Univerzita Karlova v Praze
Matematicko-fyzikalní fakulta

BAKALÁŘSKÁ PRÁCE

Petr Fanta

Real-time strategy hra s umělou inteligencí

Katedra teoretické informatiky a **matematické logiky**

Vedoucí bakalářské práce: Mgr. Tomáš Baylo

Studijní program: Informatika

Studijní obor: Programování

Praha rok 2012

Chtěl bych poděkovat vedoucímu práce Mgr. Tomáši Balyovi za jeho cenné rady, připomínky, nápady a v neposlední řadě čas, který mi věnoval. Poděkování patří také panu Milanu Rybářovi za užitečné komentáře a čas stráveny při společném zkoumání .Net Frameworku a XNA Frameworku během psaní našich bakalářských prací.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze Petr Fanta

Název práce: Real-time strategy hra s umělou inteligencí
Autor: Petr Fanta
Katedra (ústav): Katedra teoretické informatiky a matematické logiky
Vedoucí bakalářské práce: Mgr. Tomáš Balyo

Abstrakt: Práce se zabývá vytvořením real-time strategické hry ve 3D prostředí s umělou inteligencí. Práce obsahuje teoretický rozbor bojových real-time strategických her, způsoby implementace umělé inteligence v takovýchto hrách a poznatky potřebné k implementaci 3D prostředí. Součástí práce je i implementace jednoduché real-time strategické hry s umělou inteligencí.

Klíčová slova: real-time strategická hra, 3D grafika, umělá inteligence

Title: Real-time strategy game with artificial intelligence
Author: Petr Fanta
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: Mgr. Tomáš Balyo

Abstract: The aim of this thesis is to create a real-time strategy game with artificial intelligence in 3D environment. The thesis includes theoretical analysis of combat real-time strategy games, methods for implementing artificial intelligence in this kind of games, and the basic knowledge of 3D environment implementation. A simple 3D real-time strategy game with artificial intelligence has been implemented as a part of this this thesis.

Keywords: real-time strategy game, 3D computer graphics, artificial intelligence

Obsah

Obsah **1**

Předmluva **3**
Real-time strategické hry . . . 3
Struktura práce . . . 4

1 Analýza problému **5**
1.1 Vlastnosti real-time strategických her . . . 5
1.1.1 Prvky hry . . . 5
1.1.2 Herní módy . . . 6
1.1.3 Omezený pohled na hrací plochu, odkrývání mapy a Fog of war . . . 6
1.1.4 Průběh hry . . . 7
1.2 Engine . . . 7
1.2.1 3D Grafika . . . 7
1.2.1.1 Post-processing . . . 8
1.2.1.2 Světla . . . 8
1.2.1.3 Stíny . . . 9
1.2.1.4 Deferred rendering . . . 9
1.2.1.5 Particle system . . . 11
1.2.2 Terén . . . 11
1.3 Umělá inteligence . . . 13
1.3.1 Metody pro implementaci umělé inteligence . . . 15
1.3.1.1 Skriptovavá umělá inteligence . . . 16
1.3.1.2 Systém pravidel . . . 16
1.3.1.3 Case-based systémy . . . 17

2 Implementace **18**
2.1 Základní přehled 18
2.2 Použité knihovny 18
2.3 Herní engine 19
2.3.1 Správa obrazovek 19
2.3.1.1 Menu a formuláře 20
2.3.2 InputManager 20
2.3.3 Rendering 20
2.4 Hra 21
2.4.1 Herní obsah 22
2.4.2 Herní mapa 23
2.4.3 Herní objekty 24
2.4.3.1 Jednotky 24
2.4.3.2 Budovy 25
2.4.4 Hráči 25
2.4.5 Uživatelský vstup 26

3 Užívatelská dokumentace **27**
3.1 Hardwarové požadavky a instalace 27
3.2 Menu, nastavení hry a vytoření nové hry 27
3.3 Herní prostředí a ovádání hry 28
3.4 Herní objekty 30
3.5 Cíl hry 31

Závěr **32**

Přílohy **33**

Literatura **34**

Předmluva

Počítačové hry jsou v dnešní době velmi výraznou složkou softwarového průmyslu. Podle průzkumů organizace *Entertainment Software Association(ESA)*[1] *[1]* 46% obyvatel Spojených států amerických si v roce 2012 koupilo či plánuje koupit nejméně jednu počítačovou hru[2]. V roce 2011 bylo 27,6% prodaných her určených na počítače strategických a byl to nejvíce zastoupený herní žánry. Ostatní herní žánry mají na počítačích výrazně menší zastoupení, z čehož můžeme usuzovat, že tento žánr bude mít i v nadcházejících letech na trhu velké zastoupení.

Význam strategických není pouze v oblasti zábavního průmyslu, ale podle [2] jsou vhodné i pro testování poznatků z oblasti umělé inteligence na akademické úrovni.

Strategické hry se dělí na dvě základní kategorie. První z nich je skupina tahových strategií. Tento druh se nejvíce podobá deskovým hrám, kde každý hráč může během svého tahu provést pouze omezený počet úkonů a poté hraje další hráč. Druhou kategorií jsou real-time strategické(RTS) hry. V takovýchto hrách se již hráči nestřídají po tazích, ale hrají simultánně po celou dobu hry. Cílem této práce je navrhnou a implementovat právě RTS hru, proto je tomuto žánru věnovaná další část textu.

Real-time strategické hry

Real-time strategické hry jsou podžánrem strategických her, které nejsou rozděleny na tahy, ale všichni hráči provádějí úkony současně. Cílem RTS her je typicky snaha hráče optimalizovat svůj stav pomocí strategického rozhodování a nakládání se zdroji nebo plnění zadaných úkolů. Zdroji jsou v těchto hrách jednak suroviny, ale také jednotky, která plní úkoly zadávané hráčem. Hráč typicky

[1] Organizace sdružující společnosti zabývající se vývojem počítačových her.

[2] Zde bereme jako počítačovou hru i hry určené pro konzole.

ve hře působí pouze jako komandér, tedy se nevyskytuje ve hře jako postava, ale pouze zadává jednotkám úkoly, které je poté plní samostatně.

V RTS hrách všechny úkony probíhají v reálném čase, tedy každá akce v herním světě trvá určitý čas a různé akce mohou probíhat současně. Časová rovina hry je shodná s tou, kterou vnímá hráč, tedy doba trvání různých akcí je přizpůsobena tomu, aby hra byla zábavná.

Tento podžánr lze dále dělit na mnoho kategorii, ale pro účely této práce se zaměříme pouze na hry válečné. Ve válečných hrách hráč ovládá skupiny jednotek, řídí jejich chování v bitvě. Cílem těchto her většinou bývá eliminace jednotek protivníka. Významnými představiteli této skupiny jsou hry ze serii *Warcraft[23, 24, 25]* a *Starcraft[22, 21]*, *Age of Empires[26, 27, 28]*či *Command & Conquer[29, 30, 31, 32, 33, 34, 35]*.

Struktura práce

Tato práce je rozdělena do 3 hlavních kapitol. První kapitola teoreticky rozebírá obecné požadavky na vývoj RTS hry, herního engine a umělé inteligence pro RTS hry. Druhá kapitola obsahuje popis konkrétní implementace RTS podle rozboru v první kapitole. V poslední kapitola se zaměřuje na popis implementované hry z pohledu jejího uživatele.

Kapitola 1

Analýza problému

1.1 Vlastnosti real-time strategických her

V této sekci budu stručně vysvětlené prvky, strategie a módy typické pro RTS hry.

1.1.1 Prvky hry

Válečné real-time strategické hry se většinou skládají z mnoha různých prvků, i když ve většině her je mnoho z nich velmi podobných. Jeden z hlavních prvků hry je herní mapa. Hra vždy probíhá na omezené hrací ploše, která může skládat z mnoha druhů povrchu. Každý povrch může mí různé vlastnosti, například na některém může jít stavět, na jiném nemohou chodit jednotky atd. Mapa také může obsahovat ostrovy, mezi kterými se mohou pohybovat jen určité druhy jednotek. Díky tomu mohou vznikat speciální strategie, ke kterým je potřeba využít různé druhy jednotek.

Dalším výrazným prvkem strategických her jsou již zmíněné jednotky. Jednotky jsou to, co utváří tento žánr, ale z hlediska hry nejsou důležitější než jiné prvky hry. Ve většině her existuje mnoho druhů jednotek, ze kterých hráč sestavuje svojí armádu. Jednotlivé druhy bývají různě silní proti různým druhům. Například tank bude silný na ničení budov, ale slabý proti vojákům. Typicky hry obsahují tři základní druhy jednotek: dělník, jednotka útočící na blízko, jednotka útočící na dálku.

Aby bylo možné vytvářet nové jednotky jsou dalším prvkem RTS her budovy. Opět v každé RTS hře existuje mnoho druhů budov, které mají různé významy. Většinou RTS hry obsahují budovy na stavbu jednotek, ale mohou obsahovat

také budovy, které jednotky léčí, když jsou v jich blízkosti, či umožňují vyvíjet nové technologie atd.

Posledním prvkem RTS her jsou suroviny. Každá hra by měla obsahovat alespoň dva druhy surovin, aby mohli vznikat různé strategie. Na surovinách závisí celé hra, protože bez nich hráč nemůže stavět budovy, jednotky či objevovat nové technologie. Suroviny bývají rozmístěné v omezeném množství na různých místech mapy. K jejích těžení se většinou používají dělníci, ale existují i hry, které používají k těžbě specializované budovy. Dělníci je poté dopravují do různých skladů surovin, či je mění na jiné. V některých hrách také existují obnovitelné zdroje surovin, například pole.

1.1.2 Herní módy

RTS hry často obsahují tři herní módy: kampaň, hru pro jednoho hráče na libovolné mapě a hru pro více hráčů.

Kampaň je určena pro jednoho hráče a skládá z více herních map, které bývají rozdělené do různých úrovní. Na každé herní mapě hráč musí plnit předpřipravené úkoly, aby danou úroveň splnil a postoupil do další úrovně.

