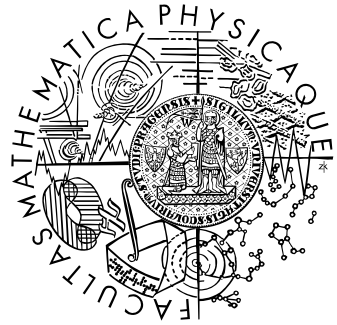
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jan Forch

Grafický engine pro FPS

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Jakub Yaghob, Ph.D.

Studijní program: Informatika, Programování

2010

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne: 20. 5. 2010 Jan Forch

Obsah

Název práce: Grafický engine pro FPS

Autor: Jan Forch

Katedra (ústav): Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Jakub Yaghob, Ph.D.

E-mail vedoucího: Jakub.Yaghob@mff.cuni.cz

Abstrakt: V této práci studuji technologie vývoje počítačových her. Práce je zaměřena především na hry typu FPS (First Person Shooter). Tento typ charakterizuje trojrozměrná grafika. V textu budu rozebírat algoritmy, datové struktury, knihovny a optimalizace, které vedou k úspěšné realizaci počítačové hry. Součástí projektu je i konkrétní implementace hry založená na popsaných řešeních.

Klíčová slova: subsystém, engine, 3D grafika, umělá inteligence, navigace, hra

Title: Graphical engine for FPS

Author: Jan Forch

Department: Department of Software Engineering

Supervisor: RNDr. Jakub Yaghob, Ph.D.

Supervisor's e-mail address: Jakub.Yaghob@mff.cuni.cz

Abstract: The aim of the present thesis are technologies of computer games development. Thesis is mainly focused on FPS (first person shooter) type of games. For that type of games is typical three dimensional graphics. Main topics of that text are related algorithms, structures, libraries and optimalizations, which lead to successful realization of computer game. Important part of the thesis is concrete implementation of game which uses described solutions.

Keywords: engine, 3D graphics, artificial intelligence, navigation, game

1. Úvod

1.1 Téma

Význam herního průmyslu v posledních letech výrazně stoupá spolu se vzestupem obratu v tomto odvětví. Jedním z nejoblíbenějších žánrů her je tzv. **FPS** (First Person Shooter), u které je základem úspěchu kvalitní grafický subsystém neboli „engine“. Dalším důležitým faktorem úspěchu je umělá inteligence nepřátel a interaktivnost objektů ve hře. Pro FPS hru je typická trojrozměrná grafika, akčnost a typický pohled do scény jakoby očima hráče. Cílem práce je prozkoumat současné technologie používané v tomto odvětví a navrhnout grafický subsystém vhodný pro tvorbu FPS. Součástí návrhu bude i prostředí pro podporu umělé inteligence a interaktivnosti prostředí. Navržený subsystém bude demonstrován na konkrétní hře.

V bakalářské práci bych chtěl porovnat přístupy k vývoji FPS her, popsat technologické možnosti, relevantní algoritmy, použitelné knihovny a strategie vytváření autonomních agentů, které mě nejvíce zaujaly. Nabyté vědomosti a postupy budou demonstrovány na konkrétní aplikaci. Při vývoji hry se zaměřím především na výběr nejvhodnější varianty řešení parciálních problémů a na diskusi jejich detailu. Slovo „nejvhodnější“ zde vystihuje poměr mezi efektivitou, realizovatelností a čitelností zdrojového kódu. Nakonec zhodnotím výběr řešení z pohledu praktických zkušeností během vývoje a po dokončení aplikace. Věnuji se také možným rozšířením do budoucna a uplatnění v praxi.

Dnešní doba klade na programátory velké nároky. Technologie se stále vyvíjejí, inovují, stávají se více komplexními a stále více přejímají poznatky z oblasti vědy a výzkumu. Oblast vývoje her je jednou z nejvíce ovlivněných oblastí. Trh počítačových her je silně konkurenčním trhem. Jakákoliv výhoda nad konkurencí se silně odráží na komerčním úspěchu či neúspěchu projektu. Základními pilíři moderní FPS hry jsou grafika, realističnost prostředí, umělá inteligence nepřátel a dobrý nápad.

Kvalita grafiky je ovlivněna hlavně použitou technologií. Dnes jsou pro tento účel ve většině komerčních projektů využívány knihovny DirectX (zaštítěny firmou Microsoft) a OpenGL. Každá z těchto knihoven má svá specifika a možnosti. Více využívaná je dnes knihovna DirectX. Kvalita 3D (trojrozměrných) modelů objektů a textur zobrazovaných ve hře je také důležitým prvkem. Pro tyto účely je nejčastěji využíván program 3D Studio Max a jiné jemu podobné grafické nástroje pro tvorbu trojrozměrných grafických modelů.

V této práci se budu zabývat použitím grafických knihoven ve vlastním projektu a zapracováním zdrojů jimi poskytnutých do objektového návrhu hry tak, aby bylo dosaženo cílů stanovených ve specifikaci při maximální efektivitě běhu programu. Dále bych rád prozkoumal možnosti návrhu a implementace umělé inteligence. Za důležitou přitom považuji i konzistenci s objektovým modelem hry a vizuálním výstupem na obrazovku.

Právě technologie trojrozměrné grafiky a umělé inteligence považuji za jedny z nejvíce perspektivních. Prostorová grafika umožňuje simulovat dění reálného světa na obrazovce počítače s velkou věrohodností. Myslím si, že 3D grafika a algoritmy spojené s jejím použitím mají velký potenciál do budoucna. V příštích letech očekávám další rozvoj jejího uplatnění v praktickém životě lidí. 3D grafika se spolu s umělou inteligencí zatím objevuje zejména v počítačových hrách. Dokážu si však představit budoucí potenciál rozvoje těchto technologií v architektuře, vojenství, simulacích, sociologických průzkumech, dopravě, vesmírném výzkumu a zábavním průmyslu.

Zajímavou vlastností postupů použitých v trojrozměrném světě programů je podobnost s postupy v reálném světě. To značně zvyšuje možnost budoucího rozšíření těchto algoritmů i mimo počítačový prostor, například v průmyslu a robotice. I když se může běžnému člověku jevit téma mé bakalářské práce značně abstraktní a vytržené z reality, věřím, že po přečtení úvodu, je již snadnější pochopit praktický dopad a uplatnění mé bakalářské práce.

Právě zmiňované výhody a potenciál výše zmíněných technologií jsou důvodem, proč jsem se o ně začal zajímat a proč se staly námětem bakalářské práce. Má práce by měla ukázat přístup k chápání a používání těchto moderních technologií včetně jejich praktického užití. Praktickým projektem „Valahalla“ vytvořeným v souvislosti s bakalářskou prací budu demonstrovat možnosti, omezení a algoritmy spojené s trojrozměrnou grafikou a navigací. Tento projekt je také určitou motivací a osnovou pro studium tohoto tématu.

1.3 Organizace textu

Obsah bakalářské práce postupuje od technologií a použitých algoritmů, ke konkrétním řešením na návrhové úrovni, až po vysvětlení implementací jednotlivých dílčích problémů. Práce začíná představením grafické knihovny DirectX, která slouží pro vykreslování, a jejím srovnáním s konkurenčními produkty. Dále budou představeny stěžejní algoritmy a datové struktury použité v projektu následované zamyšlením nad použitou architekturou

a použitím knihoven DirectX pro zvuk a ovládání hry. Zároveň budou rozebrány i návrh a možná řešení umělé inteligence. Na konci práce se objeví srovnání s komerčními projekty a zamýšlení nad splněním cílů a možná další rozšíření. Jako příloha bude připojena uživatelská příručka a organizace multimediální přílohy bakalářské práce.

2. DirectX

2.1 Představení

Stručný popis DirectX a jeho součástí jak je uvedena na stránkách [12]:

„Microsoft DirectX je v informatice sada knihoven poskytujících aplikační rozhraní (API) pro umožnění přímého ovládání moderního hardwaru. Jejich cílem je maximální využití možností hardware jak po stránce nabízených funkcí, tak z hlediska maximálního výkonu, což je využíváno pro tvorbu počítačových her, multimediálních aplikací i grafického uživatelského prostředí (viz Windows Aero).

Jak z názvu vyplývá, je DirectX produktem firmy Microsoft určeným výhradně pro operační systém Windows. Alternativním multiplatformním rozhraním je OpenGL, který je možné používat jak v Microsoft Windows, tak v Mac OS X, Linuxu a dalších operačních systémech.

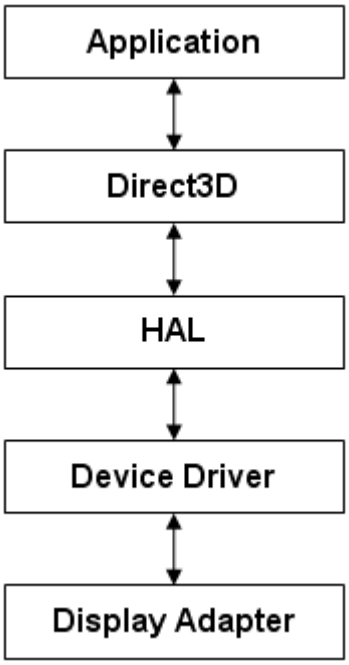
DirectX se skládá z několika částí, rozdělených podle svého účelu. Rozdělení je podstatné pouze pro programátory, protože je k dispozici jako souhrnný balík obsahující všechny komponenty. “

- **DirectX Graphics** (a jeho části **DirectDraw** a **Direct3D**) – podpora grafiky, 3D vykreslování atd.,
- **DirectInput** – podpora vstupních zařízení jako např. myši, joysticky, gamepady apod., včetně podpory technologie force feedback,
- **DirectPlay** – podpora hry více hráčů po síti,
- **DirectSound** (dříve též spolu s DirectMusic označováno souhrnným názvem **DirectX Audio**) – podpora přehrávání a záznamu zvuků,
- **DirectMusic** – podpora přehrávání a zpracování hudby,
- **DirectShow** – podpora multimediálních aplikací, přehrávání a zpracování videa a zvuku,
- **DirectSetup** – jednoduchý nástroj umožňující instalaci knihovny DirectX na počítač,
- **DirectX Media Objects** – podpora pro tvorbu multimediálních efektů, kodeků apod.

Z vyjmenovaných součástí jsou v subsystému použity DirectX Graphics, DirectInput a DirectSound. DirectX umožňuje použití svých knihoven v jazycích C++ a C# prostřednictvím „Managed DirectX“. Většina profesionálních projektů z herního průmyslu však používá verzi DirectX pro C++ spolu s tímto jazykem. Důvodem je možnost lepší optimalizace rychlosti aplikace psané v C++.

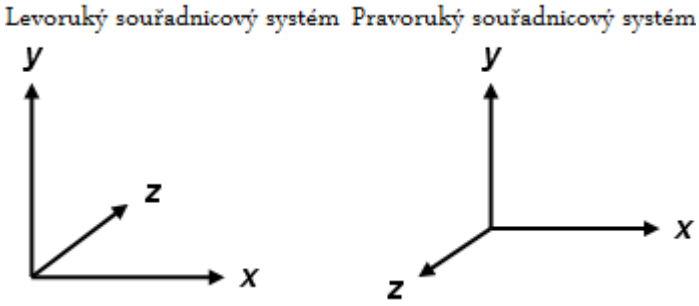
DirectX programátorům zprostředkovává velké množství hotových funkcí, interfaců, tříd a šablon. Tyto zdroje řeší za programátora časté úlohy z oblasti matematiky, přístupu k hardwaru počítače, zobrazování grafiky apod.

Důležitou stránkou hratelnosti hry je rychlost, s jakou běží na uživatelově PC. DirectX se snaží vyjít tomuto požadavku maximálně vstříc. Snaží se dosáhnout optimalizace použitých algoritmů již na úrovni architektury a komunikace s Windows. Tuto architekturu je možné vidět na obrázku č. 1. Skládá se z několika vrstev. První vrstvu tvoří samotná aplikace komunikující s vrstvou Direct3D, která je součástí DirectX a je zodpovědná za grafiku. Direct3D je API (Application Programming Interface) používané k vykreslování grafických primitiv na obrazovku počítače. Další vrstvou je vrstva HAL (Hardware Abstraction Layer). HAL je na konkrétním grafickém zařízení závislé rozhraní poskytované výrobcem. Každý grafický adaptér má své specifické rozhraní zpřístupňující jeho funkce a zdroje. HAL implementuje všechny funkce definované v Direct3D, které je příslušný adaptér schopen vykonat. Vrstva HAL vrací do Direct3D informace o schopnostech a vlastnostech použitého zařízení. Direct3D komunikuje přímo s tímto rozhraním a tím umožňuje dosáhnout maximální možné rychlosti zobrazování. Není tudíž nutné používat GDI (Graphic Device Interface), grafickou vrstvu Windows. Je totiž poměrně pomalá. Direct3D poskytuje programátorům prakticky úplnou kontrolu nad grafickým zařízením.



Obrázek č. 1 - Model architektury použití Direct3D.

DirectX používá pro umísťování objektů ve trojrozměrné scéně kartézský systém daný třemi souřadnicovými osami x, y, z. DirectX používá tzv. levoruký souřadnicový systém. Na obrázku č. 2 je zobrazen nejen levoruký souřadnicový systém, ale také jeho srovnání s pravorukým souřadnicovým systémem.



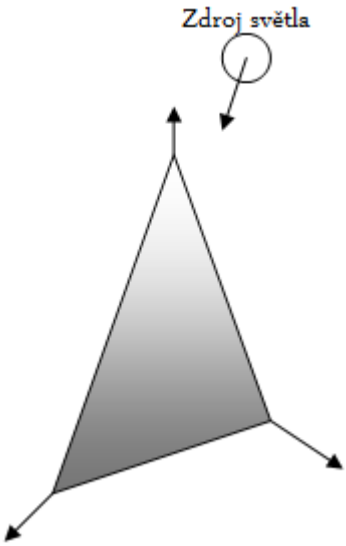
Obrázek č. 2 - Levoruký a pravoruký souřadnicový systém.

Grafická karta je schopna vykreslovat jen některá grafická primitiva. Z těchto primitiv se na aplikační úrovni skládají složité scény, obrázky, trojrozměrné objekty apod.

Seznam primitiv

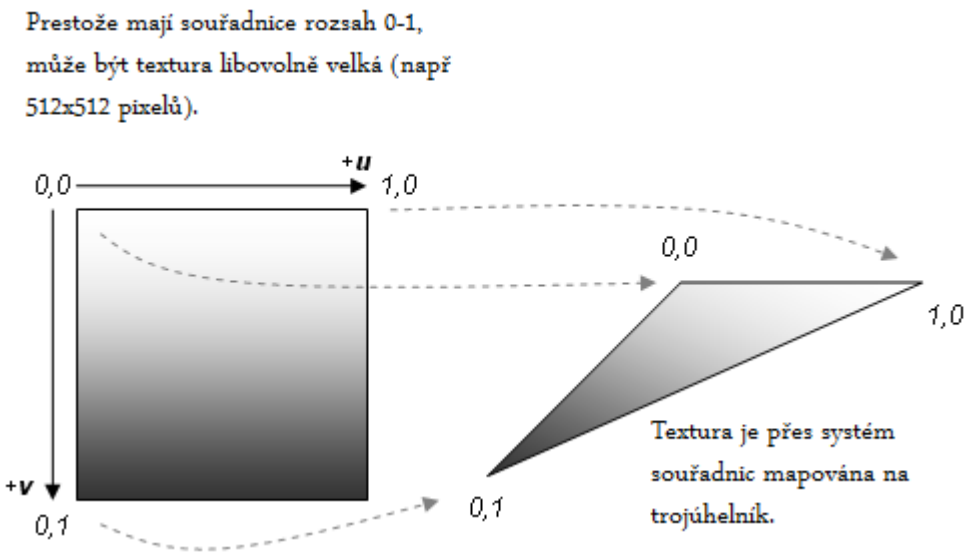
- body (pixely),
- čáry (resp. úsečky dané počátečním a koncovým bodem),
- trojúhelníkové plošky,
- vějíře trojúhelníků atd.

Trojúhelníkové plošky jsou nejpoužívanějším primitivem, se kterým se pracuje. Jelikož žádná další primitiva nejsou definována, můžeme nahlédnout jeden ze základních problémů programování trojrozměrných světů. Tímto problémem je to, že vše ve scéně musí být složeno z primitiv. Pokud tedy máme nadefinovanou mapu či 3D model, je bezpodmínečně nutné převést komplexní tvary (modely) do formátu seznamu trojúhelníků ještě před jejich zobrazením. Jen v tomto formátu je totiž umí DirectX zobrazit.



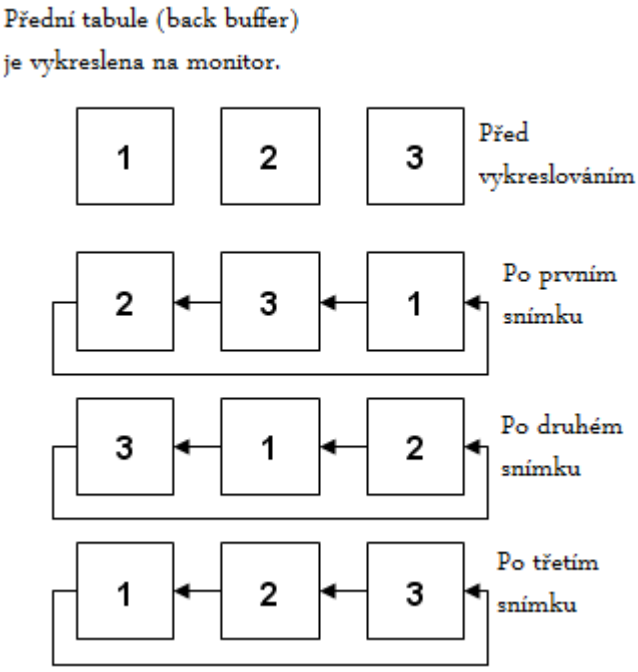
Obrázek č. 3 - Trojúhelníková ploška tvořená vrcholy (vertexy).

Před vykreslením je potřeba dodat grafickému adaptéru spolu s trojúhelníky i materiál, kterým je trojúhelník pokryt. Materiálem rozumíme texturu spolu s dalšími prvky, jako jsou odstín, odrazivost různých druhů světel, emitace světla apod. Nastavení materiálu výrazně ovlivňuje zobrazený výsledek. Textura materiálu je většinou získávána nahráním nějaké bitmapy ze souboru (.bmp, .gif, .jpg, .dds, atd.). Textuře je před použitím nutné nastavit mapování na vykreslovaný trojúhelník. Spolu s trojúhelníkem se tedy do seznamu vykreslovaných plošek uloží i souřadnice textury. Způsob mapování demonstruje obrázek č. 4.



Obrázek č. 4 - Mapování textury na trojúhelníkovou plošku.

Aby bylo vykreslování vytvořené scény plynulé, umožňuje DirectX použít tzv. „multibuffering“. Jeho principem je vytvoření několika stejných datových struktur, grafických tabulí v paměti, reprezentujících displej počítače, které se cyklicky střídají ve vykreslování na plochu monitoru. Typická frekvence zobrazení snímků animované scény hry je řádově v desítkách či stovkách snímků za sekundu. Vykreslování probíhá tak, že zatímco se vykresluje tabule č. 1, tabule č. 2 se plní grafickými daty o novém snímku. Tabule č. 2 se pak může zobrazit po dokončení zobrazení tabule č. 1 celá naráz. Tím je dosaženo plynulejšího zobrazování snímků animace. Typický počet tabulí je 2 nebo 3. Princip zobrazování tabulí je možno vidět na obrázku č. 5.

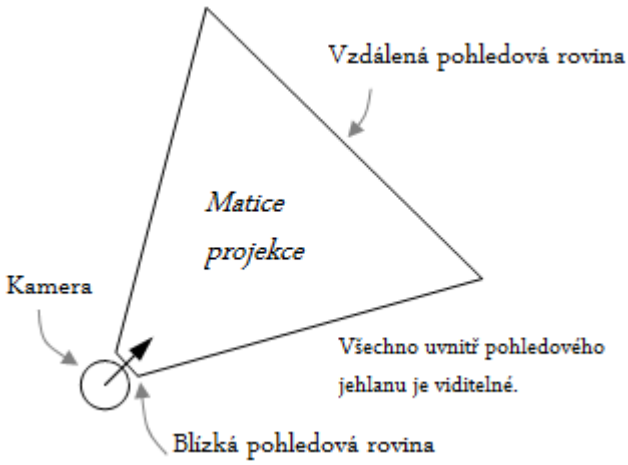


Obrázek č. 5 - Schéma cyklického střídání tabulí pro vykreslování při použití tzv. „multibufferingu“.

Scéna, ve které se hráč či postava pohybuje, je tvořena tisíci až milióny trojúhelníkových plošek. Přirozeně tedy existuje řada optimalizačních technik, jak na aplikační vrstvě, tak na vrstvě Direct3D, které umožňují s takovým počtem plošek efektivně pracovat. Hlavním cílem je zúžit okruh zpracovávaných trojúhelníkových plošek na minimum.

První technika, kterou zmíním, je použití matice projekce. Jedná se o optimalizaci na úrovni vrstvy Direct3D. Jejím principem je vytvoření matice, která po aplikaci na seznam všech trojúhelníků, vybere ty, jež mohou být viditelné pro postavu hráče s přihlédnutím k jejímu aktuálnímu umístění a natočení. Tuto matici za nás vytvoří jedna z funkcí DirectX po zadání parametrů viditelnosti vpravo a vlevo. Dále je nutné zadat vzdálenost blízké a vzdálené

roviny viditelnosti. Cokoliv, co padne do tohoto „trojúhelníku“, je zobrazeno. Cokoliv vně viditelné nebude.



Obrázek č. 6 – Tvar, jenž reprezentuje matice projekce, ovlivňuje oblast vykreslování grafiky.

