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

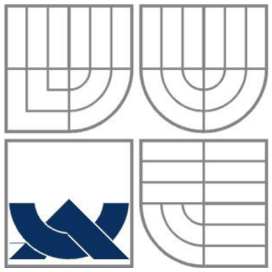
POČÍTAČOVÁ HRA SE ZAMĚŘENÍM NA MULTIPLAYER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

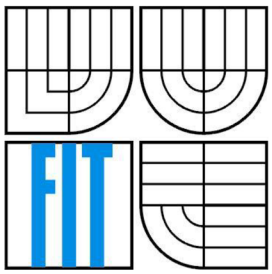
AUTOR PRÁCE
AUTHOR

TOMÁŠ KUBÍK

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

POČÍTAČOVÁ HRA SE ZAMĚŘENÍM NA MULTIPLAYER

MULTIPLAYER PC GAME

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ KUBÍK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. TOMÁŠ MIKOLOV

BRNO 2010

Abstrakt

Tato práce se zabývá tvorbou jednoduché trojrozměrné počítačové hry, která umožňuje propojení hned několika hráčů najednou prostřednictvím sítě internet (LAN). V této práci je použit a popsán Microsoft XNA Framework. Výsledná hra obsahuje jednu lokaci, kde je možné vidět oblohu, jednoduchý terén, modelované 3D objekty a zjednodušenou fyziku letu.

Abstract

This work describes implementation of a simple 3D computer game, with support of gameplay over internet for multiple gamers. In this project, Microsoft XNA framework is used and throughly described. The final game includes a skybox, simple terrain and multiple 3D objects, all rendered in real time.

.

Klíčová slova

Počítačová hra, 3D zobrazení, síťová komunikace ve hře, Lidgren network library, Tom Shane Neoforce Controls, herní vývojový cyklus.

Keywords

Computer game, 3D projection, multiplayer and game networking, Lidgren Network library, Tom Shane NeoForce Controls, game developmental cycle.

Citace

Kubík Tomáš: Počítačová hra se zaměřením na multiplayer, bakalářská práce, Brno, FIT VUT v Brně, 2010

Počítačová hra se zaměřením na multiplayer

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Tomáše Mikolova. V práci jsem uvedl všechny literární prameny, publikace a knihovny, ze kterých jsem čerpal, nebo které jsem použil.

……
Tomáš Kubík
19.5.2010

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce za pomoc, ochotu a čas, který mi věnoval.

Obsah

1 Úvod

Svět počítačových her je oblast v informačních technologiích, která vyžaduje velké množství znalostí z různých oborů jako je fyzika, matematika, informatika (hardware i software) a další. Díky moderním technologiím uplatňujícím se v posledních letech je však již jednodušší snadno se položit do psaní středně náročných počítačových her a nebrat tolik zřetel na nízkoúrovňové věci (tj. práce přímo s hardwarem počítače) a problémy s nimi spojené. Člověk pak není natolik vázán na vícečlenný tým, který je nutností pokud chcete, aby byla hra konkurence schopná a vydala se v co nejbližším časovém horizontu, nebo pokud nechcete trávit tolik času nad řešením těchto problému a raději energii vložit do samotné logiky a příběhu hry. Díky těmto aspektům jsem si i já vybral právě toto téma.

V následujících kapitolách bude přiblížena tvorba jednoduché letecké síťové hry od počátku až do konce. Účelem této hry není reálná simulace chování letadla, ale zábavná a nenáročná forma akční hry, v které je hlavní prioritou sestřelení všech svých protivníků z oblohy. Ve hře se nachází jednoduché úvodní menu, které umožňuje hru založit, vyhledat, nebo připojit se k už vytvořené relaci. Do této hry se pak kdykoliv mohou připojit další hráči. Letadlo disponuje potřebnou municí a palivem, bez kterého by pouze plachtilo. Pro znovudobytí munice či paliva se nad ostrovem vznáší tzv. checkpointy, kterými když letadlo proletí, získá novou munici, palivo, nebo extra rychlost. Ke konci bude představeno nabízející se rozšíření a případná pokračování této hry.

2 Teorie a představení použitých technologií

2.1 Teorie počítačových her

Počítačová hra je z hlediska informačních technologií program, či aplikace jako každá jiná. Ovšem velký důraz se zde většinou klade na interakci uživatele, případně náhodu skloubenou s časem. Většina typických konzolových a okenních aplikací (jako např. programy pro evidenci majetku, účetnictví, výpočty, rýsování či modelování různých objektů) není závislá na čase a má striktní formu předem daných příkazů, postupů a jejich odpovědí, které se s definicí hry moc spojovat nedají. Počítačová hra je přímo závislá na interakci s uživatelem, která se většinou opakuje v tzv. nekonečné smyčce a reakce hry se na uživatelské vstupy mohou radikálně měnit v průběhu času.

Počítačové hry vznikají už od dob, kdy se první počítače dostaly na pulty obchodů. Samozřejmě prošly velkým vývojem a získaly mnoho tváří, jež má za následek to, že nejsou dnes vídány pouze ve formě textových programů, ale i s 2D či 3D propracovaným, realistickým enginem.

Protože herní engine je jednou z nejdůležitějších částí počítačové hry, co do technické stránky, je mu věnován tento odstavec. Engine, jako takový, bývá po programátorské stránce nejkomplikovanější část celého projektu, bez které by hra vůbec nevznikla. Tato komponenta se může starat o renderování všech objektů ve scéně, jejich otexturování a osvětlení. Řeší pohyb a pohled kamery (resp. herních postav, letadel apod.), který následně přetransformuje hráče na dvourozměrnou obrazovku. Herní engine definuje vztahy mezi objekty ve hře, jejich vzájemné působení, gravitaci a další fyzikální vlastnosti v celém virtuálním světě. Do této sféry spadá i pohyb objektů v prostoru a jejich vzájemná koordinace. Ačkoli může být engine použit i na více odlišných her, může řešit i umělou inteligenci, kterou je možno modifikovat pro vlastní, specifičtější účely. V moderních počítačových hrách, tam kde se to přímo vybízí, je skoro hřích neimplementovat možnost síťové hry – proto ve většině dnešních herních enginů nechybí ani nativní podpora vlastní síťové komunikace, která umožní ze hry udělat společenskou záležitost jiného rozměru. V takovém případě engine předepisuje i síťové protokoly, pravidla komunikace a podporované druhy připojení (tcp/ip, ipx apod).

Další nepostradatelnou částí počítačové hry, která kooperuje s enginem, jsou všechna multimédia. Obrázky na pozadí menu, tlačítek a všechny zvuky, které hrají při spuštění hry, nebo které se spouští při některé akci přímo ve hře (střelba, zvuk motoru apod.). Jako celek pomáhají dotvořit věrohodnou atmosféru a uživatel je zde (na rozdíl od ostatních aplikací) vyžaduje. Další součástí multimédií ve hře jsou všechny modely vozidel, lidí, stromů, budov a propracovaných herních lokací, které nejsou generovány kódově. Je velký rozdíl, zda se jedná o modely do 2D či 3D počítačové hry. 2D modely jsou většinou obyčejné průhledné obrázky, které jsou v průběhu času posouvány, zkoseny, zvětšovány, zmenšovány a různě transformovány na obrazovce počítače. 3D modely, jak název napovídá, uchovávají i třetí rozměr, díky čemuž je většina typických operací

s těmito modely složitější a výpočetně náročnější. Zatímco u 2D modelů je bitmapový model složen z bodů na obrazovce (pixelů), tak u 3D modelu je tomu jinak. Pro jejich zobrazení musíme v paměti uchovávat pozice všech bodů v prostoru, které utváří celý model. Tyto body se většinou spojují do trojúhelníků a zakřivených ploch, které tvoří tzv. polygony. Tyto modely mohou být zhotoveny programově (procedurálně), nebo, a to ve většině případů u her, manuálně ve specializovaných programech pro tvorbu 3D modelů (3D studio max, Blender, Maya, Lightwave …) zkušenými grafiky, kteří mohou dále modely oživit animacemi a různými pohyby.

