FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Hierarchické plánování pro real-time strategické hry
Student:	Bc. Lukáš Kameník
Vedoucí:	prof. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

V rámci práce bychom chtěli prozkoumat možnosti využití hierarchického plánování (hierarchical task network planning - HTN) pro rozhodování v real-time strategických hrách typu Dune či Command and Conquer. Standardní koncept hierarchického plánování nepředpokládá protivníka, je tedy třeba koncept vhodně obohatit případně upravit, například přijetím předpokladu dynamického prostředí bez protivníka s přeplánováním, aby jej bylo možné aplikovat pro real-time strategické hry. Předpokládáme, že práce bude budována na existujících platformách pro real-time strategické hry, jako je OpenRA, a hierarchické plánování, jako je SHOP2. Úkoly pro řešitele jsou následující:

1. Seznamte se s problematikou hierarchického plánování (HTN) a platformou OpenRA z hlediska možnosti integrace automatického rozhodovacího mechanismu.
2. Navrhněte přístup pro použití HTN plánování pro řízení rozhodování v real-time strategické hře, předpokládá se návrh vhodné hierarchie metod a interakce mezi plánovačem a hlavním rozhodovacím machanismem.
3. Implementujte svůj návrh v rámci platformy OpenRA a otestujte jej v relevantních herních scénářích.

[1] The OpenRA Developers: OpenRA: Red Alert, Command & Conquer, Dune 2000, Rebuilt for the Modern Era, platforma pro vývoj real-time strategických her, https://www.openra.net/, [Accessed February 2023].

[2] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu,

---

Elektronicky schválil/a Ing. Magda Friedjungová, Ph.D. dne 16. února 2023 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Fusun Yaman: SHOP2: An HTN Planning System. J. Artif. Intell. Res. 20: 379-404 (2003)

[3] Munir Hussain Naveed: Automated planning for pathfinding in real-time strategy games. University of Huddersfield, UK, 2012

[4] Alexandre Menif, Eric Jacopin, Tristan Cazenave: SHPE: HTN Planning for Video Games. CGW@ECAI 2014: 119-132

Diplomová práce

HIERARCHICKÉ PLÁNOVÁNÍ PRO REAL-TIME STRATEGICKÉ HRY

Bc. Lukáš Kameník

Fakulta informačních technologií
Katedra aplikované matematiky
Vedoucí: prof. RNDr. Pavel Surynek, Ph.D.
4. května 2023

Obsah

Poděkování vii

Prohlášení viii

Abstrakt ix

Seznam zkratek x

Úvod 1

1 Cíl práce 3

2 Teoretická východiska 5
2.1 Herní agenti . . . 5
2.1.1 Typy prostředí . . . 6
2.1.2 Agentní funkce . . . 6
2.1.3 Typy agentů . . . 6
2.1.4 Herní agenti v RTS . . . 7
2.2 Teorie her . . . 7
2.2.1 Hra v normální formě . . . 7
2.2.2 Hra v extenzivní formě . . . 8
2.2.3 Teorie her pro RTS . . . 8
2.3 Dynamické prostředí . . . 9
2.4 Klasické plánování . . . 9
2.4.1 Konceptuální model plánování . . . 10
2.4.2 Reprezentace klasického plánování . . . 13
2.4.3 Složitost . . . 15
2.5 HTN . . . 16
2.5.1 Úlohy a metody . . . 16
2.5.2 Domény, problémy a řešení . . . 17
2.5.3 Procedura HTN plánování . . . 18
2.5.4 SHOP2 . . . 18
2.6 Real-time strategické hry . . . 24
2.6.1 Analýza cílů AI v RTS . . . 25
2.6.2 OpenRA . . . 25
2.6.3 Dune 2000 . . . 26

3 Plánování v RTS 41
3.1 Analýza hry . . . 41
3.1.1 Využívání zdrojů . . . 42
3.1.2 Skladba armády . . . 42
3.1.3 Ovládání jednotek . . . 43
3.1.4 Analýza výchozí AI . . . 43
3.2 Reflexe lidského hráče . . . 44