Ve hrách se často objevují trojrozměrné modely různých objektů. Hotovému předpřipravenému 3D modelu nějakého objektu se říká **mesh**. Tento název byl přejat z angličtiny a běžně se používá v české literatuře (např. [6]). Meshe jsou obvykle uloženy v souboru s příponou „.x“. V aplikaci je možné je nahrát do paměti pomocí knihoven DirectX a poté zobrazovat jako součást scény. Takto je možné získat modely hráčů, zbraní, lékárniček, různých dekorací apod.

2.2 Popis instalace DirectX do projektu Visual Studia 2008

Před použitím DirectX v aplikacích je nutné mít nainstalovány DirectX na lokálním počítači. Moderní Windows jako například Windows Vista mají DirectX nainstalovány standardně jako svou součást. Kvůli dostupnosti opravdu všech možných použitých dynamických knihoven je však nutné, mít nainstalovánu poslední verzi DirectX, která je volně ke stažení například na oficiálních stránkách společnosti Microsoft nebo na DVD přiloženém k bakalářské práci.

Pro potřeby vývoje aplikace je nutné mít na počítači nainstalovany DirectX distribuované ve verzi DirectX SDK (Service Developer Kit). DirectX SDK obsahuje kromě knihoven i ukázkové příklady, dokumentaci apod. Praktický projekt k této bakalářské práci „Valahalla“ (pracovní název aplikace grafického subsystému) již obsahuje všechny nutné knihovny. Jsou umístěny v adresáři projektu v podadresáři s názvem „DXSDK“. Valahalla je projekt napsaný

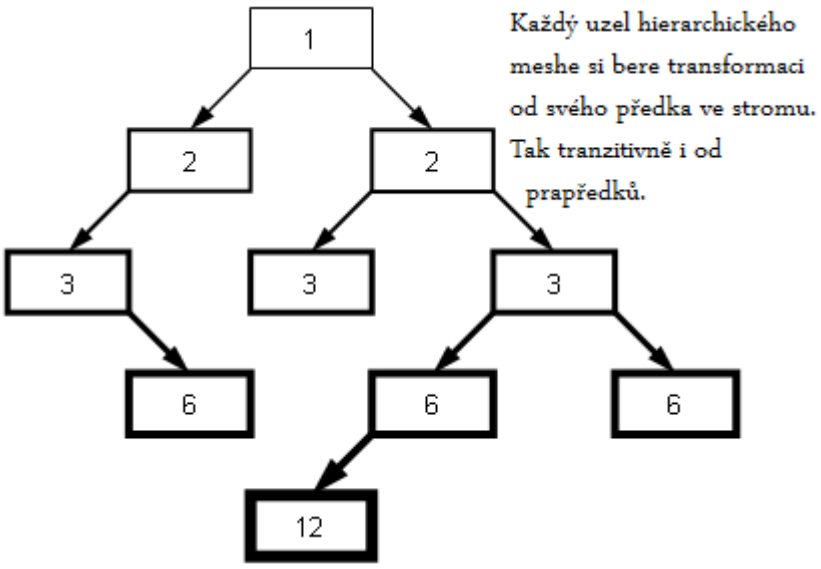
ve Visual Studiu 2008 programovacím jazykem C++. Knihovny ze zmiňovaného adresáře bylo nutné do aplikace importovat v nastavení projektu.

2.3 Použité prvky DirectX

Projekt využívá jen určitou podmnožinu funkcí, které DirectX nabízí. Aplikace používá knihovnu DirectX k získávání vstupu uživatele z klávesnice a myši, ozvučení hry, vykreslování grafických primitiv (konkrétně trojúhelníků tvořících model mapy), nahrávání předpřipravených zdrojů ze souboru (meshe, materiály, zvuky atd.) a nastavení chování grafického adaptéru.

Modely hráčů jsou tvořeny tzv. **hierarchickým meshem**. Jde o trojrozměrný model rozdělený na více částí do stromové struktury. Je používán k animovanému pohybu meshe. Model se nemusí pohybovat celý, může se pohybovat jen některá jeho část (např. končetina) reprezentovaná konkrétním podstromem. Hierarchický mesh se nahrává ze souboru podobně jako klasický mesh, jen se nahrává do stromové datové struktury. Část meshe, uložená v některém uzlu stromu, je vždy závislá na pohybu svého rodiče a její pohyb zase ovlivňuje její potomky.

Pohyb modelů těles v prostoru se realizuje násobením souřadnicových vektorů **transformační** maticí, kde vektory reprezentují vrcholy vykreslovaných trojúhelníků. Tak se provádí přesunutí (transformace) celého trojúhelníku. Bližší informace o transformacích jsou uvedeny v kapitole č. 3 věnující se algoritmům.



Obrázek č. 7 - Stromová struktura hierarchických meshů.

2.4 Srovnání s OpenGL

Hlavní konkurencí DirectX je OpenGL. Obě technologie fungují na podobném principu. Jsou zde však rozdíly. Stručný popis OpenGL, jak je uveden na stránkách [15]: „OpenGL (Open Graphics Library) je průmyslový standard specifikující multiplatformní rozhraní (API) pro tvorbu aplikací počítačové grafiky. Používá se při tvorbě počítačových her, CAD programů, aplikací virtuální reality či vědeckotechnické vizualizace apod. Standard OpenGL spravuje konsorcium označované jako ARB (Architecture Review Board), jehož členy jsou firmy jako např. SGI, Microsoft, nVidia, ATI atd. Implementace OpenGL existují pro prakticky všechny počítačové platformy, na kterých je možno vykreslovat grafiku. Kromě implementací vestavěných v grafickém hardware (na grafické kartě) existují také softwarové implementace, které umožňují používat OpenGL i na hardwaru, který ho sám o sobě nepodporuje (ale obvykle nabízejí nižší výkon). Příkladem takové implementace je open source knihovna Mesa, která ovšem z licenčních důvodů nemůže být označena jako implementace OpenGL, ale pouze jako implementace API, které je „velmi blízké“ OpenGL. Základní funkcí OpenGL je vykreslování do obrazového rámce (framebufferu). Umožňuje vykreslování různých základních primitiv (bodů, úseček, mnohoúhelníků a obdélníků pixelů) v několika různých režimech. Veškerá činnost OpenGL se řídí vydáváním příkazů pomocí volání funkcí a procedur (kterých OpenGL definuje cca 250). V OpenGL se nepoužívá objektově orientované programování. Jednotlivá primitiva jsou definována pomocí vrcholů. Každý z nich definuje bod, koncový bod hrany nebo vrchol mnohoúhelníku. Každý vrchol má přiřazena data (obsahující souřadnice umístění bodu, barvy, normály a texturovací souřadnice). Rozhraní OpenGL je založeno na architektuře klient-server. Program (klient) vydává příkazy, které grafický adaptér (server) vykonává. Díky této architektuře je možné, aby program fyzicky běžel na jiném počítači než na tom, na kterém se příkazy vykonávají, a příkazy se předávaly prostřednictvím počítačové sítě.“

Před implementací projektu Valahalla jsem se rozhodoval, zda použít k vykreslování technologii DirectX, nebo OpenGL. Velmi silným argumentem pro OpenGL byla možnost jejího použití na více platformách. Nakonec jsem se však rozhodl pro DirectX. Většina komerčních herních projektů dnešní doby totiž používá právě DirectX. Důvodem je hlavně podpora společnosti Microsoft. Její operační systém Windows je totiž stále nejpoužívanější herní platformou pro osobní počítače. Díky finanční podpoře společnosti Microsoft se DirectX stále vyvíjejí kupředu. Aktuální verze má označení DirectX 11. Je součástí Windows 7.

3. Použité algoritmy a struktury

3.1 Uplatněné návrhové vzory

Návrhové vzory jsou tématem a velmi důležitým pojmem ve světě objektově orientovaného programování. Tyto vzory jsou výsledkem diskuze a praktických zkušeností programátorů s řešeními často se opakujících problémů. Programátorům poskytují elegantní řešení objektového návrhu aplikací. Vzor je typicky pojmenován podle problému, který řeší a je tvořen spojením tříd, interfaců a struktur v hierarchii. Obvykle je znázorňován UML diagramem. Kromě nalezení elegantního řešení umožňuje programátorům i rychlejší a přesnější komunikaci založenou na popisu problému, o kterém je řeč pouze jedním slovem, názvem konkrétního vzoru. Jednotlivé návrhové vzory mají spoustu tvarů a modifikací. Je nutné volit ten pravý podle konkrétních požadavků. Několik návrhových vzorů nalezlo své uplatnění také v projektu Valahalla. Věřím, že jejich použití přispělo ke zvýšení kvality zdrojových souborů programu.

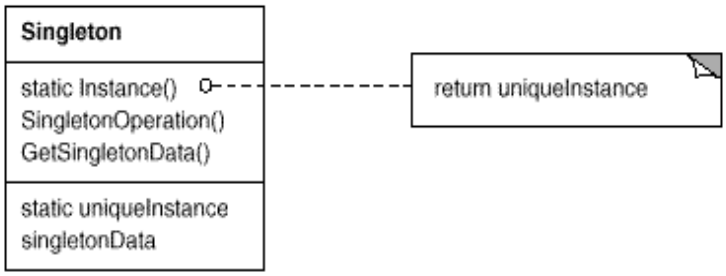
Singleton

Singleton je návrhový vzor, který zaručuje, aby měla třída pouze jednu instanci. K instanci poskytuje globální přístupový bod. V mém programu je použita tzv. Scott Meyersova verze singletonu. Nabízí automatickou destrukci objektu singletonu po ukončení programu. Ideou je použít na místo dynamické alokace funkci vracející ukazatel na statický objekt ve funkci. Jejich použití umožňuje budovat úspornější interfacy a snižovat počet kopií ukazatelů na nějakou často využívanou instanci třídy, o které se ví, že bude pouze jedna. Na druhou stranu se nedoporučuje přílišné používání tohoto vzoru, jelikož s jeho nadměrným užitím se z objektového návrhu může ztrácet přehlednost, logická souvislost mezi jednotlivými třídami, hierarchie apod. V projektu Valahalla je Singleton použit na hlavní třídu aplikace „Engine“ a stavy umělé inteligence.

Obrázek č. 8 - Scott Meyers

Implementace a schéma:

```
class Single
{
public:
 static Single& Inst();
private:
 ~Single() { /*uklid*/ }
 Single() { /*inicializace*/ }
 /*nejakadata*/
};
Single& Single::Inst() {
 static Single inst_;
 return inst_;
}
int main() {
 Single& s = Single::Inst();
 return 0;
}
```

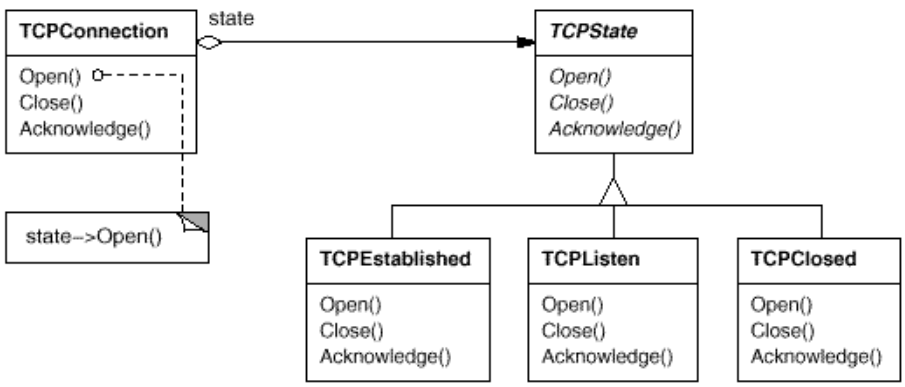


Obrázek č. 9 - Schéma návrhového vzoru Singleton.

Příklady použití: globální přístupový bod k hlavní třídě aplikace (třída Engine).

State

Tento návrhový vzor umožňuje objektu měnit svoje chování v závislosti na jeho stavu. Stav objektu se mění za běhu programu. Je použitelný tam, kde metody vykonávající jednotlivé funkce objektu obsahují větvení v závislosti na nějaké sadě výčtových proměnných. Namísto rozsáhlé konstrukce switch se použije polymorfismus.

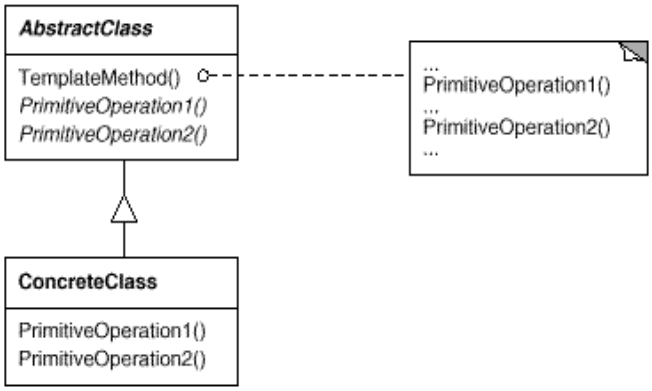


Obrázek č. 10 - Schéma návrhového vzoru State.

Příklady použití: editační a herní stavy, stavy umělé inteligence agentů

Template method

Definuje kostru algoritmu a nechává některé kroky implementovat podtřídami. Dovoluje podtřídám měnit některé části algoritmu beze změny struktury samotného algoritmu. Zamezuje vzniku duplicitního kódu. Je ji možné použít v případech, kde jsou v podtřídách dvě metody, které provádějí podobné kroky ve stejném pořadí, nicméně kroky nejsou stejné.



Obrázek č. 11 - Schéma návrhového vzoru Template method.

Příklady použití: vytváření stromové struktury hierarchických meshů

Pro detailnější informace o použitých návrhových vzorech doporučuji prezentaci [11] a knihu [4].

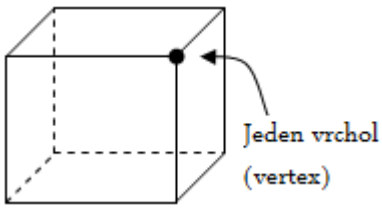
3.2 Transformace souřadnic

O souřadnicích byla zmínka již v kapitole č. 2 o DirectX. Využitím souřadnic můžeme definovat umístění bodu ve trojrozměrné mřížce kartézského systému souřadnic. Kartézský systém se používá k umísťování objektů ve scéně. Souřadnice bodu jsou dány vektorem [x, y, z]. Body se ve smyslu použití v Direct3D nazývají **vertexy**. Vertex znamená v angličtině vrchol. Vertexy se používají k definování tvaru trojrozměrných modelů. **Vertex buffer** je převzaté pojmenování pro pole vertexů.

Vykreslování grafiky na monitor uživatele probíhá zjednodušeně řečeno v těchto krocích:

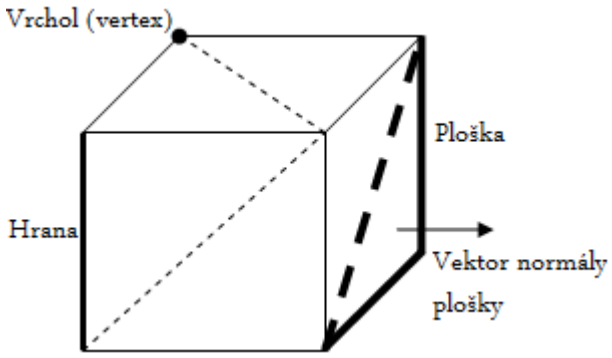
1) Pomocí rozhraní Direct3D nastavím zdroj geometrických dat pro vykreslení grafickou kartou (vertex buffer).

2) Určím materiál a typ vykreslovaných primitiv (většinou trojúhelníkové plošky).
3) Poté se zavolá funkce pro vykreslení na monitor.



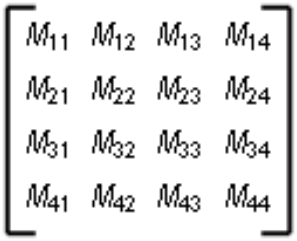
Obrázek č. 12 - Role vrcholů (vertexů) při definici geometrie 3D modelů.

Pokud jsou jako typ vykreslovaných primitiv nastaveny body, bere grafická karta z pole vrcholů jeden bod po druhém a vykresluje ho jako jeden pixel na monitor. Pokud je tímto typem čára, berou se vrcholy po dvojicích a vykreslují se na monitor jako čáry. Pokud se jedná o trojúhelníkové plošky, berou se vrcholy po trojicích. Každá trojice reprezentuje vrcholy trojúhelníku, které tvoří vykreslovanou **plošku**.



Obrázek č. 13 - Vrchol, hrana a trojúhelníková ploška.

Častými úkony při řízení logiky hry jsou přemísťování, otáčení a změna velikosti (transformace) 3D modelu herního objektu, které probíhají ještě před jeho vykreslením. Jelikož je geometrie modelů tvořena vrcholy, je možné použít k jejich **transformaci** v prostoru kartézského systému souřadnic lineární algebru. Transformace vrcholu se provede násobením vektoru souřadnic vrcholu maticí ve speciálním tvaru. K transformaci souřadnic vrcholu se používají dvě matice s rozměry 1 x 4 a 4 x 4 pole. Rozměry matic jsou typické pro trojrozměrný prostor (pro srovnání pro 2D prostor by to byly matice 1 x 3 a 3 x 3). Matice 1 x 4 reprezentuje vektor souřadnice vrcholu ve tvaru [x, y, z, 1]. Jednička na konci se používá jen pro matematické účely. Matici 4 x 4 se říká **transformační matice**.



Obrázek č. 14 - Schéma matice 4 x 4 používané k transformacím souřadnic.

Tvar transformační matice se mění v závislosti na tom, jaké transformace souřadnic chceme dosáhnout. Existují tři druhy transformací. Posunutí, otočení a změna velikosti. Každé transformaci odpovídá speciální tvar matice. Násobením vektoru transformační maticí se dosáhne žádané operace – tzn. posunutí, otočení či změny velikosti souřadnic vektoru. Tato zobrazení je možno skládat, tzn. násobit mezi sebou matice požadovaných transformací a výsledek použít na komplexní transformaci souřadnic vrcholu. Knihovny DirectX obsahují funkce, které v sobě obsahují implementaci transformací. Není tedy nutné znát maticové operace, které jsou v pozadí těchto funkcí. Poloha jednotlivých objektů v prostoru vzhledem ke kameře (pohled hráče) a jejímu natočení ovlivňuje, zda je objekt viditelný (podobně jako v reálném světě).

Posunutí

1	0	0	0
0	1	0	0
0	0	1	0
posunX	posunY	posunZ	1

Změna velikosti

změnaX	0	0	0
0	změnaY	0	0
0	0	změnaZ	0
0	0	0	1

Otočení

Transformace souřadnic otočením nemá jednu univerzální transformační matici. Otočení kolem každé z os trojrozměrného kartézského systému se musí provést zvlášť. Otočení kolem jednotlivých os mají svou vlastní matici otočení. Pro obecné otočení je potřeba násobit vektor souřadnic všemi potřebnými osovými maticemi otočení. Obecné otočení se tak provede jako kombinace elementárních (osových).

Matice otočení kolem osy x

1	0	0	0
0	cos(radiany)	sin(radiany)	0
0	-sin(radiany)	cos(radiany)	0
0	0	0	1

Matice otočení kolem osy y

cos(radiany)	0	-sin(radiany)	0
0	1	0	0
sin(radiany)	0	cos(radiany)	0
0	0	0	1

Matice otočení kolem osy z

cos(radiany)	sin(radiany)	0	0
-sin(radiany)	cos(radiany)	0	0
0	0	1	0
0	0	0	1

Příklad

Fungování transformace ukážu na jednoduchém příkladu. Vrchol umístěný na souřadnicích [1, 1, 1] chci posunout o 3 body dále ve směru osy x a o 4 body ve směru osy y.

[1, 1, 1, 1]	x	1	0	0	0	=	[4, 5, 1, 1]
		0	1	0	0		
		0	0	1	0		
		3	4	0	1		

3.3 Otočení objektu na souřadnice cíle

Při navigaci agenta, herní bytosti řízené umělou inteligencí, je často potřeba natočit ho tak, aby se díval na určitou pozici. V řeči matematiky to znamená, aby agentův dopředný vektor (reprezentace směru natočení) svíral s vektorem směru od agenta k cíly úhel menší, než je nějaká malá konstanta.

K výpočtu úhlu, o který se má postava agenta otočit se využije trigonometrie a **skalární součin**. Nechť u a v jsou dvojrozměrné vektory, pak skalární součin (u • v) těchto vektorů je tvaru:

$$u \bullet v = u1 * v1 + u2 * v2$$

Skalární součin se dá také napsat ve tvaru:

$$u \bullet v = |u| * |v| * \cos(\alpha)$$

kde α je úhel, jenž svírají oba vektory. Pokud jsou oba vektory znormalizované (velikost rovna jedné), je možné přepsat vzorec do tvaru:

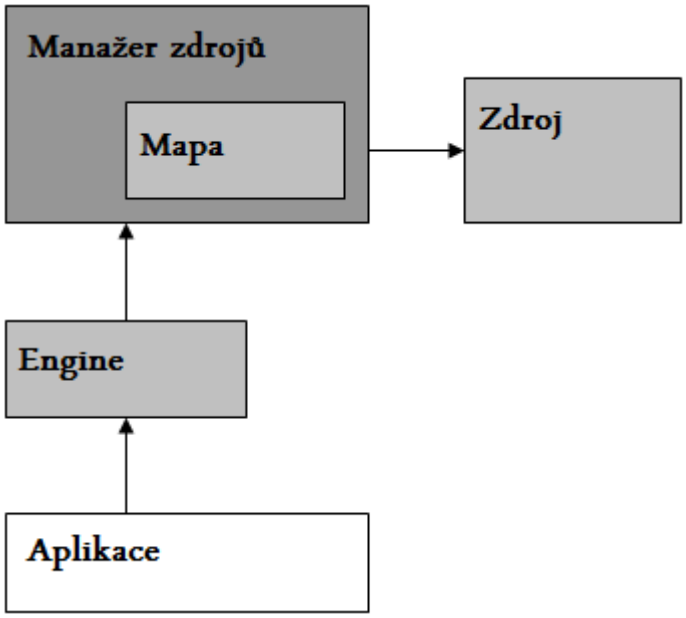
$$u \bullet v = \cos(\alpha)$$

Použitím tohoto vzorce se dá vypočítat, zda svírají dva vektory úhel větší než 90 stupňů. V případě navigace ve hře, jsou těmito dvěma vektory dopředný vektor pohledu hráče a vektor od umístění hráče k cílovému bodu. Vzorec nám tedy zároveň dává informaci, zda je cíl před, či za hráčem. Respektive, zda je výsledek skalárního součinu kladný, či záporný. Pokud na výsledek tohoto skalárního součinu aplikujeme operaci inverzního cosinu, získáme přesný úhel v radiánech, který oba vektory svírají. Při otáčení hráčů se používají pouze dvě osy, jelikož jsou všichni hráči stále stejně vysoko. Plynulosti otáčení se dá dosáhnout rozložením otočení o získaný úhel do více kroků (snímků herní grafiky).

