Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Plánování akcí botů pro platformu Pogamut

Plzeň 2012 Matěj Sutr

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 11. května 2012

Matěj Sutr

Abstract

Bot Action Planning for Pogamut Platform
The aim of this thesis is to explore efficiency and implementation of A* planning in artificial intelligence - in particular to control virtual agents. This concept is originally based on STRIPS planner and was first successfully implemented in a mainstream computer game by J. Orkin. The goal of this thesis was to verify the algorithms designed by J. Orkin using Pogamut 3 software platform. Main contribution of the thesis is in discovering weak spots of the previous implementation, improvement of performance of search algorithm and quality of plans as well as comparison with former implementation.

Obsah

1 Úvod **1**

2 Popis základních prostředků **2**
2.1 A* Algoritmus 2
2.1.1 Velikost stavového prostoru 3
2.1.2 Heuristická funkce A* 4
2.2 Agent 4
2.3 Pogamut a GameBots2004 4

3 Implementace A* pro AI bota **7**
3.1 Systém volby akcí 7
3.2 Obecná akce 9
3.2.1 Počáteční podmínka 9
3.2.2 Efekt akce 9
3.2.3 Cena akce 9
3.2.4 Provedení akce 10

4 Realizace AI **11**
4.1 Stanovený cíl 11
4.2 Heuristická funkce 12
4.3 Definované akce 13
4.4 Popis světa 14

5 Hodnocení světa **16**
5.1 Prostředky 16
5.2 Svět s vysokou hodnotou 16
5.3 Implementované hodnocení 17
5.3.1 Stanovení celkového ohodnocení stavu světa 17
5.3.2 Stanovení hodnoty vybavení bota 18
5.3.3 Hodnocení výzbroje 19

6 Model soupeřova světa **23**
6.1 Motivace . . . 23
6.2 Prostředky . . . 23
6.3 Implementace . . . 24
6.3.1 Obecný model . . . 24
6.3.2 Stanovení rychlosti růstu . . . 25
6.3.3 Ověření . . . 26

7 Systém volby prováděné akce **27**
7.1 Problémy . . . 27
7.1.1 Aktuálnost stavu světa . . . 27
7.1.2 Náročnost výpočtu plánu a potřeba náhradní akce . . . 28
7.2 Implementace . . . 29

8 Kritická místa původní implementace **32**
8.1 Prioritní fronta . . . 32
8.2 Množství objektů v paměti . . . 33
8.3 Dopad úprav na rychlost algoritmu . . . 34
8.3.1 Testovací program . . . 34
8.3.2 Parametry testovacího stroje . . . 35
8.3.3 Výsledek původní varianty . . . 35
8.3.4 Výsledek s prioritní frontou . . . 36
8.3.5 Výsledek po eliminaci objektů . . . 36

9 Významné rozdíly oproti předchozím řešením **37**
9.1 Změny zvyšující efektivitu . . . 37
9.1.1 Zrychlení algoritmu . . . 37
9.1.2 Plánování v libovolný čas . . . 37
9.2 Změny zvyšující kvalitu plánů . . . 38
9.2.1 Změna cílů v A* . . . 38
9.2.2 Konkretizace akcí . . . 39
9.2.3 Omezené prořezávání . . . 40
9.2.4 Přidán model soupeře . . . 41

10 Možnosti rozšíření **42**
10.1 Úprava konstant na základě zkušeností . . . 42
10.2 Údaj o viditelnosti . . . 42
10.3 Kooperace více botů . . . 42

11 Závěr **44**

1 Úvod

Práce navazuje na bakalářskou práci Tomáše Ettlera z roku 2010 [3], která zkoumala využití A* (A hvězda) algoritmu pro plánování, čili hledání posloupnosti akcí, vedoucích k určenému cíli. Jeho cílem bylo pokusit se vytvořit umělou inteligenci postav ve hře Unreal Tournament 2004 (dále jen UT2004) podle metody navržené Jeffem Orkinem. Ten si získal velkou pozornost akademického světa i herního průmyslu právě unikátním návrhem umělé inteligence do počítačové hry F.E.A.R.. K implementaci této metody do hry UT2004 sloužila platforma Pogamut 3.

Cílem této práce je představit čtenáři algoritmy navržené J. Orkinem, software Pogamut 3 a implentaci řešení Tomáše Ettlera. Práce popisuje nalezená místa, kvůli kterým nebyl algoritmus dostatečně efektivní pro velký počet akcí, prezentuje jejich vylepšení a porovnává výsledek s výchozím programem.

Dalším úkolem této práce je seznámit čtenáře s návrhem a implementací všech vylepšení, která umožňují volit výrazně delší plány a celkově přibližují chování bytostí řízených ve hře umělou inteligencí živým hráčům.

2 Popis základních prostředků

2.1 A* Algoritmus

A* (A hvězda)[7] je informovaná metoda prohledávání stavového prostoru v grafu, která využívá heuristickou funkci ke zmenšení prohledávané části prostoru. Během prohledávání využívá algoritmus dva seznamy (označované jako OPEN a CLOSED), kam se ukládají uzly nově přidané (OPEN) a již expandované (CLOSED). Každý uzel obsahuje název, ohodnocení a odkaz na svého rodiče.

V zásadě se jedná o prohledávání grafu do šířky, kde je ohodnocující funkce $f(i)$ rozšířena právě o heuristickou složku $f(i) = g^*(i) + h^*(i)$, kde $g^*(i)$ je odhad vzdálenosti od počátečního uzlu a $h^*(i)$ je odhad vzdálenosti do cíle. Hodnota uzlu $f(i)$ lze v A* chápat jako cena cesty od počátečního uzlu k cílovému přes uzel i. Právě heuristická funkce umožňuje výrazné zmenšení prohledávaného prostoru.

Obecný postup při hledání řešení vypadá následovně:

1. Vlož do seznamu OPEN uzel s počátečním stavem, CLOSED je prázdný
2. Pokud je OPEN prázdný, řešení neexistuje
3. Vyber ze seznamu OPEN uzel i s nejnižší hodnotou $f(i)$, odstraň jeho reprezentaci v seznamu OPEN a vlož jej do CLOSED
4. Je-li uzel i cílovým, bylo nalezeno nejlepší řešení
5. Expanduj uzel i, tzn. všechny jeho následníky, kteří nejsou v seznamu CLOSED, vlož do seznamu OPEN
6. Pokračuj bodem 2.

Podle informace o přechůdci nalezeného uzlu je možné získat průchodem přes všechny předchůdce celou cestu od počátečního bodu k cíli. Algoritmus zaručuje nalezení nejlepšího řešení podle zadané heuristické funkce.

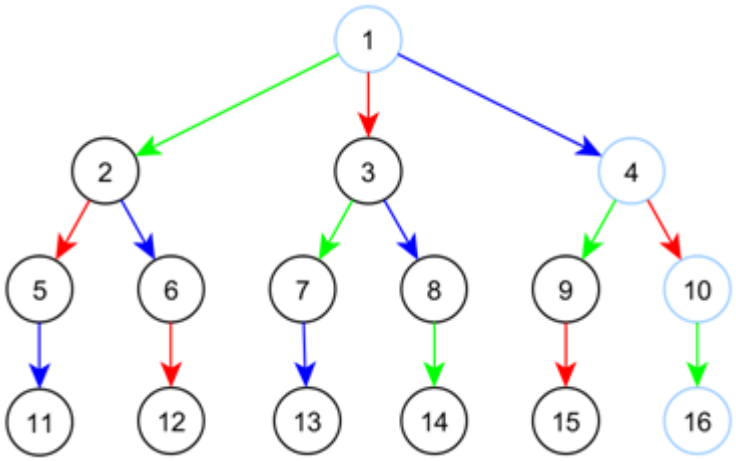
Funkcí seznamu CLOSED je pouze eliminace cyklů v grafu (zajišťuje, že nalezená cesta se nebude vracet do expandovaných vrcholů). Pokud máme

graf definovaný takovým způsobem, že cykly vznikat nemohou (všechny cesty míří stejným směrem), můžeme prohledávání zrychlit úplným vynecháním seznamu CLOSED.

A* algoritmus se obvykle používá v počítačových hrách k hledání nejvýhodnější cesty mezi dvěma body navigační sítě, zde bude využit k hledání posloupnosti akcí (plánů), která povede k danému cílovému stavu hry.

2.1.1 Velikost stavového prostoru

Velikost prostoru je daná počtem vrcholů grafu a počtem hran mezi nimi. Na obrázku 2.1 je znázorněn nejhorší možný případ pro 3 hrany. Graf má stromovou strukturu (z každého vrcholu směřuje maximálně n hran a neobsahuje cykly). Pro nejvýše n hran z jednoho vrcholu je počet expanzí až $max_n = 2(n!) + n + 1$, protože jeden uzel expandujeme na začátku algoritmu, n uzlů pak při první iteraci a $2(n!)$ v každé vzniklé větvi.



Obrázek 2.1: Stavový prostor pro 3 hrany z každého vrcholu

Takový prostor samozřejmě pro vyšší počet hran není možné kompletně prohledat v přijatelném čase, naštěstí v reálné situaci nevede z každého vrcholu všech n hran (podle počátečních podmínek) a díky heuristické funkci můžeme najít řešení výrazně dříve.

2.1.2 Heuristická funkce A*

Heuristická funkce má možnost výrazně zmenšit prohledávaný prostor, pokud je určena tak, že zvýhodňuje uzly, které se nejvíc blíží cílovému stavu a ostatní naopak znevýhodní. Teorie vychází z toho, že pokud je uzel od cíle daleko, budou při prohledávání zvýhodněny ostatní uzly. Pokud je funkce nastavena správně, pak uzly blízké cíli skutečně do cíle povedou a ostatní uzly nebude třeba vůbec expandovat.

2.2 Agent

Základní definice agenta může vypadat následovně:

> Agent je počítačový systém, který je zasazen do určitého prostředí a který je schopný autonomních akcí v tomto prostředí. [1]

> Agent je entita zkonstruována za účelem kontinuálně a do jisté míry autonomně plnit své cíle v adekvátním prostředí na základně vnímání prostřednictvím senzorů a prováděním akcí prostřednictvím aktuátorů. Agent přitom ovlivňuje podmínky v prostředí tak, aby se přibližoval k plnění cílů. [2]

Agentem tedy může být hardware (často označovaný jako robot) nebo software, který je mnohdy označován termínem bot. Srovnání vlastností objektů s agenty shrnuje následující tabulka 2.1 [2].

Za agenta tedy označujeme program, který se dá považovat za autonomní, inteligentní a plnící vlastní cíle. Každý agent se umí rozhodovat na základě vlastní paměti podle vjemů z prostředí, ve kterém se pohybuje. Na základě samostatně zvoleného cíle může provádět vlastní akce, které budou mít dopad na dané prostředí.