3.3 Aplikace plánování pro OpenRA 44

4 Návrh herních AI **47**
4.1 Obecné vlastnosti nových AI 47
4.2 Návrh nové AI pro Dune 2000 47
4.2.1 Zdroje 48
4.2.2 Energie 48
4.2.3 Počty herních objektů 49
4.2.4 Čas při plánování 50
4.2.5 Stavba budov 51
4.2.6 Stavba jednotek 53
4.2.7 Útoky 61
4.2.8 Strategie 64
4.2.9 Ovladač 66
4.2.10 Možná rozšíření 68
4.3 Alternativní návrh AI 69
4.3.1 Plánovač 69
4.3.2 Ovladač 71
4.3.3 Vhodná rozšíření 73

5 Experimentální výsledky **75**
5.1 Experimenty, jejich výsledky a jejich analýza 75
5.1.1 RBAI vs. Omnius 77
5.1.2 IBAI vs. Omnius 77
5.1.3 RBAI vs. IBAI 80
5.1.4 Malá mapa 82
5.1.5 Velké počáteční zdroje 86
5.2 Analýza výsledků mezi RBAI a IBAI 86

6 Závěr **89**

A Implementační detaily **91**
A.1 Plánování 91
A.2 Soubory výsledků měření 91

Obsah přiloženého média **95**

Seznam obrázků

2.1 Hra kámen, nůžky, papír v normální formě. 8
2.2 Příklad hry v extenzivní formě. 8
2.3 Konceptuální model pro dynamické plánování. [4] 12
2.4 Interpretace domény klasického plánování jako přechodového systému. [4] 15
2.5 PFD procedura pro HTN plánování. 19
2.6 OpenRA verze hry Dune 2000 v rozlišení 1366 × 768 [18] 34
2.7 Originální Dune 2000 v rozlišení 640 × 400 [20] 39

5.1 Mapy používané při testování. 76
5.2 Grafy týkající se měření ekonomické stránky hry na velké mapě při porovnávání RBAI a Omnius. Legenda v grafu (a) platí i pro zbylé grafy. 78
5.3 Grafy týkající se měření vojenské stránky hry na velké mapě při porovnávání RBAI a Omnius. Legenda v grafu (a) platí i pro (b). 79
5.4 Grafy týkající se měření ekonomické stránky hry na velké mapě při porovnávání IBAI a Omnius. Legenda v grafu (a) platí i pro zbylé grafy. 80
5.5 Grafy týkající se měření vojenské stránky hry na velké mapě při porovnávání IBAI a Omnius. Legenda v grafu (a) platí i pro (b). 81
5.6 Grafy týkající se měření ekonomické stránky hry na velké mapě při porovnávání RBAI a IBAI. Legenda v grafu (a) platí i pro zbylé grafy. 82
5.7 Grafy týkající se měření vojenské stránky hry na velké mapě při porovnávání RBAI a IBAI. Legenda v grafu (a) platí i pro (b). 83
5.8 Grafy týkající se měření ekonomické stránky hry na malé mapě při porovnávání RBAI a IBAI. Legenda v grafu (a) platí i pro zbylé grafy. 84
5.9 Grafy týkající se měření vojenské stránky hry na malé mapě při porovnávání RBAI a IBAI. Legenda v grafu (a) platí i pro (b). 85
5.10 Grafy týkající se měření ekonomické stránky hry na malé mapě s velkými zdroji při porovnávání RBAI a IBAI. Legenda v grafu (a) platí i pro zbylé grafy. 87
5.11 Grafy týkající se měření vojenské stránky hry na malé mapě s velkými zdroji při porovnávání RBAI a IBAI. Legenda v grafu (a) platí i pro (b). 88

Seznam výpisů kódu

2.1 Ukázka axiomu systému SHOP2 (axiom použitý v novém AI pro Dune 2000). . 21
2.2 Ukázka operátoru systému SHOP2 (zjednodušená varianta operátoru použitého v nové AI pro Dune 2000). 22
2.3 Ukázka metody systému SHOP2 (upravená varianta metody použité v nové AI pro Dune 2000). 23
4.1 Axiomy RBAI 48
4.2 Externí funkce pro výpočet délky fronty po aplikaci operátoru čekání. 49
4.3 Metoda čekání v RBAI. 50
4.4 Příklad axiomu určujícího, zda byla nějaká budova postavena nebo nějaké vylepšení provedeno, operátoru stavby budov a metody stavby budov v RBAI. Konkrétně se jedná o axiom postavení elektrárny, operátor pro postavení kasáren a dvě metody pro postavení kasáren. 52
4.5 Příklad operátoru stavby obléhacího tanku, metody jeho stavby a metody stavby transportních letadel v RBAI. 54
4.6 Pomocné operátory v RBAI. 55
4.7 Metody a operátory pro stavbu armád v RBAI. (Poslední vypsaná metoda je zjednodušená pouze na pěchotu.) 56
4.8 Metody a axiomy pro stavbu armád v RBAI. (První metoda je zjednodušena pouze na pěchotu.) 59
4.9 Metoda pro stavbu armád v RBAI. 61
4.10 Metody pro útok v RBAI. 62
4.11 Podpůrné metody pro útok v RBAI. 63
4.12 Metoda a operátor pro útok v RBAI. 64
4.13 Jedna z větví určující strategii pro RBAI. 65
4.14 Operátory aktivace a deaktivace fronty stavby v IBAI. 71
4.15 Operátory aktivace a deaktivace fronty stavby v IBAI. 72