3.4 Manažer zdrojů

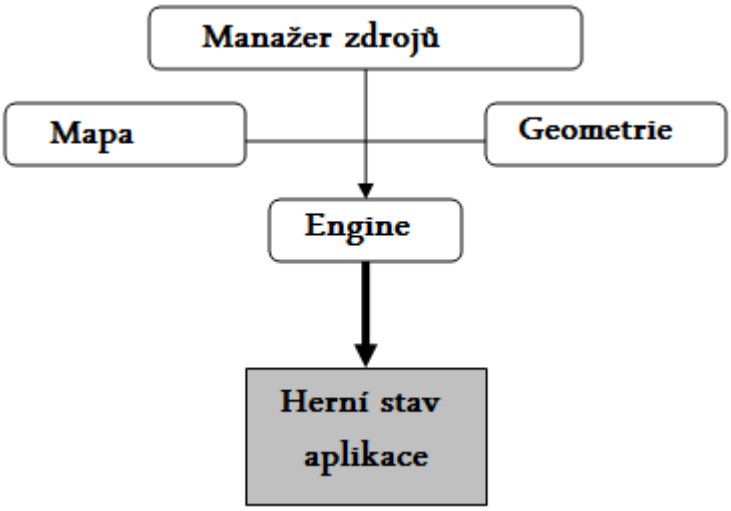
Při nahrávání dat ze souboru do paměti a vytváření některých zdrojů, jsem si uvědomil, že jsou v paměti redundantně. Bylo by tedy vhodné a žádoucí je sdílet pro všechny třídy, které mají zájem je využívat. Výsledkem použití sdílení je úspora paměti a procesoru. Zdroje, které mohou být sdíleny, jsou totiž v paměti drženy jen jednou a jejich vytvoření se provádí také jen jednou. Proto je v projektu nadefinována šablonová třída manažera zdrojů, který toto sdílení obhospodařuje.

Toto řešení má však i jistá úskalí. Tím je rychlost, s jakou je manažer schopen zjistit, zda obsahuje daný zdroj, či nikoliv. Prvním nápadem by mohlo být použít ve třídě manažera nějaké úložiště typu spojového seznamu apod.



Obrázek č. 15 - Schéma manažera zdrojů.

Toto řešení by však bylo značně pomalé. Může se nám totiž stát, že přijde několik po sobě jdoucích dotazů na zdroj, který je umístěn až na konci seznamu. Složitost vyhledávání zdroje ve struktuře seznamu je lineární.



Obrázek č. 16 - Příklad využití manažera zdrojů.

Vzhledem k tomu, že je manažer primárně využíván k vyhledávání existujících zdrojů, je vhodnější využít některé ze struktur určených k vyhledávání. Vhodnou variantou by mohly být hešovací tabulky, či stromy. Manažer zdrojů v projektu Valahalla je implementován kontejnerem mapa ze standardní knihovny C++ (knihovna STL). Tato struktura je implementována červeno-černým stromem. Má tedy asymptoticky logaritmickou vyhledávací časovou složitost vzhledem k počtu záznamů.

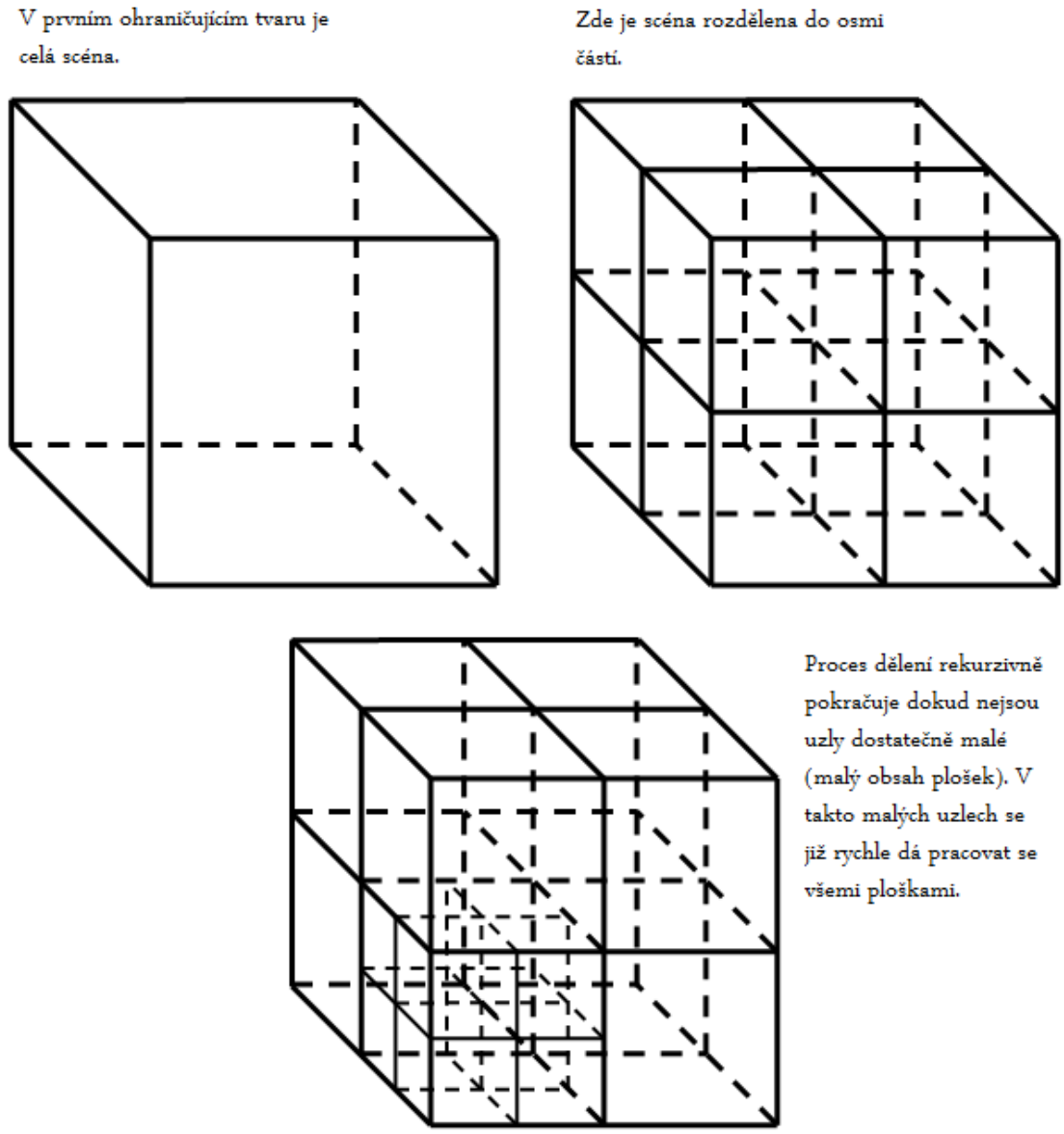
3.5 R strom

Jak už bylo uvedeno dříve, velmi důležitou vlastností hry je snímková frekvence, s jakou je schopna běžet. Animace pohybu scény se pak jeví plynulá a přirozená. Použití R stromu je jednou z optimalizačních technik, která výrazným způsobem zrychluje běh programu. Podstatou optimalizace je redukce počtu vrcholů ve vertex bufferu před vykreslováním. Grafická karta totiž musí pomocí vnitřní logiky vybrat ze zadaných vrcholů ty, které jsou viditelné pro hráče. Snížením počtu vložených vrcholů již na aplikační úrovni ulevíme grafické kartě. Ta je pak schopna pracovat rychleji. Optimalizace na straně vytížení grafické karty obvykle vede k většímu vytížení procesoru. Děje se tak, protože se část práce s filtrováním neviditelných (zakrytých, či mimo zorný úhel hráče) vrcholů přesunula z grafické karty do aplikace (na procesor).

R strom je datová struktura, která se snaží při svém vzniku rovnoměrně rozložit scénu na celky o určitém počtu prostorových objektů. Tyto celky pak obsahují objekty s podobným souřadnicovým umístěním. Při jejich vytváření je obvyklý rekurzivní postup. Tato struktura

je většinou používána ke hledání skupin objektů, které mají souřadnicové umístění blízké nějakému bodu.

V programu Valahalla je použita modifikace R stromu. Od originální verze se liší tím, že neprovádí vyvažování rozložení objektů do nových sub-částí rozkládaného celku. Jelikož pracujeme ve trojrozměrném prostoru, tvoří celky objekty, jejichž souřadnice jsou obsaženy v krychli určité velikosti umístěné na konkrétních souřadnicích. Těmito objekty jsou zde trojúhelníky, které tvoří triangulaci scény mapy. Základem je krychle, která obsahuje celou 3D scénu (všechny trojúhelníky). Ta tvoří kořen vytvářené stromové struktury. V dalším kroku se krychle rozdělí na osm sub-krychliček, do kterých se přiřadí příslušné trojúhelníky. Takto se rekurzivně postupuje, dokud nemají krychličky požadované parametry (velikost, počet trojúhelníků). Obrázek č. 17 ilustrující proces je uveden níže.



Obrázek č. 17 - Schéma rekurzivního dělení scény při vytváření modifikovaného R stromu.

Vyvažování rozložení objektů do struktury stromu není implementováno z několika důvodů. Hlavními důvody jsou zachování krychlového tvaru uzlů R stromu a relativně malý podíl na zlepšení výkonu při použití vyvažování. Trojúhelníky scény mapy jsou totiž vlivem půdorysového návrhu rozloženy v prostoru velmi rovnoměrně. Nehrozí tak kumulace početné skupiny trojúhelníkových plošek kolem nějaké souřadnice a následné výrazné zvětšení hloubky stromu. Naproti tomu získáme výrazně jednoduší vytváření této struktury, lepší čitelnost pro ostatní programátory a jednoduší (rychlejší) testování uzlu na kolize s ostatními prostorovými objekty. Tato struktura se v aplikaci používá hlavně k optimalizaci vykreslování a počítání kolizí. Razantně snižuje počet zpracovávaných trojúhelníků.

3.6 Navigační graf

Umělá inteligence je oblast, u které je žádoucí minimalizace výpočetní náročnosti jejího používání. Jednou ze součástí umělé inteligence je i navigace agentů po mapě. Pod navigací si lze představit určení cesty, po které se má agent dát, aby dorazil k cíli. Cílem může být místo na mapě, zbraň určitého typu, lékárnička apod.

Model mapy je během hry neměnný. Aby bylo vyřízení požadavku na získání cesty k určitému bodu mapy co nejrychlejší, je možné některé informace o mapě předpočítat dopředu. K tomuto účelu se dá použít řada struktur. Jednou z nejpoužívanějších je navigační graf (ve smyslu diskrétní matematiky). Vychází z toho, že pro navigaci agenta jsou důležité jen některé klíčové body. Může jít o body důležité z pohledu umístění, či vybavení a předmětů umístěných v jeho blízkosti. Navigační graf definuje, kudy se mají agenti po mapě pohybovat.

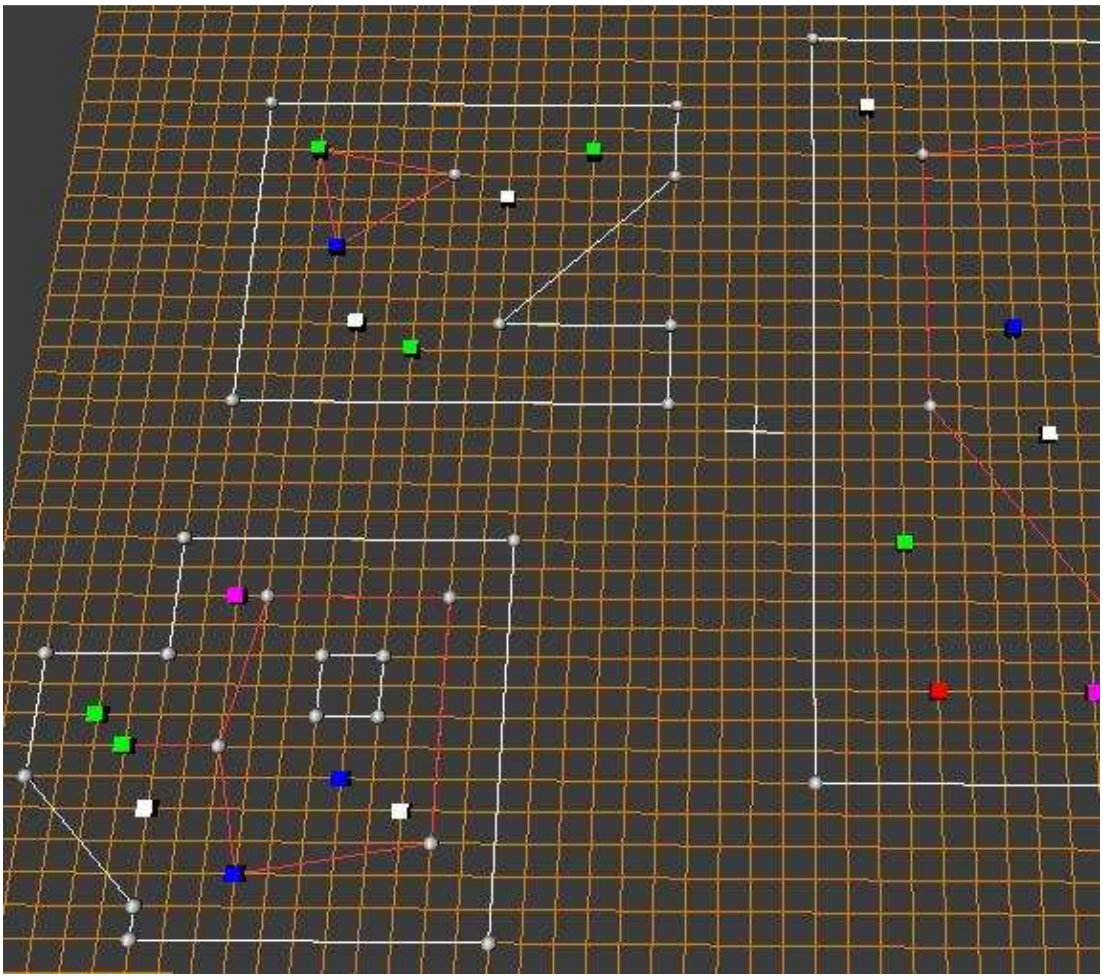
Informace o nejkratších cestách je generována pomocí známého Dijkstrova algoritmu. Jedná se o grafový algoritmus, který umí v grafu spočítat nejkratší cesty ze startovacího uzlu do všech ostatních dostupných uzlů v grafu. Pomocí Dijkstrova algoritmu se spočítají nejkratší cesty mezi všemi uzly. Dijkstrův algoritmus byl zvolen pro svou efektivitu na řídkých grafech. Navigační graf je totiž ve většině případů velmi řídký. Navíc jsou hrany grafu reprezentovány jako seznamy následníků. To vede k velmi efektivní implementaci algoritmu využívající binární haldu jako pomocnou strukturu pro získávání minima. Při této implementaci algoritmus běží v čase O((|E| + |V|) * log |V|), kde |E| je počet hran v grafu a |V| je počet vrcholů. Detailní informace o implementaci Dijkstrova algoritmu pomocí binární haldy jsou k dispozici například na internetových stránkách [16].

Pohyb probíhá jen mezi uzly grafu, po hranách, které je spojují. Výjimkou jsou nestandardní situace, jako například boj s jiným agentem, kdy by toto omezení působilo škodlivě a neužitečně.

Výpočetní úspora spočívá v redukci možných směrů, kudy se může agent pohybovat. Hlavním pozitivem úspory je možnost předpočítat cesty a možnosti pohybu po mapě dopředu během nahrávání mapy ze souboru. Předpočítání navigačních informací spočívá v kalkulaci nejkratších cest, kudy se dát, pokud se chci dostat z bodu A do bodu B. Dále je možné uložit cesty po mapě k různým druhům vybavení a důležitým místům. Tato informace je uložena pro každý uzel zvlášť přímo v daném uzlu, pro který chceme tyto cesty znát.

Použití grafu za běhu hry je velmi jednoduché a rychlé. Jakmile se agent dostane k některému z uzlů navigačního grafu, umělá inteligence agenta se rozhodne, jít směrem k některému z navigačních bodů na mapě. Objekt, který drží informace o mapě (včetně navigačních), zná nejkratší cestu mezi pozicí agenta a jeho cílem. Rovnou tak může umělé inteligenci vrátit seznam souřadnic křivky pohybu agenta po mapě k pozici cíle. Křivka je tvořena souřadnicemi uzlů grafu na této cestě. Podobně pokud se agent rozhodne získat cestu k určitému druhu vybavení, zeptá se na cestu na nejbližším uzlu. Ten ji má předpočítánou a rovnou jí vrátí s minimálním zatížením procesoru.

Navigační graf je nutno vytvořit v editoru manuálně nástrojem „Paths builder“. Uživatel má možnost ovlivnit, kudy se budou agenti pohybovat a které body mapy budou pro umělou inteligenci klíčové (zbraně, vybavení hráčů, lékárničky apod.). Obrázek č. 18 představuje příklad návrhu navigačního grafu. Tvoří ho body a červené úsečky (hrany), které je spojují.



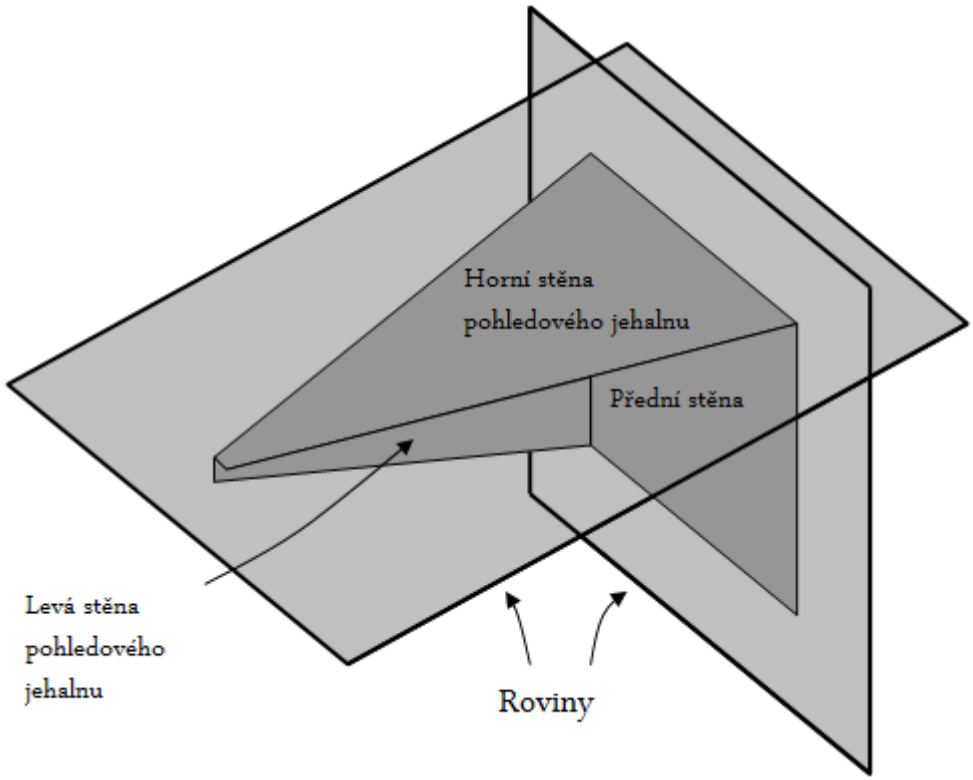
Obrázek č. 18 - Červené čáry představují hrany navigačního grafu, cesty pro agenty.

3.7 Pohledový jehlan

Pohledový jehlan je jednou z dalších optimalizačních technik. Je založena na redukci trojúhelníkových plošek před odesláním grafické kartě ke zpracování. Filtruje plošky, které nejsou v zorném poli hráče. K provedení filtrace je nutné vytvořit komolý jehlan. Jehlan si je možné představit jako pyramidu, na jejímž vrcholu je kamera (pohled hráče) nasměrovaná do těla pyramidy. Jehlan reprezentuje zorné pole hráče. Vrchol jsou oči, vzdálená rovina je dohled, blízká rovina je minimální zaostřovací vzdálenost a stěny jsou zorné úhly.