Poslední částí hry, kterou většina dnešních profesionálních tvůrců počítačových her zanedbává, je její hratelnost a celkový příběh. Je jednoduché napsat kalkulačku a říci o ni, že je to hra, ale jakmile v tom uživatel neuvidí zábavu, strhující příběh, nebo aspoň něco, co by ho nutilo propočítat se do dalšího kola, tak všechna práce byla k ničemu a hra je odkázána k zapomnění.

Na závěr stojí za zmínku, že vývoj počítačových her je jedno z odvětví moderního bussinesu, které umožňuje spolupráci více lidí s naprosto odlišnými znalostmi a zkušenostmi z jiných oborů v jednom týmu (programování, grafika, dějiny, fyzika). Tato práce (koníček) už nebývá jen o „datlování“ kódu v tmavých místnostech vývojářských týmů, ale o rozsáhlých studiích prezentované problematiky a výjezdech do terénů pro získání jedinečných podkladů a materiálů.

2.2 Microsoft XNA Framework

Firma Microsoft v roce 2004 vydala první verzi svého dnes už hojně používaného frameworku, který umožňuje hlavně jednotlivcům nebo menším vývojářským týmům redukovat množství potřebného kódu při tvorbě ne moc rozsáhlých počítačových her. Tento framework by se dal považovat za nástupce Managed DirectX 9, u kterého se vývoj již zastavil. Celé XNA je postavené nad .NET Frameworkem a DirectX 9 a využívá Common Language Runtime optimalizovaný pro běh a správu her. Vyšší verze DirectX nejsou zatím podporovány z důvodu zpětné kompatibility se staršími systémy jako Windows XP. Oficiálním jazykem pro vývoj pod XNA je C#, ale je možné importovat podporu i pro ostatní NET jazyky. XNA umožňuje vývojářům zapomenout na to jaký hardware je v tom nebo tamtom počítači. Obsahuje totiž knihovny, které vás naplno odstíní od nízkoúrovňové správy hardwaru (tak jako by tomu bylo, kdybychom vyvíjeli hru přímo pod DirectX). Jediné požadavky na běh hry jsou zajištění minimální hardwarové konfigurace pro tuto hru a softwarové vybavení. XNA také obsahuje mnoho funkcí a algoritmů na zdánlivě jednoduché věci, které bychom pod DirectX museli složitě a pracně psát sami. S XNA Frameworkem můžeme vyvíjet hry pro Windows XP, Windows Vista, Windows 7, Xbox 360, Zune a Windows Phone 7. Ačkoli poslední tři jmenované nespadají pod PC platformy, Microsoft jim (hlavně Xboxu 360) přikládá velkou roli a některé komponenty XNA frameworku jsou přizpůsobeny právě a pouze pro ně. Existuje i portace toho frameworku s název MonoXNA, která je určena pro linux a místo zmiňovaného DirectX využívá knihovny OpenGL. Při vývoji pro Windows může být využita celoá škála knihoven, kterou nabízí nainstalovaný NET Framework. Ovšem při vývoji pro Xbox 360 máme k dispozici pouze okleštěnou verzi v podobě .NET Compact Frameworku, který je viděn v různých PDA a smartphonech s Windows Mobile. Omezení je logické a vychází z hardwarové výbavy těchto zařízení. Pokud jsou ale dodržena určitá pravidla, není problém hru pro Windows lehce portovat na ostatní zmiňované platformy. XNA Framework je součástí integrovaného vývojového prostředí XNA Game Studio, což není vlastně nic jiného než plugin pro Microsoft Visual Studio 2005 a vyšší. Díky této nadstavbě může Visual studio vytvářet herní projekty a uživatel dostává přístup k několika užitečným nástrojům jako je například XACT (Cross-platform Audio Creation Tool), který slouží pro správu hudebního obsahu jak na Xbox360 tak pod Windows. Pomocí XACT lze i snadno vytvářet hudební a zvukové smyčky a jiná opakování, která se většinou horko těžko tvoří přímo v kódu hry.

Při programování hry v XNA se dají často využít následující klíčové vlastnosti tohoto frameworku, které značně urychlí vývoj. První z nich je správa samotné herní smyčky a vykreslování obsahu na monitor. XNA si automaticky zjistí zobrazovací schopnosti vašeho monitoru a jemu přizpůsobí i počet renderovaných snímku za sekundu. Bylo by velice nevýhodné, kdyby váš počítač musel renderovat 150 snímků za sekundu na monitor, který zvládne zobrazit jen 70. Dále tato smyčka automaticky volá při každém zobrazení herní scény příslušné události pro naplnění herního obsahu, inicializaci, aktualizaci dat a vykreslení, které obsluhují přímo náš kód.

Další výhodou je správa obsahu pomocí tzv. *Content Pipeline*. Tento mechanismus nás oprostí od načítání 3D modelů, textur, fontů a dalších grafických souborů z filesystému počítače a jejich následného importu do enginu počítačové hry. Díky Content Pipeline je možné spravovat

multimediální obsah právě v Xbobech, kde není nic jako filesystém a kde v okleštěné verzi NET Frameworku nejsou ani funkce pro práci s filesystémem, adresáři, právy apod. Content Pipeline je vlastně sdružení všech těchto souborů, které jsou při kompilaci převáděny do formátu .xnb, který je potom za běhu hry velice snadné importovat pomocí funkcí, které čtou z Content Pipeline. Také importace modelu s již aplikovanými texturami není o moc složitější než vložit obyčejný obrázek. Na Content Pipeline je velice výhodné to, že není potřeba se starat o to, jak nahrát a používat ve hře obrázek s příponou jpg a pak zvlášť, jak nahrát a používat obrázek typu png (ještě k tomu se zachováním průhlednosti). Správa obsahu prostřednictvím Content Pipeline totiž skýtá rozsáhlou podporu většiny známých souborů (modelů, obrázků, shaderů apod). Trochu odlišná práce je zde se zvukem a to prostřednictvím výše zmiňovaného XACT enginu, nicméně s plnou podporou XNA.

Jak bylo psáno dříve, na Xboxech není možné data „ukládat na disk“. Herní obsah je zde řešen pouze přes Content pipeline. V situaci, kdy je potřeba ukládát herní pozice, statistiky a podobné věci, XNA nabízí namespace Storage , který zprostředkovává malé uložiště právě pro tato data. Tyto třídy se dají využít i při vývoji pro Windows. Použití je jednoduché a při ukládání statistik, checkpointů nebo konverzací se přímo vybízí.