2.3 Pogamut a GameBots2004

Pogamut 3 [4] je systém vytvořený na Univerzitě Karlově za účelem rychlého prototypování agetů. Umožňuje testování navržených metod umělé inteli-

Objekt	Agent
zapouzdřuje proměnné a metody	zapouzdřuje chování
nutná synchronizace vláken	agenty jsou navzájem nezávislé
uspořádání objektů prostřednictvím dědičnosti	uspořádání agentů v organizacích
komunikuje voláním metod (posíláním zpráv)	komunikuje prostřednictvím vyšších komunikačních jazyků
prostředí (kromě kompilačního) nehraje žádnou roli	významnou roli sehrává prostředí

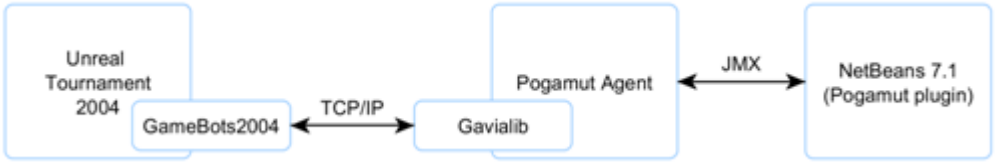
Tabulka 2.1: Srovnání abstrakce agentů a objektů.

gence v počítačové hře bez znalosti specifického skriptovacího jazyka používaného hrou (v UT2004 UnrealScript).

Pogamut 3 poskytuje velké množství hotových a jednoduše použitelných funkcí, díky kterým je proces vytváření, sledování a řízení agenta maximálně zjednodušen, což z něj dělá ideální základní platformu pro testování algoritmů vyšší úrovně, tedy umělé inteligence.

Významnou součástí platformy je plugin do vývojového prostředí Netbeans IDE 7.1, díky kterému je možné testované agenty efektivně ladit.

Tato práce se soustředí výhradně na nejvíce rozvíjený směr, tedy použití platformy Pogamut 3 [8] pro hru UT2004. Schéma architektury systému je znázorněno na obrázku 2.2.



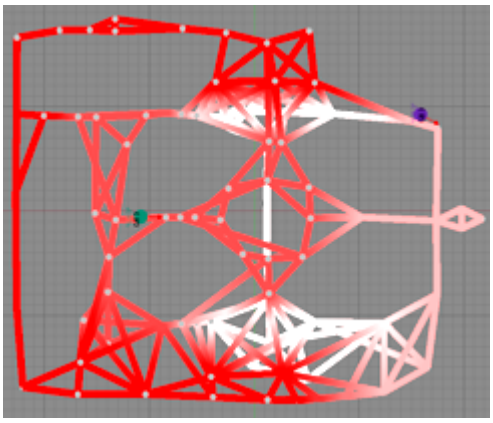
Obrázek 2.2: Obecné schéma architektury Pogamut 3

Základem je hra UT2004 [10], do které je přidán modul GameBots2004 [9]. Ten je napsán ve skriptovacím jazyce UnrealScript a jeho hlavní funkcí je zachytávat informace o stavu světa hry. Ty potom po dávkách posílá pomocí TCP/IP spojení na rozhraní knihovny Gavialib připojeného agenta. Tato knihovna přijatou zprávu přetvoří do struktur, ze kterých potom může agent čerpat informace o stavu světa ve hře. Posílaná data mohou obsahovat

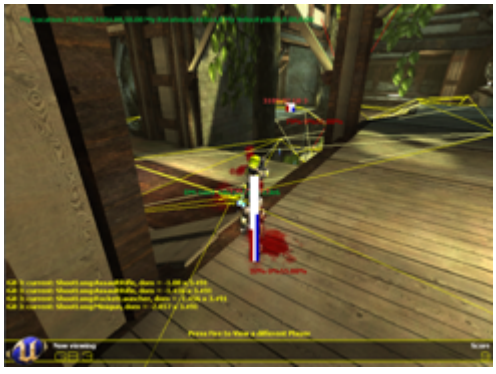
například údaje o navigačních bodech, o stavu agenta, jeho pozici, výzbroji nebo nepřátelích.

Připojovaný agent je programován v objektově orientovaném jazyce Java. Pomocí vývojového prostředí NetBeans IDE 7.1 lze díky poskytovanému pluginu vzdáleně sledovat a ladit agenta pomocí protokolu JMX.

Na obrázku 2.3 je ukázka z pluginu ve vývojovém prostředí, který umožňuje zobrazovat pozici připojených botů na zvolené mapě (v tomto případě DM-1on1-Idoma). Na obrázku 2.4 pak přímo ukázka ze hry.



Obrázek 2.3: Možnosti sledování pohybu agentů v Pogamutu 3



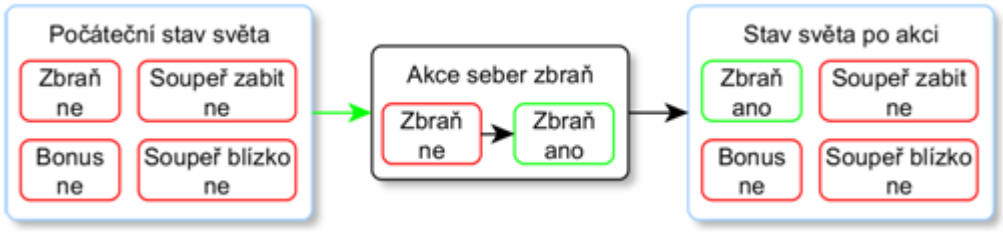
Obrázek 2.4: Sledování agentů z prostředí UT2004

3 Implementace A* pro AI bota

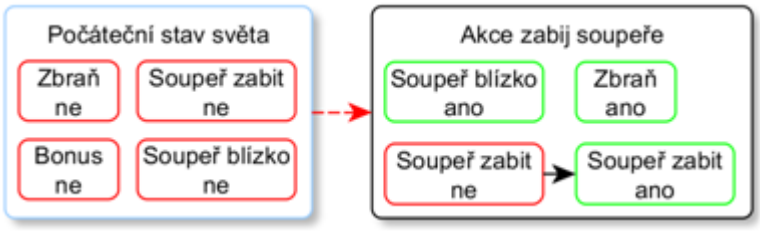
3.1 Systém volby akcí

Systém volby akcí podle J. Orkina [6] je založen na metodě STRIPS (Stanford Research Institute Problem Solver) [5], který se skládá z akcí a světů. Svět je popsán množinou atributů, které budou změněny při použití akce. Atributy světa navíc určují, které akce mohou být na daný svět použity. Postupným plněním akcí je možné dosáhnout stavu světa, který splňuje podmínky stanovené cílem. Cílový stav je definován množinou atributů, které musí být po skončení poslední akce v plánu splněny.

Na obrázku 3.1 je znázorněna ukázka aplikace akce na počáteční stav světa a její efekt. Obrázek 3.2 potom ukazuje situaci, kdy akci nelze použít.



Obrázek 3.1: Aplikace akce na počáteční stav světa a její efekt



Obrázek 3.2: Ukázka akce, jejíž počáteční podmínky nejsou splněny

Na následujícím příkladu je znázorněn obecný postup hledání řešení. V tabulce 3.1 jsou zobrazeny definované světy, v tabulce 3.2 potom definované akce. Aplikací akce na počáteční stav světa vzniká nový stav světa (pokud byly splněny podmínky akce). V dalším kroku můžeme aplikovat další akci, která má splněny počáteční podmínky. Dopředu není jasné, kolik akcí bude

nutné provést k dosažení cílového stavu světa, navíc řešení může být více a mohou být různě kvalitní.

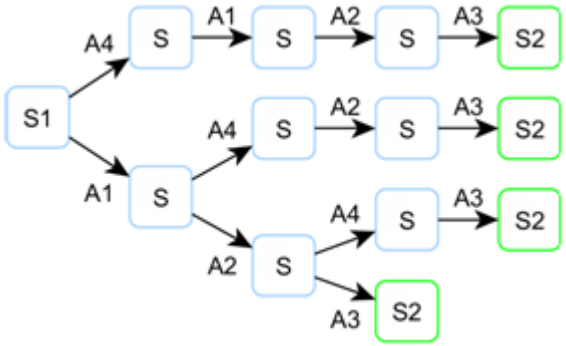
Atributy světa	Nepřítel blízko	Nepřítel zabit	Zbraň	Bonus
Počátek S1	ne	ne	ne	ne
Cílový svět S2	-	ano	-	-

Tabulka 3.1: Definované stavy světa

Definovaná akce	Nutný svět před akcí	->	Svět po akci
A1 (seber zbraň)	(- , - , ne, -)	->	(- , - , ano, -)
A2 (dojdi k nepříteli)	(ne, - , ano, -)	->	(ano, - , ano, -)
A3 (zabij nepřítele)	(ano, ne, ano, -)	->	(ne, ano, - , -)
A4 (seber bonus)	(- , - , - , ne)	->	(- , - , - , ano)

Tabulka 3.2: Definované akce

Na obrázku 3.3 je zobrazen strom všech povolených akcí od počátečního stavu až k dosažení cíle.



Obrázek 3.3: Postup hledání plánu

Varianty výsledného plánu:

S1 -> A1 -> A2 -> A3 -> S2
S1 -> A4 -> A1 -> A2 -> A3 -> S2
S1 -> A1 -> A4 -> A2 -> A3 -> S2
S1 -> A1 -> A2 -> A4 -> A3 -> S2

V tomto příkladu neexistuje jiné řešení než čtyři výše uvedené plány. Pokud bychom se snažili použít akci dvakrát nebo mimo dané pořadí, nebudou splněny její počáteční podmínky.

Při hledání nejlepšího plánu prohledáváme stavový prostor určený grafem, kde vrcholy tvoří konkrétní stavy světa a přechody mezi nimi představují akce. Naším cílem při určování plánu tedy je nalézt nejvýhodnější cestu mezi počátečním a koncovým vrcholem (stavem světa). K tomu můžeme podle [6] velmi dobře využít právě A* algoritmus, pokud určíme konkrétní akce a ohodnocující funkci.

3.2 Obecná akce

Každá akce používaná v A* prohledávání musí mít definované nezbytné parametry: počáteční podmínku, efekt akce, cenu a definici procedury, kterou bude bot během této akce vykonávat.

3.2.1 Počáteční podmínka

Akce nesmí být použita, pokud není v daném světě splněna její počáteční podmínka. Ta může vypadat například tak, že stav světa v okamžiku jejího spuštění bude obsahovat jeden konkrétní parametr, jiný parametr se ve světě nacházet nebude a třetí parametr bude mít kladnou hodnotu.

3.2.2 Efekt akce

Ještě před provedením akce musí být možné určit, jak bude stav světa vypadat po jejím dokončení. Tento její dopad může záviset na konkrétních ukazatelích světa a v některých případech je nutné následky akce pouze odhadnout na základě předchozí zkušenosti nebo pravděpodobnostního modelu. Podle efektu akce se určuje dosažení cíle.

3.2.3 Cena akce

Každá akce musí mít definovanou cenu, podle které se bude určovat ohodnocující funkce při A* prohledávání. Cena by proto měla odpovídat náročnosti vykonání akce, ale konkrétní zdroj odvozené hodnoty není nijak explicitně

určen (pouze platí, že cena každá akce by měla být určena stejným pravidlem, aby byly jednotlivé akce skutečně porovnatelné). Cenu může definovat například čas, potřebný k dokončení akce, vzdálenost, kterou bot musí urazit k cílovému předmětu nebo pouze konstanta (zde ale dojde k permanentní diskriminaci některých akcí na úkor jiných).