Ve hře pro jednoho hráče hráč hraje proti počítačem kontrolovanému protivníkovi. Oba hráči začínají za stejných podmínek a nemusí plnit žádné speciální úkoly. Hra většinou probíhá ve více hráčích.

Hra pro více hráčů je v podstatě stejná jako hra pro jednoho hráče s jediným rozdílem, že protivníci jsou kontrolování jinými lidskými hráči.

1.1.3 Omezený pohled na hrací plochu, odkrývání mapy a Fog of war

Součástí herního mechanizmu je i pouze částečný přehled o dění na hrací ploše k tomu přispívají tři vlastnosti - omezený pohled na hrací plochu, postupné odkrývání mapy a fog of war.

Protože hrací plocha většiny RTS her bývá moc velká, než aby ji mohl hráč vidět celu najedou, mívá pouze omezené informace o dění v herním světě. Většina her proto obsahuje malou informační mapku s částečnou informací o dění na hrací ploše, která zobrazuje přibližnou polohu objektů ve hře, ale nelze podle ní rozlišit o jaké objekty jde. Hráč je tím nucen sledovat dění v různých částech herního světa.

Další typickou vlastností RTS her je postupné odkývání herní mapy. Hráč na začátku hry zná pouze bezprostřední okolí své základny, tedy nemá žádné

informace o pozici svých protivníků, stavu jejich vývoje nebo vzhledu terénu. Hráč terén během hry pomocí jednotek objevuje a tím získává taktickou výhodu, jelikož může využívat tvaru terénu ke své obraně.

Fog of war je jakési rozšíření konceptu postupného odkrývání. Hráč sice vidí terén, které jeho jednotky objeví, ale pokud na daném území nemá své jednotky, nedostává aktuální informace o dění na daném místě.

1.1.4 Průběh hry

Ve hře většinou hráči usilují o odstranění protivníku z herní mapy, nebo o získání převahy jiným způsobem. K tomuto účelu používají armády sestavené z různých jednotek a různé taktiky. Strategií, jak hrát různé hry, je nekonečné množství, ale bývají založené na podobných principech. Nejčastěji se používají tři přístupy: rychlý útok na začátku hry(rush), vytvoření technologické převahy, která umožňuje stavět silnější jednotky, nebo vytvořit silnou ekonomiku a postavit armádu s početní převahou nad protivníky.

K vytvoření dobré strategie je také nutné uvažovat o real-time hledisku hry. Vše se odehrává v reálném čase, tedy je nutné, aby hráč mohl kdykoliv změnit svojí taktiku vzhledem ke stavu hry. Z toho vzniká požadavek na hráče, aby měl neustále přehled o dění ve hře a dokázal na něj dostatečně rychle reagovat.

Aby hráč ve hře mohl ve hře uspět musí znát dostatečně herní plán, stav rozvoje svého protivníka a v neposlední řadě své budovy, jednotky a dostupné technologie. Každá z těchto částí je v průběhu hry rozhodující pro všechny hráče. Pokud hráč nebude schopný využívat výhod terénu, nebude se moci efektivně bránit ani útočit. Také když hráč nedokáže postavit základnu podle požadavků hry, může jeho rozvoj být pomalejší něž ostatních hráčů.

1.2 Engine

1.2.1 3D Grafika

V dnešní době je asi nejvýraznější částí všech počítačových her zobrazovací složka. Díky velkému rozvoji v oblasti grafických karet, nevznikají v podstatě žádné nové hry, které by nebyly ve 3D. I přesto je fotorealistické zobrazování 3D grafiky[1] velmi náročné, než aby se dalo využít ve hrách, či obecněji v real-timove grafice. Ve hrách často požadujeme, aby hra měla alespoň 30 FPS, ale

[1] Různé vysoce specializované renderery vykreslují scénu v řádu minut.

běžné je 60 FPS, proto se používají různé triky, jak vykreslování urychlit na úkor realističnosti.

1.2.1.1 Post-processing

Post-processing je technika používaná k vytvoření iluze různých jevů známých z ray-tracingu či reálného světa. Místo přímého vykreslování na obrazovku je výstup přesměrován do bufferu v paměti grafické karty. Pomocí pixel shaderu jsou poté na buffer aplikovány různé operátory. Výhodou post-processingu je, že filtry jsou aplikovány pouze na pixely, které jsou poté skutečně vidět, a že je možné používat komplexní představu o dané scéně, zatímco jsou jednotlivé objekty vykreslovány samostatně.

Pro lepší představu uvádíme pár nejpoužívanějších využití post-processingu: Bloom efekt[2], SSAO[3], hloubka ostrosti, světelné paprsky, zrnění obrazu, pohybová neostrost a jiné.

1.2.1.2 Světla

Osvětlení scény se dělí do dvou skupin: globální a lokální osvětlení. Globální osvětlení zahrnuje zastínění okolím a stíny, tedy je třeba mít informace o celé scéně (viz dále).

Lokální osvětlení zahrnuje pouze stínování jednoho objektu vzhledem ke světlému zdroji, aniž bere ohled na to, jestli mezi ním a světelným zdrojem je jiný objekt. Pro výpočet lokálního osvětlení se ve většině případů používá Phongův světelný model[17], který sice není fyzikálně přesný, ale ve většině případů dostatečně věrný. Phongův světelný model uvažuje dva typy odrazu - difuzní a spekulární. Díky tomu se dají vytvářet materiály s různými vlastnostmi.

Aby scéna vypadala co nejreálněji, většinou si nevystačíme pouze s globálním osvětlením. Běžně se ve hrách setkáváme s různými světelnými zdroji - rozptýlené světlo, směrové světlo, reflektor, bodové světlo.

Problémem je způsob, jak vykreslovat scénu s více světelnými zdroji. U malého počtu světel stačí specializovaný shader, toto řešení není moc univerzální, ale asi nejrychlejší. Další možnost je vykreslovat celou scénu pro každé světlo

[2]Bloom effekt simuluje záři kolem velmi jasných objektů, nebo rozostřuje jemné detaily.

[3]*Screen space Ambient Occlusion* (SSAO) je technika jak efektivně aproximovat zastínění okolím v reálném čase. Zastínění okolním pomáhá modelům pro výpočet lokálního osvětlení dodat realistický dojem díky tomu, že započítává tlumení světla zastíněním. SSAO bylo poprvé požito v enginu od firmy *Crytek* ve hře *Crysis*.

a výsledek kombinovat. Toto řešení je velmi dobré co se tyče možností a výsledku, ale je velmi pomalé, protože počet snímku za vteřinu se dělí počtem světel ve scéně. Poslední možností je deferred rendering(viz dále).

1.2.1.3 Stíny

Dynamické stíny se vytvářejí hlavně dvěma způsoby, i když existuje i třetí ne moc dobrý způsob.

Shadow volume, v některých zdrojích označované jako *stencil shadowing*, je technika, která využívá geometrická tělesa k určení oblasti neosvětlené od světelného zdroje. Objem těchto těles rozděluje virtuální svět na dvě oblasti - oblast ve stínu a mimo něj. K popisu oblastí se používá, jak již alternativní název napovídá, stencil buffer.
Tato technika je vhodná pokud chceme vytvořit velmi ostré stíny. Pokud chceme měkké stíny, není tato technika moc vhodná a může být velmi neefektivní. Pravděpodobně poslední úspěšná hra, která používá tuto techniku, byl *Doom 3*[36].

Shadow mapping se skládá ze dvou fází. V první je vykreslena scéna z pohledu světelného zdroje do bufferu v paměti grafické karty(do bufferu se ukládá vzdálenost bodu od světelného zdroje). V druhé fázi je vykreslena scéna z pohledu kamery. Při vykreslování se porovnává vzdálenost kresleného bodu od světelného zdroje a tím se určí, zda je osvětlený.
Tato metoda je snadná na implementaci, levná s ohledem na výkon a při dostatečném rozlišení stínové mapy, dává i velmi dobré výsledky. Existují i další možnosti jak vylepšit výsledek této metody, například rozostření či jiná modifikace stínové mapy. Tato metoda je v současných hrách asi nejpoužívanější.

„Shadow matrix“ využívá speciální transformační matici ke zploštění modelu, tak aby vypadal jako jeho stín. Tato metoda dává rozumné výsledky pouze na rovných(hladkých) plochách, jinak dává velmi spatné výsledky. V dnešní době se pravděpodobně nepoužívá.

1.2.1.4 Deferred rendering

Deferred rendering[16], v literatuře občas také deferred lighting či deferred shading, je odlišný pohled na vykreslování scény vhodný pro scény s mnoha světly.

Hlavní přínos deferred renderingu je oddělení vykreslování scény a výpočtu světel, která jsou na scénu aplikovaná jako post-processing. Celý proces je rozdělen do tří kroků: vykreslení scény, výpočet osvětlení, zkombinování scény a osvětlení.

V prvním kroku je scéna vykreslena do tří bufferu(textutur či render targetu) v paměti grafické karty současně.

- Textura obsahující barvy pixelů ve výsledném obraze.
- Textura obsahující normály zakódované do barvy
- Textura obsahující vzdálenost každého pixelu od kamery.

Tento buffer se většinou označuje jako *GBuffer*[4]. Celé vykreslní scény probáhá pouze jednou během celého renderingu. Důležité je si uvědomit, že ze vzdálenosti od kamery a pozice v textuře jsme schopni kdykoliv vypočítat původní pozici pixelu ve scéně. Pokud máme původní pozici pixelu a normálu, která k němu patří, jsme schopni spočítat i osvětlení daného pixelu.