Dalším bezesporu ulehčujícím mechanismem pro tvorbu síťové hry je komplexní podpůrná technologie Live!. Této technologii se celkem zdařilou formou věnuje i Riemer Grootjans ve své knize [1]. Tato síťová platforma umožňuje spravovat jednotlivé hráčské profily a zprostředkovávat komunikaci mezi hráči. Není potřeba se starat o principy vyhledávání her, ke kterým se můžete připojit. Systém sám poskytuje informace o kvalitě připojení a o stavu vytvořeného sezení. Také nemusí být manuálně udržována spojení mezi hráči a přes celkem přívětivé API můžou být rozesílána data všem hráčům ať už v topologii klient-server nebo peer-to-peer. Velkou výhodou je to, že tento mechanismus funguje i na Xboxech 360. Live! technologie je vlastně jediný možný způsob, jak na této konzoly hrát přes síť.

Bohužel je zaměřena hlavně na Xboxy, takže pro vývojáře na PC platformě nese nemilá omezení, která i mne přinutila tuto technologii zavrhnout. Omezení se týká počtu spuštěných instancí Live! na jednom počítači na jednu – toto omezení se týká spíše nepříjemností při testování, což by ještě nebylo tak strašné, ale když připočteme větší síťovou réžii a s tím spojený maximální počet připojených hráčů 32 a hlavně nutnost nainstalovat celé XNA Game studio (potažmo Visual studio) na klientském počítači pro chybějící podporu Live! technologie v XNA redistributable balíčku, tak klasický uživatel Windows, aby spustil síťovou hru, je nucen kvůli drobné hře nainstalovat věci, které mu zaberou drahocenný čas a pár giga na disku, která nakonec stejně k ničemu nevyužije.

Nepostradatelným pomocníkem při vývoji hry jsou také metody a datové typy pro práci s maticemi, quaterniony, modely apod. Některé operace se samozřejmě musí řešit individuálně, ale určitě se předprogramované základní transformace, bez kterých je potřeba napsat pár nezáživných řádků kódu navíc, hodí.

2.3 Neoforce Controls

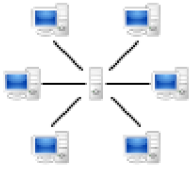
Pokud je hra vytvářena pod Windows, nabízí se možnost udělat herní menu pomocí Windows Forms a všechno nastavení realizovat přes klasickou okenní aplikaci. Toto řešení je vskutku rychlé a snadné, avšak není to to pravé. Toto řešení nevypadá moc profesionálně a následné skákání z enginu hry do aplikace a nazpět nepůsobí na uživatele moc dobrým dojmem. Krom toho, když opomeneme novější technologie jako WPF a XAML, tak WinForms nenabízí moc možností, jak takovou aplikaci graficky vyladit k obrazu svému, a je dost možné, že tak skončí u prostoduchého, v horším případě, šedivého okna klasické aplikace. Jednou z určitě lepších variant je vytvořit ovládání hry, menu a další nastavení přímo v enginu hry, které bude s hrou audiovizuálně souzněné. Nevýhoda takového řešení je jeho složitost a pracná implementace. Každý vlastní prvek je potřeba vykreslovat tak, jako bychom pracovali přímo s modelem nebo obrázkem ve hře. Při tomto řešení je nutné si navrhnout a napsat vlastní komponenty s jejich událostmi a začlenit je do běhu hry. Tento způsob je poněkud zdlouhavý a časově náročný a u hry, která je zde prezentována, i zbytečný. Naštěstí existuje zlatá střední cesta a tou je možnost použít volně šiřitelných grafických knihoven, které se snaží nahradit WinForms přímo v prostředí XNA (potažmo v DirectX). Práce s těmito knihovnami je velice podobná práci s WinForms bez klikacího designeru, kde se definují prvky, které chcete použít, styl jejich vykreslení a přidružení obslužných metod pro interakci s uživatelem. Bezplatných a volně šiřitelných knihoven pro XNA je na internetu více, jednou z nich je právě knihovna Neoforce Controls, kterou napsal Tom Shane [2]. Jako příklad ostatních mohu uvést xWinForms, Nuclex Framework nebo Window System for XNA. Neoforce Controls je výhodnou volbou právě proto, že nabízí o něco pokrokovější funkce než předchozí jmenované knihovny. Jedná se o podporu skinování všech komponent, díky které je snadné odlišit svoji implementaci od ostatních, o automatickou podporu stínů a průhlednosti oken apod. Krom těchto výhod má tato knihovna k dispozici i velkou paletu hotových komponent jako Label, ListBox, ImageBox, CheckBox, RadioButton, Window, TextBox, Dialog a další. Pokud některá komponenta schází, není problém, díky licenční smlouvě a dostupným zdrojovým kódům, podědit některé třídy a udělat si vlastní nadstavbu nebo celou komponentu.

Velkým plusem je opravdu přehledný a strukturovaný kód a hlavně podpora uživatelů na fórech tohoto projektu a tím i celkem pohotový feedback při řešení problémů.

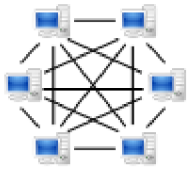
2.4 Síťová komunikace

Tato kapitola představí možná řešení síťové komunikace v prezentované hře. Jsou zde nastíněny výhody toho a toho řešení a nakonec pohled na řešení, která jsou volně k dispozici.

2.4.1 Klient-server vs. peer-to-peer



Při správě síťové komunikace u her, ale i u jiných programů, existují dva majoritní způsoby, jak zprostředkovat komunikaci mezi jednotlivými hráči. První z nich je architektura **klient-server**. V této architektuře existují dva druhy programů a to klient, který je většinou reprezentován kompletním herním enginem, a server, kde se herní engine obejde například bez hudby a grafiky, nebo nemusí být implementován vůbec (vyjma síťové podpory). Mezi klientem a serverem je obvykle stanoven komunikační protokol, pomocí kterého klient zasílá požadavky na server a server po vyhodnocení zformuluje odpověď pro klienta a odešle ji nazpět. V tomto aspektu se vývoj u her může trošku lišit od klasické architektury klient-server, kde aplikace typu klient žádá přístup k tzv. službě, kterou kompletně řídí server (např. webový klient a vzdálený http server). V počítačových hrách tomu tak být nemusí a server může pouze zprostředkovávat komunikaci mezi klienty a do herního světa vůbec nezasahovat. Tato architektura je centralizovaná a umožňuje komunikaci napřímo jen klientovi se serverem. Ostatní spojení jsou realizována pouze přes server. U dnešních her je celkem časté, že hráč vytvoří herní session přímo v jeho instanci hry a ostatní se připojují k němu. Ačkoli to tak nevypadá, ve většině případů se nejedná o nic jiného než právě architekturu klient-server, kde sice hráč založí ve svém klientském programu hru, ale nestane se nic jiného, než že se na pozadí spustí další podprogram (server), který začne pasivně přijímat žádosti od ostatních hráčů včetně zakladatele.



Dalším způsobem je tzv. **peer-to-peer** architektura, kde jsou si všechny koncové body rovny a obstarávají jak funkci serveru, tak klienta. V tomto případě klient založí hru a ostatní hráči se k němu připojí. Dalším krokem je vyhledání hráčů, kteří jsou se mnou ve stejné síti. Tuto informaci lze získat rozesíláním tzv. *discovery messages*, nebo se přímo zeptat připojeného klienta. Poté komunikace probíhá následujícím způsobem. Klient zpracuje svůj herní stav a pošle ho všem ostatním klientům, kteří jsou k němu připojeni. Tito klienti jeho stav přijmou a každý zvlášť odesílá zase svůj herní stav všem ostatním klientům.

Protože každý může mít jiné nároky na svou hru, nebo aplikaci, rád bych shrnul výhody a nevýhody jednotlivých řešení. V posledním odstavci zveřejním důvody mnou zvolené varianty.