3.2.4 Provedení akce

Kromě samotných instrukcí, kterými se bot bude řídit po spuštění akce, zde musí být definované i podmínky, podle kterých lze určit, zda prováděná akce probíhá podle očekávání, zda je stále splnitelná a zda už nebyla dokončena.

4 Realizace AI

4.1 Stanovený cíl

J. Orkin ve hře F.E.A.R. využíval velké množství cílů, k jejichž splnění stačilo provést většinou jednu nebo dvě akce. Protože bot v UT2004 může sbírat předměty a jeho cílem je zabít soupeře, mohou být plány výrazně delší, ale cíl je v podstatě pouze jeden (zabít soupeře). Jakákoliv implementace jiných cílů (vyléčení, vyzbrojení, atp.) působila velmi nevyváženým dojmem a plány, které bot těmto cílům sestavoval byly tak přímočaré, že v důsledku byly stanoveny už samotným výběrem cíle.

To je na jednu stranu sice výsledek odpovídající původnímu návrhu, ale z hlediska umělé inteligence by byla škoda nevyužít celého potenciálu UT2004.

Protože každý hráč sbírá předměty, které zvyšují jeho šanci na úspěch v budoucí konfrontaci se soupeřem, měly by všechny plány umožňovat zařazení i takových akcí, které sice nevedou k bezprostřednímu splnění cíle, ale jejichž výsledkem bude výhoda nad soupeřem.

Už tedy nehledáme ten nejjednodušší a nejkratší plán, který změní stav světa z výchozího na cílový. Hledáme takový, který povede k cíli a který přitom umožní botovi získat největší možnou výhodu nad soupeřem. Cíl jako takový nám nakonec stačí jeden jediný - zabít soupeře.

Pokud světu, ve kterém agent zabije soupeře, přiřadíme číselnou hodnotu, můžeme cíl definovat jako svět, ve kterém má bot daný počet bodů.

Přiřazením hodnoty ostatním stavům světa dokážeme rozhodnout, která akce nabízí nějvětší zisk, a proto bude pravděpodobně výhodné zařadit ji do plánu. To, který plán bude označen za výhodnější, je potom dáno heuristickou funkcí A*.

4.2 Heuristická funkce

V původním návrhu heuristická funkce pomáhá určit nejkratší možný plán. Zvýhodňuje proto při prohledávání A* takové uzly, které se liší od cílového stavu v co nejmenším počtu atributů.

V současné implementaci jde ale hlavně o kvalitu plánu, proto záleží na zisku, který daná akce přinese. Zvýhodnění tedy nezávisí na počtu odlišných atributů, ale závisí na výhodě, kterou bude mít bot nad soupeřem v daném uzlu prohledávaného stromu. Proto je nutné zavést systém hodnocení jednotlivých stavů. Protože je cílem zabít soupeře, každý plán by měl končit střelbou na nepřítele. Plán, který této akci bude předcházet potom závisí na čase, který je potřebný k vykonání všech akcí, a zisku, který z plánu plyne.

Každá akce má stanovenou cenu, která figuruje jako funkce $g*(i)$ ve výpočtu heuristické funkce $f(i)$. Tou cenou je v současné implementaci vzdálenost, kterou je nutné urazit ze současné pozice k cíli akce. Alternativně by bylo možné definovat cenu akcí časem, který uplyne, než bude akce dokončena. Pro akce, které nejsou spojeny s pohybem (střelba), je cena v obou variantách stanovena odhadnutou konstantou.

Druhá složka heuristické funkce potom určuje vzdálenost hodoty stavu světa po skončení akce od cílové hodnoty (hodnota světa po zabití soupeře). Tím je zaručeno, že v prohledávání budou zvýhodněny všechny uzly, které vytvářejí botovi lepší stav světa za co nejnižší čas.

4.3 Definované akce

V tabulce 4.1 je seznam akcí implementovaných v programu.

Název akce	Popis akce
GoToEnemy	Přibliž se k nepříteli
RandomWalk	Prozkoumávej mapu
SearchEnemy	Najdi nepřítele
ShooltLong	Střílej ze zbraně na dálku
ShootShort	Střílej ze zbraně na blízko
TakeDoubleDamage	Seber bonus DoubleDamage
TakeHealth	Seber velkou lékárničku
TakeMiniHealth	Seber malou lékárničku
TakeShield	Seber obyčejný štít
TakeSuperShield	Seber velký štít
TakeFlakCannon	Seber zbraň FlakCannon
TakeLightningGun	Seber zbraň LightningGun
TakeLinkGun	Seber zbraň LinkGun
TakeMinigun	Seber zbraň Minigun
TakeRocketLauncher	Seber zbraň RocketLauncher
TakeShockRifle	Seber zbraň ShockRifle
TakeSniperRifle	Seber zbraň SniperRifle
TakeFlakCannonAmmo	Seber náboje pro FlakCannon
TakeLightningGunAmmo	Seber náboje pro LightningGun
TakeLinkGunAmmo	Seber náboje pro LinkGun
TakeMinigunAmmo	Seber náboje pro Minigun
TakeRocketLauncherAmmo	Seber náboje pro RocketLauncher
TakeShockRifleAmmo	Seber náboje pro ShockRifle
TakeSniperRifleAmmo	Seber náboje pro SniperRifle

Tabulka 4.1: Akce používané v programu

Každá z akcí je v programu reprezentována samostatnou třídou. Všechny akce, které končí sebráním předmětu (předpona *Take*), jsou na začátku hry vytvářeny podle předmětů na mapě. V seznamu použitelných akcí tak může být například třikrát akce *TakeHealth* (pro každou lékárničku bude mít nastaven jiný cílový předmět), ale ani jednou *TakeMinigun* (protože se na dané mapě nenachází ani jeden předmět tohoto typu). Délka seznamu akcí tedy závisí na počtu předmětů na mapě.

Akce *RandomWalk* se do vytvářených plánů vůbec nehodí, ale můžeme jí využít při hledání alternativního plánu v krizových situacích. Má proto stanovenou extrémně vysokou cenu (1000000), čímž je eliminováno zařazení této akce do výsledného plánu.

Akce *SearchEnemy* je naopak téměř zadarmo (1), ale její počáteční podmínka říká, že akce může být použita jedině v případě, kdy má bot alespoň o 5 bodů lepší hodnocení světa něž jeho soupeř.

Další akce, jejichž cena nehraje v A* téměř žádnou roli jsou *ShootLong* a *ShootShort* (jejich cena je také konstanta 1). To umožňuje botovi tyto akce kdykoliv zařadit, pokud má silnou zbraň a je blízko soupeře.

Všechny ostatní akce mají cenu stanovenou podle vzdálenosti k cílovému předmětu a jsou aplikovatelné, pokud je cílový objekt dostupný.

4.4 Popis světa

Systém Pogamut 3 získává ze hry UT2004 velké množství informací. Během prohledávání dochází neustále k odvozování nových stavů světa a kopírování všech informací by bylo časově náročné. Pro rozhodování agentů naštěstí stačí daleko menší množina atributů uvedená v tabulce 4.2.

Název atributu	Popis atributu
HEALTH	hodnota zdraví
SHIELD	hodnota štítů
IS_ENEMY_DEAD	soupeř zabit
IS_NEAR_ENEMY	bot je blízko nepřítele
HAS_LOADED_WEAPON	bot má aspoň jednu nabitou zbraň
SEE_ENEMY	bot vidí nepřítele
HAS_(zbraň)	bot má v inventáři (zbraň)
HAS_(zbraň)_AMMO	počet nábojů ke (zbrani)
IS_(předmět)_REACHABLE	(předmět) je dostupný
WORLD_VALUE	hodnota světa
EN_WORLD_VALUE	hodnota soupeřova světa
IS_DOMINATING	svět bota převyšuje ten soupeřův
ENEMY_DISTANCE	vzdálenost k soupeři

Tabulka 4.2: Reprezentace světa používaná v programu

Atribut *HAS_(zbraň)* a *HAS_(zbraň)_Ammo* existuje pro každou implementovanou zbraň uvedenou v tabulce 4.3).

AssaultRifle	FlakCannon	LinkGun	ShockRifle
SniperRifle	Minigun	RocketLauncher	LightningGun

Tabulka 4.3: Implementované zbraně

Atribut *IS_(předmět)_REACHABLE* je vytvořen na začátku hry pro každý konkrétní předmět zvlášť (každá akce s předponou *Take* má ve světě vlastní atribut). Celkový počet atributů tak opět závisí na počtu předmětů, které mají implementovanou akci pro sebrání, a které se ve světě během hry vyskytují (dáno konkrétní mapou).

5 Hodnocení světa

Protože cílový plán je nyní určen na základě hodnoty světa po skončení akce, ovlivňuje systém hodnocení nejen odhad aktuální dominance bota, ale i jeho výběr akcí. Při špatně definovaném modelu může bot volit nelogické nebo nevhodné akce a může mít silně zkreslený odhad vlastní situace.

5.1 Prostředky

Na hodnocení stavu světa může mít vliv kterýkoliv atribut, kterým je svět reprezentován. Mezi hlavní atributy patří ukazatel zdraví, ukazatel štítů a ukazatel dosažitelnosti soupeře. Kromě toho je mezi atributy zastoupen ukazatel pro každou zbraň v inventáři a pro množství jejích nábojů. Každý předmět, který se vyskytuje na mapě, je ve světě bota zahrnut ukazatelem dosažitelnosti (pokud je k němu implementována adekvátní akce).

Všechna tato data mohou dohromady tvořit poměrně komplexní přehled o situaci ve hře, ale pro určení jedné konkrétní hodnoty, která bude celý svět zastupovat musí být nejprve definovány váhy jednotlivých položek a jejich provázanost.

5.2 Svět s vysokou hodnotou

Nejdůležitějším kritériem pro ohodnocení světa musí být už z principu hry informace, zda je soupeř naživu. Musí být tedy zvýhodněn stav světa, kdy je soupeř botem zabit. Tento atribut nemůže vyvážit nic jiného. Nikdy nesmí nastat situace, ve které bot označí za nejlepší plán, který končí něčím jiným než zabitím nepřítele.

Hlavním aspektem hodnocení bota je ukazatel zdraví a štítů. Ve hře bot začíná bez štítů a se zdravím na hodnotě 100. Tato hodnota se během hry může měnit v rozmezí od 0 do 199, hodnota štítů se dá navýšit na 150. Vzhledem k faktu, že některé zbraně mohou ubrat při jednom zásahu až 100 bodů ze součtu zdraví a štítů, je výhodné být před soubojem řádně vybaven.

V nejlepším možném případě může bot mít 199 bodů zdraví a plných 150 bodů štítů, reálně se za výhodný stav dá považovat hodnota 120 bodů zdraví, která už může představovat výhodu nad soupeřem. Tendence bota sbírat předměty zvyšující hodnotu zdraví by měla s ubývajícím zdravím výrazně stoupat, naopak při velmi vysokém stavu zdraví a štítů by měla být minimální.