Z programového hlediska je komolý jehlan reprezentován soustavou šesti rovin, které tvoří tvar jehlanu. Parametry jehlanu tvoří rozlišení monitoru (zorné úhly) a přilehlá respektive vzdálená rovina, jejichž vzdálenost od kamery je definovaná konstantou. Ořezávání trojúhelníkových plošek probíhá jednoduchým způsobem. Cokoliv je uvnitř pohledového jehlanu, je použito, cokoliv je venku, zahozeno. V řeči programového kódu to znamená, že ploška musí být vždy nad každou z rovin. Roviny jsou přitom definovány tak, aby slovo

„nad“ znamenalo směrem dovnitř pyramidy. Pokud je tedy ploška nad všemi rovinami, je uvnitř pyramidy a může být považována za viditelnou. Tato metoda je velmi účinným filtrem nežádoucích (neviditelných) plošek. Následující obrázek č. 19 ukazuje model pohledové pyramidy a jak je tvořena jednotlivými rovinami a stěnami.



Obrázek č. 19 - Model pohledového jehlanu.

3.8 Stínící objekty

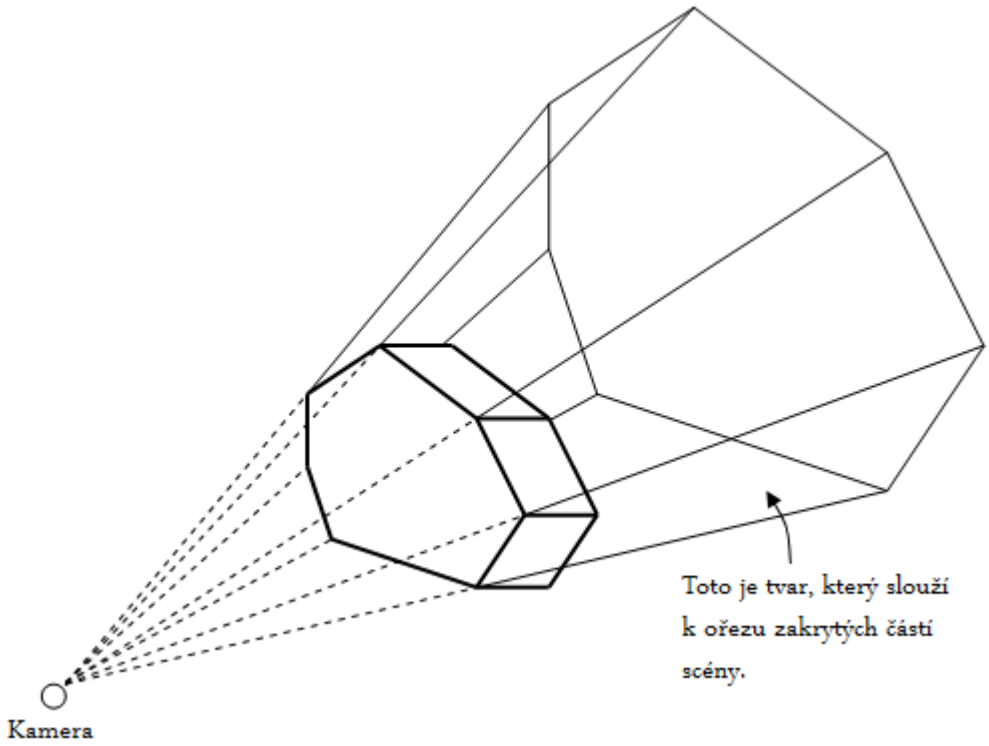
Stínící objekty jsou poslední optimalizační technikou pro grafiku. Je založena na podobném principu jako předchozí metoda. Ve scéně se často objevují předměty, které omezují viditelnost. Takovým předmětem může být například vlak, barel, velká bedna, auto apod. Tyto tělesa mají tu vlastnost, že pokud je hráč dostatečně blízko, tělesa zakryjí vše, co se nachází za nimi. Jinými slovy, vše co je za nimi, je pro hráče zakryto a mělo by být ořezáno před odesláním na grafickou kartu. Stejně jako v případě minulé metody je totiž jisté, že tyto plošky stejně na monitor vykresleny nebudou. Tím se opět získá menší vytížení grafického adaptéru.

Algoritmus filtrování plošek je podobný jako v minulém případě. Vytvoří se komolý hranol a zkoumáme, zda je ploška uvnitř.

Vytvoření tohoto hranolu probíhá následujícím způsobem: Máme těleso, které zakrývá výhled a souřadnice kamery hráče. Nejdříve se získá seznam hran, které tvoří siluetu hranolu z pohledu hráče. Prochází se všechny plošky tělesa a u každé se dívám, jestli normála této plošky svírá s vektorem mířícím od kamery k tělesu větší než pravý úhel. Pokud ano, pak je ploška odvrácena od kamery a netvoří určitě siluetu tělesa, jelikož není vůbec vidět. Pokud ne, přidají se její hrany do seznamu kandidátů na hranu siluety tělesa. Před přidáním jednotlivých hran se ale zkontroluje, zda již není hrana obsažena v této množině hran (kandidátů). Pokud ano, tak se nic nepřidává a smaže se i ta hrana, co je tam již vložena. Pokud ne, hrana se přidá. Hlavním smyslem tohoto postupu je to, že hraniční hrany siluety tělesa jsou přidávány pouze jednou.

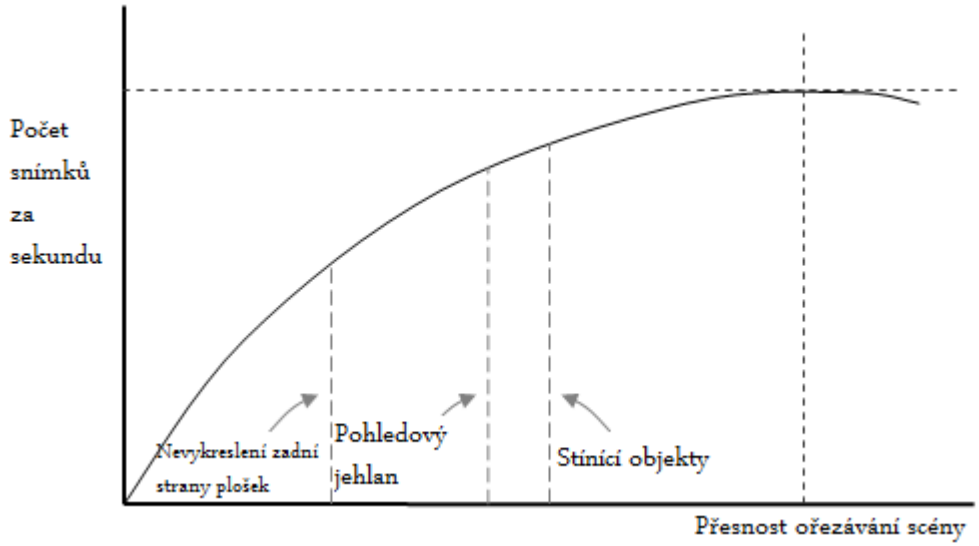
Ostatní hrany, které jsou na společné hranici dvou viditelných plošek, netvoří siluetu tělesa. Z hran siluety a bodu umístění kamery se vytvoří roviny tvořící hranol. Nejdříve se však přidá přední rovina kolmá na kameru ve vzdálenosti umístění tělesa. Test na viditelnost probíha naprosto stejně, jako v případě testu pohledovým jehlanem, s tím rozdílem, že cokoliv je uvnitř hranolu naopak zobrazeno není (zakryto tělesem).

Tato metoda nebyla nakonec v projektu použita, jelikož nárůst výkonu během testování byl minimální. Je však možné kdykoliv metodu přidat. Zejména pokud by se do mapy daly vkládat nějaké větší objekty, které by zakrývaly významnou část scény.



Obrázek č. 20 - Model hranolu, který tvoří stín objektu.

Jelikož se jedná o poslední z metod optimalizace vytížení grafického adaptéru, dovolil bych si malé shrnutí. Metodami uvedenými v podkapitolách 3.5 až 3.8 se sníží vytížení grafické karty. Stále jsou zde však plošky, které nejsou viditelné, a přesto jsou odeslány k vykreslení. Cílem optimalizací není dokonalé filtrování, hlavním cílem je maximální rychlost běhu hry. Pokud bychom se zabývali ořezáváním neviditelných plošek příliš detailně, snížilo by to výkon hry vlivem přetížení procesoru. Filtrace totiž probíhá na aplikační úrovni. Cílem je najít optimální poměr mezi vytížením procesoru a grafické karty. Objektivním měřítkem vyváženosti tohoto poměru je snímková frekvence, s jakou hra běží. Obrázek č. 21 demonstruje vliv optimalizačních technik na výkon hry podle [8].



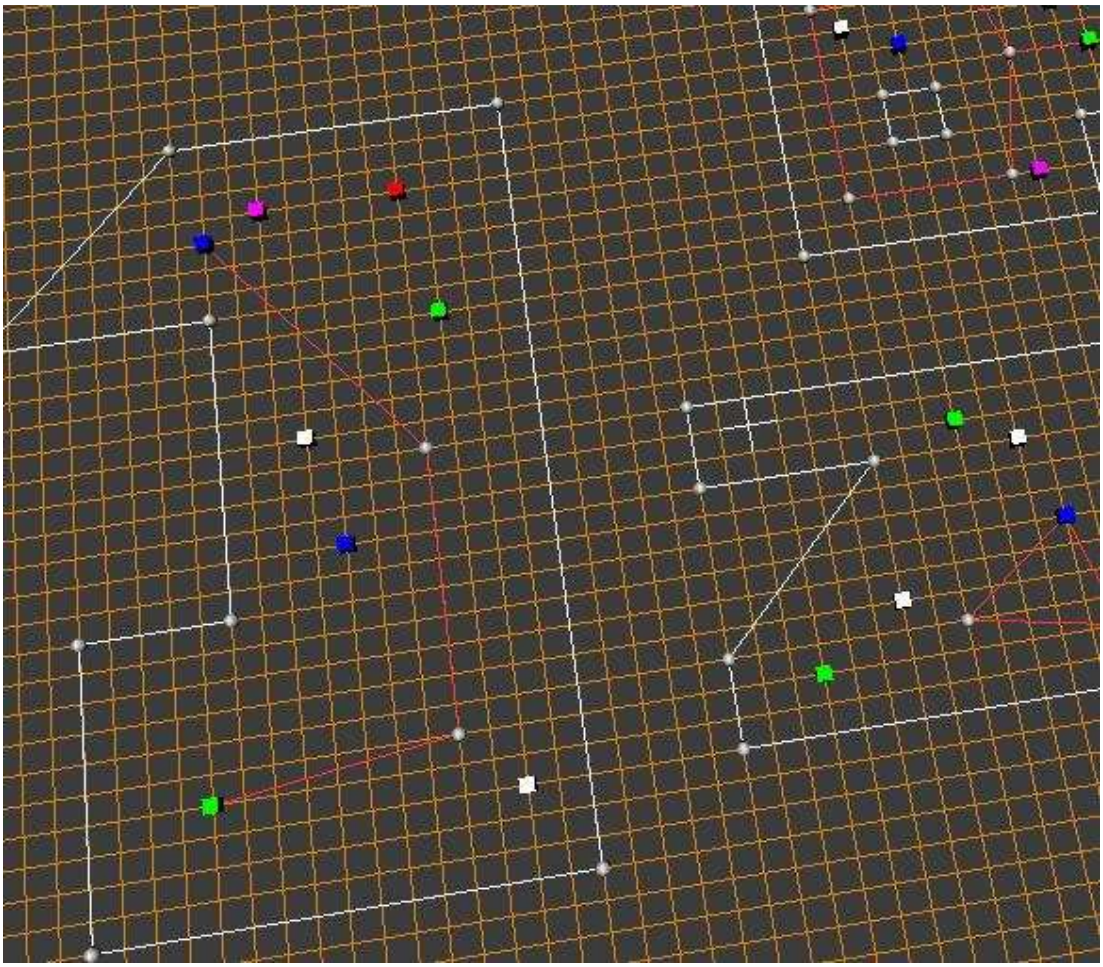
Obrázek č. 21 - Závislost snímkové frekvence na použití optimalizačních metod.

Metoda na obrázku výše nazvaná „Nevykreslení zadní strany plošek“ je obsažena již v samotných DirectX. Spočívá ve vykreslování plošek jen z jedné strany (přední). Pokud máme například ve scéně mapy zeď, není nutné, aby se grafika zabývala vykreslením plošky stěny zdi i ze zadní strany. Není totiž stejně viditelná. Z grafu lze vidět závislost snímkové frekvence na použití optimalizačních technik. Pokud je optimalizací na straně aplikace příliš mnoho, přetížení procesoru sníží vlivem příliš detailní filtrace plošek celkový výkon hry (snímkovou frekvenci).

3.9 Triangulace mapy

Jak již bylo zmíněno, grafická karta umí vykreslovat pouze určitá grafická primitiva. Proto je nutné před začátkem vykreslování převést celou scénu do tvaru seznamu trojúhelníků, který může být prostřednictvím DirectX zobrazen na ploše monitoru.

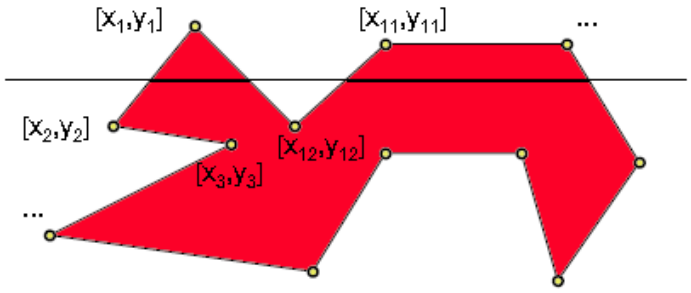
Popis algoritmu této procedury bude rozdělen na tři části. Nejdříve popíšu, jaké struktury a data máme po nahrání mapy ze souboru k dispozici, poté popíšu metodu transformace těchto dat na seznam trojúhelníků a nakonec využití dat získaných touto metodou.



Obrázek č. 22 - Bílé čáry (stěny) spolu se čtvercovou sítí vymezují oblast pro triangulaci.

Model mapy je tvořen půdorysovým návrhem. Návrh je reprezentován grafem ve smyslu diskrétní matematiky, tedy množinou vrcholů a hran. Hrany reprezentují stěny mapy a vrcholy jsou tedy logicky rohy, tzn. místa, ve kterých se spojují dvě stěny mapy. Stěny mohou svírat s osou x souřadnicového systému pouze úhel rovný násobku 45 stupňů. Tím je umožněno zjednodušení a zrychlení algoritmu triangulace scény a mapování textur materiálů na vzniklé trojúhelníky. Nevýhodou tohoto řešení je ztráta jisté obecnosti návrhu.

Než se pustím do osvětlení metody vykonávající triangulaci, rád bych představil jeden obecnější a známější algoritmus s oblasti 2D grafiky, který mě inspiroval při psaní triangulační metody. Jde o algoritmus vyplňování n-úhelníku barvou z přednášky [10]. Následující text a myšlenky jsou převzaty z textu k této přednášce.

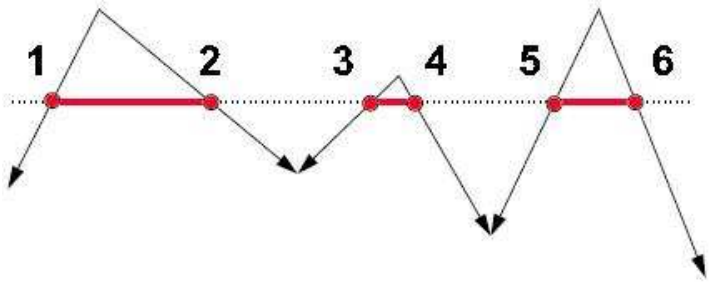


Obrázek č. 23 - Polygon vyplňovaný barvou pomocí řádkového algoritmu.

Řádkový algoritmus

N-úhelník je zadán posloupností svých vrcholů. Dá se říci, že polygon jakoby rozřízneme na proužky a ty pak zpracováváme jeden po druhém v cyklu zvlášť. N-úhelník rozložíme na jednotlivé hrany. Vodorovné odstraníme. Pro ostatní hrany vytvoříme pracovní záznamy. Všechny předzpracované hrany setřídíme do vstupního seznamu S podle kritérií.

1) Vzestupně podle y-ové souřadnice
2) Vzestupně podle x-ové souřadnice
3) Vzestupně podle dxy



Obrázek č. 24 - Jeden krok řádkového algoritmu.

Aktuální seznam hran A bude obsahovat všechny hrany, které protínají aktuální řádku. Seznam budeme udržovat setříděný podle kritérií.

1) Vzestupně podle x-ové souřadnice
2) Vzestupně podle dxy

Na začátku zařadíme do A počáteční úsek seznamu S, hrany se shodným (tj. minimálním) y. Vykreslení aktuální řádky probíhá následujícím způsobem. Je potřeba projít aktuální seznam A a vykreslit úseky odpovídající vnitřku n-úhelníka. Kreslím každý úsek mezi lichým a sudým záznamem.

Aktualizace seznamu A:

a) Délka hrany = délka hrany – 1
b) If(délka hrany == 0) vyhoď hranu ze seznamu
c) x = x * dxy
d) kontrola setřídění A
e) zatřídění nových hran z S do A (počáteční úsek S)

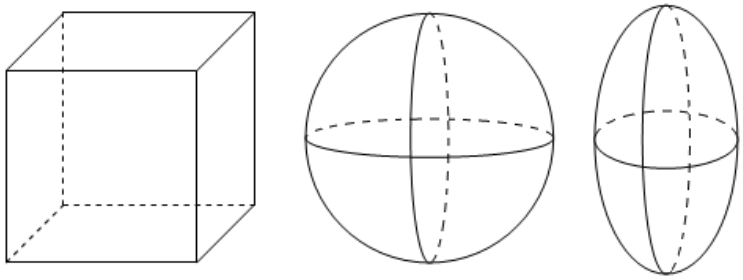
Algoritmus končí, pokud je seznam S u konce a seznam A je prázdný. Jinak se pokračuje krokem Aktualizace seznamu A. Nyní bych rád popsal modifikaci řádkového algoritmu užitou v aplikaci.

Algoritmus triangulace scény

Algoritmus připraví potřebné struktury a proměnné. To znamená, že vytvoří dva pracovní seznamy hran obsahující setříděné (ve smyslu výše popsaného řádkového algoritmu) a setříděné horizontální (rovnoběžné s osou x) hrany. Vytvoří seznam horizontálních hran pro první kolo algoritmu. To jsou ty s nejmenší z-ovou souřadnicí. Poté se spustí hlavní cyklus triangulace. Prochází scénu řádek po řádku a vytváří triangulaci scény. V prvním podcyklu přidává do seznamu aktuálně zpracovávaných hran (ekvivalent seznamu A řádkového algoritmu) všechny hrany, které ještě nebyly zpracovávány a začínají na pozici aktuálně zpracovávaného řádku (z-ová souřadnice). Vymění seznamy horních horizontálních hran za spodní. To proto, že hrany, které tvořily spodní okraj zpracovávaného řádku v minulém kole cyklu, tvoří nyní horní okraj v pásku o řádek níže. Ve druhém podcyklu se přidají nové spodní horizontální hrany se z-ovou souřadnicí shodnou s novou z-ovou pozicí aktuálně zpracovávané řádky. Třetí podcyklus provádí zpracování aktuální řádky mapy. Získám vždy dvě paritní hrany. Z nich se vytvoří zdi, ale pouze pokud jsou ve zpracování řádky prvně, aby nevznikaly redundantní zdi se stejným umístěním. Dále se vytvoří zdi ze všech horizontálních hran, které jsou na řádek napojeny. Nakonec se přidá podlaha (případně strop), proužek z trojúhelníků. Takto se pokračuje, dokud je co zpracovávat. Konečným produktem algoritmu je seznam trojúhelníkových plošek. Z něj už je snadné vytvořit vertex buffer, datovou struturu čitelnou pro DirectX sloužící ke zobrazení scény na monitoru uživatele.

3.10 Kolize s objekty

Kolize objektů, jako jsou například kulky, lékárničky, agenti atd., jsou jedny z nejčastějších jevů. V této části textu bych nastínil, jaké funkce jsou použity pro jejich detekci. Největší podíl na způsobu detekce kolizí mají tvary těles, které spolu kolidují.



Obrázek č. 25 - Modely kolizních objektů.

Většina algoritmů na výpočet detekce kolize těles je definována v hlavičkovém souboru „Geometry.h“ projektu Valahalla. Jedná se o testy:

- Zda krychle obklopuje krychli,
- Zda je ploška uvnitř krychle,
- Zda je krychle uvnitř tvaru daného rovinami,
- Zda je krychle uzavřena do koule,
- Kolize dvou pohybujících se koulí v prostoru.

Pro pochopení technických detailů těchto testů odkazuji na zmiňovaný hlavičkový soubor.

3.11 Kolize se scénou

Pro pohyb dynamických objektů ve scéně mapy je nutná přesná detekce kolizí. Ta zabrání výskytu nereálného chování hry, jako je například procházení zdí apod. Kolize se počítají pro pohyb všech dynamicky se pohybujících objektů v každém snímku hry. Tím je dosaženo maximální přesnosti pohybu.