Výhody „klient-server“:

- Menší zátěž u jednotlivých klientů
- Snadná autentizace a správa přístupu k serveru (různé nároky kladené na klienty jako verze hry apod.)
- Naprostá kontrola nad provozem dat po síti
- Aktualizace herních stavů probíhá taktéž centralizovaně a není nutné říkat, každému klientovi zvlášť svůj herní stav.
- U her, u kterých je to žádoucí, je umožněna mnohem vyšší a kvalitnější podpora pro „fair play“, protože server uchovává údaje o hráčích a dá se například snadno zamezit tomu, že by v určitou chvíli jeden hráč viděl tři spoluhráče a druhý hráč (díky špatné konektivitě mezi jedním spoluhráčem) pouze dva.
- Celkově vyšší bezpečnost komunikace

Nevýhody „klient-server“:

- Možné přetížení a zahlcení serveru, který bude muset najednou zpracovat mnoho požadavků z různých koutů světa.
- Při výpadku serveru se zhroutí celá síť a klienti nebudou schopni komunikace mezi sebou.
- Náročnější implementace (nutné zavést zvlášť server)

Výhody „peer-to-peer“ :

- Rozložení zátěže mezi všechny klienty. Minimální možnost přetížení sítě.
- Hráč, který hostoval vytvořenou hru, může snadno předat hostování někomu jinému a sám se ze hry odpojit. Ostatní hráči výpadek serveru nemusí vůbec zaznamenat a mohou dále pokračovat ve hře.
- Snazší implementace jednodušších problémů (chat apod.)

Nevýhody „peer-to-peer“ :

- Složitější dolování aktuálních stavů hry ostatních hráčů (např. kolize)
- Nutnost objevování ostatních hráčů a následné udržovat spojení se všemi připojenými hráči ve hře.
- Nízká podpora pro „fair play“ – hráči se nemusí vidět apod.
- O něco větší zátěž přímo u klienta.

Shrnutí všech těchto výhod a nevýhod mě donutilo uchýlit se k architektuře *klient-server*. Složité navazování spojení s každým hráčem, řešení kolizí na každém klientovi zvlášť a zmíněná „fair play“ mě přinutily zavrhnout architekturu *peer-to-peer*, která sice má svá pro, ale já osobně bych ji využil u věcí jako chat, file sharing apod. Navíc je výhodou spravovat připojené uživatele z jednoho místa. Z místa, které má vyšší oprávnění než kterýkoli připojený klient. Centrální správa je také nepostradatelná při distribuci patchů, záplat a nových rozšířeních ke hře. Toto vše může hravě nabídnout architektura *klient-server*.

2.4.2 Lidgren network library

Jak již tu bylo zmíněno, samotný XNA framework obsahuje celkem propracovanou podporu pro síťovou komunikaci – Live!. Nicméně díky nevýhodám, kterým se věnovala předchozí kapitola o XNA Frameworku, jsem se uchýlil k jinému řešení. Jednou z variant by bylo napsat vlastní síťové API, které by nad klasickým UDP nebo TCP protokolem implementovalo (nejlépe víceprocesový) konkurenční server a k němu připojitelné klienty. Řešení postavené na TCP protokolu je sice bezpečné, tj. data, která byla odeslána, skutečně najdou příjemce a jsou obdržena ve stejném pořadí jako byla odeslána. Tento způsob se může zdát vyhovující, ale má určité nevýhody. Při pohledu na hru, která je např. pro 4 hráče, kteří jsou závislí na svých tazích, a kde průměrná uživatelova interakce vznikne po 10 sekundách, je vidět, že toto řešení je dostatečné. Ale při zaměření na některou online válečnou akční hru, kde je na bojišti zároveň 100 lidí a komunikace klienta se serverem probíhá takřka bez ustání, je toto řešení nešťastné. Důvodem je princip fungování TCP protokolu, který při zahájení a ztrátě spojení navazuje tzv. *three-way handshake*, kterým není nic jiného než po sobě jdoucí série ověřovacích packetů a teprve až poté se ustanoví spojení. Udržet toto spojení vyžaduje také jistou režii. Z tohoto hlediska a s faktem, že hlavička TCP packetu je mnohem větší než hlavička UDP packetu, se většina vývojářů přiklání k implementaci s využitím UDP protokolu. Ve 100 hráčích na bojišti s využitím UDP protokolu a v situaci, kdy po sobě střílejí, je jasné, že když se náhodou packet, který aktualizoval pozici vystřelené kulky, neobjeví u cílové stanice a dorazí až následující, tak z celé trajektorie kulky bude chybět malý kousek, který ve výsledku hráč nemá šanci postřehnout. Na tomto příkladu je vidět, že využití protokolu TCP je v tomto případě opravdu zbytečné a značně neefektivní. Samozřejmě i UDP má své nevýhody. Jednou z nevýhod je absence mechanismu, který by zajistil příchod dat ve stejném pořadí, jako byly odeslány. U některých her je toto vyžadováno a je nutné tuto funkcionalitu doprogramovat.

Ve výsledku je tedy výhodné a někdy situací vyžadované implementovat server a klienta nad UDP protokolem s prvky spolehlivého přenosu. Jedná se o vcelku obecný problém, který až tak s tvorbou her nesouvisí. Naštěstí díky tomu, že je tato problematika obecně velice známá a hojně diskutována, existuje několik propracovaných knihoven, které přímo toto řeší. Jednou z nich je právě knihovna pana Michaela Lidgrena [3], která je napsaná pod NET Frameworkem a pomocí UDP protokolu zprostředkovává jednoduché API pro propojení klienta a serveru. Tato knihovna podporuje metody spolehlivého přenosu, které mohou zcela nahradit i TCP protokol. V konfiguraci aktuálního spojení je možné určit, zda se odesílaná zpráva má poslat v režimu spolehlivého doručení, nebo zda má či nemusí být zachováno pořadí odesílaných dat. Mezi další důvody proč použít zmíněnou knihovnu patří kontrola šířky pásma, statistiky spojení, možnost vytvořit peer to peer síť, předimplementované metody pro vyhledání lokálních serverů a testovací systém, který je součástí samotné knihovny a který umožňuje simulovat lagy, ztráty packetů a duplikace dat na síti.

3 Implementace

3.1 Herní jádro

Herním jádrem v XNA Frameworku se rozumí třída Game. Od této třídy je poděděna hlavní třída této hry (třída GameRoot). Výsledek poskytuje vše potřebné ke spuštění XNA enginu, zapouzdření implementace tzv. *herní smyčky* a řízení počtu zobrazených snímku za sekundu (FPS). Ve třídě GameRoot je konstruktor, ve kterém se inicializuje GraphicsDeviceManager (ten se stará o práci s grafickou kartou). Prostřednictvím toho manageru je následně nastaveno pár základních věcí jako preferované rozlišení hry, zobrazení na celou obrazovku, viditelný kurzor apod.. Tato třída obsahuje vlastnost Content (typu ContentManager), které hned v konstruktoru určíme cestu k veškerému hernímu obsahu (modely, textury apod.) Další důležitou metodou v této třídě je Initialize, která, jak jméno napovídá, obstarává inicializaci všech herních komponent, z kterých se hra může skládat, a všech podpůrných tříd (v případě této hry Audio systému). LoadContent je název další metody, uvnitř které dochází k načítání veškerých dat z Content Pipeline prostřednictvím vlastnosti Content a její metody Load. Mezi nejdůležitější metody ovšem patří Update a Draw. V metodě Update se řeší logika hry – zpracování uživatelské vstupu, výpočty pozic objektů, síťová komunikace apod. Metoda Draw je volána ihned po metodě Update, jejím cílem je vykreslení všech objektů na scéně. Při běhu hry se tyto metody volají navzájem dokola, jedna podruhé počínaje metodou Update. Navíc v metodě Draw jsou nastaveny parametry třídy GraphicsDevice, která se stará přímo o vykreslování objektů na obrazovku. Zde je také prostor pro aplikaci různých efektů, shaderů apod. Jedním z profesionálů, kteří se ve své práci zabývají problematikou herního jádra XNA Frameworku, je i Aaron Reed [4].