V nejlépe ohodnoceném světě by měl bot v inventáři všechny dostupné typy zbraní a k nim maximální možné množství nábojů. Ve skutečnosti je ale bot na souboj dobře připraven už ve chvíli, kdy má alespoň jednu silnou zbraň na blízko a jednu na velkou vzdálenost (za silnou zbraň můžeme považovat v podstatě jakoukoliv, kromě základní zbraně AssaultRifle).

Každá ze zbraní má specifické vlastnosti, takže její užití v konkrétní situaci by mohlo být výhodnější než jiné. Bot by proto měl mít tendence ke sbírání dalších zbraní. Rozdíl mezi hodnocením světa s jednou silnou zbraní na blízko a se dvěma by rozhodně neměl být tak výrazný, jako mezi stavem, kdy nemá žádnou a kdy má alespoň jednu silnou zbraň.

5.3 Implementované hodnocení

5.3.1 Stanovení celkového ohodnocení stavu světa

Vzhledem ke stanoveným předpokladům, se hodnocení světa bota zakládá na dvou hlavních složkách. První složkou jsou body za zabití soupeře a druhou složkou je jeho celková vybavenost (hodnota zdraví, štítů a zbraní). Aby bylo zaručeno, že bot zůstane motivován k další konfrontaci, musí být počet bodů za zabití soupeře vždy větší nebo roven počtu bodů, které může získat při plné výbavě. V programu jsou obě veličiny nastveny na maximální hodnotu 50 bodů. Tím je zaručeno, že bot bude muset při sestavování plánu vždy volit akce zakončené nalezením a zabitím soupeře.

Zde je důležité zdůraznit, že hodota celkové vybavenosti je dále přenásobená (health/199). To snižuje celkový výsledek jednotlivých složek, pokud je zdraví na nízké úrovni nebo naopak zesiluje, pokud má bot hodnotu zdraví na vysoké úrovni. Můžeme to interpretovat i tak, že jakkoliv dobře je bot vybaven zbraněmi, hodnocení jeho stavu světa nemůže být vysoké, pokud má nízkou hodnotu zdraví.

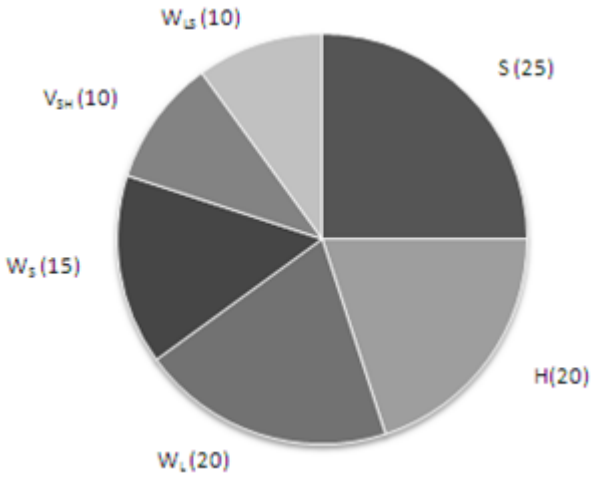
5.3.2 Stanovení hodnoty vybavení bota

V následující tabulce 5.1 je zobrazen seznam definovaných proměnných, používaných při výpočtu hodnoty světa.

Název	Jméno v programu	Význam
W	worldValue	agregovaná hodnota vybavení
S	shieldValue	hodnota štítů
H	healthValue	hodnota zdraví
W_L	weaponryLongValue	hodnota výzbroje na dálku
W_S	weaponryShortValue	hodnota výzbroje na blízko
V_{SH}	shieldAndHealthValue	součet hodnoty štítů a zdraví
W_{LS}	weaponryLongAndShortValue	součet hodnoty W_L a W_S
B_W	bestWeaponBonus	bonus za nejlepší zbraň
B_A	bestAmmoBonus	bonus za náboje do nejlepší zbraně
A_A	allAmmoBonus	bonus za všechny náboje
D_D	doubleDamageBonus	bonus za aktivní doubleDamage

Tabulka 5.1: Definované proměnné

Na obrázku 5.1 je vidět, jakým podílem přispívají jednotlivé složky hodnocení světa do celkové hodnoty za vybavení (celkem lze získat až 50 bodů).



Obrázek 5.1: Základní rozdělení vah při hodnocení světa

Toto rozdělení je založeno na dlouhodobém testování a vychází z něko-

lika elementárních úvah. Každá z těchto částí představuje základní položku v hodnocení stavu světa, která může silně ovlivnit jeho reálnou hodnotu. Kromě zvýhodnění stavu s vysokým zdravím a stavu s vysokým štítem, by měl být zvýhodněn i stav, kdy má bot současně zdraví i štíty, aby neměl tendenci trvale zvýhodňovat zdraví nebo štíty a aby měl naopak tendenci navyšovat obě hodnoty současně. Stejně tak by měl být zvýhodněn stav, kdy má bot zbraně na blízko a zároveň zbraně na dálku.

Pokud bude mít bot například plný ukazatel zdraví i štítů, celkové hodnocení bude minimálně 65 bodů ze 100. Pokud ale bude mít plné zdraví a žádné štíty, bude hodnota jeho světa v rozmezí 35 a 70 bodů. Analogicky se stejně dá popsat případ, kdy má sice všechny zbraně a maximální množství nábojů, ale nízké zdraví a štíty. V takovém případě by byla hodnota jeho světa 35 bodů plus několik bodů za zdraví a štíty. Na přiloženém CD lze pro ilustraci najít rozsáhlý srovnávací graf, zobrazující velké množství takto předdefinovaných parametrů a výslednou jimi generovanou hodnotu světa.

Každá složka celkového hodnocení je normována svou maximální hodnotou a vynásobena přiřazenou váhou. Hodnocení celé výbavy má tedy následující tvar:

$$W = 25\frac{S}{150} + 20\frac{H}{199} + 10\frac{V_{SH}}{349} + 20\frac{W_L}{100} + 15\frac{W_S}{100} + 10\frac{W_{LS}}{200} \tag{5.1}$$

5.3.3 Hodnocení výzbroje

Každá zbraň má přidělenou váhu, která určuje její kvalitu. Ve většině map některé zbraně chybí, proto váhu musíme chápat pouze jako prostředek pro porovnání dvou zbraní vůči sobě a ne konkrétní hodnotu. Stejně je s ní nakládáno i v následujících výpočtech. Váhy zbraní jsou založeny na jejich teoretické účinnosti a konečné hodnoty byly doupraveny podle úspěšnosti ve hře.

V tabulce 5.2 a 5.3 je vidět výčet zbraní a jim přidělených hodnot.

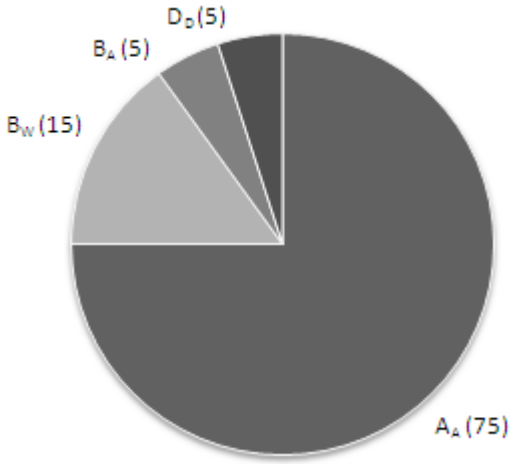
zbraň	váha	max. nábojů	zásobník
AssaultRifle	1	200	50
FlakCannon	10	35	10
LinkGun 2.mód	9	220	50

Tabulka 5.2: Parametry zbraní na blízko

zbraň	váha	max. nábojů	zásobník
ShockRifle	17	50	20
LinkGun	16	220	50
SniperRifle	21	35	10
Minigun	13	300	50
LightningGun	14	40	10
RocketLauncher	19	30	9

Tabulka 5.3: Parametry zbraní na dálku

Výzbroj na blízko i na dálku je hodnocena podle čtyř složek, uvedených na obrázku 5.2.



Obrázek 5.2: Rozdělení vah při hodnocení výzbroje

Největší bonus je přidělen situaci, kdy má bot k dispozici alespoň jednu silnou zbraň. Ke zjištění nejlepší zbraně musíme nejdřív definovat každé zbrani její hodnotu a k tomu slouží následující vztah:

$$value_i = has_i \cdot weight_i \cdot \frac{ammo_i}{maxAmmo_i} \quad \text{, kde } i \text{ je typ zbraně} \tag{5.2}$$

Parametr *has* nabývá pouze hodnoty 0 nebo 1, podle toho, zda bot má (1) danou zbraň v inventáři nebo nemá (0). Parametr *weight* je definován pro každou zbraň podle tabulky 5.2 nebo 5.3 a reprezentuje výhodnost konkrétní

zbraně (každá zbraň má definovanou vlastní váhu, podle které je určena její výhodnost). Proměnná *ammo* představuje počet nábojů k dané zbrani, které má bot v inventáři a *maxAmmo* je jejich maximální možný počet. Když známe hodnoty jednotlivých zbraní, můžeme z nich vybrat tu nejlepší a určit bonus:

$$B_W = 75\frac{bestValue}{maxWeight} \tag{5.3}$$

Parametr *bestValue* představuje hodnotu nejlepší vlastněné zbraně, *maxWeight* pak představuje nejvyšší váhu ze všech zbraní kategorie.

Bonus za náboje se vztahuje opět k nejlepší vlastněné zbrani a nabývá hodnot od nuly do pěti. Řídí se následujícím předpisem:

$$B_A = 5\frac{bestAmmo}{maxAmmo} \tag{5.4}$$

Parametr *bestAmmo* udává počet nábojů nejlepší vlastněné zbraně, *maxAmmo* je maximální počet nábojů k této zbrani, které je možné mít v inventáři.

Další složkou je bonus za celkový počet nábojů. Ten nabývá také pouze hodnot od nuly do pěti. Ve výpočtu je nutné uvažovat různé váhy jednotlivých zbraní, protože plný zásobník základní zbraně AssaultRifle nemá zdaleka takový význam, jako například 3 náboje do zbraně SniperRifle. Vztah pro výpočet tedy vypadá následovně:

$$A_A = 5\frac{\sum_i(ammo_i \cdot weight_i)}{\sum_i(maxAmmo_i \cdot weight_i)} \quad \text{, kde } i \text{ je typ zbraně} \tag{5.5}$$

Jedná se tedy o podíl součtu všech nábojů v inventáři přenásobených váhou přidružené zbraně a počtu všech dosažitelných nábojů přenásobených opět váhou zbraně. Proměnná *ammo* udává počet nábojů konkrétní zbraně, *weight* její definovanou váhu a *maxAmmo* je maximální počet nábojů k dané zbrani.

Pokud je v hodnoceném světě navíc údaj o aktivovaném bonusu Double-Damage a bot má aspoň jednu silnou zbraň, získává k celkovému hodnocení další body.

$$D_D = 15 \cdot hasDoubleDamage \cdot hasStrongWeapon \tag{5.6}$$

Celkové hodnocení výzbroje je tedy dáno součtem B_W, B_A, A_A a D_D. Výsledek vždy nabývá hodnot od 0 do 100. Hodnocení výzbroje bude vždy větší než 50, pokud má bot v inventáři alespoň jednu silnou zbraň a k ní alespoň polovinu zásobníku nábojů.