Detekci kolizí pohybujících se objektů, jako jsou například agenti, osvětlím na konkrétním příkladu. Mějme agenta umístěného na souřadnicích „xyz“, který se pohybuje se rychlostí „v“ ve směru dopředného vektoru „q“. V prvním kroku program zjistí seznam plošek, které jsou v těsném sousedství agenta. Využije k tomu již zmiňované struktury modifikovaného R stromu. Poté prochází plošky ze získaného seznamu a zjišťuje, zda nedojde ke kolizi s některou z nich (vzhledem k xyz, v, q). Využije funkce D3DXIntersectTri pro výpočet

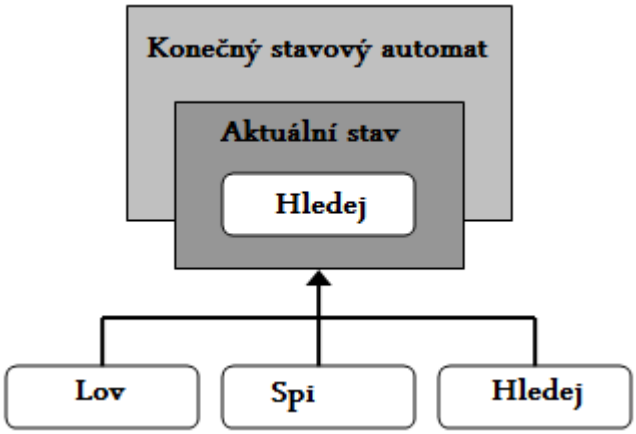
potenciálního průsečíku trojúhelníkové plošky a paprsku poskytované DirectX. Kromě samotného trojúhelníku se testují na kolize i hrany tvořené jeho stranami. To pro případ, že se objekt pohybuje rovnoběžně s testovaným trojúhelníkem. Pokud dojde ke kolizi, algoritmus se pokusí dodržet pohyb alespoň v rámci možností, až do okamžiku, než dojde ke kolizi. Pokud k žádné kolizi nedojde, agent se pohybuje určenou rychlostí a směrem nerušeně dál. Algoritmus pro detekci kolizí je inspirován článkem „Improved Collision detection and Response“, který je součástí multimediální dokumentace knihy [8]. Zmiňovaný matematický text, který detailně popisuje teorii kolizí užitou při implementaci detekce kolizí v projektu Valahalla, je volně dostupný na internetu a je též součástí multimediální přílohy této bakalářské práce.

3.12 Konečný stavový automat umělé inteligence

O umělé inteligenci se podrobně mluví dále v kapitole č. 5 „Umělá inteligence“. Zde jen nastíním datovou strukturu reprezentující mysl agenta. Při jejím návrhu jsem vycházel z těchto vlastností agentů:

1) Všichni mohou myslet stejně
2) Cílem je minimální náročnost na výpočet dalšího kroku

Z prvního bodu vyplývá možnost sdílet „mozek“ všemi agenty a tím dosáhnout paměťové úspory. Výsledky přemýšlení však musí mít každý agent uloženy zvlášť. V kapitole č. 5 „Umělá inteligence“ je navržena reprezentace stavů mysli agenta pomocí konečného stavového automatu. Konkrétní implementace automatu je realizována návrhovým vzorem „State“ a třídou představující stavový automat. Tato třída obsahuje ukazatel na předka konkrétních stavů (boj, spánek, prohledávání mapy). Výhodou tohoto řešení je nahrazení rozsáhlých větvících se konstrukcí polymorfismem. To přináší možnost dále rozvíjet počet stavů mysli agenta bez zpomalení běhu programu a udržet čitelný kód.



Obrázek č. 26 - Příklad konkrétního použití konečného stavového automatu.

4. Architektura

4.1 Analýza problému

Žánr 3D akčních her je poměrně náročný na realizaci. Pro úspěšnou implementaci jsou totiž nutné vědomosti hned z několika oborů (matematiky, grafiky i informatiky). Každý obor s sebou přináší otázky a problémy, pro které je třeba najít při realizaci co nejoptimálnější řešení.

Grafika

Úspěšnost hry je v dnešní době dána z velké části grafickým zpracováním. Program tedy musí obsahovat co nejdetailnější textury a modely vykreslovaných grafických objektů. Dobrá hra také nemá moc vysoké hardwarové nároky vzhledem k viditelné vizuální kvalitě. Uživatel má rád, když hra běží plynule s vysokým počtem snímků za sekundu. Program proto musí dbát na co nejlepší optimalizaci zobrazování grafiky.

Informatika

Při tvorbě složitého programu, jako je 3D akční hra, se dá předpokládat velká rozsáhlost zdrojových kódů. Program tedy musí být napsán tak, aby byl zdrojový kód udržovatelný pro potřeby dlouhodobého rozšiřování. Aplikační logika by neměla být příliš složitá, aby nebrzdila hru. Velkou část výkonu stroje totiž spotřebuje zpracování samotné grafiky. Program by měl být napsán s maximálním ohledem na optimalizaci použitých algoritmů a na kvalitu návrhu programu. Kromě již zmiňované úspory výkonu na úrovni algoritmů je možné dosáhnout i určité úspory výkonu procesoru na úrovni samotného programovacího jazyka. Příkladem muže být předávání instancí tříd referencí místo hodnotou. V projektu Valahalla je tento přístup dokonce vynucen vhodnou některých deklarací tříd. Kopírovací konstruktor a operátor přiřazení jsou deklarovány jako privátní položky tříd bez implementace. Při pokusu o kopírování tak překladač oznámí chybu ve zdrojovém souboru.

Matematika

Některé problémy při tvorbě programu nejsou bez znalostí relevantních částí matematiky vůbec řešitelné, nebo jsou řešitelné jen s velkou časovou složitostí. Bez znalosti řešení informatických problémů s pomocí matematiky také může vzniknout nečitelný kód plný okrajových případů a podmínek, které není ani autor schopen udržovat a ladit. Pro účely 3D akční hry jsou nejužitečnější diskrétní matematika a lineární algebra. Lineární algebra má velké využití při zpracování grafiky, zejména při transformaci souřadnic bodů v prostoru a diskrétní matematika je využívána pro navigaci a návrh map. Pro orientaci v trojrozměrném prostoru je také nutná analytická geometrie.

Při analýze konkrétních problémů spojených s navrženou hrou, se vyskytlo několik problémů. Každý bylo nutné promyslet, zejména z pohledu výkonnostních dopadů na běh aplikace.

Jedním z hlavních problémů, které jsem řešil v průběhu návrhu hry, byla reprezentace herních map (úrovní) jak v souboru, tak v paměti za běhu programu. Zpočátku jsem myslel, že nejlepší a nejflexibilnější variantou bude jakýsi seznam bodů (souřadnic), který bude podle určitého klíče tvořit zcela obecné trojrozměrné modely mapy. To by umožnilo velkou svobodu při tvorbě mapy v editoru. Po určité době a několika pokusech jsem však omezil obecnost návrhu mapy na půdorysový, kde jsou elementy mapy reprezentovány třídami a strukturami. To umožnilo lepší pochopitelnost reprezentace mapy v aplikační logice. Struktura souboru uložené mapy tak může být velmi elegantně vytvořena jako seznam objektů tvořících mapu v paměti. Samozřejmě je vždy nutné ještě připojit informace, podle kterých je možno znovu vytvořit naprosto stejné kopie objektů. Půdorysový návrh představuje mapu z pohledu shora na plán mapy. Je podobný architektonickým půdorysovým nákresům staveb. Tvoří jej stěny, místa pro objevování hráčů na začátku hry, místa se zbraněmi a vybavením, dekorační předměty (např. dřevěné bedny) a konečně navigační mapa. Další výhodou je o něco jednoduší implementace. Pokud člověk implementuje tak rozsáhlý projekt sám, je měřítko časové náročnosti implementace řešení velmi důležité. Dopředu jsem si rozvrhl, že reprezentaci map věnuji kolem třetiny času stráveného na projektu. Poslední výhodou je myslím poměrně snadné pochopení ovládání editoru a vytváření map (herních úrovní) uživatelem.

Další problém, který je potřeba vyřešit, je triangulace scény. Jde o převedení ploch jako je podlaha či zdi na trojúhelníky, tedy na tvar srozumitelný pro grafickou kartu (vytvoření a naplnění pole vrcholů).

Jednou z klíčových věcí pro herní subsystém je zbraňový systém, pohyb vystřelených kulek, získávání nových zbraní, výměna aktuálně používané zbraně za jinou z hráčova inventáře apod. Toto je další problém, který je potřeba vyřešit a navrhnout jeho řešení.

Dále je potřeba vyřešit detekci kolizí herních objektů a následnou reakci. Příslušná reakce musí být provedena u obou objektů. Příkladem může být například sebrání zbraně. Zbraň musí reagovat tak, že po jejím sebrání hráčem zbraň zmizí. Do hráčova inventáře je zároveň přidána další zbraň.

Jednou z otázek k řešení je i uživatelské rozhraní. Tato část by měla být co nejméně náročná na vývojový čas. Profesionální hry sice často mívají velmi hezké a nápadité menu a ovládací prvky, při jejich vývoji je však k dispozici více lidských a časových zdrojů. V rámci plánu rozložení časových zdrojů má mnohem větší prioritu jádro subsystému.

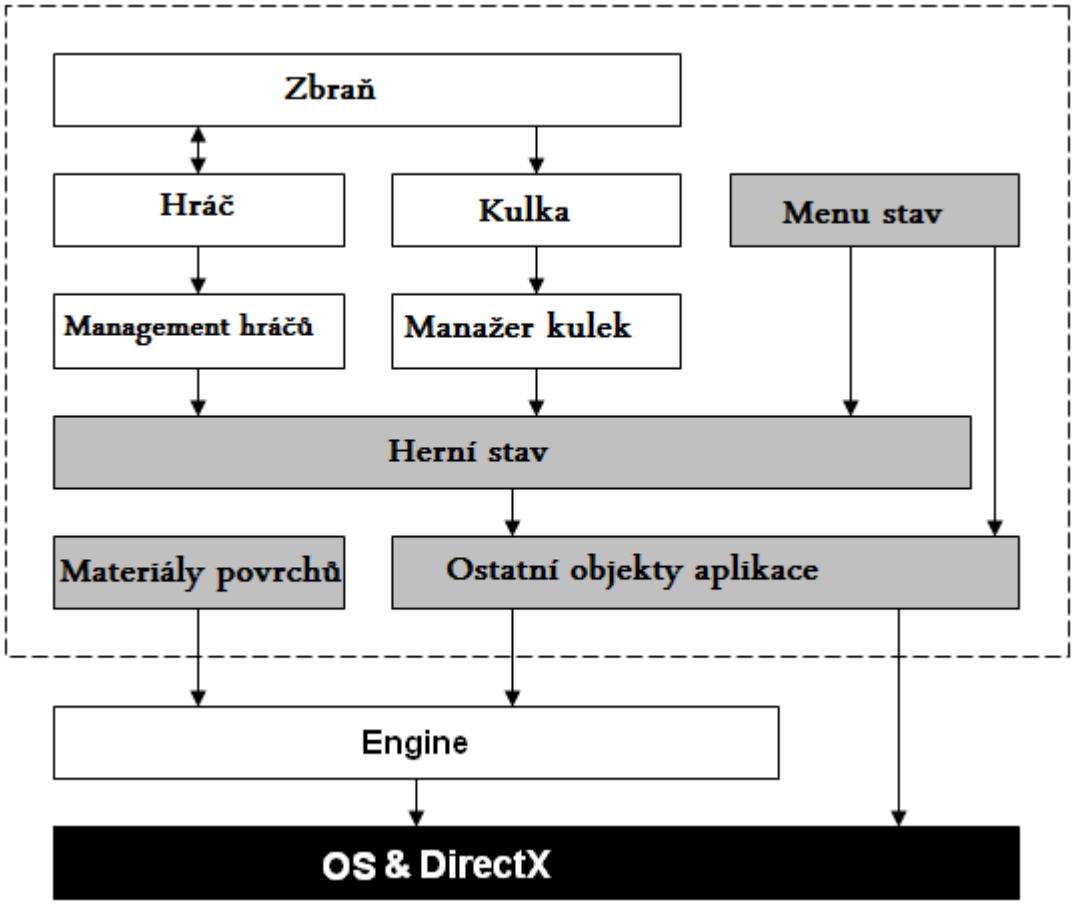
Další otázkou je řešení umělé inteligence, zejména otázka její hloubky a reprezentace. S tím souvisí i vzájemná interakce agentů a pohyb hráčů po mapě.

Technickými částmi hry bude ovládání a zvuk, které musí korespondovat s pohybem objektů po mapě. Důležitá je i hudba dotvářející atmosféru.

Je jisté, že snímková frekvence bude v programu reprezentována velkým cyklem, který poběží stále dokola až do konce hry. Z pohledu designu je velmi zajímavé pořadí a vzájemné ovlivňování okruhů (kolize, umělá inteligence, pohyb hráčů, …) procedur, které budou v tomto cyklu přímo či nepřímo přes vnořená volání funkcí, obsaženy.

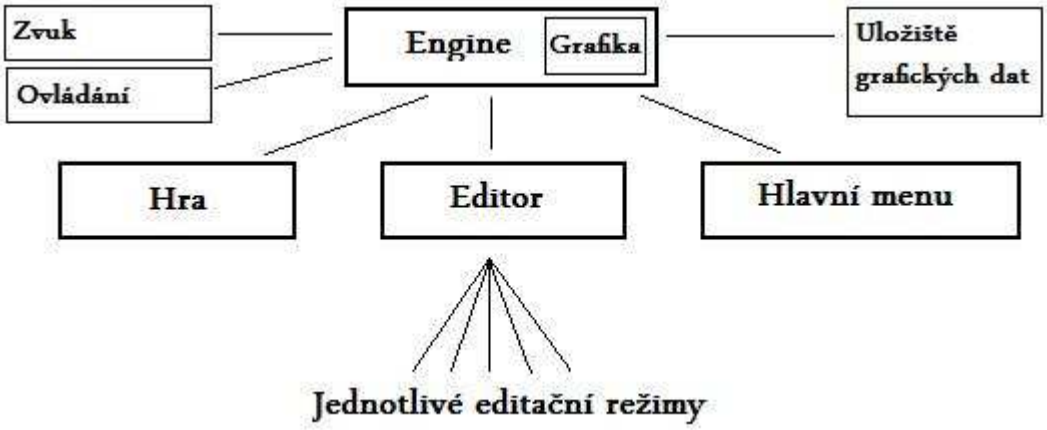
4.2 Klíčové objekty a jejich kooperace

Kooperace mezi objekty charakterizuje fungování logiky hry. V této sub-kapitole přiblížím klíčové třídy projektu Valahalla, jejich funkcionalitu a vztah k ostatním třídám. Obrázek č. 27 ukazuje hlavní tok informací mezi jednotlivými objekty a částmi herní logiky. Šipka představuje směr toku.



Obrázek č. 27 - Klíčové objekty hry.

Jiný náhled na architekturu může být z pohledu kódu. To znamená rozdělení funkcionality do objektové hierarchie. Zde hraje hlavní roli již zmiňovaný návrhový vzor „State“.



Obrázek č. 28 Základní architektura projektu.

Hlavním objektem architektury je třída Engine. Ta v sobě integruje technické prvky (abstrakce zařízení pro přehrávání zvuku, obsluhu příchozích vstupů z klávesnice a myši a abstrakci grafické karty). Klíčovou součástí třídy Engine je objektová abstrakce grafické karty. Přes rozhraní této abstrakce probíhá vykreslování grafiky na monitor uživatele. Uživatel se za běhu programu přepíná mezi hlavní obrazovkou, editorem a vlastní hrou. To je v architektuře řešeno návrhovým vzorem „State“. Třída Engine obsahuje implementace abstraktního předka herních stavů (herní stav, editor atd.). Dále obsahuje ukazatel na tohoto abstraktního předka. Ten ukazuje na herní stav, který je právě aktivní. Podobná architektura platí o úroveň níže u jednotlivých editačních režimů editoru (vkládání grafických objektů, změna textur, budování navigačních cest atd.).

Následuje popis některých důležitých tříd herních objektů, jejich funkcí a začlenění v hierarchii objektového návrhu hry.

Weapon

Třída Weapon implementuje funkcionalitu zbraní používaných hráči. Zbraň určuje dostřel, rychlost, kadenci a poškození, které mohou projektily vystřelené ze zbraně udělit protihráči. Pro každého hráče existuje jedna instance dané zbraně (nejsou datově společné).

Obrázek č. 29 - Model zbraně.

Player

Třída Player dědí od třídy AnimatedObject (kvůli animaci běhu, úkroků, apod.) a reprezentuje programovou logiku hráče. Každý hráč má přiřazenu právě jednu instanci tohoto objektu či jeho potomka. Třída umělého agenta dědí od této třídy a rozšiřuje ji o samostatné přemýšlení.

Obrázek č. 30 - Model hráče.

Bullet

Reprezentuje kulku vystřelenou ze zbraně (třída Weapon). Od chvíle, kdy je kulka vystřelena cestuje prostorem a kontroluje, zda s něčím nekolidovala. Pokud ano, podívá se, s čím kolidovala. Pokud je to hráč, zraního ho o hodnotu danou členskou proměnnou reprezentující velikost poškození. Pokud je to něco jiného třeba stěna, nebo kulka dorazí za oblast dostřelu zbraně, ze které byla vystřelena, sama zanikne. Případné rakety a jiné podobné projektily musí dědit od třídy MeshObject, aby byly viditelné a mohlo se s nimi zacházet, jako s grafickými objekty scény.

BulletManager

Spravuje a zpracovává všechny kulky ve hře.

Menu

Třída je odvozena od předka State, tudíž se jedná o jeden z herních stavů. Tento stav je implicitně nastaven při spustění aplikace. Menu slouží k zadávání informací o zobrazení (rozlišení, barevná hloubka). Spouští se z něj vlastní hra a editor map. Slouží i k nastavení hlasitosti zvuku.

Game

Je odvozena od třídy State. Jedná se opět o jeden z herních stavů. Tento stav bude během hry asi nejčastější, jde totiž o stav odpovídající vlastní hře (běhání po mapě, střílení po nepřátelích atd.). Implementuje virtuální funkce předka. Tento stav jako jedinný spolu

s editorem může zpracovávat uživatelské vstupy pro pohyb kamery (pohyb hráče, střelba, atd.).

Engine

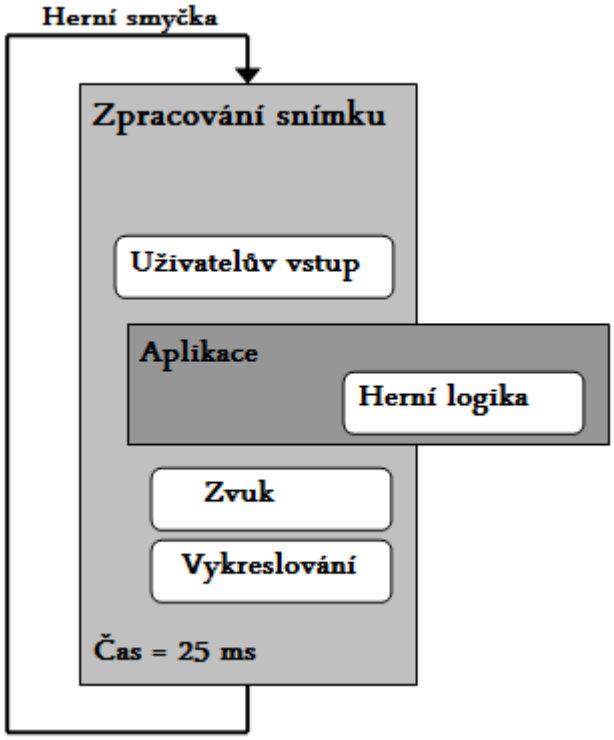
Klíčová, hlavní třída, singleton. Zapouzdřuje instance všech hlavních funkčních objektů. Funguje jako jedinný přístupový bod k celému subsystému. Přes tutu třídu je subsystém napojen na grafický adaptér. Třída Engine reprezentuje kořen v objektové hierarchii tříd programu. Obsahuje sadu metod, které reagují na různé události spojené s grafickou kartou (ztráta kontroly, získání kontroly, restart atd.). Tyto metody jsou aktivovány například při ztrátě kontroly nad grafickou kartou, když se uživatel přepne do jiného okna apod. Jsou nutné pro korektní zotavení aplikace při opětovaném návratu do okna hry a získání kontroly nad grafickou kartou. Stejnou sadu metod obsahují také podobjekty, které na tyto události musí reagovat. Zprávy o těchto událostech se tak vždy lavinově šíří objektovou hierarchii programu směrem od kořene (třídy Engine) do nižších pater (podobjektů). Stejným způsobem je realizována také aktualizace herní logiky a vykreslování herních objektů na obrazovku.

Projekt obsahuje celou řadu dalších tříd nezbytných k jeho fungování (grafy, třídy editoru, konečné automaty atd.). Ty jsou většinou použity právě k fungování výše jmenovaných tříd a jejich potomků. Projekt se řídí filozofií objektového programování, z menších a technických tříd postupně sestavuje složité celky (modularita).

4.3 Hlavní cyklus

Klíčovou metodou třídy Engine je metoda Run, ve které běží „nekonečný“ cyklus. V tomto cyklu se detekuje správná činnost grafického zařízení. Pokud je zařízení nečinné, aplikace ho restartuje a znovu nahraje data, která v něm byla uložena. Pokud je zařízení v pořádku, proběhne aktualizace logiky aktivního stavu a vykreslení na monitor (prvky grafického uživatelského rozhraní, mapa, zbraně apod.).

Hlavní cyklus je místo v programu, kde se zpracovává aktualizace a vykreslování všech herních objektů. Metoda Run se spouští přímo ve funkci Main a běží až do konce běhu aplikace. Nejdříve se zobrazí okno Windows, do kterého se bude vykreslovat grafika. Poté začne běžet cyklus, který může být ukončen až obdržením zprávy o ukončení programu od Windows. Tato zpráva může vzniknout jak uvnitř aplikace (tlačítko „Quit“ apod.), tak i mimo ni.



Obrázek č. 31 - Schéma průběhu hlavního cyklu hry.

V každém cyklu se nejdříve spočítá čas, který uběhl od minulého kola. Ten se používá pro aktualizaci hry. Poté se aktualizuje logika hry. Aktualizuje se stav všech herních objektů. To se provede zavoláním metody OnFrameMove třídy Engine. Většina vnořených tříd obsahuje tutéž metodu, a tak se aktualizace propaguje voláním této metody na vnořených objektech (scéna, hráči, umístění zdrojů zvuků, kamera atd.). Aktualizuje se také vstup z myši a klávesnice od uživatele (ovládání).