Všechny jmenované metody jsou virtuální a uvnitř frameworku mají základní implementaci zajišťující chod celého systému. Díky tomuto faktu je nutné v každé metodě spustit i její bázovou implementaci.

Celá hra mohla být implementována přímo ve třídě GameRoot poděděné ze třídy Game, ale při složitějším projektu by takové řešení bylo celkem nepřehledné. XNA nabízí celkem užitečné východisko v podobě herních komponent, z kterých se hra může skládat a které jsem zmínil o kousek výše.Nejedná se o nic jiného než o třídy GameComponent nebo DrawableGameComponent, které obsahují totožné metody jako třída Game (až tedy na třídu GameComponent, která nemá metodu LoadContent, Draw a podobně zaměřené). Kouzlo tkví v tom, že takto vytvořenou herní komponentu je snadné zaregistrovat v inicializační části GameRoot třídy a ta pak automaticky zajistí volání metod jako Initialize, LoadContent, Update v registrované herní komponentě.

3.2 Tvorba úvodního menu

Ačkoli je menu většinou takřka poslední věcí, na kterou se při vývoji hry myslí, její implementace bude popsána právě teď. Celé menu je zapouzdřené do třídy GameMenu, která dědí z DrawableGameComponent, a tedy vystupuje jako herní komponenta. Tato třída je implementována pomocí návrhového vzoru Singleton. V konstruktoru této třídy se nastavuje viditelnost kurzoru na obrazovce pomocí vlastnosti bázové třídy – IsMouseVisible. Dále zde bylo nutné inicializovat Neoforce Controls a to tak, že se vytvořila instanci *Manageru* této knihovny a jako parametry se jí nastavily odkaz na třídu GameRoot, odkaz na třídu GraphicDeviceManager a volba varianty vzhledu s cestou, kde se nacházejí skin soubory těchto komponent. Všechny tyto údaje jsou potřebné k tomu, aby knihovna s komponentami pracovala správně. Samotný *Manager* je vyžadován v konstruktoru každé komponenty NeoForce knihovny a stará se o jejich vykreslování - což znamená, že má uvnitř vlastní implementaci klasické metody Draw. V inicializační metodě jsou následně inicializovány všechny používané komponenty jako Button, PictureBox, TextBox apod. a nastaveny metody pro obsluhu jejich událostí. Ve výsledku se například při stisknutí tlačítka „Nápověda“ zavolá vlastní obslužná metoda ShowHelpMenu, která zařídí viditelnost potřebných prvků a skryje momentálně nepoužívané. Stejně tak pro spuštění enginu hry se jednoduše schovají a deaktivují všechny komponenty menu a zavolá vlastní GameSystemInitialize metoda, která načte potřebné herní části a spustí hru. Výsledné menu poskládané ze dvou obrázků, pár tlačítek a jednoho textboxu vypadá následovně.

Obrázek 1 : Vzhled úvodní obrazovky hry

3.3 Pohyb letadla a kamera

O načtení různých 3D modelů do hry se stará vlastní třída ModelManager. Tato třída je potomkem třídy DrawableGameComponent, je implementována pomocí návrhového vzoru Singleton a zapouzdřuje v sobě kolekci modelů, které jsou manuálně aktualizovány a vykreslovány v metodách Update a Draw. Kolekce modelů je generický list, do kterého je možné vložit modely resp. třídy typu BasicModel a její potomky. Třída BasicModel reprezentuje základní 3D model ve hře. V konstruktoru této třídy je předáván odkaz na načtený 3D model z Content Pipeline a v metodě Draw této třídy následně celý model vykreslen. Vykreslení bude detailně popsáno v další kapitole. Všechny tyto tři matice jsou spolu s trojrozměrným vektorem pozice a *quaternionem* rotace kamery zapouzdřeny ve třídě Camera. Poslední dvě jmenované vlastnosti jsou ve třídě pro ulehčení výpočtu matice *view* při pohybu kamery za letadlem.

Quaternion [5] je rozšíření klasických imaginárních čísel, které umožňuje jiným způsobem zapsat 4 rozměrný vektor. Quaterniony lze vyjádřit pomocí 4x4 matice a i naopak, rotační matice lze vyjádřit pomocí kvaternionu. Pokud chceme docílit složení několika rotací najednou, stačí quaterniony mezi sebou vynásobit. Operace vynásobení rotačních quaternionů bývá několika násobně rychlejší než násobení rotačních matic.

Model letadla představuje třída Aircraft. Ta je potomkem třídy BasicModel a rozšiřuje základní implementaci o informace o munici, zásazích, jménu hráče, rychlosti, palivu, rotaci a pozici letadla. Tato třída pomocí matice, která reprezentuje pozici letadla a quaternionu, kterým je určena jeho rotace, modifikuje matici *world* uvnitř předka (BasicModel) a aplikuje na model transformace při jeho výsledném vykreslení. U takového druhu tříd se přímo vybízí implementovat metody jako „DestroyAirCraft“, která spustí audiovizuální explozi na pozici letadla, doplní výchozí hodnoty všech zásobníků a respawnuje jej na jiném místě.

Protože letadel ve hře lítá spousty, ale ovládat lze jen jeden konkrétní model, musí se od ostatních lišit. Z toho důvodu existuje třída MainAircraft, která reprezentuje letadlo, které je přímo lokálního hráče. V této třídě je přenášena hráčská interakce na model letadla a pohyb kamery. Hlavní funkcionalitu přebírá z nadřazených tříd Aircraft a BasicModel. Navíc rozšiřuje virtuální metodu Update, kde zachytává stisknuté klávesy, aktualizuje pozici a směr letadla i kamery a stará se o ucházející palivo. V následující části je pomocí pseudokódu popsán algoritmus pohybu letadla a zpožděné kamery.

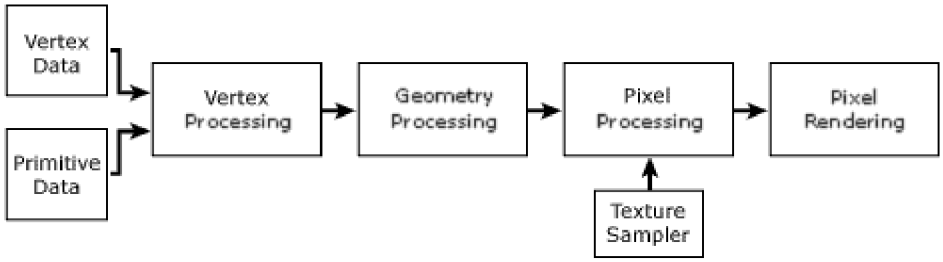
1. Získáme vstup od uživatele
2. Pomocí zvolených konstant a vstupu od uživatele vytvoříme nový quaternion rotace letounu
3. Získaný quaternion vynásobíme se stávajícím rotačním quaternionem letounu a získáme nové natočení letounu.
4. Pomocí konstanty rychlosti a natočení letounu vytvoříme nový prostorový vektor a sečtením tohoto vektoru s aktuálním pozičním vektorem letadla získáme novou pozici letounu.
5. Následně vezmeme quaternion natočení letounu a stávající quaternion natočení kamery a provedeme lineární interpolaci mezi nimi. Dle té jsme schopni určit zpoždění rotace kamery za letadlem.