6 Model soupeřova světa

6.1 Motivace

Pro správné určení hodnoty světa konkrétního bota je samozřejmě nutné uvažovat hodnotu světa jeho soupeře. Právě srovnáním obou hodnot je možné určit, který z botů je ve výhodě, což je zásadní při sestavování plánů.

Pokud je svět jednoho bota ohodnocen výrazně lépe, může se pustit do hledání soupeře, protože je na případnou konfrontaci lépe připraven a má tak vysokou šanci v boji nepřítele porazit.

Naopak výrazně horší ohodnocení by mělo bota směrovat ke sbírání bonusů a tendenci vylepšit své ohodnocení světa nad úroveň soupeře.

6.2 Prostředky

Protože hra UT2004 poskytuje připojeným hráčům jen velmi omezené informace o stavu světa soupeře (neposkytne víc, než by mohl při hře získat živý hráč), je nezbytné jeho model sestavovat na základě zaznamenaných událostí a odhadovat nezaznamenané změny.

Není tedy možné zjistit například konkrétní hodnotu zdraví soupeře, což je zásadní parametr ve výpočtu hodnoty. Co se ale ze hry zjistit dá, je velikost poškození, které soupeř utržil při zásahu. Dále víme, že hodnota bota na začátku hry a po každé jeho smrti je 100.

Už z těchto údajů lze odvodit minimální hodnotu soupeřova zdraví, pokud předpokládáme, že během hry nezraňuje sám sebe. Na to, abychom mohli odhadnout jeho skutečnou aktuální hodnotu zdraví ale potřebujeme vědět, kdy navíc sebral které bonusy, přidávající zdraví a jak byly účinné.

Stejně jako by živý hráč během hry slyšel zvuk, který každý předmět vydává během sebrání, může být i bot informován o zaznamenaném zvuku. Protože uvažujeme hru jeden na jednoho a víme, které předměty jsme v ten který okamžik sebrali, snadno určíme, jaký předmět soupeř získal a o kolik mu pravděpodobně stoupne hodnocení stavu světa.

Dalším ukazatelem, ze kterého je možné získat potřebné údaje, je zaznamenaná střelba. Podobně jako sebrání předmětu může bot slyšet střelbu, ze které lze rozpoznat typ soupeřovy zbraně.

6.3 Implementace

6.3.1 Obecný model

Kdyby se bot pohyboval stále v blízkosti soupeře tak, že by slyšel každé sebrání předmětu, můžeme model světa stanovit velmi přesně. Budeme vědět jaké zbraně má soupeř v inventáři, víme kolik sebral lékárniček a kolik nábojů. Protože taková podmínka ale ve hře splněna být nemůže, nezbývá než odhadovat, o kolik bodů se jeho stav mohl zvýšit za dobu od jeho poslední smrti nebo od zmizení z dohledu.

Při hře jeden na jednoho hráče není pravděpodobné, že by se hodnocení světa jednoho bota snížilo bez účasti druhého. Sběrem předmětů může bot jedině získat a hodnota zdraví, štítů i nábojů by mimo konfrontaci se soupeřem klesat také neměla (předpokládáme, že bot střílí pouze na soupeře a sám sobě zranění nezpůsobuje). Navíc je přirozené zvýhodnit hráče, který se na mapě pohybuje delší dobu, aniž by byl zabit a narozdíl od čerstvě připojeného hráče měl čas na sbírání předmětů.

Můžeme tedy tuto situaci zobecnit do podoby, kdy hodnota světa bota, který není v kontaktu se soupeřem, od počátku hry nebo poslední smrti konstantně roste. *Rychlost růstu* závisí na jeho aktivitě, rozmístění předmětů a jejich počtu na konkrétní mapě. Navíc je nutné si uvědomit, že bot v okamžiku hodnocení soupeřova světa nezná ani jeho pozici. Na druhou stranu, s jakoukoliv stanovenou funkcí získáme pouze odhadovanou hodnotu, takže v zásadě není chyba, pokud použijeme průměrnou rychlost růstu naměřenou na průměrně velké a zaplněné mapě.

Svět soupeře ve výsledku můžeme hodnotit podle stejných pravidel jako ten vlastní, pokud k němu přičteme takto dopočtenou hodnotu. Stejně jako při výpočtu hodnoty vlastního světa se tak v hodnocení projeví množství zdraví, které má bot navýšené o hodnoty ze sebraných lékárniček (které náš bot zaregistroval) nebo naopak snížené o množství zdraví, které soupeři ubral v souboji. Analogicky získáme ohodnocení všech zbraní, štítů a počet nábojů.

Pokud například bot zaregistruje střelbu z konkrétní zbraně, může ji ve stavu soupeřova světa rovnou započítat společně s jedním zásobníkem, ze kterého ale odečte průměrnou dávku nábojů spotřebovaných při jedné střelbě.

V jiné situaci bot soupeře například jen lehce zasáhne. Pokud má v hodnocení jeho světa údaj o tom, že soupeř sebral štít, neubere zásahem hodnotu zdraví, ale pouze štítu.

6.3.2 Stanovení rychlosti růstu

K měření jsem použil mapy pro hru jeden na jednoho průměrné velikosti DM-1on1-Irondust a DM-1on1-Roughinery. V obou mapách lze dosáhnout stejného maximálního ohodnocení stavu světa, protože jsou na nich rozmístěny předměty stejných typů. V hodnocení není uvažována zbraň BioRifle, kapsle adrenalinu a MegaHealthKeg, které nejsou implementovány v akcích bota. Na mapě DM-1on1-Irondust je tak k dispozici celkem 32 předmětů, na mapě DM-1on1-Roughinery lze sebrat 42 předmětů.

body				procenta			
0	60	120	180	0	60	120	180
3,258	15,318	22,269	30,449	10,395	48,874	71,052	97,15
3,258	17,28	20,577	27,956	10,395	55,134	65,653	89,197
3,258	12,139	26,168	27,966	10,395	38,73	83,492	89,229
3,258	13,625	21,327	31,297	10,395	43,472	68,046	99,856
3,258	6,86	16,966	25,144	10,395	21,888	54,132	80,225

Tabulka 6.1: Ukázka z výstupu provedeného měření

Průměrný bonus za vybavení v hodnocení stavu světa byl v první minutě podle uvedeného měření 11,694 bodu, což odpovídá zvýšení celkového bonusu za výbavu ve stavu světa z 10,395% na 47,706%. Ve druhé minutě dochází k průměrnému zvýšení bonusu o 9,587 bodu, což ve výsledku odpovídá 78% za celkový bonus v hodnocení položky vybavenost ve stavu světa.

Protože bot má v této hře většinou výrazně méně času než minutu na to, aby se připravil na setkání se soupeřem, ukázala se jako vhodné tempo růstu jeho bonusu za vybavenost právě hodnota naměřená na první minutě experimentu. V programu je tedy soupeři, který se nachází mimo dosah bota,

každých 20 milisekund zvýšen stav bonusu za výbavu o 0,0039 bodu, což odpovídá růstu o 11,7 bodů za vteřinu.

6.3.3 Ověření

Ověření hodnoty bylo provedeno tak, že byl bot upraven způsobem, kdy neustále počítal stav světa, jako kdyby se v něm nacházel i jeho soupěř, kterého nevidí. Rozdíl skutečné hodnoty stavu světa bota a odhadované hodnoty imaginárního soupeře se po celou dobu měření (dvě minuty) pohyboval velmi těsně okolo nuly. Tím je ověřeno, že stav soupeřova světa určený podle definované funkce není výrazně podceňován, ani přeceňován.

Pro rychlost růstu obecně platí, že čím více je na mapě předmětů, tím je vyšší. Naopak na rozsáhlých nebo málo zaplněných mapách, kde je větší vzdálenost mezi předměty, může být tempo zvyšování hodnoty světa pomalejší. Rozdíl ale není nijak výrazný a univerzální rychlost naměřená na průměrné mapě se dá velmi dobře použít pro malé i velké mapy.

7 Systém volby prováděné akce

7.1 Problémy

7.1.1 Aktuálnost stavu světa

Program agenta v systému Pogamut figuruje jako klient, připojený k serveru hry pomocí TCP/IP protokolu. Informace o stavu světa ve hře dostává ve formě zpráv od modulu GameBots2004 po překladu knihovnou Gavialib. Tyto informace nejsou posílány kontinuálně, ale v jednotlivých balíčcích po uplynutí stanového intervalu GameBots2004. Tento interval je implicitně nastaven na 250 ms. Agent tak získává přístup k aktuálním informacím vždy až po uplynutí daného intervalu, dokončení procesu synchronizace obou modulů, výměně zpráv a překladu knihovnou Gavialib.

Frekvenci předávání dat lze sice zvýšit z původních 250 ms až na 100 ms pomocí parametru VisionTime v nastavení GameBots2004, ale při vyšším počtu hráčů se pak může stávat, že synchronizace nestihne vůbec proběhnout a data v Pogamutu se tak agentovi neaktualizují vůbec.

Pro dva připojené agenty je použitelné ještě nastavení intervalu zasílání zpráv na 120 ms, bohužel pak ale selhávají některé funkce Pogamutu, které v současné verzi ještě nejsou univerzální. Nebude tak fungovat například třída *UT2004PathAutoFixer*, která během navigace automaticky odstraňovala chybně definované hrany mezi navigačními body. Velikost mapy na rychlost synchronizace nemá vliv.

Živý hráč může na změny světa ve hře reagovat okamžitě, v reálném čase. Pokud má na změnu reagovat bot, musí o ní být nejdříve informován ze systému Pogamut. Protože odesílání dat z GameBots2004 do Pogamutu probíhá v pravidelných intervalech, může se v nejhorším případě stát, že bot bude o změně informován téměř po dvou periodách, což už bohužel nemůžeme označit za okamžitou reakci.

Je také nutné si uvědomit, že jakékoliv údaje o světě, které v programu použijeme, se nebudou vztahovat k času, kdy jsme je přijali, ale k době poslední odeslané dávky. Pokud tedy bot například střílí ze zbraně s kadencí 20 nábojů za sekundu, můžeme od Pogamutu dostat informaci, že mu náboje

ubývají po dvou, přestože každý vystřelený náboj bude samostatný objekt vykreslený na obrazovce a každý bude mít jiné místo dopadu.

7.1.2 Náročnost výpočtu plánu a potřeba náhradní akce

V obvyklé implementaci reaktivního plánování lze rychle reagovat na změnu světa, protože určení akce, která má být prováděna, je obecně časově nenáročné. Oproti tomu rozhodování na základě A* algoritmu může v závislosti na velikosti stavového prostoru hledat plán výrazně delší dobu. V programu tedy musí existovat systém, který v případě, že není možné čekat na výsledek A*, určí agentovi zástupnou akci. Tím se zajistí jeho reaktivita. To, jestli bude akce zvolena na základě nějakých definovaných podmínek, nebo jestli bude použit nejlepší dosud nalezený plán, už záleží na konkrétní situaci.