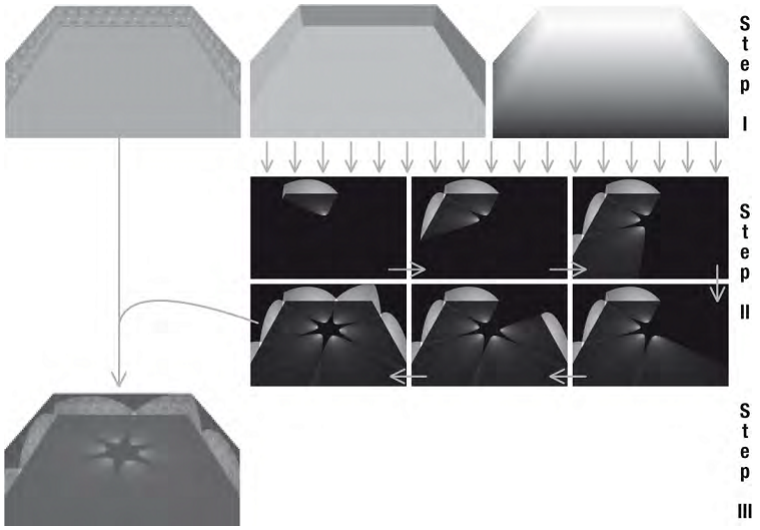
V druhém kroku je vytvořen nový render target nazývaný shading map, který bude obsahovat osvětlení scény. Jak již bylo zmíněno, díky GBufferu jsme schopni spočítat osvětlení scény pro jakýkoliv druh světla. Tento výpočet již neprobíhá pro celou scénu, ale pouze pro pixely, které jsou poté skutečně vidět a není k tomu zapotřebí vykreslovat pokaždé celou scénu.

Pro ostatní světla stačí pouze přidat jejich výsledky do shading mapy. Zde je hlavní úspora času, protože počet světel již nedělí počet snímku za vteřinu. V poslední fázi proběhne pouze zkombinování textury s barvami a shading map.

Kromě toho, že deferred rendering je velmi výkonný při velkém počtu světel ve scéně, také se dobře kombinuje s post-processingem, například se SSAO. Aby vykreslování světel bylo ještě výkonnější, je možné vykreslování světel ještě více optimalizovat tak, že se stínování počítá pouze pro pixely, na které dané světlo skutečně působí. Další výhodou je snadné kombinování se shadow mappingem. Kromě jiného je také možné do GBufferu ukládat informace o materiálu použitém pro daný pixel, například jde ukládat sílu speculární složky pro Phongův osvětlovací model.

I když má deferred rendering mnoho výhod zejména v rychlosti renderingu, má i nevýhody. Například vykreslování poloprůhledných objektů je paměťově náročné a může být pomalejší než přímé vykreslování. Také je náročnější pro programátora, protože pro implementaci deferred renderingu je potřeba mnoho velmi specializovaných shaderu. Poslední a asi největší nevýhodou je závislost

[4]Geometry buffer



Obrázek 1.1: Kroky deferred rendreingu

na grafické kartě, protože starší grafické karty nemusí podporovat vykreslování do více render targetů.

1.2.1.5 Particle system

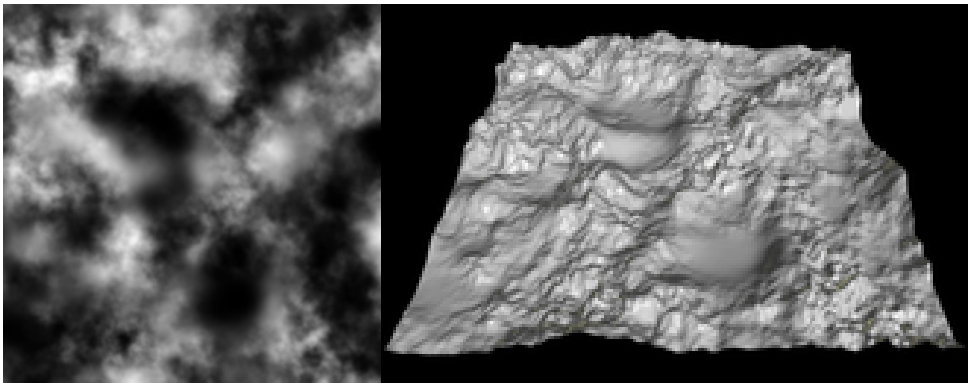
Particle system způsob jak vykreslovat složité efekty, kterých je těžké dosáhnout běžnými renderovacími technikami. Základem particle systémů, jak již název napovídá, je vykreslování velkého množství velmi malých 2D obrázků, či jiných objektů.

Typická implementace particle systémů uchovává u každé částice její pozici v 3D prostoru, směr pohybu a čas, po kterém částice zanikne. Tyto údaje přiřazuje částicím emitor, který se chová jako zdroj částic, tady určuje pozici systému ve scéně. Výhodné je využívat pro implementaci možností grafických karet[3], což umožňuje vykreslovat velké množství částic v reálném čase.

Výsledný efekt určuje emitor způsobem generování parametrů pro částice. Mezi příklady jevů simulovaných pomocí částicových systémů patří oheň, kouř, mlha, sníh, kouzla atd.

1.2.2 Terén

Základ každé hry je nějaká hrací plocha. Pro šachy to bude šachovnice, pro různě FPS hry například interiér nějaké budovy. Ve hrách odehrávajících se na volném



Obrázek 1.2: Výšková mapa a výsledný terén

prostranství je touto hrací plochou nějaký terén.

Terén může být buď velký 3D model vytvořený v nějakém programu na pro práci s 3D grafikou[5], ale častěji bývá generován procedurálně. Pro procedurální generovaní terénu se požívají hlavně dva přístupy: terén generovaný pomocí výškové mapy a terén generovaný pomocí voxelů[3][6]. Každý z přístupů má své výhody a nevýhody. Výhodou voxelových terénů je možnost vytvářet velmi složité terény, které mohou obsahovat jeskyně či skalní útvary s převislými částmi, proto jsou vhodné pro first-person hry. Pro RTS hry, které většinou používají pohled shora, se více hodí použití výškové mapy, která vytváří hladší povrch, tedy jednodušší a přehlednější.

Výškovou mapu si lze asi nejlépe představit jako dvourozměrné pole hodnot, které bodům z 2D prostoru přiřazuje nadmořskou výšku. Výhodou výškové mapy je snadné uložení do obrázku, tedy je zle jednoduše vytvářet v jakémkoliv editoru pro 2D grafiku. Výškové mapy lze také generovat algoritmicky, například pomocí Perlinova šumu či diamond-square algoritmu[7].

Protože dnešní hry používají velké terény, model vytvořený podle výškové mapy musí mít mnoho grafických primitiv (polygonů), aby vypadal realisticky, proto je vhodné použít nějaké optimalizace. Asi nejpoužívanější a nejintuitivnější vylepšení je rozdělení terénu na menší části a vykreslování pouze částí, které jsou viditelné. Jelikož lze terén považovat za dvourozměrný objekt, nečastěji se používá dělení na čtverce či obdélníky. Tyto části je výhodné ukládat do quadtree[18], což umožňuje vyhledání zobrazovaných částí v logaritmickém čase.

[5]Například 3D Studio Max, Maya, Blender atd.

[6]Případně kombinací těchto přístupů. Například použité v CryEngine

[7]http://gameprogrammer.com/fractal.html

Další možností, jak urychlit vykreslování, je snížení počtu primitiv u částí, které jsou hodně vzdálené od kamery a zobrazují se jako pár pixelů, čímž nedochází k viditelnému snížení kvality výsledného obrazu. Tento přístup je označován jako level of detail(LOD). LOD lze dobře kombinovat s quadtree tak, že čím je vykreslovaný čtverec dál od kamery, tím řidší má sít vrcholů. Nevýhodou je, že na hranicích čtverců mohou vznikat „trhliny“ v terénu, proto je třeba tento speciální případ ošetřit.

1.3 Umělá inteligence

I když pro hráče je často rozhodující grafický vzhled hry, málo kdy si uvědomují, že právě umělá inteligence je hlavní část hry, která zajišťuje zábavnost a hratelnost.

RTS hry na rozdíl jiných herních žánrů obsahují mnoho oblastí, které ke svému řešení vyžadují prvky umělé inteligence. Hlavní okruhy problémů, které musí umělá inteligence v RTS hrách řešit, jsou tyto:

Chování jednotlivých jednotek je jedním ze úkolů pro umělou inteligenci. Do této kategorie by se dalo zařadit například hledání cest na hrací ploše, ale také jak se jednotky mají chovat v boji. Do chování v boji se řadí bránění, pokud je na jednotku utočeno, pronásledování jednotky, na kterou jednotka útočí atd.
Otázkou je, jak moc mají být jednotky inteligentní, jak moc je může ovlivňovat hráč či jak moc mají umět blokovat a obkličovat protivníkovy jednotky.

Chování pracovních jednotek(dělníků) se od ostatních jednotek velmi liší. Pracovní jednotky jsou většinou slabé v boji, proto je pro ně lepší z boje utíkat, nebo se jinak efektivně bránit.
Jelikož hlavním účelem pracovních jednotek je těžba a stavění budov, musí tyto jednotky umět efektivně hledat zdroje a dopravovat je do skladu surovin. Také musí umět stavět budovy a případně si i rozdělovat práci.

Strategie malých skupin jednotek. Tato část umělé inteligence ovládá skupiny o 5-30 jednotkách, pokud jednotky společně útočí. Měla by zajišťovat maximální využití síly jednotek v dané situaci. Například určuje jestli je dobré na pozici nepřítele nejprve útočit z dálky a až poté z blízka, aby nedocházelo k zbytečným ztrátám.

Zatímco předchozí části umělé inteligence využívali i lidští hráči, tuto část najdeme již pouze u počítačového hráče.

Celková strategie je nejvyšší úroveň umělé inteligence, jejímž účelem je rozvrhování celkové taktiky. Tato úroveň rozhoduje jestli by měl počítačový hráč zaútočit, vybudovat silnou obranu atd. Také určuje průběh útoku a místo útoku. Jinak řečeno tato část umělé inteligence určuje chování počítačem řízeného hráče, jeho herní styl. Změnou této vrstvy je možné vytvořit různé varianty protivníků.