Rád bych poděkoval svému vedoucímu, prof. RNDr. Pavlu Surynkovi, Ph.D., za veškerou pomoc a vedení při tvorbě diplomové práce. Také bych chtěl poděkovat svému otci za pomoc s jazykovou korekturou této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. května 2023 .....

Abstrakt

Tato diplomová práce řeší hierarchické plánování pro real-time strategické hry. Cílem práce je vyzkoušet vhodnost aplikace HTN plánování pro počítačem řízeného hráče RTS hry tím, že bude HTN plánování použito ve hře Dune 2000.

V práci byly navrženy a implementovány dva mírně odlišné přístupy, pojmenované jako Resourced Based AI (RBAI) a Income Based AI (IBAI), modifikující původní umělou inteligenci jen do té míry, která je potřeba. První jmenovaný přístup modeluje fungování hry pro plánování přesněji. Druhý zmíněný testuje jednoduší přístup. Z experimentálního měření bylo zjištěno, že jsou oba přístupy funkční a ve většině případů poráží výchozí umělou inteligenci, ale nakonec se ukázalo HTN plánování zbytečně robustní a pracné pro zvolený přístup k vytvoření AI.

Klíčová slova plánování, HTN, strategické hry, RTS, AI, OpenRA, Dune 2000

Abstract

This thesis addresses hierarchical planning for real-time strategy games. The aim of the thesis is to test the suitability of applying HTN planning to a computer-controlled RTS game player by applying HTN planning to the game Dune 2000.

In this thesis, two slightly different approaches, named Resourced Based AI (RBAI) and Income Based AI (IBAI), have been proposed and implemented, modifying the original AI only to the extent needed. The former approach models the game for planning more accurately. The second mentioned tests a simpler approach. From experimental measurements, both approaches were found to work well and beat the default AI in most cases, but in the end, HTN planning proved unnecessarily robust and laborious for the chosen approach of creating AI.

Keywords planning, HTN, strategy games, RTS, AI, OpenRA, Dune 2000

Seznam zkratek

AI	Artificial Intelligence
HTN	Hierarchical Task Network
RBAI	Resourced Based AI
IBAI	Income Based AI
RTS	Real-time strategy

Úvod

Real-time strategické hry (RTS) patří mezi jeden ze známých dobře zavedených herních žánrů. Uživatelem a počítačem řízení hráči mezi sebou soupeří o to, kdo vyhraje. Samotní hráči mohou být vnímáni jako agenti s určitými cíli často obsahující poražení všech soupeřů. Jedním z hlavních aspektů hry je tedy boj, povětšinou reprezentovaný útočnými interakcemi mezi herními objekty nacházejícími se na herní mapě. Tato práce se zaměřuje na chování počítačem řízeného hráče, označovaném jako umělá inteligence (AI). Konkrétně na její strategická rozhodnutí. Přičemž k tomu bude využité HTN plánování.

Plánování, zastoupené mimo jiné HTN plánováním, je abstraktní proces uvažování, který vybírá a organizuje akce předvídáním jejich očekávaných výsledků. Snahou je co nejlépe dosáhnout předem stanovených cílů. Automatizované plánování je oblastí umělé inteligence, která tento proces uvažování studuje.

V této práci bude tento přístup aplikován na open source herní engine OpenRA umožnující tvorbu RTS her podobající se známé tvorbě od studia Westwood. Tři známé a vzájemně velice podobné hry byly kompletně vytvořeny přímo vývojáři OpenRA. Jedná se o:

- Command & Conquer: Tiberian Dawn;
- Command & Conquer: Red Alert;
- Dune 2000.