V ukázkových příkladech Pogamutu i v implementaci Tomáše Ettlera dochází k rozhodování o další prováděné akci s každou aktualizací světa (podle nastavené frekvence v GameBots2004). Při rozhodování na základě výsledku A* algoritmu ale při větším prohledávaném prostoru není většinou možné s každou změnou světa spouštět nové plánování, protože hledání řešení může trvat relativně dlouho.

Výsledkem A* je ale celý plán, který ve většině případů zahrnuje více než jednu akci. Pokud už máme nějaký plán připravený a nedokončený z předchozího prohledávání a stav světa se výrazně nezměnil (nedošlo ke konfrontaci se soupeřem, potřebné předměty jsou stále dosažitelné, atp.), není problém spustit hledání plánu a vymezit mu delší čas (v programu 1 s), čímž lze často získat lepší řešení. Po dobu výpočtu je pak možné provádět akce z předchozího plánu nebo akci, která byla při hledání označena za výhodnou, i když prohledávání stále probíhá.

Problém nastává ve chvíli, kdy se stav světa razantně změní (nejčastěji při setkání se soupeřem) během probíhajícího prohledávání. V takové případě nemá smysl pokračovat, protože nalezený plán by vycházel ze stavu světa před změnou (počáteční stav světa ve výpočtu musí být konstantní) a zcela jistě by nebyl vhodný k aplikaci na vzniklou situaci. V takovou chvíli je nutné prohledávání ukončit a spustit znovu s aktuálním stavem světa.

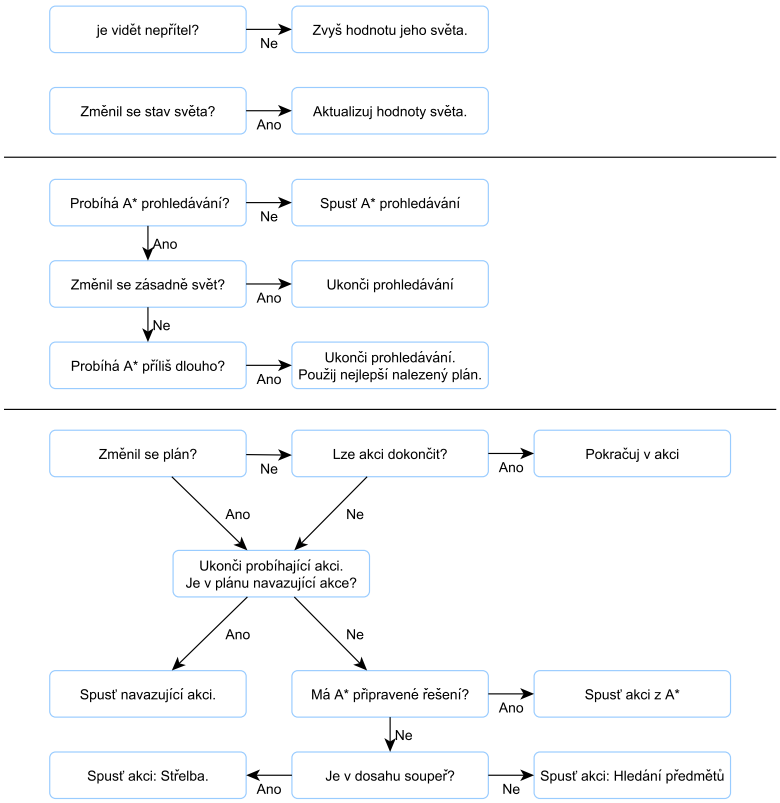
7.2 Implementace

Protože doba A* prohledávání se může v různých situacích velmi lišit a ve většině případů je delší než stanovený interval mezi dvěmi aktualizacemi světa, je vhodné spouštět plánování na tomto intervalu nezávisle. Z toho důvodu je prohledávání realizováno na jiném vlákně, než kde se aktualizuje stav světa a každé prohledávání má stanovenou maximální dobu běhu. Tím je umožněno spustit prohledávání v libovolnou chvíli.

V programu je nastaven limit plánování vymezeného času na jednu vteřinu, což je nejdelší doba, po kterou bot může v nejhorším případě zůstat bez plánu. Pokud do té doby algoritmus nenajde posloupnost vedoucí k cíli, použije se nejlepší dosud nalezený plán.

V případě výrazné změny světa je možné probíhající plánování přerušit nebo naopak použít plán, který sice nevedl k cíli, ale byl označen za dobrý, ještě před skončením prohledávání. To je vhodné v případě dokončení probíhajícího plánu, kdy není jasné, jakou akcí by bylo vhodné pokračovat.

Na schématu 7.1 je nastíněn obecný systém výběru prováděné akce. Protože bot by měl být při hře stále v pohybu, je důležité neustále kontrolovat, jestli má zvolenou akci a jestli stále může dosáhnout jejího cíle. K tomu v programu slouží vlákno třídy *AIManager*, které každých 20 milisekund projde podle uvedeného schématu dané podmínky a zajišťuje tak kompletní správu nad prováděnými akcemi.



Obrázek 7.1: Obecné schéma výběru prováděné akce

Tím, že tento proces probíhá pravidelně každých 20 milisekund, je zajištěna rychlá reakce na nově zvolený plán nebo případné změny stavu světa.

V každé iteraci tohoto cyklu je tedy možné reagovat na dokončení akce, zahrnout do stavu světa novou událost (objevení nebo zmizení předmětu nebo soupeře, smrt) nebo nový plán získaný výpočtem A* algoritmu.

V prvním kroku se aktualizují hodnoty světa, pokud od minulé iterace došlo ke změně.

V druhém kroku se ověří stav A* prohledávání. Stejně jako by měl být bot neustále v pohybu, není důvod neudržovat jeho plánování stále aktivní. Po nalezení řešení tak dochází k opětovnému spuštění prohledávání podle aktuálního stavu světa.

V posledním kroku se rozhoduje o konkrétní akci, která je v danou chvíli aktivní. Pokud už není splnitelná (cílový předmět není k dispozici, při střelbě dojdou náboje, atp.), byla dokončena, nebo pokud bylo od minulé iterace do-

končeno plánování a akce v novém plánu se liší od té prováděné, musí dojít k volbě akce nové. Zde se tedy nachází ona reaktivita. Agent má definovaný systém volby náhradní akce pro případ, že dokončil poslední akci z právě prováděného plánu a teprve spustil plánování, nebo došlo k zásadní změně světa a plánování muselo být ukončeno. Pokud taková situace nastane, má agent možnost ihned zareagovat (nemusí čekat na další zvolení akce A* prohledáváním).

V první řadě je volena akce z aktuálního plánu (nebo nového, pokud byl právě určen A*). Pokud už je plán kompletně dokončen, může bot zkusit použít nejlepší dosud nalezený plán z probíhajícího hledání. Pokud není ani tato akce dostupná, nezbývá než zvolit akci jednoduchým rozhodnutím. Volba akce se potom řídí základními ukazateli stavu světa a namísto plánu je určena pouze jedna obecná akce.

8 Kritická místa původní implementace

Po důkladném prostudování původnho programu T.Ettlera byly nalezeny dva zásadní problémy, které výrazně snižovaly efektivitu prohledávání. Ten hlavní problém vznikl během implementace (nevhodně zvolená struktura pro reprezentaci seznamu v A*), druhý je dán návrhem programu (během prohledávání se vytváří příliš velké množství objektů).

8.1 Prioritní fronta

Prohledávání v A* probíhá v obecné teorii pomocí dvou seznamů (OPEN a CLOSED). Protože je zajištěno, že graf sestavený ze stavů světa (vrcholy) a akcí (hrany) neobsahuje žádné cykly, není potřeba seznam CLOSED vůbec používat (ten by sloužil právě k eliminaci cyklů).

Prohledávání tedy vypadá tak, že do seznamu OPEN expandujeme všechny aplikovatelné akce. Potom ze seznamu vybereme nejlevnější akci, expandujeme ji (opět do OPEN) a ze seznamu odstraníme. Po expanzi ověříme, že odstraněná akce nevedla k řešení (pokud vede, končí prohledávání) a opět vybíráme nejlevnější akci ze seznamu.

V původní implementaci byl tento seznam tvořen spojovým seznamem, který musel být řazen podle ceny uzlů po každé expanzi. Zde samozřejmě docházelo při velkém počtu uzlů v seznamu k neúnosným zpomalením celého algoritmu.

Nejlepším řešením je v tomto případě zcela jistě nahrazení seznamu prioritní frontou, která má složitost vkládání i výběru nejlevnějšího uzlu pouze $O(log_2(N))$. Vkládané elementy jsou v ní umísťovány vždy přímo na pozici definovanou komparátorem (zde byly uzly řazeny podle jejich ceny).

8.2 Množství objektů v paměti

Během prohledávání stavového prostoru A* algoritmu je potřeba v programu udržovat velké množství informací o používaných uzlech. Musíme stále udržovat informaci o akci (hraně vedoucí k uzlu), o předchozím uzlu, stavu světa před akcí, stavu světa po akci a ceně uzlu.

Protože celý program je napsaný v programovacím jazyce Java, který je objektově orientovaný, nabízí se možnost použít v algoritmu nově definovaný objekt reprezentující svět (*World*) a objekt, který bude představovat uzel a jeho obsahem budou všechny výše uvedené parametry (*AStarNode*). Tímto způsobem byl řešený původní program, ale vzhledem k vysokému počtu uzlů během prohledání drasticky narůstá režie Javy a dochází tak k výraznému zpomalení.

Celá množina atributů reprezentující svět je ale složená pouze z integerů. Dokonce má každý svět stejný počet atributů, takže pro jeho reprezentaci stačí pole integerů, které v paměti počítače zabírá výrazně méně místa a práce s ním je znatelně rychlejší než s vytvářeným objektem. Pokud vytvoříme pole těchto reprezentací (dvourozměrné pole integerů), může být konkrétní svět určen pouhým indexem do tohoto pole, které je v programu nazváno *WorldBuffer*. Ještě výraznější zlepšení ale nastane v případě použití pole integerů místo původního objektu, který reprezentoval uzel grafu. Pokud totiž vytvoříme opět dvourozměrné pole (v programu *NodeBuffer*), ve kterém budeme uzly udržovat, mohou být v poli uzlu neseny všechny potřebné informace pěti integery. Rozdíl mezi oběma implementacemi shrnuje tabulka 8.1

Reprezentace	Objekty	Integery
Akce	AbstractAIAction	index v seznamu akcí
Předchozí uzel	AStarNode	index uzlu v poli NodeBuffer
Svět před akcí	World	index světa v poli WorldBuffer
Svět po akci	World	index světa v poli WorldBuffer
Cena uzlu	integer	integer

Tabulka 8.1: Možnosti implementace uzlů grafu

Výhodou je použití jediné integerové hodnoty (index do pole všech uzlů) místo velkého objektu vytvářeného za běhu (*AStarNode*), což může mít výrazný dopad na rychlost A* algoritmu. Důležitým aspektem je ale nutnost definovat v programu počet uzlů a počet světů, které mohou být používány během jednoho prohledávání. Uzly i světy musí totiž být recyklovány a ne-

smí chybět ani systém přerušení prohledávání, který zamezí přepsání světa nebo uzlu, na který stále existuje odkaz v seznamu OPEN. Tento problém se dá nejjednodušeji vyřešit přiřazením další integerové hodnoty každému z polí, která bude obsahovat identifikátor prohledávání. Potom už není problém testovat, zda nedochází k přepisu uzlu nebo světa, který byl vytvořen s aktuálním identifikátorem A*. Všechny výše uvedené postupy jsou implementovány ve finálním programu.