Analýza terénu úzce souvisí téměř se všemi úlohami v RTS hrách. Protože každá herní mapa bývá jiná, musí umělá inteligence umět terén nějak analyzovat a podle toho upravovat svou strategii. Například pro mapu složenou z mnoha ostrovů bude strategie jiná než pro mapu bez překážek. Také množství a dostupnost surovin na herní mapě hraje významnou roli ve volbě taktiky.

Dalším úkolem analýzy terénu je využití jeho tvaru během boje. Pokud hráč zná dobře terén, může vyhrát bitvu díky využití výhodných míst na mapě. Například pokud postaví jednotky, tak aby blokovali úzký průchod, může zpomalit útok na svou základnu.

Analýza terénu by měla také sloužit jako základ pro stavbu základny. Při stavbě základny je nutné, aby budovy neblokovali jednotky a aby byl co nejsnazší sběr surovin. Zároveň se stavbou je nutné plánovat i obranu, k tomu lze využít jak přirozené překážky na mapě, ale také zdi, věže, rozhledny atd.

Aktivní prozkoumávání hrací plochy. Jak bylo zmíněno v 1.1.3, hráč v počátku hry nevidí celou herní mapu, ale jen části, ve kterých má své jednotky. Proto je nutné prohledávat herní mapu, aby hráč zjistil pozici protivníka, zdrojů a v neposlední řadě tvar terénu.

Průzkum neznámé oblasti také může přispět k odhalení vlastní pozice, vyprovokování protivníka nebo ztrátu jednotek, proto průzkum různých částí herní mapy nemá vždy stejnou hodnotu a je důležité zkoumaná místa dobře vybírat.

Sledování protivníka je také velmi důležitou úlohou pro umělou inteligenci. Pokud počítačový hráč zná jednotky, které protivník staví, místa, kde protivník shromažďuje jednoty atd., může přizpůsobit svou strategii například jinak vedeným útokem, či obranou.

Stavba základen. Jak již bylo zmíněno výše, při stavbě základny je nutné vycházet z povahy terénu, jednak kvůli obraně základny, jednak kvůli rychlému rozvoji.
Základna by také měla vyhovovala zvolené strategii. To kromě rozmístění budov zahrnuje i správné pořadí výstavby, aby by hráč mohl vytvářet potřebné jednotky a technologie.

Správa zdrojů je základní hledisko RTS her. Pokud hráč správně nezachází se zdroji, nemůže reagovat na změny ve vývoji hry a je brzy poražen. Cílem dobré správy zdrojů je vybudování silné ekonomiky a armády. Hráč musí být schopen se rozhodnout kdy a co stavět. Také musí umět řídit těžbu surovin, to znamená určit počet dělníku, kteří těží jednotlivé suroviny. Vhodné je pokud hráč umí regulovat potřeby na těžbu a stavbu podle herní situace, jelikož stavění podle předem určeného plánu nemusí být optimální.

Podvádění počítačem řízených hráčů je často používané v různých hrách. Možností jak umělá inteligence může podvádět je mnoho. Například může znát celou mapu od začátku hry. Další formou podvádění je předem naplánovaná stavba základny, podle tvaru terénu. V některých hrách dokonce počítačem řízený hráč rychleji těží suroviny. Problém je pokud tím hra přestává být vyvážená.

1.3.1 Metody pro implementaci umělé inteligence

Jak bylo nastíněno v předchozí části textu, oblast problémů řešených pomocí umělé inteligence je velmi rozsáhlá. Metod řešení je mnoho, ať v komerčním prostředí, nebo v prostředí akademickém. Jelikož umělá inteligence ve firemním prostředí není moc veřejně zpracovaná, můžeme většinou pouze odhadovat jaké techniky používají. I přesto jsou některé způsoby implementace v komerčním uveřejňovány ve sbornících jako Game AI Wisdom[4, 5, 6] a Game Programming Gems[7, 8, 9, 10, 11, 12].

I v akademickém prostředí na toto téma vzniká mnoho prací, jako příklad uveďme [2, 13], které se zaměřují zejména na aplikaci poznatků z klasické umělé inteligence na RTS hry, a to zejména na nedeterministické řešení řízení RTS her.

Jako úvod do problému umělé inteligence v počítačových hrách poslouží [14]. Autoři v knize uvádí přehled elementárních technik používaných při řešení úloh umělé inteligence. V následujícím textu boudou popsány základní metody řízení RTS her, která se požívají v komerčním prostředí, pro složitější metody doporučujeme již zmíněné zdroje.

1.3.1.1 Skriptovavá umělá inteligence

Skriptovanou umělou inteligencí budeme brát nějaký popis chovaní počítačového hráče pro určitou mapu. Z toho důvodu se hodí hlavně pro herní mód kampaň. Autoři skriptu předem naprogramují, jak se bude počítačový hráč chovat. Jelikož autor skriptu přesně zná herní mapu, může předem odhadnout pro ni nejlepší herní strategii. Přesněji řečeno, může předem určit místa, kde se budou stavět budovy, jaké se mají stavět jednotky, ale také reakce na nějaké události. Díky tomu může umělá inteligence působit velmi dobře, ale má i špatné stránky.

Hlavní nedostatkem je, že strategii připravuje autor hry. Takto předpřipravené chování postihuje pouze případy, které autor předpokládal v době návrhu. Pokud hráč zvolí taktiku, která nebyla předvídána, nemusí být chování hráče konzistentní, a nebude splňovat očekávání hráče.

Z hlediska programátora je toto řešení nevýhodné, protože musí pro každou herní mapu vytvářet nový skript, jelikož řada událostí je spojena s konkrétní oblastí na mapě. Z toho vyplívá, že vývoj hry s tímto typem řízení počítačového hráče je velmi zdlouhavý. Délka vývoje také souvisí s tím, že skripty musí postihovat velké množství herních situací a variant, aby co nejméně docházelo k neočekávaným řešením.

Pro hráče je sice na první pohled protihráč velmi inteligentní, ale pokud hrají stejnou mapu vícekrát za sebou, brzy je omrzí stále stejná taktika soupeře. Tento problém jde částečně vyřešit tím, že pro každou mapu bude připraveno více skriptů, mezi kterými se podle nějakého kritéria bude vybírat.

1.3.1.2 Systém pravidel

Umělá inteligence realizovaná pomocí systému pravidel je založena na podobném principu jako skriptovaná umělá inteligence. Systém obsahuje pravidla, podle kterých se umělá inteligence chová v určitých situacích. Rozdíl oproti skriptovaným umělým inteligencím je v tom, že pravidla nejsou určena pro jednu konkrétní mapu, tedy musí být co nejuniverzálnější. Typicky mají pravidla co nejmenší oblast působnosti, například výstavbu jedné budovy, či akci jednotky.

Pravidla mají většinou podobu if-then konstrukce: if *podmínka* then *akce*. *Podmínka* specifikuje situaci, ve které má být pravidlo použito, a může určovat objekty, na které má být použito. *Akce* pak může být zadání povelu jednotce, či jiný kód, který se při splnění podmínky spustí.

Pokud hra má herní situace dostatečně pokryté pravidly, může počítačový hráč hrát poměrně dobře. Stejně jako skriptované umělé inteligence mají i sys-

tém základní nedostatek: neobsahují učení. Díky tomu nemůže počítačový hráč přizpůsobovat své chování protivníkům a proto zkušenému soupeři přestane brzo stačit.

Podobně jako u skriptované umělé inteligence je i pro systém pravidel vhodné použit více alternativních sad pravidel. Pravidla mohou být určena pro různé druhy map(například velké pevniny, malé ostrovy atd.), nebo se mohou střídat během hry podle situace, ve které se hráč nachází. Také může jedna sada pravidel obsahovat více akcí pro stejnou situaci, ze kterých se pak nějakým způsobem vybere jedna možnost. Všechna tato vylepšení výrazně zvýší zajímavost hry.

1.3.1.3 Case-based systémy

Základní myšlenkou case-based systémů je řešení nových problémů podle podobných problémů řešených v minulosti. Základem case-based systému je databáze řešených problému. Pokud se systém setká s novým ještě neřešeným problémem, vyhledá v databázi co nejpodobnější již vyřešený problém a pokusí se ho upravit, aby vyhovoval nové situaci.

Case-based systémy jsou spíše způsobem či myšlenkou, jak řešit problémy řízení. V RTS hrách se dají použít k výběru skriptu, pravidla či elementární úlohy pro jednotku. Na rozdíl od předchozích přístupů case-based systémy umožňují nedeterministické rozhodovaní, tedy se počítačový hráč nemusí ve stejné situaci vždy zachovat úplně stejně.

Kapitola 2

Implementace

2.1 Základní přehled

Hlavní programovací jazyk použitý v tomto projektu je C# 4.0 s .NET frameworkem a XNA Frameworkem 4.0. K procházení či spuštění zdrojového kódu doporučeme použít Microsoft Visual Studio 2010. Ke spuštění projektu ve Visual Studiu potřeba nejprve nainstalovat XNA Game Studio 4.0[1], které obsahuje i XNA Framework. Ke spuštění zkomplikované hry je poté potřeba mít nainstalovány pouze XNA Framework a .NET Framerwork.

Program je rozdělen do několika projektů. Toto rozdělení je hlavně kvůli požadavkům architektury XNA, proto se v dalším textu spíše budeme řídit logickým rozdělením na herní engine a samotnou hru.

2.2 Použité knihovny

XNA Framework. *XNA framework* je součástí balíku *Microsoft XNA* umožňující snadný vývoj her v jazyce C#. Obsahuje rozsáhlou sadu knihoven typických pro vývoj her. Běhové prostředí pro *XNA Framework* je dostupné pro PC(převážně s operačním systémem *Windows*), *Windows Phone 7* a herní konzoli *Xbox*.

Framework je postavený nad knihovnami *Direct X* a snaží se odstínit platformové rozdíly. Framowork rozhodně nejde považovat za herní engine, ale obsahuje prvky, které jeho vývoj usnadňují. Ve frameworku je k dispozici herní smyčka, základní možnosti práce s grafikou, zvukem, sítí a vstupem.