Pozici kamery získáme vytvořením vektoru, který reprezentuje vzdálenost a úhel, který svírá kamera s letounem. Na tento vektor aplikujeme transformaci rotace dle získaného quaternionu rotace kamery. Nakonec sečteme tento vektor s aktuálním vektorem kamery a získáme novou pozici kamery.

Obrázek 2 : Na obrázku je vidět trajektorie letu letadla s kamerou, která ji zpožděně kopíruje.

3.4 Vykreslování, skybox a tvorba terénu

XNA Framework je založený na kartouzském souřadnicovém systému, kde hlavní roli hrají 3 osy (x,y,z). V 3D prostoru se používá pravidlo *pravé ruky*, tzn. že kladné hodnoty na ose *z* rostou směrem ke kameře a záporné směrem od kamery. XNA podporuje 2 typy projekce. První je tzv. *perspektivní projekce*. Jedná se o nejpoužívanější techniku v FPS hrách pro zobrazení 3D prostoru na monitoru počítače. Záleží zde na vzdálenosti objektu od zobrazované plochy a vzdálenosti kamery od zobrazované plochy. Tyto parametry mají za následek tzv. *scale* zobrazovaných objektů. Ty potom nezachovávají své proporce a jejich obraz je zmenšený. Dalším typem projekce je *ortogonální projekce* . U tohoto zobrazení jsou zachovány všechny proporce zobrazovaného objektu a nezáleží na vzdálenosti objektu od zobrazovací plochy a plochy od kamery. Základním prvkem je *vertex,* což je vrchol trojúhelníku resp. polygonu, z kterého se skládá většina 3D objektů. XNA framework obsahuje metodu DrawPrimitives, která jako parametr dostává hodnotu z enum pole PrimitiveType (PointList - každý bod je vykreslován izolovaně od ostatních, LineList – body jsou vykreslovány v po sobě jdoucích párech a tvoří úsečky, LineStrip – všechny body jsou spojeny do jedné linky, TriangleList – 3 po sobě jdoucí body tvoří trojúhelníky, TriangleStrip – trojúhelníky jsou pospojovány, TriangleFan – trojúhelníky jsou poskládány tak, že sdílejí společný vrchol) a jako další hodnotu VertexBuffer, v kterém jsou uloženy jednotlivé vertexy. Tato metoda je volána v každém průchodu efektu, který je použit pro vykreslení scény. Efekt zde reprezentuje shader model, pomocí kterého grafická karta získává informace o barvách vrcholů, texturách, osvícení apod. Samotný efekt může mít několik průchodů tj. technik vykreslení (EffectPass). Pro použití 3D modelu, který byl vytvořen v některém z modelačních nástrojů, nabízí XNA framework datový typ *Model.* Pomocí tohoto typu a systému Content Pipeline jsme schopni lehce naimportovat model do hry. Poté je model ve hře reprezentován kolekcí *Meshes,* která sdružuje síťovinu modelu (vertexy tvořící celý model). Pro každý mesh je pak volán jeho effect a následuje jeho nastavení. Často se zde vyskytují spojení odvozená od slova „rendering“. Pod tímto slovem se skrývá transformace 3D scény do 2D obrázku. O toto se v XNA stará tzv. *Rendering Pipeline*, kterou popisuje následující obrázek.



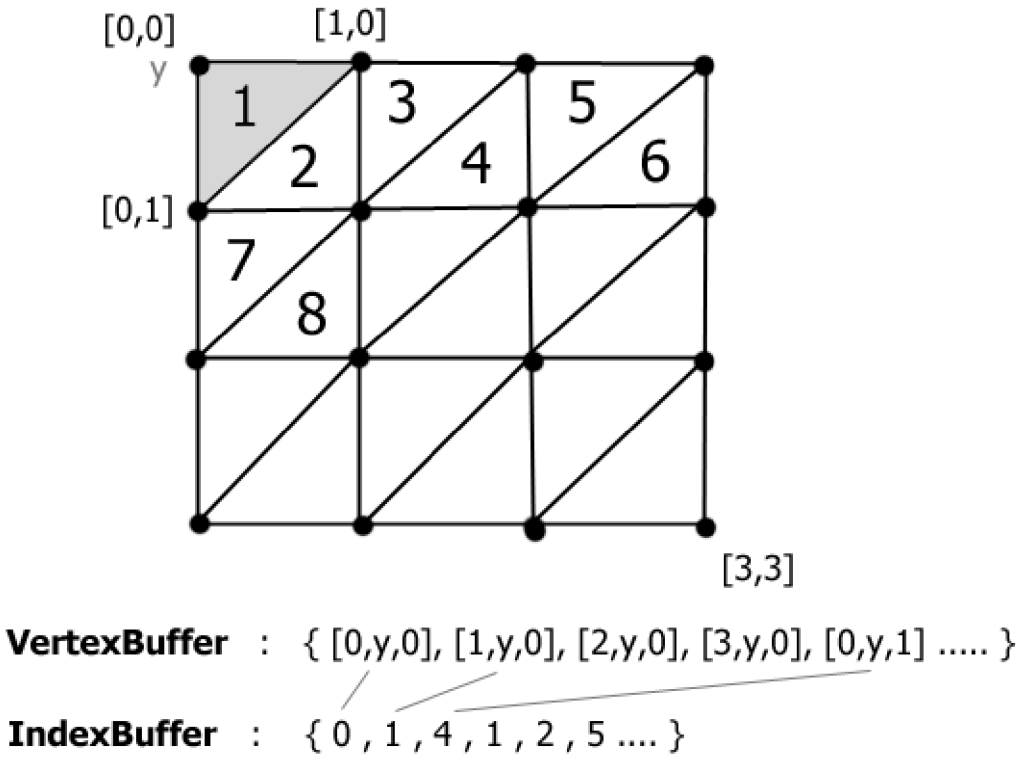
Obrázek 3 : Rendering Pipeline

Vertex Data a *Primitive Data* slouží jako uložiště pro ještě netransformované vrcholy, trojúhelníky, polygony apod. *Vertex Processing* transformuje uložené vrcholy pomocí vertex shaderu použitých efektů. *Geometry Processing* ořízne objekty a zbaví je bodů, které by při pohledu od kamery stejně nebyly vidět, a aplikuje rasterizaci na transformované vrcholy. Pixel Processing a Rendering obstará otexturování objektů, jejich nasvícení a korektní vykreslení na obrazovku.

V případě řešení skyboxu této hry se nejedná o nic jiného než o 3D model velké kostky, na kterou jsou zevnitř aplikovány textury (pomocí UV mapování), které simulují panoramatickou krajinu. Existují i skyboxy ve tvaru koule či polokoule. Toto vše je stvořitelné v 3D modelačním

nástroji jako je 3D studio Max a v kódu následně stačí tento model zvětšit či zmenšit (aplikace transformací na matici *world*) a vykreslit. K vykreslení tohoto modelu je vytvořena vlastní třída Skybox, která je potomkem třídy DrawableGameComponent, a tedy přímo ona může vykreslení provést. Zpracování a renderování tohoto modelu na obrazovku je až na minimální změny stejné jako u zpracování ostatních modelu.