Všechny herní objekty jsou poté v novém stavu (např. souřadnice, úhel otočení), některé nové dokonce mohly vzniknout (např. kulky). Nyní je čas na překreslení scény z herní logiky na monitor uživatele. Vykreslování je podobně jako aktualizace logiky realizováno jistou metodou (OnFrameRender), která se volá na třídě Engine a prostupuje dále skrze vnořené objekty. Z aktuálního stavu hry se získají požadavky na vykreslování. Každý stav je má totiž jiné (viz stavy Editor a Game). Nejdříve se vykreslí scéna a poté všechny grafické objekty v ní obsažené (agenti, zbraně atd.). Obrázek č. 31 ukazuje co vše se děje během jednoho cyklu. Čas 25ms zde naznačuje, že vykonání jednoho kola cyklu musí proběhnout velmi rychle, aby obraz působil plynule.

5. Implementace

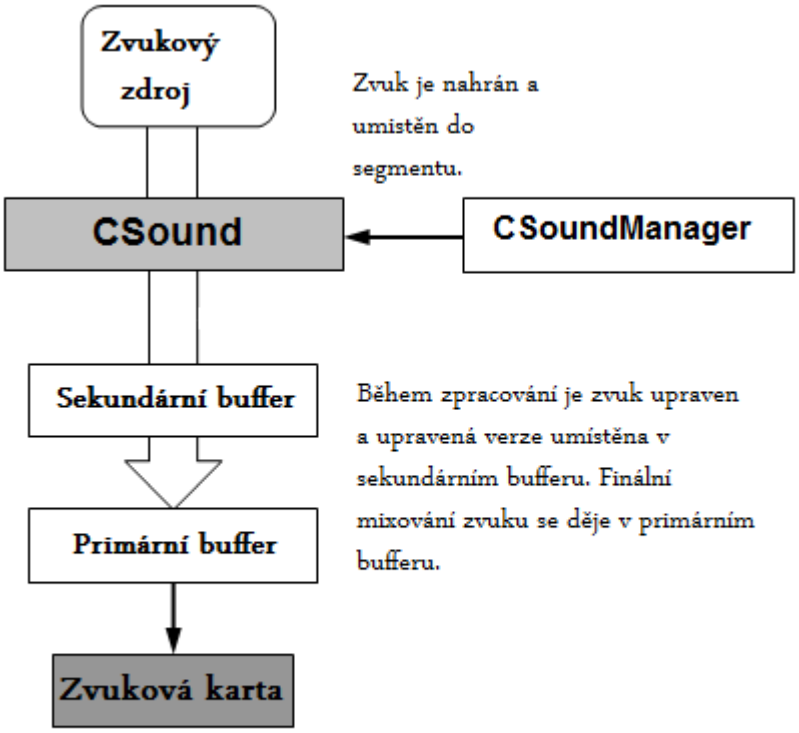
5.1 Ovládání

Ovládání hry je řešeno pomocí knihovny DirectInput, která je součástí DirectX. Ta zaručuje rychlou odezvu na povely uživatele. Z pohledu architektury je ovládání samostatným modulem, třídou Input. Detekce stisku kláves je získána přes třídu IDirectInputDevice8 představující klávesnici. Do metody GetDeviceState této třídy se vloží ukazatel na pole znaků o velikosti 256 bytů. Každý index v tomto poli představuje jeden znak klávesnice. Pokud je potřeba zjistit, zda je některý ze znaků na klávesnici stisknut, stačí se podívat na obsah pole na odpovídajícím indexu. Podobně funguje i detekce stisku ovládacích prvků myši. Jen je zde místo pole použita struktura DIMOUSESTATE. Aktualizace stisku kláves probíhá ve zpracování každého průchodu cyklem (snímku) zvlášť, z důvodu co nejrychlejší odezvy.

5.2 Zvuk

Potřebnou funkcionalitu pro použití ozvučení ve hře poskytuje opět DirectX, konkrétně část knihoven DirectSound. Zvukový systém tvoří třídy CsoundManager a Csound definované v pomocné vrstvě DirectX DXUT, která obsahuje tyto předpřivané třídy pro přehrávání zvuku. Vrstva DXUT je tvořena mnoha obsáhlými zdrojovými soubory, které mají za úkol snížit vývojový čas her a obsahují implementaci různých grafických a technických prvků. Zdrojové soubory vrstvy DXUT jsou v projektu umístěny ve stejnojmenném kontejneru. Použití DirectSound pro přehrávání zahrnuje spolupráci několika tříd a rozhraní. DXUT tento systém zjednodušuje díky zmiňovaným předpřipraveným třídám, které logiku této spolupráce obsahují a poskytují výrazně jednodušší rozhraní pro přehrávání zvuků .

Než je zvuk možno přehrát, je nutné ho nahrát do paměti. K tomu slouží třída CSound. CSound nahrává zvuky ze souboru do paměti. Instance třídy Csound se vytvářejí voláním metody Create na instanci třídy CsoundManager spolu s názvem a cestou ke zvukovému souboru, který se bude přehrávat ve hře. Při vytváření instance třídy Csound jsou zvuková data nahrána do operační paměti. Zvuky je možno přehrávat nezávisle na ostatních. Každá instance objektu zvuku má vlastní sekundární buffer, který obsahuje zvukovou informaci. Při přehrávání se posbírají data ze všech sekundárních bufferů a odešlou se na zvukovou kartu jako primární buffer (celkový zvuk). Postup přehrávání ukazuje obrázek č. 32.



Obrázek č. 32 - Zpracování přehrání zvuku.

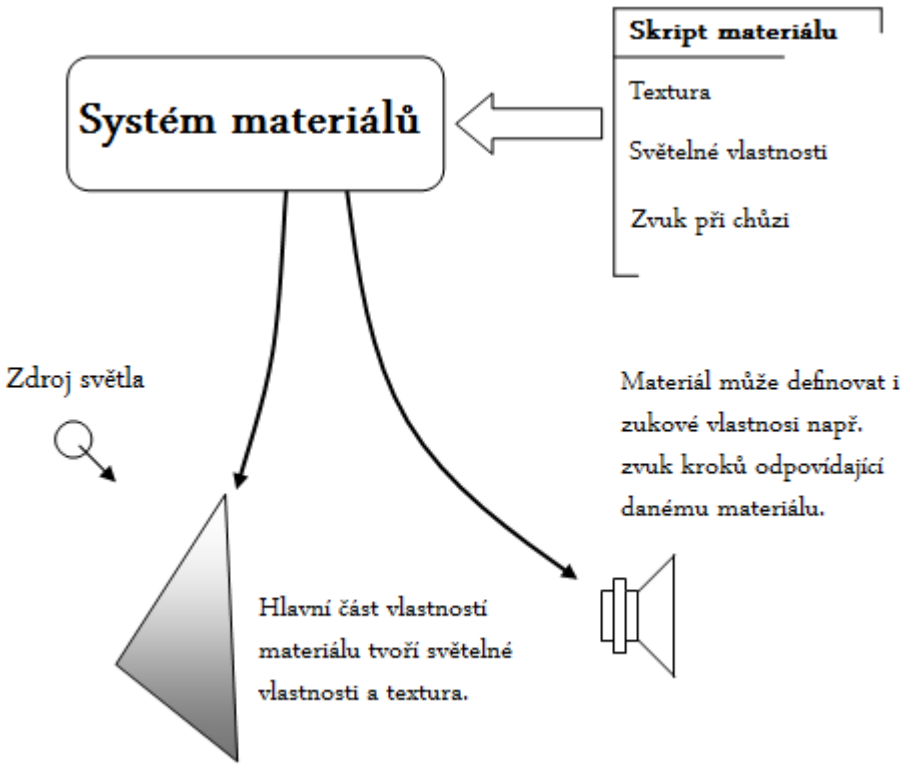
Zvukům je možné měnit hlasitost. Pomocí tlumení hlasitosti je ve hře vytvářena iluze vzdalujících se herních objektů a tím zvýšena realističnost hry. Velikost tlumení se zvyšuje se čtvercem vzdálenosti zvukového zdroje od hráče, což by mělo lépe odpovídat reálnému šíření zvuku.

Ve hře je ozvučeno hlavní menu, kde na pozadí hraje hudba, výstřely ze zbraní, kroky po různých materiálech a sebrání nějakého předmětu vybavení (lékárnička, kulomet). Každá zbraň má svůj specifický zvuk výstřelu. K lepší realističnosti přispívají i zvuky kroků. Podlahové materiály mají opět své specifické zvuky. Jinak zní chůze po trávě, asfaltu či dlažbě. Některé materiály jsou však bez jakékoliv ozvučení.

5.3 Použití materiálů

Každý povrch ve scéně musí být vykreslen určitým materiálem. Materiál není jen textura či barva daného povrchu. Je to soustava parametrů, které ovlivňují vykreslení dané plošky. Parametry materiálu jsou reprezentovány skriptem. Každý materiál má svůj vlastní skript. Kromě textury obsahuje definice materiálu informace o světelných vlastnostech. Ty ovlivňují to, jak materiál odráží nebo pohlcuje jednotlivé složky barevného spektra.

Definice materiálu dále obsahuje kolizní parametry a také seznam zvuků použitých při chůzi hráče po ploše vykreslené daným materiálem.



Obrázek č. 33 - Schéma systému materiálů. Aplikace na konkrétní plošku.

Architektura systému materiálů je tvořena šablonou managementu zdrojů. Tím je dosaženo výrazné úspory paměti a zjednodušení údržby.

5.4 Vykreslování

Grafika a její vykreslování na monitor jsou jedny z nejdůležitějších součástí subsystému. V této části textu bych rád ukázal proces, který se provádí před vykreslením každého jednotlivého snímku. Tento proces musí zajistit maximální optimalizaci, konzistenci obrazu s herní logikou a kvalitu zobrazení. Vykreslování je velmi úzce spojeno s kapitolou věnovanou algoritmům.

Pro grafiku je klíčová vykreslovací metoda třídy SceneManager. Ta totiž vykresluje scénu a všechny grafické objekty v ní obsažené. V průběhu zpracování obrazu se často používá tzv. snímkové razítko. Je to číslo uložené jako členská proměnná třídy managementu scény. Před každým zpracováním snímku se toto číslo inkrementuje. Tak se identifikuje snímek v rámci aplikační logiky hry.

Nejdříve se vykreslovací metoda ujistí, že existuje struktura modifikovaného R stromu (viz. kapitola věnovaná algoritmům a strukturám č. 3). Uvnitř vykreslovací metody se zavolá metoda RecursiveSceneFrustumCheck. Tato metoda rekurzivně kontroluje uzly R stromu proti pohledovému jehlanu. Kontroluje, zda aktuální uzel zasahuje do pohledového jehlanu. Pokud ne, rovnou skončí. Totéž bude totiž platit i pro potomky tohoto uzlu (nebudou viditelní). Uvnitř metody RecursiveSceneFrustumCheck se uzlu nastaví razítko aktuálního snímku, to znamená, že uzel R stromu bude v tomto snímku viditelný a vše uvnitř může být tedy vykreslováno. Rekurzivně se zkontroluje, zda jsou viditelní i potomci. Pokud není viditelný žádný potomek, metoda skončí. Takto se rekurzivně pokračuje až k listům R stromu, které obsahují indexy na plošky do vertex bufferu. Poté se prochází kontejnery pro trojúhelníky, které odpovídají použitým materiálům a ploškám vykresleným těmito materiály (třída RenderCache). Všechny tyto kontejnery se připraví na vykreslování volaním Begin. Následně se přidají indexy plošek viditelných uzlů do příslušných kontejnerů (instancí třídy RenderCache) k vykreslení. Zavolá se metoda End na všech instancích třídy RenderCache a tím se vykreslí aktuálně viditelné plošky pro každý materiál. Nakonec vykreslovací metoda vykreslí všechny dynamické objekty scény. Nejdříve ale vždy zkusí, zda je dynamický objekt v pohledovém jehlanu hráče.

5.5 Uživatelské rozhraní

Pro implementaci uživatelského rozhraní bylo původně použito Windows Forms, pro jejichž tvorbu má Visual Studio 2008 nástroj. Před vykreslením nabídek ale bylo potřeba, aby aktuálně vykreslovaná grafická tabule v paměti (back buffer) byla ta s pořadovým číslem nula. Jinak by se nabídka nevykreslila. V praxi to bylo řešeno tak, že těsně před požadavkem na vykreslení nabídky se několikrát naprázdno pustilo vykreslení scény, než se přerotovalo na požadovanou tabuli. To nebylo příliš efektivní. Viděl jsem tuto oblast jako velkou příležitost budoucího rozvoje. Po určité době zkoumání se mi ale podařilo nalézt alternativu. Ukázkové projekty DirectX SDK totiž používají grafické uživatelské rozhraní pomocné vrstvy DXUT. Vrstva DXUT je tvořena mnoha obsáhlými zdrojovými soubory obsahující implementaci různých grafických a technických prvků. Předpřipravuje také šablonu herního cyklu. Jejím problémem je však pomalost. Z této vrstvy se mi podařilo vyčlenit pouze grafické uživatelské rozhraní a napojit ho na mnou vytvořenou implementaci herního cyklu a abstrakci grafické karty. Nejen že se tím velmi zjednodušila implementace grafického uživatelského rozhraní, ale grafický projev uživatelského rozhraní je teď

mnohem profesionálnější. DXUT obsahuje všechny potřebné grafické prvky (popisky, textová pole, rozbalovací lišty, přepínače apod.).

5.6 Převzaté zdrojové soubory

V programu jsou kvůli lepší modularitě a úspoře vývojového času použity převzaté zdrojové kódy. Do projektu byly vloženy zdrojové soubory zmiňované vrstvy DXUT, která je součástí vývojového balíku DirectX. Dále byly vloženy zdrojové soubory projektu CMarkup. Ten je volně ke stažení na internetu. Slouží k editaci a čtení XML souborů v programech C++. Jedná se o odlehčenou, objektovou implementaci zpracování XML souborů s velmi jednoduchým použitím. Je realizována pouze dvěma soubory. Ty jsou v projektu umístěny v kontejneru XML. Funkcionalita integrovaných souborů byla testována.

5.7 Použití XML

Je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Jazyk je určen především pro výměnu dat mezi aplikacemi a pro publikování dokumentů. Jde od standardní formát pro výměnu informací. V programu je použita řada souborů, které obsahují konfigurace herních objektů různého typu (vlastnosti materiálů, herní mapy, prostorové vlastnosti hráčů, vlastnosti zbraní apod.). Všechny tyto soubory jsou reprezentovány v XML. Důvodem je čitelný a unifikovaný formát konfiguračních souborů. V programu jsou ke zpracování XML souborů použity zdrojové soubory zmiňovaného projektu CMarkup.

6. Umělá inteligence

6.1 Popis vlastností

Téměř v každé informatické úloze nalezneme problémy, nad kterými je potřeba se hlouběji zamyslet před samotnou implementací. Umělá inteligence je jistě jedním z nich. Umožní to specializaci některých algoritmů a přesnost cílů. To se pozitivně projeví na kvalitě návrhu systému umělé inteligence, na následném zkrácení vývojového času a výpočetní efektivnosti. Z pohledu této bakalářské práce chápu následující řádky jako argumenty, které pomohou v rozhodování, jakou z variant návrhu umělé inteligence zvolit.

Klíčové charakteristiky:

- Prostředí je velmi dynamické, dochází k časté interakci mezi entitami.
- Scéna pro pohyb agentů nebude obvykle moc velká, dlouho by se hledali.
- Agentů může být více než jeden, principy chování však mají stejné.
- Předpokládá se, že agent zná mapu. Ví, kde co najít a jak se tam dostane nejrychleji.
- Agent neví, kde je hráčův avatar (postava hráče), pokud ho nevidí, případně neviděl v předešlých chvílích.
- Agent má určitou dočasnou paměť na události.
- Obecný pilíř chování je se před vlastním bojem co nejlépe připravit (zdraví, zbraně).
- Chceme co nejvíce snímků za sekundu.
- Konzistence vzájemného působení agentů.
- Implementace návrhu by měla být realistická.

6.2 Obecný přístup k návrhu

Ve velké většině případů je nejlepší cestou ke kvalitě návrhu jeho podobnost s realitou. Proto se zkusím na agenta dívat jako na sebe. Jak bych reagoval na různé podněty? Jak bych přemýšlel, kdybych chtěl dosáhnout nějakého strategického cíle? Dalo by se využít nějaké vlastnosti plánů na splnění takových cílů? Tento postup má za cíl snížení časových nároků algoritmu umělé inteligence, lepší pochopitelnost a čitelnost zdrojového kódu následné implementace.

6.3 Herní objekty

Systém objektové reprezentace všech entit hry (včetně scény) je již vyřešen a účelem této kapitoly není jeho diskuse. Uvedu však logické objekty přímo související s umělou inteligencí spolu s popisem jejich funkce.

- Hráč – reprezentace agentů a avatara (herní postava ovládaná samotným uživatelem),
- Správce mapy – informace o scéně, navigační informace,
- Vybavení – lékárničky, zbraně, …

6.4 Navigační systém

Navigační systém považuji za část programu, která výrazně ovlivňuje jeho rychlost a komfort pohybu agenta po mapě. Přiznám se, že před absolvováním přednášky umělých bytostí jsem považoval tuto část za něco podobného černé magii. Byl jsem přesvědčen, že tak složitý problém, jako je navigace po mapě, se nedá spočítat tak rychle, aby mohl být aktualizován dostatečně rychle a upokojil tak požadavky na přesnost umělé inteligence. Když jsem však vyšel ze zmiňovaných charakteristik žánru a to jmenovitě „Předpokládá se, že agent zná mapu. Ví, kde co najít a jak se tam dostane nejrychleji.“ a „Agentů může být více než jeden. Principy chování však mají stejné.“, napadlo mě, že znalosti o mapě mohou byt sdíleny a předpočítány dopředu při tvorbě scény. Podle mého názoru je tedy nejvhodnější reprezentace pro účel a rozsah mého subsystému 3D akční hry systém bodů s pomocnými informacemi podobný tomu ve hře Unreal Tournament 2005. Systém bodů bude tvořen grafem, v jehož každém uzlu jsou uloženy navigační informace. Například cesty do ostatních uzlů grafu jako vektor souřadnic, po kterých se tam dostanu, či cesty k různým typům vybavení. Hráč se zkrátka jen dotáže na nejbližším uzlu, kterou cestou se má dát. Uzel mu vrátí referenci na cestu k požadovanému bodu. Výhodou tohoto přístupu je velká rychlost, jelikož se navigační informace netvoří za běhu hry, ale jsou známy již předem (předzpracovány během nahrávání mapy). Negativem je větší paměťová náročnost spojená s uložením této struktury a zdržení v počátku při nahrávání mapy, kdy se tyto cesty generují pro všechny uzly.

6.5 Možnosti návrhu umělé inteligence

Při výběru vhodného typu umělé inteligence jsem vyšel ze specifik žánru. Vzhledem k argumentům z [1] 3. část je podle mého názoru nejlepší variantou reaktivní plánování. Před výběrem umělé inteligence bych přiblížil pojmy z definice reaktivního plánování konkrétnímu případu zmiňovaného subsystému.

Stav

Agent FPS hry je z hlediska variability stavů velmi jednoduchý. Vyplývá to z jednoduchých principů vítězství ve hře.

a) Pokud je na tom agent dobře ve smyslu výbavy a zdraví, chce zvýšit své skóre.
b) Pokud je na tom špatně, raději se bojům vyhýbá, protože šance na vítězství při konfrontaci jsou malé.
c) Když bojovník nikoho nevidí, snaží se co nejvíce posunout od principu b) k a).

Možnosti posunu vidím jak na úrovni fyzické (výbava), tak mentální. Například pokud zrovna nejsem vytížen bojem, mohu začít s analýzou předchozího jednání protivníka. K jednoduchému povelu na náhodný pohyb po mapě tak mohu připojit výpočet části analýzy, aniž by se aplikace zpomalila. Po troše zamyšlení se tak zdá, že nejvhodnější variantou umělé inteligence je reaktivní plánování s hybridními prvky v těch na výpočetní složitost nenáročných stavech.

Reaktivita

Vzhledem k dynamičnosti a časté interakci zcela přirozený požadavek.

Reaktivní plán a efektivita

Půjdou proti sobě. Čím dokonalejší plán, jemnější v detailech, tím více rozhodování a času zabere výběr následující elementární akce. Maximalizace detailnosti chování při minimálním výpočetním času je hodnotící funkcí návrhu umělé inteligence.

Pokud jde o metody, jak toho dosáhnout, je myslím vzhledem realističnosti implementace (diskutuji možnosti realizace pouze jedním vývojářem) vhodnější dát přednost symbolickým metodám před konekcionistickými a to konkrétně if-then pravidlům. If-then pravidla jsou popisem chování agenta ve tvaru: „Pokud platí něco (např. agent vidí nepřítele), reaguj na to akcí definovanou v těle podmínky (např. střílej) a vyskoč z bloku if-then pravidel.“ Tato pravidla jsou pod sebou v bloku seřazena podle priority tak, že v podmínce testované okolnosti s vyšší prioritou jsou umístěny výše. Tím je zajištěno, že reakce na události s vyšší

prioritou jsou vykonány přednostně. Jejich konkrétní návrhová podoba je předmětem následujícího odstavce. Možnost analýzy chování nepřítele nechávám jako prostor budoucího rozvoje. Stejně tak konekcionalistické implementace agentů (např. pomocí neuronových sítí).

6.6 Realizace vybraného návrhu

Teď, když je zhruba znám konkrétní směr, kudy se dát, zkusím popsat konkrétní realizaci. Vzhledem k otázkám a pravidlům v části Obecný přístup k návrhu je myslím zcela přirozené, reprezentovat stav mysli agenta jako stavový automat, kde stavy mají svou vlastní hierarchii if-then pravidel. Konkrétní implementaci stavu je nejlepší reprezentovat návrhovým vzorem „State“.