[1] http://www.microsoft.com/en-us/download/details.aspx?id=23714

Kromě jiného je součástí technologie *Microsoft XNA* koncept *Content Pipeline*, který usnadňuje načítání dat z různých formátů do hry.[2]

2.3 Herní engine

Herní engine je část programu, která rozšiřuje knihovny XNA Frameworku o třídy, které usnadňují vývoj samotné hry. Tato rozšíření jsou umístěna ve jmenných prostorech *RTS.Screens*, *RTS.Forms*, *RTS.Rendering* a *RTS.Input*. Implementace hry se nachází ve jmenných prostorech *RTS* a *RTS.Game*.

2.3.1 Správa obrazovek

Jelikož většina dnešních her obsahuje mnoho obrazovek, mezi kterými hráč během hry přechází, je součástí programu i několik tříd pro jejich správu. Tyto třídy jsou součástí jmenného prostoru *RTS.Screens*.

Hlavní třídou správy obrazovek je třída ScreenManager, která dědí od *DrawableGameComponent*. Tato třída spravuje načtené obrazovky v zásobníku, kde nejvrchnější obrazovka je aktivní, což znamená, že se vykresluje a může zpracovávat vstup. Při návratu z obrazovky(odstranění ze ScreenManageru) se aktivní obrazovkou stává ta pod ní v zásobníku.

ScreenManager zprostředkovává instance objektů společné pro všechny obrazovky v něm uložené: *ContentManager* pro načítání obsahu pomocí Content Pipeline, *SpriteBatch* na rendering textur a fontu, InputManager pro zpracování vstupu, jednopixelovou texturu kvůli malování jednobarevných ploch a základní fonty. Kromě toho ScreenManager volá aktualizace a vykreslování na viditelných obrazovkách.

Abstraktní třída Screen je bázovou třídou pro jednotlivé obrazovky v programu, které jsou spravovány ve ScreenManageru. V programu se vyskytují dva typy obrazovek podle toho, jakým způsobem se chovají při vykreslování vůči obrazovkám pod nimi v zásobníku. Pokud je obrazovka označena jako „Pop Up“ je vykreslována přes obrazovku pod ní v zásobníku, jinak jsou obrazovky pod ní skryté, nevykreslují se.

Protože koncept screen manažeru umožňuje ukrývání a obnovování obrazovek, může obrazovka se vyskytovat v několika stavech: aktivní, skrytá a přechody mezi těmito stavy. Během stavu přechodu může probíhat nějaké animace.

[2] http://create.msdn.com

Speciálním potomkem třídy Screen je třída LoadingScreen, která slouží k naplnění ScreenMangeru novou sadou obrazovek.

2.3.1.1 Menu a formuláře

Jmenný prostor *RTS.Screens* kromě výš zmíněných tříd obsahuje také bázovou třídu pro obrazovky obsahující menu. Třída *MenuScreen* vykresluje menu, spravuje jednotlivé položky menu reprezentované třídou MenuEntry a na základě uživatelského vstupu vyvolává příslušné události.

Herní engine také obsahuje třídy pro vytváření formulářů(podobné například knihovně *Windows Forms*). Tyto třídy jsou součástí jmenného prostoru *RTS.Forms.*

Bázovou třídou pro obrazovky obsahující formuláře je třída Form. Umožňuje přidávání jednotlivých ovládacích prvků, na kterých jsou vyvolávány události podle uživatelského vstupu podobně jako v jiných knihovnách pro práci s GUI.

Jednotlivé ovládací prvky jsou potomky třídy Control, která určuje obecné vlastnosti ovládacích prvku a obsahuje metody obsluhující jednotlivé události. Herním engine jsou dostupné tyto ovládací prvky: *Button*, *ColorPallete*, *ComboBox*, *CheckBox*, *ListBox*, *TextBox* a *TrackBar*.

2.3.2 InputManager

InputManager je třída z jmenného prostoru RTS.Input zapouzdřující zpracování vstupu z klávesnice a myši. Poskytuje různé metody jak pracovat se vstupem, která XNA Framework sám o sobě nemá. Například oproti samotnému XNA poskytuje metodu určující kliknutí myší atd.

2.3.3 Rendering

Ve jmenném prostoru *RTS.Rendering* jsou umístěné třídy pro práci s 3D grafikou, které implementují postupy popsané v 1.2.1. Veškeré vykreslování 3D grafiky je uzpůsobeno konceptu deferred renderingu(viz 1.2.1.4). Tento přístup nelze v XNA implementovat stejně efektivně jako v *Direct3D*, nebo OpenGL, protože XNA nepodporuje některé možnosti práce s grafickým adaptérem. Kvůli deferred renderingu musí XNA být přeloženo v HiDef[3] modu, ve kterém je možné využí-

[3]Podrobnější informace o módech v XNA jsou k nalezení na blogu Shawna Hargreavese: http://blogs.msdn.com/b/shawnhar/archive/2010/03/12/reach-vs-hidef.aspx

vat více možností dnešních grafických karet, zejména vykreslování do více render targetů.

Hlavní třídou pro vykreslování je třída *DeferredRenderer*, která obsahuje funkce pro vykreslování modelů. Veškeré kreslení pomocí deferred renderingu musí být obaleno metodami Begin a End. Mezi těmito metodami může probíhat jakékoliv kreslení do GBufferu, pokud použijeme shader, který do nej umí zapisovat, jinak může dojít k neočekávaným výsledků.

Deferred rendering implementovaný v programu umožňuje, kromě částí popsaných v 1.2.1.4, ke každému pixelu na obrazovce určit také intenzitu a velikost lesklé složky v Phongově světelném modelu, tedy lze vytvářet různé materiály.

Aby bylo možné používat pro načítání modelů, které lze vykreslovat pomocí třídy *DefferedRendering*, pomocí Content Pipeline, je součástí programu i rošíření Content Pipeline. Content Pipeline je řošířena o nový procesor, který umožňuje načítání modelů s normálovou mapu a specularní mapou a pro uložení používá třídu Model z XNA.

Součástí deffered rendreringu je i definice světel. V programu jsou zatím dostupná jen globální směrová světla, ale počítáno i s reflektory a bodovými světli.

GBuffer použitý v implementovaném deferred renderingu lze snadno využít implementaci SSAO. Jako příklad post-processingu(viz 1.2.1.1) program obsahuje třídu *BloomPostprocess*, která je součástí jmenného prostoru *RTS.Rendering*.

2.4 Hra

Hra obsahuje převážně třídy reprezentující mapu, jednotky, suroviny, ale i jednotlivá menu a formuláře potřebné k vytvoření hry. Implementace hry by se dala rozdělit do několika částí, které mezi sebou více či méně komunikují. Jsou to tyto části: menu a formuláře potřebné k vytvoření hry, herní smyčka vyvolávající aktualizace a vykreslování, herní mapa, hráči, herní objekty a jejich správa.

Nejdůležitější částí je správa objektů, která zapouzdřuje jednotlivé úkony prováděné s aktivními objekty ve hře, například útočení, těžbu, stavění atd. K této části během aktualizace přistupují jednotliví hráči, které pomocí manažeru herních objektů ovládají dění ve hře, a uživatelský vstup, který reprezentuje uživatele programu.

Manažer herních objektů během aktualizací využívá herní mapu, aby mohl umisťovat jednotlivé objekty na správné pozice a vyhledávat nejkratší cesty.

Kromě toho jsou během aktualizací aktualizovány objekty zajištující grafický výstup, například je vytvářena minimapa.

Během vykreslování je postupně vyvoláváno na všech objektech, které se mají kreslit, vykreslení. Nejprve je vykreslována mapa, herní předměty, střely a nakonec 2D grafické prvky jako je minimapa či informační panely.

K implementaci hry jsou použity třídy popsané v 2.3, kromě nich jsou také v programu využity možnosti tříd z XNA Frameworku, zejména třídy umístěné ve jmenném prostoru *Microsoft.Xna.Framework.Content* umožňující načítání herního obsahu pomocí Content Pipeline a třídy z jmenných prostorů *Microsoft.Xna.Framework.Audio* a *Microsoft.Xna.Framework.Media* umožňující přehrávání zvukových efektů a hudby na pozadí.

Implementace hry je rozdělena do několika jmenných prostorů, které svým významem přibližně odpovídají výše popsanému rozdělení. Hlavní třídou programu je třída *RTSGame*, která je potomkem třídy *Game* z XNA Frameworku obsahující herní smyčku, tedy vyvolává proces aktualizace a kreslení. Třída RTSGame obsahuje dvě komponenty: Cursor vykreslující kursor myši a ScreenManager(viz 2.3.1).

Program obsahuje mnoho obrazovek s menu a jinými formuláři, pro bližší představu doporučujeme kapitolu 3. Z hlediska implementace se jedná pouze o potomky tříd popsaných v 2.3.1.1.

Hlavní třídou samotné hry je třída *GameScreen* umístěná ve jmenném prostoru *RTS.Game*, která spravuje herní obsah a výše popsaný průběh aktualizací a vykreslování hry. Kromě jiného tato třída také pracovává uživatelský vstup či kontroluje konec hry.

2.4.1 Herní obsah

Protože každá hra obsahuje mnoho externích zdrojů (například zvuky, textury atd.), připravila firma Microsoft do XNA Game Studia koncept Content pipeline, který umožňuje snadné načítání herního obsahu do hry. Content Pipeline se skládá z několika kroků, které lze rozdělit do dvou skupin. V první skupině jsou kroky, které probíhají pouze jednou během kompilace projektu. V teto části je načten původní zdroj dat, zpracován do podoby, které rozumí XNA Framework a následně zapsán. Při běhu programu je načítán předpracovaný soubor, což umožňuje jednak rychlejší zpracování, ale také zamezí běžným uživatelům počítače měnit zdrojové soubory, protože zpracovaná data jsou binární. Microsoft také do XNA Frameworku zařadil třídy, které umí zpracovat mnoho grafických a zvukových formátů, což dělá Content Pipeline asi nejvýznamnější část XNA Frameworku.