8.3 Dopad úprav na rychlost algoritmu

8.3.1 Testovací program

Rozdíly mezi implementacemi byly testovány na verzi programu, která umožnila výběr typu seznamu, typ reprezentace uzlů a navíc obsahovala třídu pro výpis statistik do souboru.

Svět byl tvořen 50 původními atributy a jako akce byly použity akce z původního programu, ke kterým byly přidány akce pro nově implementované typy zbraní (celkem tedy 27 akcí). Všechny akce měly ale shodnou cenu a nutná podmínka jejich splnitelnosti vždy platila (každá akce se dala použít při jakémkoliv stavu světa). Bylo tak možné eliminovat dopad rozdílných informací o stavu světa při opakovaném měření a zachovat jim shodné podmínky. Heuristická funkce byla určena stejným způsobem jako v původním programu (součet ceny předchozího uzlu a počet nesplněných cílových atributů).

Cílem byl stav světa obsahující následující definované atributy.

- HAS_MINIGUN_AMMO
- HAS_LOADED_WEAPON
- IS_ENEMY_DEAD
- HAS_FLAK_CANNON
- HAS_DOUBLE_DAMAGE

V takovém případě bude nutné expandovat řádově 11000 uzlů a délka seznamu OPEN bude obsahovat v době nalezení řešení okolo 300000 uzlů,

protože kvůli stejné ceně akcí je strom velmi široký a k dosažení cíle je zapotřebí plánu složeného minimálně ze čtyř akcí.

8.3.2 Parametry testovacího stroje

V tabulce 8.2 jsou uvedeny parametry počítače, na kterém bylo testování provedeno.

Operační systém	Windows 7 Professional (x64) SP1
Procesor	Intel Core i5-2500K @ 3,30 GHz
Operační paměť	2x4GB DDR3, 1600 MHz, 9-9-9-27
Grafická karta	MSI N560GTX-Ti OC
Základní deska	ASUS P8Z68-V PRO
Pevný disk	WD Caviar Blue 1TB

Tabulka 8.2: Parametry testovacího stroje

8.3.3 Výsledek původní varianty

V tabulce 8.3 jsou zobrazeny výsledky měření v případě použití spojového seznamu a objektové reprezentace (původní varianta).

Průměrná doba [ms]	116950,7
Počet měření	10
Medián	111715,5
Směrodatná odchylka	6653,4
Maximum	126220
Minimum	111252

Tabulka 8.3: Výsledky měření (spojový seznam a objekty)

Průměrná doba potřebná k nalezení řešení dosahuje času okolo dvou minut. Už z toho je zřejmé, že taková implementace se bez zásadních omezení a zjednodušení pro plánování nedá moc dobře použít. Lze snížit počet akcí, zjednodušit cíle nebo omezit délku plánů, ale i tak bude prohledávání pomalé.

8.3.4 Výsledek s prioritní frontou

Naprosto odlišná situace nastane, když použijeme namísto spojového seznamu prioritní frontu. Výsledky měření jsou shrnuty v tabulce 8.4.

Průměrná doba [ms]	2869,63
Počet měření	40
Medián	2886,579
Směrodatná odchylka	65,776
Maximum	2924,8
Minimum	2602

Tabulka 8.4: Výsledky měření (prioritní fronta a objekty)

Tento výsledek už je výrazně lepší. Situace sice stále není ideální, ale pokud by tato implementace byla použita v obvyklých podmínkách (méně komplikovaný cíl, některé akce nebudou splnitelné a budou mít různou cenu), byla by potřebná doba stále přijatelná.

8.3.5 Výsledek po eliminaci objektů

V tabulce 8.5 jsou výsledky měření, které bylo provedeno na programu využívajícím prioritní frontu bez vytváření nových objektů během prohledávání.

Průměrná doba [ms]	1153,359
Počet měření	40
Medián	1154,999
Směrodatná odchylka	4,69
Maximum	1155,954
Minimum	1127

Tabulka 8.5: Výsledky měření (prioritní fronta a pole)

S tímto výsledkem už můžeme být spokojeni, protože i přes všechny překážky dané testovacími podmínkami trvá průměrně plánování méně než 1155 ms, což už je pro plánování přijatelná doba i bez jakýchkoliv nutných zjednodušení plánu nebo cíle. V reálných situacích navíc ubyde množství neaplikovatelných akcí, což zmenší prohledávaný prostor. Navíc budou mít různé akce různou cenu a bude tak výrazně účinnější heuristická funkce, která opět sníží počet expandovaných uzlů.

9 Významné rozdíly oproti předchozím řešením

9.1 Změny zvyšující efektivitu

9.1.1 Zrychlení algoritmu

Díky úpravám provedeným na původním programu se povedlo zrychlit algoritmus prohledávání na únosnou hodnotu. Test ukázal snížení potřebného času na hodnotu menší než 1 procento z původní hodnoty (po úpravách je prohledávání až 101× rychlejší).

K dalšímu podstatnému zrychlení ale došlo i úpravou heuristické funkce a změnou definice cílů. Cíl je nyní ve stavovém prostoru nalezen výrazně dříve, protože je expandován výrazně menší počet uzlů. Současné hledání plánu neomezené délky tak trvá v závislosti na konkrétní situaci přibližně 10 milisekund. Tento údaj už není možné porovnat s původní implementací, protože plány budou ve většině situací od původní implementace odlišné. Už z rychlosti nalezení plánu je ale zřejmé, že nárůst efektivity je nezanedbatelný.

9.1.2 Plánování v libovolný čas

Původní řešení hledalo nový plán při každé aktualizaci stavu světa (k té docházelo každých 250 ms). Z toho důvodu měl A* na nalezení řešení pouze interval mezi dvěma změnami a každá změna ve stavu světa se neprojevila dřív než po této době. Přes všechna zjednodušení a silně redukovaný počet akcí v A* navíc algoritmus velmi často řešení najít nestihl.

Současný program je rozdělen do tří samostatných vláken tak, jak je znázorněno na schématu 9.1. První vlákno (*GoalBot*) přijímá informace o změnách světa a vykonává zadané akce, druhé vlákno (*AIManager*) spravuje probíhající akce a spouští A* prohledávání, které probíhá na třetím vlákně (*AStar*).

Obrázek 9.1: Schéma základního rozdělení programu do vláken

Oddělení plánovacího vlákna nám umožnilo prodloužit maximální délku prohledávání na současnou jednu vteřinu (nebo jinou libovolnou hodnotu, popřípadě dynamicky upravovanou).

Druhé vlákno může v nastaveném intervalu (každých 20 ms) kontrolovat stav světa, podle kterého má možnost rozhodnout o přerušení plánování nebo vynutit rychlé hledání náhradní akce. Tím jsme získali kratší dobu odezvy bota, který má k dispozici aktuálnější data a celkově rychlejší reakce i v případě, že interval mezi zasílanými zprávami necháme na původních 250 ms. Pokud ale chceme odezvu ještě zrychlit, můžeme tento interval s jistými omezeními zkrátit až na 120 ms.

9.2 Změny zvyšující kvalitu plánů

9.2.1 Změna cílů v A*

Protože původní program vycházel z implementace umělé inteligence použité v počítačové hře F.E.A.R., byly cíle A* hledání definovány analogicky k této počítačové hře. Vzhledem k tomu, že F.E.A.R. a UT2004 jsou ale velmi rozdílné hry a rozhraní Pogamut nabízí výrazně odlišné prostředky, působily použité cíle příliš přímočaře. Ve hře UT2004 je totiž základním předpokladem úspěchu sbírání volně dostupných předmětů, které se ve hře F.E.A.R. na straně umělé inteligence vůbec nevyskytují.

Na druhou stranu boti ve hře F.E.A.R. se velmi dobře pohybovali podle překážek na mapě a většina jejich cílů toho velmi dobře využívala. Z Pogamutu se ale ani základní informace o výskytu stěn bohužel nedá získat (existuje metoda pro odhalední překážky, ale je velmi náročná na výpočetní výkon a lze tak využít pouze ve specifických případech) a tak tyto cíle nelze

vhodně implementovat.

V původním programu v důsledku toho zbyl pouze základní cíl *KillEnemy* a k němu byly přidány cíle *Cure*, *Protect* a *Explore*. At' už byl potom zvolen kterýkoliv z cílů, vypadal výsledný plán vždy velmi podobně. Cíl *Explore* mohla splnit dokonce pouze jedna jediná akce (*RandomWalk*).

Původní myšlenka rozšířit množinu cílů a dodefinovat potřebné akce byla nakonec nahrazena zcela novou koncepcí. Hra UT2004 se zkrátka příliš liší a potřebné cíle by nebyly nikdy rovnocené ani optimální. Z toho důvodu je v programu použita nová definice cíle, který už není představován množinou atributů, které musí cílový svět obsahovat, ale hodnotou, kterou musí být výsledný svět ohodnocen. Pro hru ve skutečnosti existuje jediný cíl, a to je získat více bodů za zabití než soupeř. Hodnota ale navíc přidává možnost porovnat jednotlivé plány a zahrnuje i stav soupeřova světa.

Protože došlo k výraznému zefektivnění algoritmu a je možné sestavovat plány složené z velkého počtu akcí (nezřídka posloupnosti dvanácti i více akcí), můžeme botovi zadat jako cíl přímo stav světa, kterého může dosáhnout jedině zabitím soupeře. Pokud by měl bot na výpočet potřebný čas, teoreticky by každá posloupnost musela nutně končit akcí pro střelbu na soupeře, což je naprosto logický postup. V praxi často dochází k přerušení plánování dříve než dojde k zařazení posledních akcí. To ale v zásadě nevadí, protože nalezené řešení je díky heuristické funkci A* pro bota nejvýhodnější ze všech testovaných plánů a k cílovému stavu je tak nejblíže. Pokud není posloupnost akcí příliš dlouhá, k zařazení takové akce vždy dochází.

Výsledkem změny cíle jsou výrazně delší plány, které vždy končí zabitím soupeře.

9.2.2 Konkretizace akcí

Vzhledem k tomu, že velký počet předmětů se na mapách v UT2004 koncentruje ve více exemplářích stejného předmětu do jedné oblasti (náboje, minilékárničky, atp.), velmi často by bylo dobré řešení vysbírat všechny předměty na jednom místě. To ale není možné, pokud není pro každý předmět definována konkrétní akce, protože dokud se v A* bude objevovat pouze akce typu TakeNearestMiniHealth, není možné sebrat podle jednoho plánu víc než jeden takový předmět.