Právě poslední zmíněná hra bude využita pro testování vhodnosti využití HTN plánování pro chování AI.

Aplikace automatizovaného plánování na AI je motivována potřebou strategického rozhodování při hraní RTS her. Tyto strategie se mohou týkat například stavby nových herních objektů nebo způsobů využívaní zdrojů.

Kapitola 1
Cíl práce

Cílem práce jako celku je vyzkoušet vhodnost aplikace HTN plánování pro RTS hry. Cílem teoretické části práce je pak vysvětlit potřebné pojmy týkající se agentů, automatizovaného plánování a hry Dune 2000. Dalším cílem je odůvodnit použití HTN plánování.

Cílem kapitoly plánování v RTS je zanalyzovat hru Dune 2000 použitou v práci a způsob aplikace plánování pro OpenRA hry. Cílem praktické části práce je návrh nové herní AI využívající HTN plánování ke hraní hry. Dalším cílem praktické části práce je implementace navržené nové AI. V neposlední řadě je cílem experimentálně otestovat novou AI a porovnat ji s již existující výchozí AI. Nakonec výsledky zanalyzovat.

Kapitola 2
Teoretická východiska

Tato část bude zaměřena na popsání, co jsou agenti, jaké jsou jejich typy a další přidružené pojmy, jako je pracovní prostředí a agentní funkce. Následně bude rozebrána vhodnost teorie her pro RTS a alternativní přístup využívající automatizované plánování. Pak bude vysvětleno klasické plánování s pomocí konceptuálního modelu. Poté bude rozebráno plánování pomocí hierarchických sítí úloh (HTN) spolu s konkrétním plánovacím systémem JSHOP2. Nakonec bude v této kapitole část věnující se real-time strategickým hrám (RTS) obsahující základní popis RTS her, analýzu cílů z hlediska uživatele, herní engine OpenRA a popis hry Dune 2000.

2.1 Herní agenti

Agent je cokoli, co dokáže vnímat prostředí pomocí svých senzorů a působit na toto prostředí prostřednictvím svých efektorů. [1]

Například lidský agent má oči, uši a další orgány jako senzory a ruce, nohy, hlasové ústrojí a tak dále jako efektory. Robotický agent může mít kamery jako senzory a různé motory pohybující nohami, koly nebo pásy jako efektory. Softwarový agent přijímá stisky kláves, čte obsah souborů a síťové pakety jako senzorické vstupy a působí na prostředí zobrazováním na obrazovce, zapisováním do souborů a odesíláním síťových paketů. Herní agent vnímá prostředí z dostupných dat ve hře jako senzorické vstupy a působí na prostředí ovládáním herních objektů. [1]

Problém agentních systémů lze specifikovat pomocí pracovního prostředí, anglicky task environment. To se skládá ze čtyř částí: [1]

Metriky úspěšnosti určují, co je pro agenta podstatné při působení v prostředí. Například rychlost a bezpečnost při transportu pasažéra nebo počet zničených nepřátel v počítačové hře.

Prostředí definuje, kde agent může působit a nacházet se. Příkladem může být silniční síť nebo herní mapa počítačové hry.

Efektory jsou prostředky, kterými agent může působit na prostředí. Například se může jednat o nohy, kola nebo dávání rozkazů herním objektům.

Senzory jsou prostředky, kterými agent vnímá své okolí. Příkladem můžou být kamery, radary nebo možnosti číst data z nějakého příslušného zdroje.

Pro různá pracovní prostředí jsou důležité různé vlastnosti agentů. Tedy návrh agenta musí nutně zohledňovat specifikace pracovního prostředí. [1]

2.1.1 Typy prostředí

Rozsah pracovních prostředí, která mohou v AI nastat, je samozřejmě obrovský. Lze však identifikovat poměrně malý počet dimenzí, pomocí kterých lze kategorizovat pracovní prostředí. Tyto dimenze do značné míry určují vhodný návrh agenta. Následuje stručný popis některých nejdůležitějších charakteristik pracovního prostředí: [1]

Plně pozorovatelné vs. částečně pozorovatelné Pracovní prostředí je plně pozorovatelné, pokud jsou vždy dostupné kompletní informace o stavu prostředí; jinak je částečně pozorovatelné.