Koncept Content pipeline lze snadno lze snadno rozšířit o různé podporo-

vané formáty a u stejných formátů lze zvolit třídy, které se budou používat pro načtení(importer), zpracování(processor), zápis(writer) a sním spojené načítání za běhu(reader). Kromě jiného je možné v procesoru načítat i jiné zdroje, které jsou následně zapsány do stejného souboru jako právě zpracovávaný, nebo vyvolat zpracování jiných zdrojů a uchování referencí na ně.

Této možnosti program využívá a pomocí Content Pipeline načítá celé objekty, například herní mapu či prototyp jednotky i s modely, texturami atd.

Kromě Content Pipeline obsahuje XNA Framework i třídu *ContentManager*, která se stará o načítání a uvolňování herního obsahu přímo do hry. Ta také jiného zajišťuje, aby nevznikali zbytečné kopie stejných zdrojů v paměti.

2.4.2 Herní mapa

Herní mapa se skládá z několika částí. Její součástí jsou jednak grafické prvky jako terén či modely rozmístěné na hrací ploše, jednak informace o počátečním rozestavění hráčů, surovin atd.

Celá mapa je načítána jako jeden objekt pomocí Content Pipeline Extension (viz předchozí část textu), která jako vstup bere XML soubor validní podle XML Schema souboru terrain.xsd. Terén je tvořen jedním velkým 3D modelem, který je v procesoru Contant Pipeline vytvářen podle výškové mapy(viz 1.2.2), která je určena v XML. Na terén je možné aplikovat až 4 různé materiály, které jsou kombinované pomocí „blend“ mapy (textury, v niž každá barevná složka určuje procentuální zastoupení daného materiálu). V XML souboru lze také definovat modely tvořící detaily na mapě a jejich umístění. Pozice, na které jsou umístěné tyto modely, jsou zároveň místa na mapě, pres která nelze chodit. Pokud model zabírá více než jedno pole, je možné v XML souboru označit i ostatní pole jako neprůchodná. Kromě těchto více či méně grafických prvků lze v XML souboru označit místa, kde se na začátku objeví hráči a místa kde budou suroviny. Rozměry výsledné mapy odpovídají velikosti výškové mapy. Součástí mapy je i definice hudby na pozadí, při hraní hry na dané mapě.

Třída Map z jmenného prostoru *RTS.Game*, podle které se vytváří instance mapy v programu, obsahuje, kromě výše popsaných prvků, metody pro převod souřadnic na mapě na souřadnice světa, výpočet nadmořské výšky pozice na mapě, kliknutí na terén atd.

2.4.3 Herní objekty

Jak již bylo uvedeno v úvodu táto části textu, jsou herní objekty jednou z nejdůležitějších částí programu. Každý herní objekt je vytvářen podle prototypu, který je do programu načítán promocí Content Pipeline Extension. Samotné herní objekty jsou pouze instance těchto prototypů aktivně se vyskytující ve hře. Všechny herní objekty jsou potomky třídy *GameObject* z jmenného prostoru *RTS.Game.GameObjects*. Každý herní objekt obsahuje pozici na mapě, kvůli kolizím s jinými herními objekty, a měl by mít svého majitele z řad hráčů, jedinou výjimkou jsou suroviny, které nepatří nikomu. Součástí bázové třídy je také model, kvůli výběru objektu pomocí uživatelského vstupu.

Herní objekty jsou vytvářeny a spravovány pomocí třídy *GameObjectManager* ze stejného jmenného prostoru. Objekty samy o sobě umí pouze elementární operace, například posunout se z pole na pole, či přehrát zvuk při útoku. O nejnižší vrstvu umělé inteligence, která je společná pro všechny hráče, se stará právě *GameObjectManager*, který umí úkony jako přesouvání jednotek na větší vzdálenost podle nalezených waypointů, útočení na jiné herní objekty, stavbu budov či těžbu surovin.

Protože všechny tyto úkony vyžadují efektivní[4] vyhledávání nejkratší cesty v grafu, využívá manažer třídu *PathFinder* umístěnou ve jmenném prostoru *RTS.Game.Support*. Asi nejpoužívanějším algoritmem pro vyhledání nejkratší cesty v ohodnoceném grafu, který se v dnešní době požívá, je algoritmus A*[20, 14]. Právě tento algoritmus používá i třída PathFinder. Jako graf, ve kterém jsou cesty hledány, slouží terén převedený na čtvercovou síť. Překážky v grafu si třída bere jednak z instance mapy, jednak umí sama uchovávat přechodné překážky.

2.4.3.1 Jednotky

V předchozí sekci bylo zmíněno, že všechny objekty jsou vytvářeny podle nějakého prototypu, ani jednotky nejsou výjimkou. Prototyp je podobně jako mapa načítán z XML souboru pomocí s Content Pipeline extension, která umí zpracovat XML soubor validní podle XML Schema unittype.xsd, a je v programu reprezentován jako instance třídy UnitType. Tato třída definuje vlastnosti jednotky, například rychlost pohybu, model, maximální počet životů, zvuky, sílu útoku, způsob pohybu, zrychlení, identifikátor budov, které umí stavět atd.

Instanci jednotky ve hře reprezentuje třída *Unit*, která je potomkem třídy *GameObject*. Tato třída obsahuje metody, které obsluhují jednotlivé události, které

[4] Podle vývojářů Age of Empires[15] zabere hledání cesty asi 70% času věnovaného umělé inteligenci.

se týkají jednotek, například pokud jednotka útočí, je zasažena, těží surovinu, či se přesouvá z pole na pole.

2.4.3.2 Budovy

Práce s budovami je velmi podobná té s jednotkami. Jejich konfigurační XML soubor je popsán v souboru building.xsd. Informace potřebné k vytvoření budovy jsou jednodušší než u jednotek, v podstatě je potřeba určit pouze počet životů budovy, model a jaké jednotky umí trénovat, ale program umožňuje určit budově pro jaký druh surovin slouží jako sklad a základní bonusy.

Budovy sice nemají možnost se přemisťovat či využívat jiné prvky nejnižší vrstvy umělé inteligence, ale mohou, jaký bylo zmíněno, trénovat jednotky. K tréninku používají frontu jednotek, které jsou postupně stavěny, pokud na ně uživatel má dostatek surovin.

2.4.4 Hráči

Hráči jsou v programu reprezentováni pomocí tříd, které dědí od abstraktní třídy Player. Hra obsahuje dva hráče, jeden prezentuje uživatele programu, druhý reprezentuje počítačového hráče.

Uživatel programu je zvláštní případ hráče, protože většinu jeho úkonů nezadává třída, která ho reprezentuje, tedy třída *LocalPlayer*, ale jsou zadávány pomocí uživatelského vstupu. Třída *LokalPlayer* pouze uchovává informace o hráči, které dědí z bázové třídy. V objektu jsou ukládány informace o herních objektech, které hráč vlastní, identifikátory hráče, mezi které patří i barva zadané při vytvoření hry.

Zajímavější je třída AIPlayer, která reprezentuje počítačového hráče. Tato třída má během aktualizace, jakožto potomek třídy *Player*, přístup k *GameObjectManageru*. Tím pádem může během aktualizace ovládat své jednotky a reagovat na dění ve hře. Implementovaný počítačový hráč používá během aktualizace systém pravidel(viz 1.3.1.2), podle něhož řídí své úkony.

Pravidla pro řízení hry jsou přímo v kódu program. Tento přístup má své výhody i nevýhody. Výhodou je, že můžeme pro vytvoření systému pravidel využít veškeré možnosti jazyka C#, nejsme tedy omezeni možnostmi žádného skriptovacího jazyka. Velkou nevýhodou tohoto přístupu je nemožnost snadné změněny umělé inteligence bez zásahu do kódu, což výrazně omezuje škálovatelnost programu pomocí konfiguračních souborů.

Pravidla jsou převážně typu if-then: „Pokud je protivník v dané vzdálenosti od základny zaútoč všemi prostředky,“ nebo: „Pokud máš dostačené množství

surovin na stavbu budovy, která není postavena, tak ji začni stavět.“ Takových to pravidel musí být velké množství, aby vznikalo málo případů, kdy umělá inteligence nebude umět zareagovat.

2.4.5 Uživatelský vstup

Zajímavým problémem vyskytujícím se ve 3D prostředí je výběr předmětu, na který bylo kliknuto. 3D RTS hra musí umět zpracovat několik druhů kliknutí: kliknutí na jeden předmět, kliknutí na terén a výběr více předmětů.

Hlavní myšlenkou výběru objektu pomocí kliknutí je vytvoření paprsku od bodu na near plane kamery směrem do scény. Souřadnice bodu lze snadno získat z pozice myši na obrazovce a následné projekce do prostoru scény.

Pro výběr jednoho objektu poté stačí spočítat průsečíky s obalovou koulí tohoto objektu. Podle vzdáleností průsečíků od počátku paprsku lze jednoznačně určit, na který předmět uživatel kliknul.

U kliknutí na terén vyžadujeme mnohem větší přesnost, protože obalová tělesa nám nedají ani přibližnou aproximaci. K tomuto účelu se používá výpočet průniku paprsku s trojúhelníkovou sítí[19], čímž jsme schopni získat přesnou pozici na terénu i vzdálenost od počátku paprsku.

Nejsložitější je výběr více objektu. Opět nám stačí aproximace pomocí obalových těles, ale nemůžeme využít paprsků k výpočtu průsečíků. Tento problém jde řeší dvěma způsoby, bud nějak převedeme obdélník do herního světa, čímž dostaneme komolý jehlan, a vypočteme, které objekty jsou alespoň částečně uvnitř vzniklého tělesa, nebo můžeme všechny objekty promítnout na rovinu obrazovky a vypočítat průniky ve 2D.