Z toho důvodu jsou v programu všechny akce spojené se sbíráním předmětů generovány na začátku hry podle zvolené mapy a každá je vázaná na konkrétní předmět. Každá taková akce je definovaná cílovým předmětem. Může tak potom vzniknout například pět různých reprezentací akce TakeHealth a tři akce TakeShockRifleAmmo. Touto změnou výrazně stoupne počet akcí a naroste stavový prostor prohledávaného stromu, ale protože se akce pro střílení dají naopak zobecnit na střelbu na blízko a střelbu na dálku, není výsledný počet akcí o tolik větší. Zobecnění akcí pro střelbu si můžeme dovolit z toho důvodu, že ve výsledném plánu se akce objeví maximálně jednou a konkrétní zbraň lze definovat přímo v proceduře pro provedení akce podle aktuální výzbroje bota.

Pro mapu DM-1on1-Idoma tak například vznikne místo 21 akcí (v původní implementaci T. Ettlera) celkem 33 akcí, z čehož 28 akcí bude vázáno na konkrétní předmět. Pro mapu DM-1on1-Albatross by už bylo vygenerováno celkem 49 akcí, což ale stále není po provedených optimalizacích problém, pokud bude fungovat stanovená heuristika. Odměnou jsou výrazně zajímavější plány, kdy bot může sbírat konkrétní předměty namísto jednoho nejbližšího.

9.2.3 Omezené prořezávání

Protože už není nutné za každou cenu zmenšovat stavový prostor (heuristická funkce výrazně omezuje počet prohledávaných posloupností), mohl být z programu úplně vynechán systém prořezávání stromu při expanzi uzlů v A* prohledávání. Ten původně zamezoval zařazení dvou akcí stejné kategorie do jednoho plánu. To mělo samozřejmě negativní dopad na kvalitu plánu, protože každý plán byl tak automaticky omezen na délku danou počtem kategorií, kde se z každé kategorie mohla v plánu objevit maximálně jedna akce.

V programu už k takto drastickému prořezávání nedochází. Všechny uzly, které obsahují splnitelné akce mohou být dále expandovány. Není omezen ani počet akcí, které se v plánu objeví. Maximální povolená hloubka zanoření při A* prohledávání (původně 5) je tak dána celkovým počtem použitých akcí.

9.2.4 Přidán model soupeře

Narozdíl od předchozích implementací je v programu uvažován kromě stavu světa řízeného bota i stav světa jeho soupeře. Díky tomu je možné lépe definovat následky akcí a hodnotit jejich výhodnost vzhledem ke konkrétním situacím.

10 Možnosti rozšíření

10.1 Úprava konstant na základě zkušeností

Stejně jako hodnocení stavu světa, i důsledek každé akce je založen na odhadu, který je v tomto programu konstantní. Jako efekt akce *ShootLong* je stanoven svět, ve kterém má bot snížené zdraví o 20 bodů, štíty o 10 bodů a náboje z jeho nejlepší zbraně ubydou o třetinu zásobníku. Protože se jedná o odhad, který vychází z nejčastějších případů, můžeme ho při plánování bez problému využít. Tento odhad ale nemůže být nikdy tak kvalitní, jako odhad založený na zkušenostech s konkrétním soupeřem na konkrétní mapě.

10.2 Údaj o viditelnosti

Informace o prostředí získané ze hry jsou velmi omezené. Bot například nemá možnost určit, zda je vidět z pozice soupeře nebo je schovaný za překážkou. Pro vytváření akcí, jejichž cílem by bylo například překvapit soupeře ze zálohy nebo skrýt se před ním, je taková informace zásadní.

Jednou z možností by bylo před začátkem hry definovat každému navigačnímu bodu, po kterých se bot pohybuje, údaj o místech na která je z dané pozice vidět. Ten by pak podle svého nejbližšího navigačního bodu snadno zjistil, zda je vidět na nejbližší navigační bod soupeře. Získaná informace sice nebude úplně přesná, ale pro základní odhad situace by měla stačit.

Bylo by tak možné uplatnit nové akce, které by mohly v určitých situacích nahradit střelbu, kterou končí současné plány.

10.3 Kooperace více botů

Protože je prohledávání stavového prostoru časově náročné, uvažovali jsme pouze souboj jeden na jednoho. Kdyby ale byl prostor výrazným způsobem zmenšen (za cenu horších nebo kratších plánů), bylo by možné připojit do hry soupeřů víc. V základním typu hry DeathMatch by to zřejmě nemělo zásadní

dopad, ale pro typ hry CaptureTheFlag (boj o soupeřovu vlajku) už by bylo možné nějakého způsobu kooperace mezi boty patřičně využít. Pro takový typ hry by ale musely být definovány nové akce i cíle. Stejně tak by musel být přepracován i systém hodnocení stavu světa.

11 Závěr

V práci byly představeny algoritmy navržené J. Orkinem, princip agentních systémů a způsob implementace A* prohledávání pro efektivní plánování. Za tím účelem byl prostudován také software Pogamut 3, který slouží k rychlému prototypování agentů.

Dalším krokem byla důkladná analýza programu Tomáše Ettlera, ve kterém byla odhalena hlavní kritická místa. Kromě návrhu jejich ošetření práce nabízí i porovnání implementovaných vylepšení s původním programem.

Oproti starším verzím byl program vylepšen v následujících aspektech:

- Doba prohledávání byla snížena z původních téměř dvou minut na něco málo přes jednu vteřinu, při zachování stejných podmínek, což představuje zkrácení potřebného času o 99 procent. Prohledávání je tedy oproti původní implementaci více než stonásobně rychlejší. Tohoto výsledku bylo dosaženo nahrazením objektů z původního programu efektivnějšími reprezentacemi.
- Díky oddělení algoritmu prohledávání na samostatné vlákno bylo umožněno spouštět plánování v libovolný čas. Přitom přibyla možnost plánování kdykoliv zastavit, pokud se stav světa významně změní.
- V nové verzi programu už není délka času přiděleného plánování omezena frekvencí příchozích aktualizací světa. Plánování může probíhat až do nalezení řešení, úplného prohledání stavového prostoru nebo uplynutí zadané hodnoty.
- Do programu byl implementován systém hodnocení stavu světa, který umožňuje objektivně porovnávat různé plány.
- Původní cíle, které se ve hře UT2004 příliš neosvědčily, byly nahrazeny cílem, který je definován požadovanou hodnotou stavu světa.
- Heuristická funkce byla upravena pro hledání plánů podle jejich výhodnosti bez omezení délky plánu. To má velmi pozitivní dopad na kvalitu plánů.
- Ze seznamu akcí byly vynechány univerzální akce pro sbírání předmětů. Naopak přidány byly akce, které se vážou na konkrétní předmět. Tato úprava umožňuje volit výrazně delší a zajímavější plány.

- Použitím novější verze platformy Pogamut (verze 3.2) byl snížen počet chyb, ke kterým docházelo během navigace agentů.
- Kromě stavu světa je v nové verzi uvažován i model světa soupeře. Ten je sice z velké části založen na odhadu podle obvyklého postupu ve hře, ale dává agentovi alespoň základní možnost srovnání vlastního stavu světa se soupeřovým.
- Díky výraznému zvýšení efektivity programu bylo možné upustit od výrazného prořezávání prohledávaného stromu z předchozích verzí, které mělo velmi nepříznivý vliv na kvalitu plánů.

Výsledná implementace bohužel není stoprocentně stabilní. Její pády způsobují chyby v platformě Pogamut 3.2, které se, i přes neustálý vývoj a obrovský pokrok od minulé verze, stále nepodařilo jejím autorům zcela odstranit.

Jako možné vylepšení byla v této práci navržena úprava konstant používaných v programu na základě vlastních zkušeností agenta. Tato vlastnost by umožňovala agentům alespoň částečné přizpůsobení vlastního chování konkrétní mapě a nepříteli.

Dalšího zlepšení by se dalo dosáhnout, kdyby byl každému navigačnímu bodu přidán seznam míst, která jsou z daného bodu viditelná. To by agentovi umožňovalo podle pozice soupeře snadno určit, zda je vůči němu skrytý nebo na očích. S přidáním nových akcí by tak mohly být tvořeny zajímavější plány.

Využití navrženého řešení pro koordinaci agentů v týmové hře by vyžadovalo kompletní přepracování použitých primitiv. Musely by být vytvořeny nové akce i systém hodnocení světa. Z hlediska studia umělé inteligence by ale taková implementace mohla přinést zajímavé výsledky.

Program včetně vygenerované dokumentace zůstane přístupný na webových stránkách katedry.

Literatura

[1] *M. Wooldridge, N. R. Jennings:* **Intelligent agents: Theory and practise**. Cambridge University Press, Cambridge, 1995.

[2] *A. Kubík:* **Inteligentní agenty**. Computer Press, Brno, 2004. ISBN 80-251-0323-4.

[3] *T. Ettler:* **Implementace inteligentních botů pro UT2004 v Pogamut podle J. Orkina**. Bakalářská práce, ZČU, 2010.

[4] *J. Gemrot, et al.:* **Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. In: Agents for Games and Simulations**. LNCS 5920, Springer, 2009, pp. 1-15.

[5] *N. J. Nilsson, R. E. Fikes:* **STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving**. Stanford Research Institute, 1970.

[6] *Orkin, J.:* **Agent Architecture Considerations for Real-Time Planning in Games. Proceedings of the Artificial Intelligence and Interactive Digital Entertainment**. AAAI Press, 2005.

[7] **Algoritmy umělé inteligence: Řešení úloh ve stavovém prostoru**. (Květen 2012).
http://autnt.fme.vutbr.cz/brezina/alg_umele_int.pdf

[8] **Pogamut 3: home page**. (Květen 2012).
http://diana.ms.mff.cuni.cz/main/tiki-index.php?page=About

[9] **Pogamut 3: GameBots2004**. (Květen 2012).
http://diana.ms.mff.cuni.cz/main/tiki-index.php?page=GameBots

[10] **Unreal Tournament 2004: home page**. (Květen 2012). http://liandri.beyondunreal.com/Unreal_Tournament_2004

[11] **Wikipedia: A* search algorithm**. (Květen 2012). http://en.wikipedia.org/wiki/A*_search_algorithm

Definice používaných zkratek

AI	Umělá inteligence (z anglického termínu Artificial Intelligence)
A*	A hvězda; algoritmus, který prochází stavový prostor a pomocí zadané heuristické funkce nachází nejvýhodnější řešení bez nutnosti prohledat ho celý.
Bot	Agent, počítačem řízená bytost, která v počítačové hře reaguje na podněty z prostředí podle programem definovaných pravidel.
GameBots2004	Modifikace hry UT2004, vytvořená pro Pogamut 3, která umožňuje ovládat postavy ve hře pomocí textových zpráv a současně pomocí textových zpráv předává informace o stavu světa.
Java	Objektově orientovaný programovací jazyk.
NetBeans IDE	Integrované vývojové prostředí, které lze využít při vývoji v programovacím jazyce Java.
Pogamut 3	Platforma vytvořená za účelem rychlého prototypování agentů.
UT2004	Počítačová hra Unreal Tournament 2004, která je využita při testování implementovaného algoritmu.

Přílohy na CD

Příloha č.1 Vybrané parametry světa a výsledné ohodnocení (graf)

Příloha č.2 Naměřené hodnoty: Rychost růstu ohodnocení (tabulka)

Příloha č.3 Naměřené hodnoty: Objekty a spojový seznam (tabulka)

Příloha č.4 Naměřené hodnoty: Pole a prioritní fronta (tabulka)