Jeden agent vs. multiagentní Pracovní prostředí se považuje za multiagentní, pokud je vhodné v prostředí rozpoznat určité objekty jako jiné agenty; jinak se jedná o pracovní prostředí s jedním agentem. V případě, že se jedná o multiagentní prostředí, kde agenti soupeří, tak se v určitých případech problémem zabývá teorie her (ta je popsaná v sekci 2.2).

Deterministické vs. stochastické Pracovní prostředí je deterministické, pokud je další stav prostředí zcela určen aktuálním stavem a akcí provedenou agentem; jinak je stochastické.

Epizodické vs. sekvenční Pracovní prostředí je epizodické, pokud lze rozdělit opakované vnímání a následné konání agenta do samostatných na sobě nezávislých atomických epizod, tedy v každé epizodě agent obdrží vjem a poté provede jedinou akci, přičemž každá další epizoda nezávisí na akcích provedených v předchozích epizodách. Jinak je prostředí sekvenční.

Statické vs. dynamické Pracovní prostředí je dynamické, pokud se může změnit působením procesů, které agent nemá pod svou kontrolou; jinak je statické.

Diskrétní vs. spojité Pracovní prostředí je diskrétní, pokud jsou stavy prostředí a akce konečné množiny a čas lze vnímat diskrétně; jinak je prostředí spojité.

Známé vs. neznámé Pracovní prostředí je známé, pokud je známé, jak dané prostředí přesně funguje; jinak je neznámé.

2.1.2 Agentní funkce

Běžným cílem v oboru umělé inteligence je vytvoření programu agenta, který implementuje agentní funkci, ta mapuje vnímané sekvence na akce. Tento program nejspíše poběží na nějakém druhu výpočetního zařízení se senzory a efektory, tomu se říká architektura. Agent se skládá z konkrétního programu agenta a využité architektury. [1]

2.1.3 Typy agentů

Lze rozlišit čtyři úrovně agentů z hlediska jejich komplexnosti: [1]

1. Jednoduchý reflexní agent je nejjednodušší typ agenta. Agent vybírá akci podle aktuálních vjemů (nebere v potaz historii). Jeho chování je pevně definované sadou pravidel určujících zvolenou akci podle aktuálních vjemů (někdy nazývané jako „if-then“ pravidla).

2. Reflexní agent se stavem si na rozdíl od jednoduchého reflexního agenta ukládá aktuální stav světa. Akci vybírá stejným způsobem, tedy pomocí sady pravidel, ale v jeho případě volí akci podle uloženého stavu světa.

3. Agent s cílem má zadaný cíl, který má splnit. Akce nevolí podle pevné sady pravidel, ale využívá technik jako je prohledávání a plánování k nalezení sekvencí akcí, které vedou k zadanému cíli.

4. Agent s utilitní funkcí využívá pro určení kvality stavu utilitní funkci. Ta mapuje stav na číslo, které určuje jeho kvalitu (povětšinou platí, že čím vyšší, tím lepší). Díky tomu agent může řešit konfliktní cíle (jako je například rychlost a cena řešení).

2.1.4 Herní agenti v RTS

Jak bylo v úvodu zmíněno, tak v real-time strategických hrách spolu soupeří uživatelem a počítačem řízení hráči. Každý tento hráč může být vnímán jako agent s cílem nebo utilitní funkcí.

Podle představených specifikací může být pracovní prostředí většiny real-time strategických her definované následujícím způsobem:

- částečné pozorovatelné – real-time strategické hry často využívají mechaniky neprozkoumaného prostředí a mlhy války;
- multiagentní – různí hráči mohou být vnímáni jako různí agenti s vlastními cíli;
- stochastické – hry často obsahují mechaniky využívající náhody;
- sekvenční – pracovní prostředí nelze rozdělit na sobě nezávislé epizody;
- dynamické – ve hře můžou probíhat procesy, které agent nemá pod svou kontrolou;
- spojité – pracovní prostředí RTS her většinou nelze považovat za diskrétní, protože některé vlastnosti mohou být reprezentované reálným číslem (to má v počítačové reprezentaci sice limitovanou přesnost, ale ve výsledku by bylo stavů takové množství, že vnímat tak pracovní prostředí je nevhodné);
- známé – zde záleží, zda je kompletně známé, jak hra přesně funguje, pokud jsou přístupné zdrojové kódy hry, tak je to možné zjistit.

Vzhledem k tomu, že se jedná o pracovní prostředí, kde je více agentů a ti mezi sebou soupeří, tak se nabízí zaměřit se na teorii her.