Kapitola 3

Užívatelská dokumentace

3.1 Hardwarové požadavky a instalace

Součástí přiloženého CD je i složka Instal obsahující instalační balíček. Během instalačního procesu je kontrolováno, zda systém obsahuje i potřebné knihovny pro spuštění aplikace(.NET Framework, XNA Framework a potřebné části DirectX). Pokud v systému tyto balíčky knihoven nainstalovaný nejsou, pomůže instalátor s jejich nainstalováním.

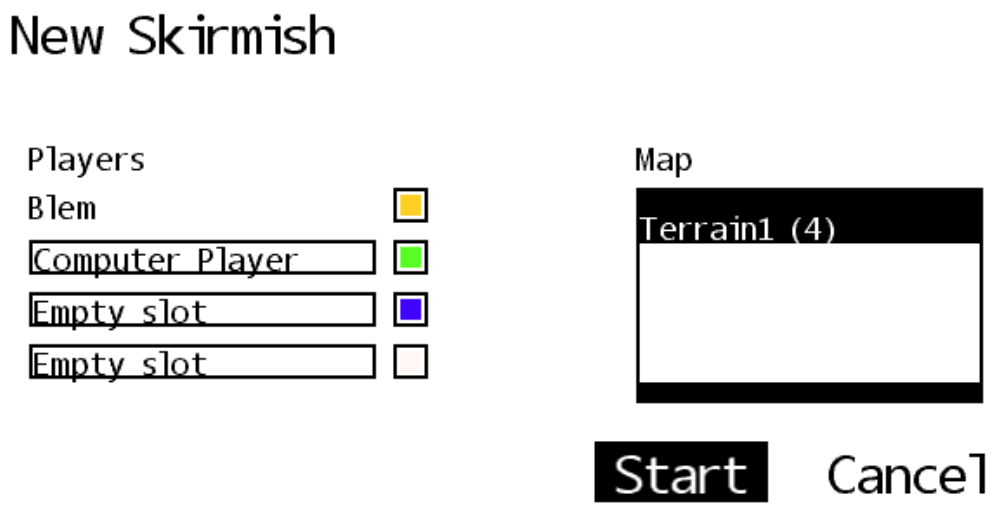
Kromě softwarových požadavků kontroluje instalační program i základní hardwarové požadavky, bez kterých program nelze spustit. Konkrétněji se jedna o grafickou kartu podporující Shader Model 3.0.

3.2 Menu, nastavení hry a vytoření nové hry

Po spuštění programu se uživatel dostane do hlavního menu. V menu se lze pohybovat jednak pomocí myši, jednak pomocí klávesnice(pomocí šipek, klávesy Enter a Escape. Hlavní menu obsahuje 3 volby: Hru pro jednoho hráče(Single player), nastavení(Options) a ukončení programu.

Po výběru volby Options se dostaneme na jednoduchý formulář s nastavením hry. Ve formuláři se lze opět kromě myši pohybovat i klávesnicí. K pohybu po formuláři slouží klávesa Tab a k ostatním úkonům lze použít klávesy Enter, šipky, písmena, číslice a klávesu Backspace. V nastavení si hráč může zvolit svojí přezdívku a nastavit hlasitost zvuků ve hře, či je úplně ztišit. Aby se změny projevily, je třeba stisknout tlačítko pro uložení formuláře(Save).

Pod možností Single player v hlavním menu se skrývá submenu, které obsa-



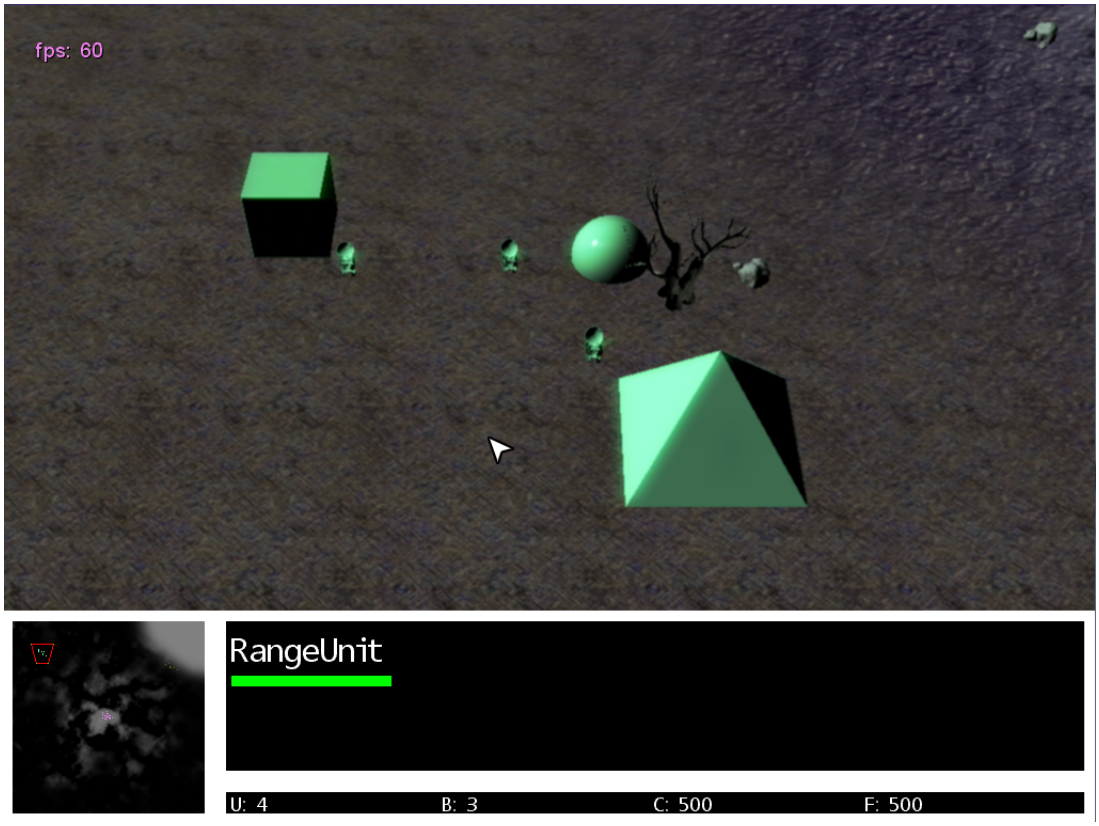
Obrázek 3.1: Nastavení nové hry

huje výběr typu hry pro jednoho hráče. Současná verze obsahuje pouze možnost vytvořit hru pro jednoho hráče na určité mapě, která bývá v RTS hrách označovaná jako šarvátka(Skirmish).

Po zvolení možnosti Skirmish se uživateli zobrazí formulář(viz obrázek 3.1) s nastavením nové hry. Na formuláři si lze v pravé části vybrat mapu, kterou bude uživatel hrát. V seznamu map je vždy uveden název mapy a v závorce za ním maximální počet hráčů, kteří mohou na dané mapě hrát. V levé části formuláře lze poté vybírat jednotlivé hráče a barvy, pod kterými budou vystupovat ve hře. Možnost "Empty slot" při výběru hráče označuje možnost, že z této pozice nevznikne ve hře hráč. Po stisku tlačítka "Start" se vytvoří nová hra s daným nastavním.

3.3 Herní prostředí a ovádání hry

Po načtení hry se zobrazí základní obrazovka(viz obrázek 3.2) s pohledem na uživatelovu základnu. V levém dolním rohu je zobrazena situační mapa(minimap), která usnadňuje orientaci ve hře. Na mapě jsou zobrazeny všechny aktivní objekty ve hře, ať objekty jednotlivých hráčů(vyznačeny barvou hráče), nebo neutrální objekty jako suroviny(označeny růžově). Také je na minimapě znázorněna viditelná oblast v daném okamžiku a bod, na který směřuje kamera. Pomocí kliknutí na minimapu je možné pohnout kamerou na vybraný bod na



Obrázek 3.2: Herní prostředí

herním plánu.

Kromě této možnosti pohybu kamerou, je také možné s ní pohybovat pomocí šipek, či najetím myší na okraj obrazovky. Kamerou je také možné otáčet. Pro otáčení kamery do stran lze použít klávesu Home(pohyb doleva) a klávesu End(pohyb doprava). Kameru lze také oddálit a měnit úhel pohledu na mapu pomocí kolečka myši - čím je kamera dál od bodu, na který se dívá, tím větší je úhel pohledu.

Ve spodní části obrazovky je zobrazen informační panel. Na informačním panelu jsou zobrazeny statistiky hráče. Jednak jsou zde zobrazeny počty hráčových jednotek(U) a budov(B), ale také množství surovin(F,C), kterými hráč disponuje.

Pokud uživatel vybere nějaký aktivní objekt, pak se nad informačním panelem zobrazí i panel s informacemi o vybraném objektu. Pokud je vybraný objekt vlastněn uživatelem zobrazí se podrobnější informace o objektu, například tla-

čítka na vytváření nových jednotek či budov. U jednotek ostatních hráčů jsou zobrazovány pouze životy daného objektu a jeho typ. U surovin je poté zobrazené množství, které daný zdroj ještě obsahuje.

Herní objekty lze vybírat pomocí levého tlačítka myši, ať kliknutím, nebo stisknutím a tažením. Výběr lze rozšiřovat pomocí výběru myší během něhož je stisknuta klávesa Shift. Vybrat jde najednou pouze jeden objekt, výjimkou jsou jednotky uživatele, kterých lze vybrat neomezené množství.

Pravým tlačítkem myši lze zadávat jednotkám úkoly. Pokud je kliknuto pravým tlačítkem na volnou ploch, vybrané jednotky se přesunou na danou pozici. Pokud hráč klikne na suroviny, vybrané jednotky, které jsou zároveň dělníky, začnou danou surovinu těžit a nosit ji do nejbližšího skladu dané suroviny. Další možností je výběr protivníkovy jednotky, která bude poté vybranými jednotkami pronásledována a napadána. Polední možností využití pravého tlačítka je stavba budov. Pokud je vybrána jedna pracovní jednotka a zvoleno stavení budovy, po kliknutí na volnou pozici na mapě na ní jednotka začne stavět budovu. Během stavby budovy je možné poslat stavět tuto budovu i jiné dělníky.

