu pomocí octree. Také by bylo možné relativně jednoduše implementovat lokální split-screen multiplayer až 4 hráčů pomocí Render targetů.

Z časových důvodů je ve hře dokončen pouze první herní režim a radar není odladěný. Přestože také nebyl implementován pooling billboardů a dochází k aktivacím garbage collectoru, hra je natolik nenáročná, že si zachovává plynulost i na konzoli Xbox 360. Tyto chyby budou opraveny v budoucnu.

Literatura

1. **Evjen, Bill, a další, a další.** *C# 2008 Programujeme profesionálně.* 2009. ISBN 978-80-251-2401-7.

2. **ECMA International.** *C# Language Specification.* 2006.

3. **Microsoft Corporation.** Platform Invoke Tutorial. *MSDN.* [Online] [Citace: 8. 5 2012.] http://msdn.microsoft.com/en-us/library/aa288468(v=vs.71).aspx.

4. **Reed, Aaron.** *Learning XNA 4.0.* 2010. ISBN 978-1-449-39462-2.

5. **Hargreaves, Shawn.** New built-in effects in XNA Game Studio 4.0. *Shawn Hargreaves' blog.* [Online] http://blogs.msdn.com/b/shawnhar/archive/2010/04/28/new-built-in-effects-in-xna-game-studio-4-0.aspx.

6. **Microsoft Corporation.** What Is a Profile? *MSDN.* [Online] http://msdn.microsoft.com/en-us/library/ff604995.aspx.

7. **Kolektiv autorů.** *Moderní počítačová grafika.* Brno : autor neznámý, 2004. ISBN 80-251-0454-0.

8. The Xbox Operating System. *http://blogs.msdn.com/b/xboxteam/.* [Online] http://blogs.msdn.com/b/xboxteam/archive/2006/02/17/534421.aspx.

9. Xbox Troubleshooting: Red Ring of Death Fix. *www.squidoo.com.* [Online] http://www.squidoo.com/xbox-troubleshooting-red-ring-of-death-fix.

10. Xbox Development Kit arrives with 'significantly reduced price,' Sidecar attachment. *www.engadget.com.* [Online] http://www.engadget.com/2011/04/05/xbox-development-kit-arrives-with-significantly-reduced-price/.

11. Is XNA good for professional development for the Xbox 360? *http://gamedev.stackexchange.com/.* [Online] http://gamedev.stackexchange.com/questions/4915/is-xna-good-for-professional-development-for-the-xbox-360.

12. **Mike Schramm.** Kinect: The company behind the tech explains how it works. *www.joystiq.com.* [Online] http://www.joystiq.com/2010/06/19/kinect-how-it-works-from-the-company-behind-the-tech/.

13. **Matt Peckham.** Kinect Has a Thousand Eyes Viewed With Night Vision Goggles. *www.pcworld.com.* [Online] http://www.pcworld.com/article/209881/kinect_has_a_thousand_eyes_viewed_with_night_vision_goggles.html.

14. Exclusive: How does Microsoft Xbox Kinect work? *www.t3.com.* [Online] http://www.t3.com/features/exclusive-how-does-microsoft-xbox-kinect-work.

15. Gesture Recognition using Depth Sensors (PrimeSense PS1080, Kinect, Xtion). *www.youtube.com.* [Online] http://www.youtube.com/watch?v=e0c2B3PBvRw.

16. **James, Sean.** *3D Graphics with XNA Game Studio 4.0.* 2010. ISBN 978-1-849690-04-1.

17. Chování duchů ve hře Pac-Man, část 1. *http://programujte.com/.* [Online] http://programujte.com/clanek/2011010500-chovani-duchu-ve-hre-pac-man-cast-1/.
18. **Microsoft corporation.** .NET Compact Framework version 2.0 Performance and Working Set FAQ. *MSDN Blogs.* [Online] http://blogs.msdn.com/b/netcfteam/archive/2005/05/04/414820.aspx.
19. **Rico Mariani.** Taming the CLR: How to Write Real-Time Managed Code. *MSDN Blogs.* [Online] http://blogs.msdn.com/b/ricom/archive/2006/08/22/713396.aspx.
20. **Microsoft corporation.** Managed Code Performance on Xbox 360 for XNA: Part 2 - GC and Tools. *MSDN Blogs.* [Online] http://blogs.msdn.com/b/netcfteam/archive/2006/12/22/managed-code-performance-on-xbox-360-for-xna-part-2-gc-and-tools.aspx.
21. Twin paths to garbage collector nirvana. *Shawn Hargreaves blog.* [Online] http://blogs.msdn.com/b/shawnhar/archive/2007/07/02/twin-paths-to-garbage-collector-nirvana.aspx.
22. **Microsoft Corporation.** Connecting to Your Xbox 360 with XNA Game Studio 4.0 Refresh. *MSDN.* [Online] http://msdn.microsoft.com/en-us/library/bb975643.aspx.
23. —. Xbox 360 Programming Considerations. *MSDN.* [Online] http://msdn.microsoft.com/en-us/library/bb203938.aspx.
24. **Tom Miller, Dean Johnson.** *XNA Game studio 4.0 Programming.* 2011. ISBN 978-0-672-33345-3.
25. XNA Games Programming. *www.toymaker.info.* [Online] http://www.toymaker.info/Games/XNA/index.html.
26. **Grootjans, Riemer.** XNA Tutorial for C# overview – Series 1. *Riemer's XNA tutorials.* [Online] http://www.riemers.net/eng/Tutorials/XNA/Csharp/series1.php.

Seznam příloh

Příloha 1. DVD se zdrojovým kódem hry.