Při tvorbě terénu je zvolena varianta vyžadující tzv. *výškovou mapu*, dle které jsou vytvořeny trojúhelníčky, které se spojují a tvoří různé nerovnosti a zakřivené plochy. Tuto problematiku detailně popisuje na svých webových stránkách například Tomáš Herceq, Microsoft MVP [6]. Výšková mapa je vlastně obyčejný bitmapový obrázek v odstínech šedi. Pokud máme obrázek s 255 stupni šedi, můžeme vytvořit 255 stupňové pohoří. Černá barva zde reprezentuje nejnižší bod na mapě, zatímco bílá je nejvyšší bod na mapě. O vykreslení terénu se stará třída Landscape. Ta prochází po řádcích každý pixel této mapy a určuje jeho výšku (souřadnice osy y), kterou si ukládá do dvourozměrného pole. Indexem tohoto pole jsou hodnoty os x a z. Následně se nabízí dvojice tříd : VertexBuffer a IndexBuffer, kterou obsahuje přímo XNA Framework. Do VertexBufferu jsou uloženy všechny získané body v prostorové reprezentaci. Aby to nebylo pouze seskupení nic neznamenajících bodů v prostoru, nastavuje se i tzv. IndexBuffer, který slouží pro určení toho, které body spolu tvoří trojúhelníky, potažmo celou plochu. Jedná se vlastně o pole čísel - indexů z VertexBufferu. Nebýt IndexBufferu a VertexBufferu museli by se udržovat body všech trojúhelníků a ještě k tomu by se sousedící body mezi jednotlivými trojúhelníky duplikovaly. XNA Framework obsahuje přímo mechanismus na vykreslení těchto trojúhelníků. Následuje podobná procedura renderování scény jako při zobrazení jiných modelů, kde nastavím texturu, její mapovací souřadnice, osvětlení a matice view, projection a world.



Obrázek 4 : Princip ukládání bodů do bufferů

3.5 Zásahy , checkpointy a ostatní kolize

Ve hře jsou vidět dva druhy kolizí. První je kolize letadla s terénem, který je generován a nemá předem daný model. Ve třídě Landscape se nachází metoda GetHeight, která požaduje jako parametry souřadnice na ose x a z a vrací výšku terénu reprezentovanou hodnotou na ose y. V aktuální chvíli není problém zjistit výšku letadla z jeho pozičního vektoru. Pomocí této informace a hodnoty z funkce GetHeight se posoudí, zda se letadlo nachází nad terénem, nebo zda má už explodovat. Všechny kolize jsou testovány ve třídě Collision, která dědí z DrawableGameComponent a vše se děje v její metodě Update.

Dalším typem kolize je kolize typu : letadlo – letadlo, střela – letadlo, checkpoint – letadlo. Checkpoint je 2D průhledný obrázek pozicovaný v 3D prostoru, který je převeden na 2D obraz a vždy otočen k hráči. Tento způsob vykreslení se nazývá *PointSprite* a inspirace spojené s touto technologií můžete nalézt na webu Grootjanse Riemera [7]. Důležité je, že se velikost tohoto obrázku na monitoru mění při změně vzdálenosti kamery. Mechanismus vykreslení obrázku na monitor počítače je podobný jako všechny ostatní vykreslování v XNA Frameworku. Liší se jen v parametrech toho, co se vykresluje a jak se to vykresluje. Jedná se například o nastavení alpha kanálu u obrázků (textury) apod. Dalším způsobem, který se může zvolit při řešení popisovaných situací, je používání tzv. *BoundingSphere* resp. *BoudingBox* tříd. Tyto třídy jsou přímo implementovány v XNA Frameworku a pomáhají vývojářům řešit kolize modelů. Na letadle, nábojích i checkboxech v této hře je aplikován BoundigSphere. Jedná se o neviditelnou kouli, která zaobaluje všechny zmíněné předměty, případně se s nimi i hýbe. Tyto kostky resp. koule overridují metody obvyklé C# metody jako *Contains [8]*, podle kterých jsme schopni zjistit, zda se dvě koule, nebo kostky protínají.

Pro nekomplikovanost sestřelení letadla je na něj aplikována pouze jedna BoundingSphere o odpovídající velikosti střední části letadla. Pro realističtější zpracování by bylo možné vytvořit letadlo z několika menších BoundingSphere a detekovat například zásahy do samotného křídla apod.

3.6 Síťová hra

3.6.1 Serverová aplikace – založení hry

Serverová aplikace je obyčejná okenní aplikace vytvořena pomocí WinForms. Sama o sobě neobsahuje žádnou herní logiku a obstarává pouze komunikaci mezi jednotlivými hráči. Skládá se ze dvou důležitých částí a každá běží ve svém vlastním vlákně. První částí je uživatelské GUI, které umožňuje uživateli založit hru a definovat port, na kterém bude hra naslouchat. Dále umožňuje definovat maximální počet připojených uživatelů a to, zda bude hra vyhledatelná v lokální síti. Po vytvoření hry je možné sledovat připojené hráče ve hře, zasílat jim zprávy, případně je ze hry odpojit. Všechny události jako připojení a odpojení hráče jsou aplikací logovány a zobrazovány v příslušném okně.

Další zmiňovanou částí je třída Network, která běží ve vlastním vlákně, aby její naslouchání příchozím požadavkům nebránilo v ovládání aplikace. Tato třída implementuje již zmiňovanou Lidgren Network knihovnu a komunikační protokol. Prostřednictvím třídy NetServer je vytvořen a nabindován socket pro naslouchání vzdálených připojení. Všechna příchozí a potvrzená připojení knihovna automaticky udržuje a umožňuje listovat mezi připojenými hráči, přijímat od nich zprávy a odesílat zprávy všem připojeným. Po založení herního sezení běží naslouchání v nekonečné while smyčce, která nedělá nic jiného, než že přijme příchozí zprávu, rozpozná její typ a podle domluveného protokolu na ni reaguje.

Obrázek 5 : Úvodní obrazovka serverové aplikace

3.6.2 Vyhledávání a připojení k vytvořené hře

Lidgren network library reprezentuje server třídou NetServer a klienta třídou NetClient. Klientská třída obsahuje metodu Connect, kde jedním z parametru je IP adresa a port vzdáleného serveru – toto je jedna z cest, jak nalézt server a připojit se k němu. Tato možnost je implementována přímo ve hře, kde uživatel může zadat adresu ve formě IP:PORT a zkusit štěstí. Další možností, kterou poskytuje přímo jmenovaná knihovna, je zaslání požadavku na vyhledání lokálních serverů. Tuto funkcionalitu poskytuje metoda DiscoverLocalServer třídy NetClient. Její princip je následující - klient prostřednictvím této metody rozešle po síti UDP packety, které obsahují identifikátor hry (ten je stejný jak u serveru, tak u klienta a nastavuje se při konfiguraci socketu) a požadavek na zaslání odpovědi. Server, který je nastavený jako veřejný a je ve stejné síti (má stejný identifikátor), odpoví UDP packetem, který v hlavičce udává svůj typ a v těle nese strukturu *IPEndPoint*, která obsahuje informace o serveru, tj. o IP adrese, portu atd. V další části navázání spojení klient žádá server o povolení se připojit. Součástí žádosti jsou i tzv. *haildata*, která obsahují uživatelské jméno hráče. V tuto chvíli nemusí dostat klient od serveru povolení k připojení. Může to být z důvodu zaplněného serveru nebo konfliktu uživatelského jména již hrajícího hráče. V případě neúspěchu je klient informován o nedostupném serveru nebo o existenci hráče se stejným jménem.