Tento systém dle mého názoru zcela přirozeně popisuje myšlenkové postupy člověka. Člověk například při vlastním boji také nepřemýšlí, kudy se dostane na své oblíbené místo se zbraní. Naopak při nenáročných činnostech, jako je běh po mapě, analyzuje chování nepřítele. Vůbec při tom nepřemýšlí, jestli udělat úhyb vpravo, či vlevo. Stavy se zkrátka velmi přirozeně snaží kopírovat lidské myšlení, které tím zcela přirozeně a chytře zjednodušuje myšlenkové postupy při řešení každodenních lidských úloh. Stavy také zcela přirozeně postihují jeden z problémů reprezentace lidského chování. Z [1] 3. část: „V druhé řadě nemůžeme měnit pozornost virtuálního člověka z jednoho úkolu na druhý, jak se nám zamane – člověk při „přechodu“ mezi dvěma úkoly dělá určité netriviální mezikroky. Například pokud při zalévání zahrady dostane hlad, položí nejprve konev, umyje si ruce, než se dojde najíst.“ Návrhový vzor „State“ reprezentuje tyto mezikroky obligátními funkcemi Enter a Exit.

Při přemýšlení, jak reprezentovat konkrétní if-then pravidla jsem si uvědomil některé souvislosti, které ovlivnili návrh.

1) Velká část operací je společná všem stavům.
2) Pokud v těle podmínky provedu dvě operace, které si v některém ohledu protiřečí, dojde k nekonzistenci výsledku, například pokud bych provedl operace „jdi dopředu“ a „jdi dozadu“ ve stejný moment. Tento příklad je nesmyslný, ale tyto operace mohou být včleněny do větších celků. Pak může být složitější nahlédnout jejich kolize.

3) V podstatě se dá říci, že složité motorické funkce jsou kombinací ortogonálních elementárnějších pod-operací. Ortogonální znamená, že jsou nezávislé ve smyslu 2).

Vzhledem k tomu je podle mého názoru nejvhodnější implementovat nejdříve zcela elementární (ortogonální) operace přímo v předkovi všech stavů či ve třídě reprezentující agenta, kvůli opakování téhož ve více konkrétních stavech znovu, a poté z nich skládat složitější operace opět v předkovi (nebo třídě agenta), či již v konkrétních stavech.

Myslím, že je dobré organizovat složené operace v hlavičkových souborech projektu tak, že jsou po skupinách v odstavcích, kde odstavec je skupina operací, které jsou závislé. Z nich pro další vyšší vrstvu operací mohu vybrat pro každou novou operaci jen jednu operaci. Tak se zajistí konzistence výsledku nově vzniklé operace. Co do čitelnosti vzniklých if-then pravidel se mi osvědčil systém jedné složené operace v těle každé z podmínek. Nemusí se řešit konzistence těla podmínky. O vložené operaci vzniklé předešlým postupem totiž vím, že je konzistentní.

6.7 Zhodnocení

Po implementaci části popsaného systému umělé inteligence musím potvrdit nízké časové nároky algoritmu a dobrou přehlednost implementace definující chování agentů v různých stavech. Po naimplementování dostatečné základny složených operací je již velmi rychlé přidat nový stav nebo chování agenta, přestože výsledek vypadá velmi pracně a efektně. Konečná konkrétní podoba umělé inteligence agentů v programu Valahalla je odpovídající vzhledem k časovým prostředkům na její realizaci, není dokonale detailní a reálná. Nabízí automatickou navigaci po mapě, sběr lékárniček při zranění, sběr lepších zbraní pro zvýšení šancí na úspěch při boji a implementaci chování při samotném boji (úkroky, zaměřování a střelba po nepříteli). Co se týče paměti agenta, ta obsahuje aktuálně viditelný cíl, na který agent útočí a aktuální cestu k uzlu, kam má agent namířeno.

6.8 Možný rozvoj

Dle mého názoru je dobrý návrh sytému jednou z nejdůležitějších věcí při vývoji. Tato kapitola víceméně popisovala argumenty, které mě vedly k cíli. Trochu závidím lidem disponujícím většími lidskými zdroji a vědomostmi, že mohou přivést k životu dokonalejší

aparáty umělé inteligence, které jsou schopné analýzy protihráčů a přizpůsobení taktiky boje, typu neuronových sítí, genetických algoritmů apod. Člověk ale někde začít musí a tento projekt to umožňuje. Je tím pravým místem, kde se dají zkoumat a demonstrovat jednotlivé přístupy řízení agentů ve vizuální podobě.

7. Porovnání s existujícími projekty

7.1 Duke Nukem 3D

Hra Duke Nuken 3D je popsána na příslušných stránkách wikipedie [13].“ Duke Nukem 3D je 3D akční počítačová hra, vytvořená společností 3D Realms, vydaná v roce 1996 společností Apogee Software. Patří mezi nejznámější akční hry. Námětem navazuje na starší 2D akční hry Duke Nukem a Duke Nukem 2. Nástupcem této hry je dosud stále nevydaný Duke Nukem Forever.“ Od projektu Cybotech se liší většími možnostmi při tvorbě map a přítomností zvláštních druhů vybavení. „Ve hře jste mohli pomocí tzv. Jetpacku létat, pomocí nočního vidění vidět i do tmavých koutů, díky steroidům jste mohli běhat jako chrt a kopat jako kůň. Navíc tu existovala věc jako tzv. Holoduke, vaše vlastní holografická projekce, která sloužila k oklamání nepřátel. Převratné bylo, že firma také nabídla svým hráčům software pro výrobu vlastních úrovní. Hlavními komponenty tohoto softwaru jsou build a editart. Kolem tohoto softwaru se nakonec vytvořila tak rozsáhlá hráčská a budovatelská komunita, že vzniklo mnoho nových utilit usnadňujícím nadšencům do této hry práci.“ Právě styl editoru jako mříže pro zakreslení půdorysu mapy je touto hrou inspirován.

7.2 Unreal Tournament 2005

Hra Unreal Tournament je výhradně multiplayer 3D akcí. Vytvořil ji tým Epic Games. Hra se specializuje výhradně na boj hráč proti hráči. Je tak stylem podobná projektu Valahalla. Unreal Tournament se od mého projektu liší je především propracovaností. Na této hře totiž spolupracovalo velké množství odborníků ze všech oblastí informatiky, do kterých hra zasahuje. Má vynikající grafiku, spoustu herních módu a propracovanou umělou inteligenci. Pro navigaci agentů po mapě používá navigační graf. Tento systém byl inspirací pro realizaci navigace v mém projektu (skrze přednášku Umělé bytosti). Systémy navigace jsou tedy u obou projektů podobné.

7.3 Quake 3 Arena

Tato 3D akční FPS hra byla vydána v roce 1999 společností Id Software. Je jednou z prvních her výhradně zaměřených na hru více hráčů (případně agentů). Stala se tak inspirací pro řadu dalších projektů. Hra obsahuje 26 map (arén), které hráč musí projít postupně a všude vyhrát. V každé další úrovni se zvyšuje obtížnost nepřátel. S touto hrou nebyl distribuován subsystém pro tvorbu nových map. Tento subsystém byl velmi úspěšný

z hlediska prodejnosti licencí jiným herním studiím pro tvorbu dalších herních projektů, které subsystém hry Quake 3 Arena využívaly jako platformu.

7.4 Half life

Pár slov o hře Half Life lze nalézt na příslušné stránce wikipedie [14]. „Half-Life je akční počítačová hra ve stylu sci-fi, typ FPS, vyvinutá v roce 1998 firmou Valve Software a vydaná společností Sierra Studios. Half-Life běží na vylepšeném grafickém subsystému her Quake. Hra získala mnohá ocenění od herních kritiků i samotných hráčů a byla mnohými herními časopisy po celém světě vyhlášena jako akční hra roku, načež se posléze stala kultovní. Jedná se o jednu z nejlepších FPS her. Existují stovky modifikací, které tuto hru upravují, většinou jsou vyráběny amatéry, mezi nejznámější modifikace patří Counter-Strike." Tato hra se od projektu Valahalla liší především stylem. Half life je totiž hrou založenou převážně na příběhu. Myslím, že hraní této „střílečky“ velmi připomíná spíše interaktivní film. Half life je však důkazem, že 3D akční hry mají vždy něco společného. Příkladem může být modifikace této hry Counter-Strike, která se stala populárnější než samotný Half Life. Myslím, že po přidání určité funkcionality by mohl být projekt Valahalla použit i pro tvorbu her s příběhem, herním stylem podobným hře Half Life.

7.5 Editor 3D hry

Tento projekt je diplomovou prací Jaroslava Staňury. V této práci je navržen a implementován editor 3D hry. Systém umožňuje vytvářet úrovně a interaktivní objekty pro hry. Z vytvořených souborů lze definovat kompletní hru. Modelář poskytuje funkce pro vkládání a editaci geometrických primitiv a dalších objektů potřebných pro 3D hry. Dále je implementováno například generování terénů a stínování scény. Je umožněno psaní skriptů pro řízení animací a interakci s hráči. Této práce jsem si všimnul, při hledání inspirace pro svůj vlastní projekt. Od projektu Valahalla se primárně liší tím, že se jedná pouze o samostatný editor úrovní. Naproti tomu nabízí navíc některé nástroje. Zmínil bych například možnost modelování terénu, import modelů 3D objektů ze souboru 3ds, možnost psaní skriptů ovlivňujících chování některých elementů či detailní nastavení světel ve scéně. Co mi však v projektu chybí, je podpora zvuků. Projekt Valahalla naproti tomu nabízí herní jádro (implementaci herní logiky a základní umělou inteligenci agentů).

8. Závěr

8.1 Dosažené cíle

Na počátku tohoto projektu byla specifikace definující cíle a vlastnosti produktu, který má vzniknout. Nyní, po dokončení jsem až překvapen, jak moc se výsledek kryje se zadáním. Některé z cílů se samozřejmě musely omezit, jiné rozšířit.

Prvním z cílů bylo přiblížení k moderním hrám. Vzhledem k časovým prostředkům se, myslím, tento bod velmi zdařil. Projekt Valahalla má opravdu tvar hry. Není sice zdaleka tak rozvinutá co do funkcionalit a detailnosti grafiky jako profesionální hry od firem, jako je například Id Soft (série Quake), obsahuje však plně funkční prvky (editor map, herní stavy, reprezentace hráčů a zbraňových systémů, umělá inteligence nepřátel), které tvoří kostru klasické 3D akční hry.

Jednou z hlavních otázek byla práce s mapami jednotlivých úrovní. Podařilo se dosáhnout systému map uložených v XML souborech, které je možné nahrát a upravit v editoru, nebo použít pro samotnou hru. Obecnost mapy však musela být z realizačních důvodů omezena na půdorysovou variantu ze specifikace. Téma reprezentace map poskytuje možnost budoucího rozvoje projektu. Uživatelům projektu se navenek nijak nespecifikuje formát souborů (možnost ruční editace map v textovém editoru), stejně jako u téměř všech komerčních projektů. K práci s mapami je určen pouze editor přístupný z aplikace.

Co se týče předpokládaných objektů, je vidět, že v době specifikace projektu nebyly známy všechny potřeby a vazby mezi budoucími třídami. Náčrt objektů ze specifikace se však v základních rysech přibližně shoduje s dokončeným projektem. Hlavní změnou oproti specifikaci je rozšíření obecnosti práce s kamerou. Ta měla být původně centralizovaná (jedna instance). Ukázalo se však velmi výhodné, mít možnost přepínat pohled do scény mezi jednotlivými objekty (především hráči). To také umožňuje použít nezávislou kameru nad scénou pro její pozorování (chování agentů). Přibylo také velké množství tříd, jejichž potřeba přišla až při řešení konkrétních témat.

Použitá literatura byla rozšířena o české tituly, které pomohly při začátcích práce s DirectX. Později přibyla i nutnost nastudovat si některá témata z počítačové grafiky (např. řádkový algoritmus pro vyplňování polygonů). Celkově jsem s dosaženými cíli spokojen. Projekt však obsahuje některé zajímavé oblasti, které by stály za rozšíření. Tyto oblasti by se mohli stát námětem pro budoucí rozvoj či samostatné zkoumání.

8.2 Možnosti budoucího rozvoje

Subsystém pro hru představuje velmi široké téma. Projekt Valahalla tak i po dokončení obsahuje velké možnosti pro budoucí rozvoj. V této části textu bych rád uvedl seznam oblastí, kterým bych se chtěl věnovat. Některé z nich by se mohli stát i námětem pro zkoumání v diplomové práci.

Uživatelské rozhraní

Většina moderních her má velmi efektní a graficky propracovaná uživatelská rozhraní. Jeho realizace by měla využívat DirectX. Otázkou však je použití efektních nabídek v editoru. Editory jsou totiž oblastí, kde je myslím účelnost a snadná pochopitelnost nadřazena nad grafickou bohatost.

Umělá inteligence

Tato oblast je jednou z nejvhodnějších pro budoucí rozšíření. Myslím, že by se mohla stát hlavním tématem mého budoucího studia a námětem na diplomovou práci. Umělá inteligence se totiž dá kombinovat s teorií neuronových sítí, genetickými algoritmy, fuzzy logikou, neprocedurálním programováním, různými konstrukcemi stavových automatů apod. Pomocí statistik by například bylo velmi zajímavé sledovat, jak si jednotlivé druhy inteligencí vedou při vzájemné konfrontaci.

Nové 3D modely

Jednou ze slabších stránek subsystému je menší množství 3D modelů (meshů). V budoucnu by bylo dobré tuto paletu rozšířit, aby se mapy staly vizuálně zajímavější.

Modelování terénu

Toto téma je typické zejména pro exteriéry. Představuje zvlnění plochy podlahy tak, aby připomínala přírodní nerovnosti. Bylo by to velmi zajímavé rozšíření funkcí editoru map. Otázkou by asi byla reprezentace plochy s kopci a nížinami v souboru mapy.

Obecná triangulace mapy

Pro jednodušší triangulaci a mapování textur na trojúhelníkové plošky je použito omezení úhlu stěn vzhledem k ose x kartézského systému na násobky 45 stupňů. Zajímavým rozšířením by bylo zobecnění na libovolný úhel stěn. Další možností by mohl být plně trojrozměrný návrh map, který by umožňoval úplnou svobodu při vytváření map nebo umisťování světelných zdrojů do scény (lampy, oheň).

3D model mapy

Editory map moderních počítačových her umožňují modelovat mapu libovolně ve 3D prostoru. Přechod návrhu scény mapy z půdorysového na zcela trojrozměrný představuje programátorsky a časově asi nejnáročnější rozšíření.

Síťová podpora

Režim více hráčů propojených lokální sítí, či internetem představuje jeden z nejpopulárnějších způsobů hraní 3D akčních her. Rozšíření o síťovou komunikaci by tedy bylo velmi zajímavé. Otevírá však kapitolu synchronizace jednotlivých instancí hry. Komunikace po sítí pomocí zpráv je totiž asynchronní a může způsobovat nekonzistenci stavů jednotlivých instancí hry.

8.3 Zhodnocení práce na projektu

Práce na tomto projektu mě naučila mnoho z oblastí aplikované matematiky, grafiky a umělé inteligence. V praxi jsem si vyzkoušel mnoho postupů při návrhu a psaní zdrojového kódu. Asi nejproblematičtějším místem při vývoji byl převod logické reprezentace mapy na úrovni struktur do tvaru, který je schopný zpracovat grafická karta, a časová náročnost implementace vzhledem k velké rozsáhlosti praktické projektu. Často jsem musel bolestivě volit, co je pro projekt podstatné a co ne. Dalším velmi náročným místem byla práce s hardwarem a objekty, které s ním umožňují pracovat. Pokaždé se mi však po určité době strávené studiem možných řešení, podařilo vyřešit problémy, které mi tento projekt přichystal a dotáhnout ho do myslím celkem povedené formy.

A. Reference, použité knihovny a zdroje

A.1 Použitá a prostudovaná literatura

1. Brom Cyril: Action Selection for Virtual Humans in Large Environments" (in Czech) dizertační práce, MFF UK, CZ, 2008
2. Buckland Mat: Programming Game AI by Example
   Wordware Publishing, USA, 2005, ISBN: 1-55622-078
3. Ericson Christer: Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology)
   Elsevier, USA, 2005, ISBN: 1-55860-732-3
4. Gama Erich, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns Elements of Reusable Object-Oriented Software
   KevinZhang, USA, 1997
5. Luna Frank D.: Introduction to 3D Game Programming with Direct X 9.0c: A Shader Approach (Wordware Game and Graphics Library)
   Wordware Publishing, USA, 2006, ISBN: 1-59822-016-0
6. Miller Tom: Programujeme 3D hry v jazyce C#
   Computer Press, CZ, 2006, ISBN: 80-251-1126-1
7. Walnum Clayton: Programujeme grafiku s Microsoft Direct3D
   Computer Press, CZ, 2004, ISBN: 80-251-0136-3
8. Young Vaughan: Programming a Multiplayer FPS in DirectX (Game Development Series)
   Charles River Media, USA, 2005, ISBN: 1-58450-363-7

A.2 Elektronické zdroje informací

9. Brom Cyril: Slajdy z přednášky Umělé Bytosti
   MFF UK, CZ, 2008
10. Pelikán Josef: Slajdy z přednášky Grafika I
    MFF UK, CZ, 2007
11. Studenti: Slajdy ze semináře Návrhové vzory
    MFF UK, CZ, 2008
12. DirectX, Wikipedia, http://cs.wikipedia.org/wiki/DirectX, 28. 11. 2009
13. Duke Nuken 3D, Wikipedia, http://cs.wikipedia.org/wiki/Duke_Nukem_3D, 28. 11. 2009
14. Half Life, Wikipedia, http://cs.wikipedia.org/wiki/Half-Life, 28. 11. 2009
15. OpenGL, Wikipedia, http://cs.wikipedia.org/wiki/OpenGL, 29. 11. 2009
16. Dijkstra's algorithm, Wikipedia, http://en.wikipedia.org/wiki/Dijkstra's_algorithm, 8. 5. 2010

A.3 Použité knihovny

- WinAPI
- DirectX SDK, August 2009
- Zdrojové soubory z elektronické přílohy knih [2] a [8]

A.4 Ostatní

- Ilustrace a schémata z knihy [8] Programming a Multiplayer FPS in DirectX (viz Použitá a prostudovaná literatura). Obrázky byly lehce upraveny a přeloženy do češtiny.

- Trojrozměrné modely hráčů, textury a zvuky z multimediální přílohy knihy [8].

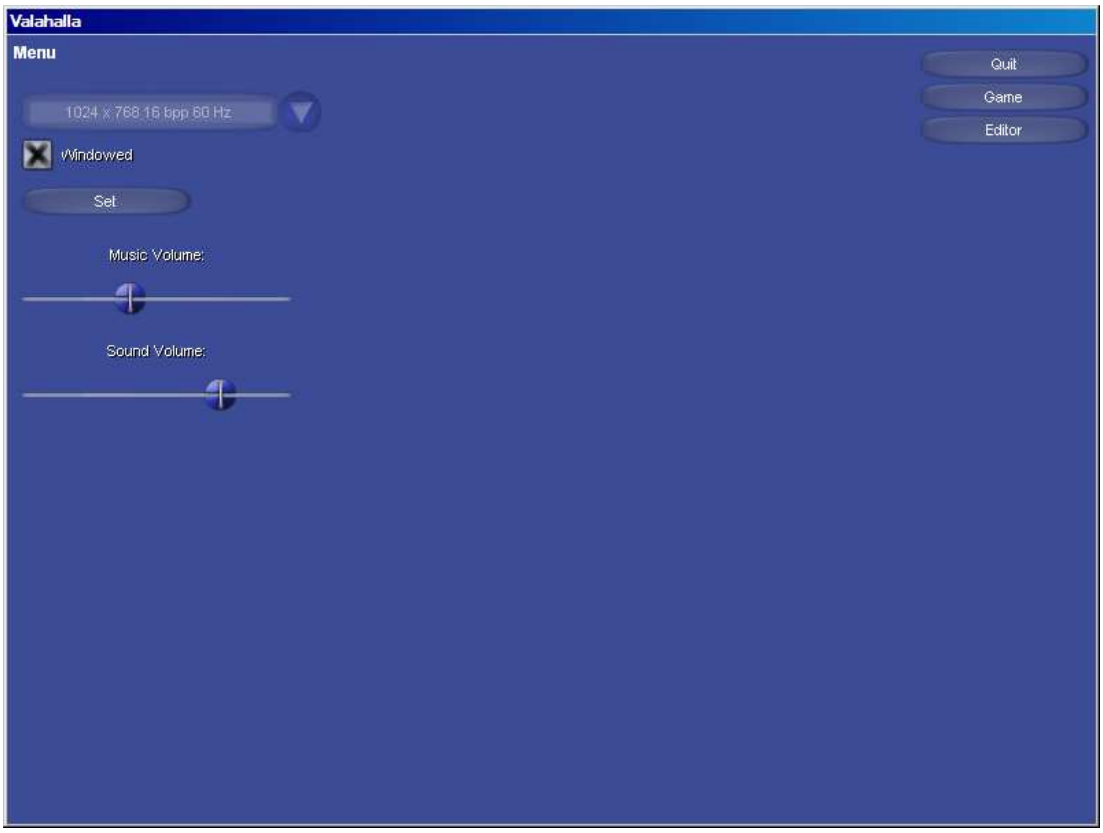
B. Uživatelská příručka

B.1 Úvod

Nezbytnou součástí jakéhokoliv programu je uživatelská příručka. Poslední kapitolou této práce bude tedy příloha s tímto obsahem. Obsah je rozdělen na celky příslušné různým částem programu. Postupně zde vysvětlím používání nastavení grafiky, hry, editoru a ovládání vlastní hry. Po spuštění aplikace se jako první objeví menu s nastavením grafiky a volbami pro přepnutí do hry a editoru. V editoru je možné vytvářet, ukládat a editovat vlastní mapy, které je poté možno spouštět ve hře.