Pokud jsou vytvářeny nějaké jednotky v budově, budou se po dostavění objevovat na obvodu této budovy. Když jsou všechna pole na obvodu budovy zabraná, jednotka je zahozena, ale suroviny nebudou vráceny, proto by měl hráč věnovat pozornost i své základně. Jednotky se také mohou zastavit, pokud jsou zablokovány jiným objektem či skupinou objektů, které nedokáže obejít. Tuto situaci musí vyřešit uživatel.

Poledním prvkem herního prostředí je zobrazování zpráv ze hry. Zprávy se zobrazují po levé straně obrazovky a informují uživatele o různých událostech ve hře. Například pokud byla dostavěna nová budova, smazána nějaká jednotka, protože kolem budovy nebylo místo, na kterém by se objevila, pokud byl některý soupeř poražen atd.

3.4 Herní objekty

Ve hře se vyskytuje osm druhů herních objektů, které lze rozdělit do 3 skupin.

První skupinou jsou suroviny. Ve hře jsou obsaženy dva druhy surovin: palivo(Fuel) a krystaly(Crystals). Tyto objekty nemohou patřit žádnému hráči a lze je těžit pomocí dělníku. Pokud je naleziště suroviny vytěženo, zmizí z herní mapy.

Druhou skupinou jsou budovy. Budovy již náleží určitému hráči. To je naznačeno barvou budovy, která je shodná s barvou, která byla vybrána při založení hry. Hlavním smyslem budov je trénování nových jednotek, proto počet druhů jednotek odpovídá počtu druhů jednotek. Jsou tedy tři: hlavní bu-

dova(MainBuilding), ve které lze vytvářet dělníky, kasárny pro trénink jednotek útočících na blízko, kasárny pro jednotky útočící na dálku.

Poslední skupinou objektů jsou jednotky. Jak již bylo naznačeno hra obsahuje tři druhy jednotek. Asi nejdůležitějším type jednotky je dělník(Builder), který umožňuje stavbu nových budov, těžbu surovin, ale může i bojovat. Co se týče bojových schopností je ze všech druhů nejslabší a má nejmenší výdrž. Na druhou stranu je to jednotka nejlevnější a nejrychleji se trénuje. O něco silnější v boji jsou jednotky, které útočí na dálku(RangeUnit), což je jejich největší předností. Posledním, v boji nejsilnějším druhem jednotek, jsou jednotky útočící na blízko(MeeleUnit).

3.5 Cíl hry

Cíl hry je, jako obvykle u bojových strategických her, zničení všech budov a jednotek protivníka, proto jsou možné tři stavy jak může hra skončit. Buď jednotky uživatele ovládnou celou herní plochu a zničí všechny jeho protivníky, čímž hráč vyhraje, nebo bude více štěstí na straně protivníků a bude zničen hráč, tedy prohraje. Poslední možností jak může hra skončit je remíza, pokud se poslední dvě jednotky zabijí ve stejný okamžik, ale pravděpodobnost tohoto výsledku je velmi malá.

Závěr

V této práci jsme se seznámili s žánrem real-time strategických her a se základními vlastnosti, které se vyskytují u podžánru zaměřeného na boj. Uvedli jsme metody, které se využívají k jejich implementaci ve 3D prostředí, a základní techniky pro implementaci umělé inteligence v nich. Podle nabytých vědomostí jsme implementovali jednoduchý herní engine a RTS hru, která obsahuje i jednoduchého počítačového hráče založeného na systému pravidel.

Jak bylo popsáno v 1.1 a v 1.3, je problém real-time strategických her velmi rozsáhlý, zejména pak umělé inteligence. V komerčním prostředí vytváří podobné hry velmi specializované týmy, kterým vývoj hry trvá i několik let, tudíž není v silách jednotlivce dosáhnout stejných výsledků jako mají tyto týmy, proto výsledný program může sloužit spíše jako ilustrační.

Možností jak vylepšit stávající program je mnoho. Asi nejzápadnější je optimalizace výkonu, které jde dosáhnout příklad pokročilejšími způsoby implementace terénu popsanými v 1.2.2. Také je mnoho prostoru pro zlepšení v použité grafice. Použití particle systémů(1.2.1.5) či modelů a textur vytvořených dobrým grafikem výrazně ovlivní celkový dojem hráče ze hry.

Na druhou stranu bychom chtěli vyzdvihnout možnosti změny vlastností hry pomocí konfiguračních souborů. Přidáním konfiguračního souboru[1] můžeme snadno vytvořit novou jednotku, budovu či mapu. Také lze jejich změnou modifikovat celkový dojem hry(zvuky, grafické prky atd.), proto lze velmi snadno vytvořit jinou real-time strategickou hru.

I když program z mnoha důvodů nelze označit za dokonalý, základní principy vývoje real-time strategických her celkem dobře demonstruje.

[1]Soubor musí být zpracován pomocí Content Pipeline(viz 2.4.1), která se spouští během překladu programu.

Přílohy

Příložené CD obsahuje tyto soubory:

- Elektronickou verzi této práce ve formátu PDF
- Zdrojové kódy programu
- Instalační soubor programu

Literatura

[1] The ESA: *Essential facts about the computer and video game industry.* 2012 [cit. 2012-7-16]. Dostupné na: http://www.theesa.com/facts/pdfs/ESA_EF_2012.pdf.

[2] Sýkora, O.: *Umělá inteligence pro real-time strategické hry.* diplomová práce, Univerzita Karlova v Praze, 2007

[3] Nguyen, H.(editor): *GPU Gems 3.* Nvidia, 2007, ISBN-10: 0321515269.

[4] Rabin, S.(editor): *Game AI Wisdom.* Charles River Media, 2002, ISBN 15-8450-077-8.

[5] Rabin, S.(editor): *Game AI Wisdom 2.* Charles River Media, 2003, ISBN 15-8450-289-4.

[6] Rabin, S.(editor): *Game AI Wisdom 3.* Charles River Media, 2006, ISBN 15-8450-457-9.

[7] DeLoura, M.(editor): *Game Programming Gems.* Charles River Media, 2000, ISBN 15-8450-049-2.

[8] DeLoura, M.(editor): *Game Programming Gems 2.* Charles River Media, 2001, ISBN 15-8450-054-9.

[9] Treglia, D.(editor): *Game Programming Gems 3.* Charles River Media, 2002, ISBN 15-8450-233-9.

[10] Kirmse, A.(editor) : *Game Programming Gems 4.* Charles River Media, 2004, ISBN 15-8450-295-9.

[11] Pallister, K.(editor): *Game Programming Gems 5.* Charles River Media, 2005, ISBN 15-8450-352-1.

[12] Dickheiser, M.(editor): *Game Programming Gems 6.* Charles River Media, 2006, ISBN 15-8450-450-1.

[13] Walther, A.: *AI for real-time strategy games.* diplomová práce, University of Copenhagen, 2006.

[14] Bourg, D., Seenam, G.: *AI for Game Developers.* O'Reilly, 2004, ISBN: 0-596-00555-5.

[15] Pottinger, D.: Terrain Analysis in Realtime Strategy Games, Game Developers Conference Proceedings, 2000

[16] Grootjans, R.: *XNA 3.0 Game Programming Recipes.* Apress, 2009, ISBN-13: 978-1-4302-1855-5.

[17] Phong, B. T.: *Illumination for computer generated pictures.* Communications of the ACM, 1975, roč. 18, č. 6, s. 311-317.

[18] Samet, H.: *The Quadtree and Related Hierarchical Data Structures.* Computing Surveys, 1984, roč. 16, č. 2, s. 187-260.

[19] Möller, T., Trumbore, B.: *Fast, minimum storage ray/triangle intersection.* SIGGRAPH '05 ACM SIGGRAPH 2005 Courses, článek č. 7, ACM, 2005.

[20] Hart, P. E., Nilsson, N. J., Rapheal, B.: *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* Transactions on Systems Science and Cybernetics, 1968, roč. 4, č. 2, s. 100-107.

[21] Blizzard Entertainment: *StarCraft*[software]. Blizzard Entertainment, 1998.

[22] Blizzard Entertainment: *StarCraft II: Wings of Liberty*[software]. Blizzard Entertainment, 2010.

[23] Blizzard Entertainment: *Warcraft: Orcs & Humans*[software]. Blizzard Entertainment, 1994.

[24] Blizzard Entertainment: *Warcraft 2*: *Tides of Darkness*[software]. Blizzard Entertainment, 1995.

[25] Blizzard Entertainment: *Warcraft 3*: *Reign of Chaos*[software]. Blizzard Entertainment, 2002.

[26] Ensemble Studios: *Age of Empires*[software]. Microsoft Game Studios, 1997.

[27] Ensemble Studios: *Age of Empires II: The Age of Kings*[software]. Microsoft, 1999.

[28] Ensemble Studios: *Age of Empires III*[software]. Microsoft Game Studios, 2005.

[29] Westwood Studios: *Command & Conquer: Red Alert*[software]. Virgin Interactive, 1996.

[30] Westwood Studios: *Command & Conquer: Tiberian Sun*[software]. Electronic Arts, 1999.

[31] Westwood Pacific: *Command & Conquer: Red Alert 2*[software]. EA Games, 2000.

[32] EA Pacific: *Command & Conquer: Generals*[software]. EA Games, 2005.

[33] EA Los Angeles: *Command & Conquer 3: Tiberium Wars*[software]. Electronic Arts, 2007.

[34] EA Los Angeles: *Command & Conquer: Red Alert 3*[software]. Electronic Arts, 2008.

[35] EA Los Angeles: *Command & Conquer 4: Tiberian Twilight*[software]. Electronic Arts, 2010.

[36] id Software: *Doom 3*[software]. Activision, 2004.