3.6.3 Komunikace během hry

Na straně serveru je obsluha řešena pomocí nekonečné smyčky, ale u klienta toto obstarává samotná herní smyčka XNA frameworku, která je sice pomalejší než samotný běh while cyklu na pozadí WinForms aplikace, ale pro tyto účely naprosto dostačující a méně náročná. Ve hře je komunikace obstarávána třídou Network. Tato třída je samostatnou herní komponentou a odesílání aktuálního stavu hry a přijímání dat ze serveru je řešeno přímo v metodě Update. V prvním kroku klient odešle svá data o pozici a natočení letounu, informace o skóre a pozici vlastních vystřelených nábojnic. Hned poté se role otočí a klient přijímá stejná data ostatních hráčů od serveru. Situace, kdy server odešle klientovi více dat, která nestačí přijít v jednom zobrazovacím cyklu hry je řešena třídou NetBuffer, kterou poskytuje přímo použitá knihovna. Tato data je pak možné dočíst v následujícím cyklu hry. Při komunikaci mezi klientem a serverem je nutné stanovit komunikační protokol. V této hře je protokol založený na jednoduchém zasílání zpráv různého typu. Typ zprávy je obsažen v její první textové části. Zbylá část obsahuje data příslušná k typu zprávy. Tento protokol obsahuje tyto druhy zpráv:

- **KICKPLANE** – zprávu odesílá server všem klientům. Každý klient zjistí, zda se jedná přímo o něj - v tomto případě se odpojí od serveru, v opačném případě vyhledá letadlo soupeře a vyřadí jej z ModelManageru.
- **MESSAGE** – zprávu může zaslat jak server, tak samotný klient. Výsledkem je zobrazení jejího obsahu u klienta v „chatboxu“.
- **BLOWUPPLANE** – touto zprávou se řeší synchronizace kolizí a výbuchů všech letadel v online módu. Zprávu zasílá klient serveru a ten poté určí, které letadlo má explodovat a to u všech připojených hráčů zároveň.
- **DATA** – tuto zprávu odesílá klient na server a ten poté všem připojeným hráčům. Ve zprávě jsou informace o aktuálním stavu hráče.

4 Shrnutí výsledků a možná rozšíření

Výsledkem této práce není hra s úchvatnou grafikou a propracovanou fyzikou letu, ale slušný základ, pomocí kterého by se takové hry dalo dosáhnout. Cílem této práce byl stručný popis tvorby multiplayerové hry pomocí moderních technologií a NET platformy. Bez implementačních detailů, které jsou viditelné v samotné práci, se tato práce věnovala hlavním částem 3D počítačové hry. Ve skutečnosti určitě existuje více postupů a lepších řešení jednotlivých částí her, které by byly obsahově na samostatnou práci a které by zacházely více do implementačních detailů. To ovšem nebylo záměrem této práce. Tato práce měla za úkol představit dostupné technologie a řešení jednoduché nenáročné síťové hry pro širší veřejnost. Jak jsem zmínil už na začátku, technologie využívané v této práci, většinou nenajdou uplatnění u větších profesionálních studií, kterým jde o absolutní kontrolu nad celým systémem, o implementaci nejnovějších technologií jako DirectX 11 a o optimální výkon. Ačkoli je XNA framework celkem zdařilá technologie pro vývoj her pro rozličné platformy, bez dalších rozšíření a optimalizací bude určena spíše pro menší projekty a herní nadšence.

Nakonec bych uvedl některá možná rozšíření této hry, jako je implementace systému pro načítání různých map - což by se stávajícím návrhem nebyl takový problém. Další inovací by mohlo být rozšíření herních statistik o dobu připojení jednotlivých hráčů. Také by bylo možné zpestřit zbraňový arsenál s různou účinností zásahu a s tím spojená rozšířená detekce kolizí jednotlivých částí letadla jako trupu, křídel apod. Letadlo by tímto mohlo získat i tzv. *identifikátor zdraví* a po zásahu rovnou neexplodovat, ale jen doutnat. K lepší rozlišitelnosti hráčů by se nabízely jmenovky nad letadlem nebo více druhů letadel. V neposlední řadě by bylo možné implementovat i umělou inteligenci v podobě nepřátelských letounů nebo obranných systémů.

Literatura

[1] Riemer Grootjans. *XNA 3.0 Game Programming Recipes.* Apress, USA, 2009.

[2] Tom Shane. *NeoForce Controls.* [online], 2010. [cit. 2010-04-22]. Dokument dostupný na URL http://www.tomshane.cz

[3] Michael Lidgren. *Lidgren Network Library.* [online], 2010. [cit. 2010-04-22]. Dokument dostupný na URL http://code.google.com/p/lidgren-network/w/list

[4] Aaron Reed. *Learning XNA 3.0.* O'Reilly, USA, 2009.

[5] Wikipedia. *Quaternion*. [online], 2010. [cit. 2010-04-22].

[6] Tomáš Herceg. *Generování terénu.* [online], 2008. [cit. 2010-04-22]. Dokument dostupný na URL http://vbnet.cz/clanek--77-xna_2_0_ve_vb_net_dil_3_generovani_terenu.aspx

[7] Riemer Grootjans. *Point sprites.* [online], 2008. [cit. 2010-04-22]. Dokument dostupný na URL http://www.riemers.net/eng/Tutorials/XNA/Csharp/Series2/Point_sprites.php

[8] Christian Nagal a spol. *C# 2005 programujeme profesionálně.* ComputerPress, Czech republic, 2006.

Seznam příloh

Příloha 1 : Použité zdroje a metriky hry

Použité modely, obrázky a skiny

3D model letadla, lodí
vlastní tvorba pro tuto hru. (3D studio Max)
Výšková mapa terénu
vlastní tvorba pro tuto hru (Photoshop CS4)
Textury oblohy
vlastní tvorba pro tuto hru (Photoshop CS4)
Textura povrchu ostrovu
vlastní tvorba pro tuto hru (Photoshop CS4)
Šablona komponent (tlačítka apod.) – výchozí vzhled NeoForce Controls
(URL http://www.tomshane.cz)
Obrázek na pozadí hry – volně dostupná tapeta
(URL http://www.freedesktopwallpapers.net/)

U nejmenovaných částí projektu se předpokládá vlastní tvorba.

Použité zvuky a hudba

Zvuky ve hře (střelba, exploze, motor, příchozí zpráva, checkpointy)
volně dostupné a šiřitelné zvukové zdroje (URL : http://www.freesound.org)
Hudba na pozadí hry
vlastní tvorba (kapela Broken Doors)

Metriky hry

3D model letadla :
počet objektů v modelu : 5 částí
počet vrcholů : 840
počet polygonů : 750
Terén ve hře :
počet vrcholů : 262144
počet polygonů : 522242

Přibližný počet polygonů ve scéně : 522992

Paměťová náročnost hry : cca 33 MB
Paměťová náročnost puštěného serveru bez připojených klientů : cca 5 MB
FPS : závisí na klientském počítači (minimum: graf. karta s podporou DX9 a Shader Model 1.1, Windows XP (Vista,7))

Příloha 2 : Obrázky ze hry