B.2 Hlavní menu

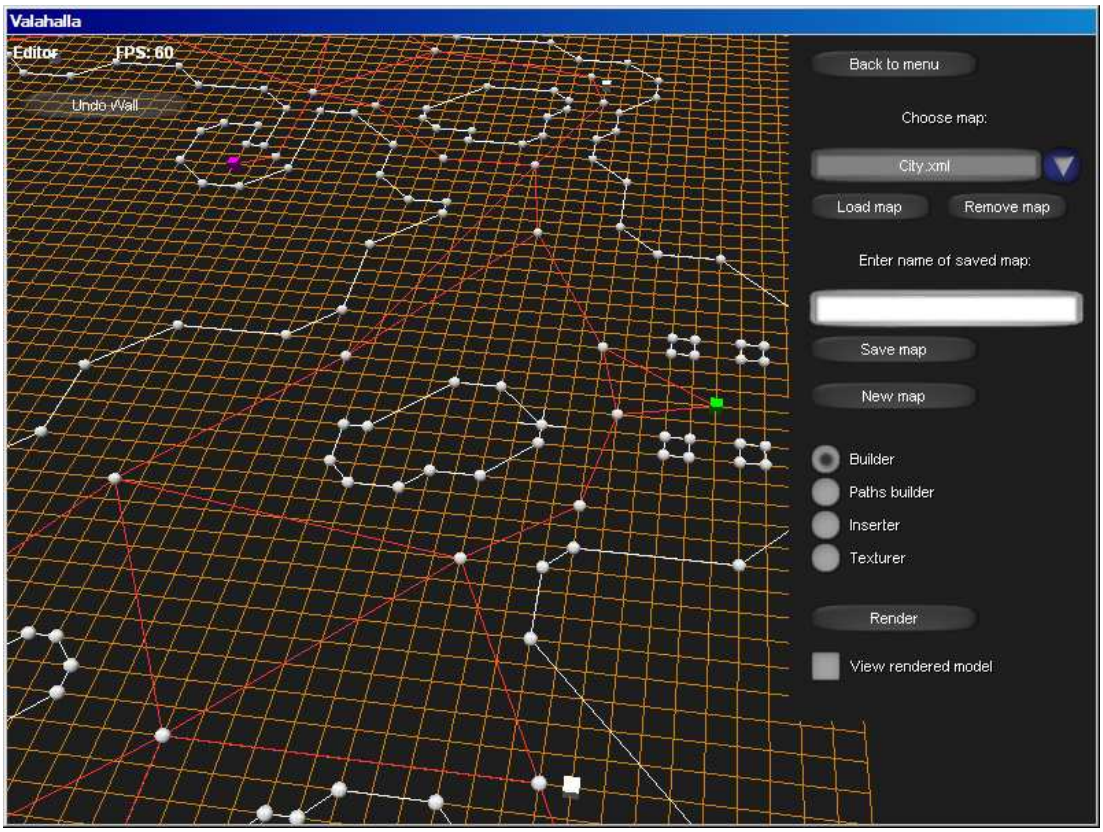
Při spuštění programu se jako první zobrazí hlavní menu. V hlavním menu můžeme vidět několik polí s nastaveními. Obrázek č. 34 ilustruje ovládací prvky hlavního menu. V pravé části je několik tlačítek. Tlačítko „Quit“ ukončí aplikaci, tlačítko „Game“ přepne uživatele do herního stavu (nahrávání map, boj proti agentům) a tlačítko „Editor“, které přepne uživatele do editační části programu, kde je možné vytvářet, ukládat a editovat vlastní mapy. V levé části obrazovky hlavního menu je nabídka grafických módů, které lze nastavit. Jednotlivé volby mají tvar „Rozlišení, Barevná hloubka, Obnovovací frekvence“. Dále je vidět zaškrtávátko „Windowed“. To určuje, zda se aplikace spustí v malém okně, nebo přes celou obrazovku. Rozlišení a ostatní parametry zobrazení je možné nastavit pouze, pokud je nastaveno zobrazení přes celou obrazovku. Posuvné ovládací prvky umožňují nastavení hlasitosti hudby a zvukových efektů. Posledním prvkem menu je tlačítko „Set“, které aktivuje grafické nastavení zvolené uživatelem. Aktivace nastavení může trvat několik vteřin.



Obrázek č. 34 – Hlavní menu s nastaveními grafiky, hry a zvuku.

B.3 Editor

Jednou z nejkomplikovanějších součástí hry z hlediska ovládání je právě editor map. Přesto je v porovnání s editory profesionálních produktů triviální. Editor je tvořen hlavním menu a několika editačními nástroji. Ty jsou realizovány jako módy editoru. Mezi editačními módy se přepíná pomocí přepínače v pravé části obrazovky (Builder, Paths builder, Inserter, Texturer). Mapa se zakresluje do mřížky pomocí různých nástrojů. Pohyb nad mřížkou a nad návrhem mapy se realizuje skrze klávesnici a myš. Na klávesnici se ovládá směr pohybu klasicky pomocí tlačítek W (vpřed), A (vlevo), S (vzad), D (vpravo). Aplikace nastaveného editačního nástroje na editovanou mapu se provede stiskem levého tlačítka myši. Myší se také ovládá směr pohledu a pomocí přetáčecího kolečka se přibližuje a oddaluje od zakreslovací mřížky kamera. Ovládání je tedy velmi podobné tomu z vlastní hry (obzvláště při zapnutí „ptačí perspektivy“ viz. dále). Ovládání pohybu je však aktivní pouze, pokud nejsme v režimu menu editoru, kdy je aktivní grafické uživatelské rozhrání. Přepínání mezi ovládáním grafického uživatelského rozhraní a vlastní editací se provádí stiskem klávesy Esc.



Obrázek č. 35 - Pohled na editovanou mapu a menu editoru.

Klávesnice

Esc - přepínání mezi ovládáním grafického uživatelského rozhraní a vlastní editací

W - posun pohledové „kamery“ dopředu

S - posun vzad

A - posun vlevo

D - posun vpravo

Myš

Pohyb myši - pohyb kurzorového kříže po mříži a natočení kamery.

Levé tlačítko - editace, provedení v menu nastavené operace (např. vložení zbraně, uzlu stěny, aplikace nastavené textury na zaměřenou zeď)

Pravé tlačítko - vymazání zaměřeného navigačního uzlu spolu s na něj napojenými hranami

Kolečko - přibližování a oddalování od mřížky

Tlačítko „Undo“ u jednotlivých editačních módů slouží k vrácení zpět naposledy provedeného editačního kroku. Vrátí poslední operaci v daném módu (např. odebrání stěny, cesty, zbraně).

Hlavní menu

V hlavním menu editoru je možné nahrát mapu ze souboru. Nastavenou mapu nahrajeme tlačítkem „Load map“. Pokud chceme uložit editovanou mapu, stačí do pole pro text napsat jméno, pod kterým se má mapa uložit a klepnout na tlačítko „Save map“. Stejný postup jako pro nahrávání mapy, použijeme i pro mazání uložené mapy. Jen se použije tlačítko „Remove map“. Přepínač v sekci editačních režimů slouží k přepínání editačních režimů editoru. Jednotlivé editační režimy jsou popsány níže. Tlačítko „Back to menu“ vrátí uživatele zpět do hlavního menu. Dále je přítomno tlačítko „Render“, které vytvoří grafický model mapy tak, jak ho uživatel uvidí během hry. Pokud tento model chceme vidět, je nutné stisknout tlačítko „Render“ a poté zaškrtnout pole „View rendered model“. Při prohlížení modelu není možné mapu editovat. Po první změně mapy je nutné znovu stisknout tlačítko „Render“. Starý model po editaci totiž už není platný a neodpovídá aktuálnímu půdorysovému návrhu. Neplatný grafický model není možné zobrazit a je nutné ho před vykreslením obnovit zmiňovaným tlačítkem „Render“.

Obrázek č. 36 - Hlavní menu editoru.

Builder

Editační stav pro tvorbu půdorysových návrhů mapy. Pomocí levého tlačítka myši a kurzorového kříže se do mříže zakreslí tvar nových místností. Začne se počátečním bodem a poté se postupně přidávají stěny. Nakonec se spojí první uzel tvořící tvar místnosti s posledním a tvar místnosti se uzavře. Žádné dvě stěny se nesmí křížit. Nové stěny mohou být zakresleny jen pod úhlem násobku 45 stupňů. To, že má stěna požadovaný úhel a nekříží se s žádnou jinou stěnou, se pozná vizuálně podle změny barvy vkládané hrany z červené na bílou. Uvnitř editoru se totiž testuje, zda hrana neporušuje tyto požadavky. Pokud jde o místnosti, doporučuji vzhledem k lepší hratelnosti vytvářet mapy typu jedné velké strukturované místnosti, kde jsou všechny prostory propojeny. Důvodem je, že agenti se mohou volně pohybovat po celém prostoru. Jinak by se mohlo stát, že se agent objeví v některé separované místnosti sám a bude jen čekat na nepřítele. Při vytváření půdorysového návrhu mapy je tedy nejlepší, vytvořit první jednu velkou místnost, do které se postupně zakreslují menší místnosti pomocí stěn. Hra již obsahuje několik map, které je možno otevřít v editoru a inspirovat se jejich návrhem a strukturou pro lepší pochopení návrhu map a funkcí editoru. Před vkládáním stěn je nutné se první přepnout do texturovacího režimu a nastavit texturu nově vkládaných zdí.

Obrázek č. 37 – Při vytváření stěn není potřeba žádné nastavení. Tlačítko „Undo Wall“ umožňuje vracení naposledy přidaného uzlu (stěny).

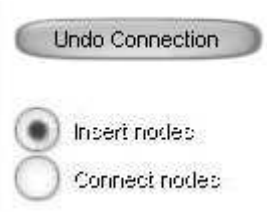
Inserter

Slouží ke vkládání různých objektů. Vložení objektu se provede zaměřením kurzoru na zvolené místo editační mříže a stiskem levého tlačítka myši. Na výběr jsou spawnery pro hráče, to jest body v mapě, kde se hráči objevují na začátku hry a po své smrti, když se znovu oživí. Výběr přesného typu vybavení se provede nastavením příslušného záznamu v nabídce. Mezi vybavení patří základní puška, kulomet, ostřelovací puška a lékárnička. Dále je možné vložit dřevěnou bednu jako dekorativní předmět.

Obrázek č. 38 - Dialog vkládání vybavení.

Paths builder

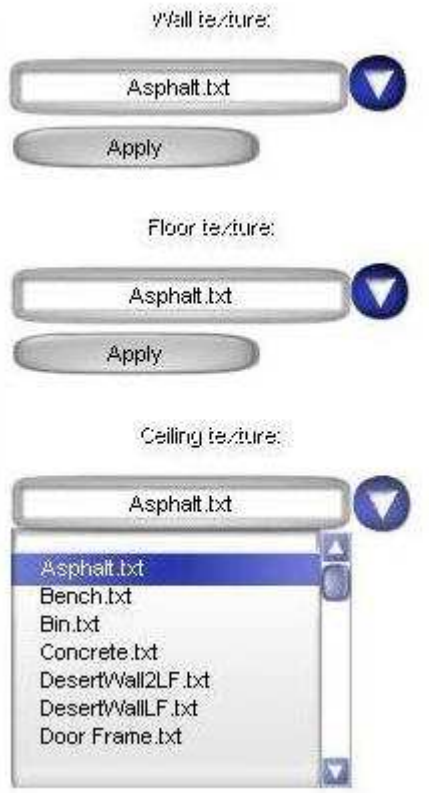
Pro úspěšnou realizaci navigace umělých agentů po mapě je nutné připravit určitou sadu cest a informací, které jsou nutné pro umělou inteligenci. Cesty jsou tvořeny uzly a spojovacími hranami. V uzlech se automaticky nastavuje dodatečná informace o umístění vybavení v místě uzlu. Tato informace je dostupná inteligenci autonomních agentů k jejich úspěšné navigaci k těmto instancím vybavení. Pomocí levého tlačítka myši a kurzorového kříže se do mříže zakreslí tyto navigační cesty vkládáním uzlů a jejich následným spojováním. Přepínačem lze nastavit režim pro tvorbu nových uzlů, či jejich spojování spojnicemi. Ty musí začínat a končit v již existujícím uzlu. Spojování uzlů probíhá tím způsobem, že se zapne režim spojování přepínačem „Connect nodes“, poté se kurzorovým křížem zaměří první spojovaný uzel a označí stiskem levého tlačítka myši. Pak se zaměří druhý spojovaný uzel a opětovným stiskem levého tlačítka myši se dokončí spojení. Informace o vybavení se vytváří v uzlech automaticky tak, že na místo uzlu vložíme spawner příslušné zbraně, či jiného vybavení. Logika editoru již sama tuto informaci uzlu nastaví. Jako obvykle je možné odebrat posledně vloženou spojnici tlačítkem „Undo Connection“ či uzel zaměřením uzlu kurzorovým křížem a stiskem pravého tlačítka myši. Spojnice by se kvůli přehlednosti a realističtějšímu chování agentů neměli křížit mezi sebou ani se stěnami. Je nutné mít v paměti, že po spojnicích se budou agenti opravdu pohybovat a umísťovat je tedy v dostatečné vzdálenosti od stěn a neprůchozích objektů tak, aby při hře nedocházelo k zadrhávání agentů o tyto překážky.



Obrázek č. 39 - Dialog nastavení informací o navigačním uzlu.

Texturer

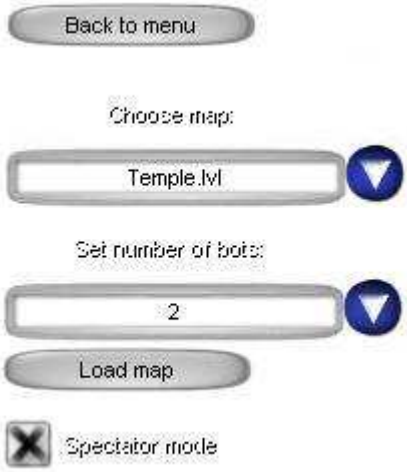
Poslední editační mód slouží ke změně textur stěn, podlahy a stropu mapy. Pokud je zapnut editační mód „Texturer“ je nejprve nutné nastavit cílené textury. To se provede nastavením požadované textury v rozbalovací nabídce a stiskem tlačítka „Apply“. Je možné nastavit texturu zdí, podlahy a stropu. Pokud nastavíme texturu stropu, automaticky se vypne zobrazení oblohy a místo ní se objeví strop. Takto je možné vytvářet mapy s interiéry. Je také možné nastavit vypnutí stropu (pokud byl dříve zapnut) nastavením poslední z možných textur stropu „NO CEILING“. Po potvrzení se automaticky aplikuje nová textura podlahy či stropu (pokud byla nastavena). Nově nastavené textura pro zeď se aplikuje zaměřením zdi kurzorem a následným stiskem levého tlačítka myši. Takto můžeme pohodlně pokračovat s aplikací i na další stěny mapy bez nutnosti nového nastavení textury. Před uložením mapy je nutné mít nastavenou alespoň texturu podlahy. Před vkládáním stěn je nutné mít nastavenou texturu stěn, jinak se objeví zpráva s upozorněním o nutnosti jejího nastavení. Pro vizuální změnu textur zdí je nutné mít již hotov alespoň základní půdorysový návrh, vytvořit jeho grafický model tlačítkem „Render“ a zaškrtnout jeho zobrazení. Poté lze již velmi jednoduše levým tlačítkem myši měnit texturu zaměřené zdi na nastavenou texturu zdí. Při změně textury podlahy či stropu je nutné přepočítat grafický model mapy před jeho zobrazením opětovným stiskem tlačítka „Render“.



Obrázek č. 40 - Nastavení textury zdi a podlahy.

B.4 Nastavení hry

Ovládací nabídka herního režimu programu ovlivňuje konkrétní instanci hry. Umožňuje zvolit mapu a počet agentů. Tlačítko „Back to menu“ vrátí uživatele zpět do hlavního menu. Dále následuje nabídka, kde je možné zvolit mapu a počet agentů. Pod ní je tlačítko „Load map“, které spustí hru podle nastavení. Posledním prvkem je zaškrtávátko „Spectator mode“. Pokud je zaškrtnuto, umožňuje pozorovat hru a chování agentů z ptačí perspektivy.



Obrázek č. 41 - Natavení hry.

Ovládání kamery během používání ptačí perspektivy je podobné, jako při klasickém pohybu hráče po mapě. Tedy pomocí klávesnice a myši. Na klávesnici se ovládá směr pohybu klasicky pomocí tlačítek W (vpřed), A (vlevo), S (vzad), D (vpravo). Myší se ovládá směr pohledu a pomocí přetáčecího kolečka přibližuje a oddaluje. Účelem toho režimu náhledu na scénu je pozorování chování agentů (umělé inteligence). V komerčním produktu by však přítomnost tohoto režimu působila jako značná výhoda pro hráče.

Obrázek č. 42 – Používání ptačí perspektivy při pozorování chování hráčů.

B.5 Hra

Dostáváme se k poslední a pro uživatele nejdůležitější části, ovládání samotné hry. Po nastavení hry (mapa, počet agentů) a nahrání mapy (může trvat několik sekund) se postava hráče objeví uvnitř mapy. Pro aktivaci a deaktivaci grafického uživatelského rozhraní stačí stisknout klávesu „Esc“. Pokud hráč umře, objeví se během několika sekund na místě některého ze spawnerů pro hráče, které byly nastaveny v editoru. Aplikace vždy kontroluje zde v určité vzdálenosti od spawneru není jiný hráč. V tom případě vybere pro oživení hráče jiné místo. Pokud není volné žádné místo, hráč čeká, dokud se některé neuvolní. „Ptačí perspektiva“ je standardní pohled na scénu po přepnutí aplikace do herního režimu. Před vlastním bojem je ji tedy nutné ji v menu vypnout přepínačem.

Obrázek č. 43 - Obrázek ze hry.

Klávesnice

Esc - zrušení hry a návrat do menu

W - pohyb dopředu

S - pohyb vzad

A - úkrok vlevo

D - úkrok vpravo

R – restartování pozice hráče

Myš

Pohyb myši - pohyb zaměřovacího kříže a natočení hráče.

Levé tlačítko - střelba

Kolečko - změna zbraně

B.6 Instalace a adresářová struktura programu Valahalla

Hra je určena pro operační systém Windows XP a novější. Dále je nutné mít nainstalovány DirectX August 2009 a novější. Před spuštěním vlastní hry je tedy nutné nainstalovat běhové prostředí a knihovny DirectX úspěšnou instalací pomocí souboru „directx_aug2009_redist.exe“ umístěném na DVD přiloženém k bakalářské práci. Instalační

soubor DirectX je také možné volně stáhnout na oficiálních stránkách společnosti Microsoft. Instalace samotné hry vyžaduje pouze několik kroků. Stačí překopírovat adresář „Valahalla“ se hrou Valahalla z DVD přiloženého k bakalářské práci kamkoliv na lokální disk počítače. Aplikace je od té chvíle připravena k použití. Pro spuštění a plynulý chod aplikace je potřeba, aby počítač, na kterém je hra spouštěna, splňoval alespoň minimální konfiguraci. Stejná minimální konfigurace je vyžadována i pro práci s herním projektem Visual Studia 2008.

Minimální konfigurace

- frekvence procesoru alespoň 1.2 GHz

- operační paměť alespoň 1 GB

- grafický akcelerátor podporující DirectX9

- místo na pevném disku 15MB

- operační systém Windows XP a novější

- nainstalovány DirectX August 2009 (soubor „directx_aug2009_redist.exe“ z multimediální přílohy bakalářské práce) a novější

C. Organizace bakalářské práce

C.1 Organizace souborů projektu

Praktická část je napsána ve Visual Studiu 2008 programovacím jazykem C++. Soubory jsou rozděleny do kontejnerů podle příslušnosti k určitým logickým celkům programové logiky, které spolu souvisejí. Každá třída nebo struktura má nad svou deklarací uveden krátký popis, některé metody nad svou definicí. Podobně atributy třídy. Projekt obsahuje také soubory „Changelog.txt“, který obsahuje důležité změny v projektu a soubor „TODO.txt“. Ten obsahuje seznam věcí, které by se měli udělat, vylepšit, opravit apod. (označené znakem -), nebo se již opravily (označené znakem +).

Adresáře v projektu a jejich obsah:

AI – umělá inteligenci

Common – běžné algoritmy a datové struktury bez závislosti na projektu

DXUT – vrstva DirectX používaná ukázkovými programy v dokumentaci

Engine – hlavní třída aplikace

GameStates – herní stavy

GrapDevManager – modul obsluhující programovou abstrakci grafické karty

Geometry – třídy definující geometrii ve hře

Graph – algoritmy a datové struktury spojené s reprezentací grafu

Graphics – třídy používající přímo knihovnu DirectX

InputDevManager – vstup z klávesnice a myši

WindowManager – modul obsluhy okna WinAPI

XML – do projektu integrované soubory projektu CMarkup

C.2 Obsah přiloženého DVD

K této práci je přiloženo DVD s veškerými zdrojovými soubory projektu, dokumentací, použitou grafikou, hudbou a další související data.

Adresář „Valahalla Project“ obsahuje praktický projekt Visual Studia 2008, adresář „Valahalla“ obsahuje samotnou hru, určenou pro používání koncovými uživatele (hráči). Dále DVD obsahuje elektronickou verzi této bakalářské práce a instalační soubory DirectX pro koncové uživatele („directx_aug2009_redist.exe“) a vývojáře („DXSDK_Aug09.exe“).