2.2 Teorie her

Teorie her je matematická disciplína zabývající se situacemi, kde utilitní funkce agentů jsou v konfliktu. Účelem vytvořených modelů her je přesné definování hry jako problému a jejich následná analýza s cílem pochopení problému a nalezení nejlepších strategií pro jednotlivé hráče. [2]

V teorii her jsou hry formálně definované pojmy. Každá hra je definovaná hráči, jejich možnými akcemi a jejich utilitními funkcemi (ty určují, jak dobré jsou stavy po zvolení příslušných akcí). Nejběžnější formy reprezentace hry jsou následující dva. [2]

2.2.1 Hra v normální formě

Hra je definovaná maticí, která určuje hráče, jejich možné akce a hodnoty utilitních funkcí pro daný stav. Na obrázku 2.1 je příklad hry v normální formě, jedná se o hru kámen, nůžky, papír, kde jsou dva hráči, přičemž každý z nich má tři možnosti akcí. První hráč vybírá řádek, druhý vybírá sloupec. Hodnota utilitní funkce každého hráče pro daný stav je zapsaná uvnitř matice, první číslo patří prvnímu hráči a druhé číslo patří druhému hráči. [2]

Pro hry v normální formě platí, že buď všichni hráči vybírají svoje akce zároveň, nebo nevědí, které akce zvolili soupeři. [2]

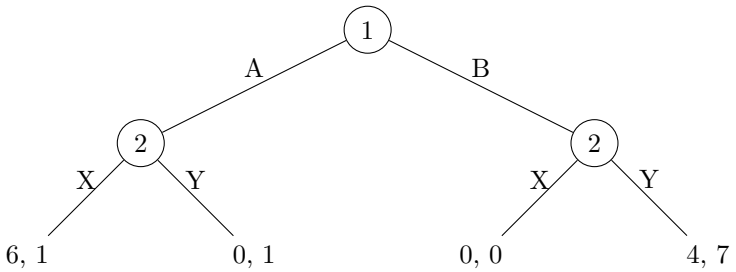
	kámen	nůžky	papír
kámen	0, 0	1, -1	-1, 1
nůžky	-1, 1	0, 0	1, -1
papír	1, -1	-1, 1	0, 0

Obrázek 2.1 Hra kámen, nůžky, papír v normální formě.

2.2.2 Hra v extenzivní formě

Hry v extenzivní formě lze využít k formalizaci her, kde záleží na pořadí akcí. Pro vizualizace se používají stromy. Vrcholy reprezentují místa, kde hráči vybírají své akce, přičemž obsah vrcholu určuje, který hráč akci vybírá. Hrany určují možné tahy hráčů. V listech stromu jsou zapsány hodnoty utilitních funkcí všech hráčů. [2]

Na obrázku 2.2 je vymyšlený příklad hry v extenzivní formě. Hráč 1 vybírá jako první a má možnosti A a B, jako druhý volí akci hráč 2 a ten může zvolit X nebo Y. Například pokud hráč 1 vybere A a hráč 2 zvolí Y, pak první hráč získá hodnotu 0 a druhý 1.



Obrázek 2.2 Příklad hry v extenzivní formě.

2.2.3 Teorie her pro RTS

Jak bylo v úvodu nastíněno, tak v real-time strategických hrách hráči ovládají relativně velké množství jednotek a budov na herní mapě. V případě her v OpenRA se jedná o mapu s čtvereč-kovanou mřížkou.

Vzhledem k tomu, že velikost středně velkých čtvercových map může být například 64 herních polí a většina je pro jednotky přístupná, tak jen při jediné jednotce by větvící faktor této hry modelované v extenzivní formě byl blízký 4096. (Použití normální formy není vhodné, protože hra má velké množství na sobě závislých rozhodnutí.) Zároveň platí, že při standardní rychlosti se hra 25krát za sekundu aktualizuje. Pokud by tedy každá aktualizace byla vnímána jako volba akce jedním hráčem, pak jen pro rozhodnutí na minutu dopředu by hloubka stromu byla 1500 za jednoho hráče.

Ze zmíněných důvodů je tedy zřejmé, že formulovat hru exaktním způsobem je výpočetně nezvladatelné. Nabízí se využít nějaké abstrakce a jako akce modelovat jen určitá strategická rozhodnutí. Těmito rozhodnutími by mohli být například volby stavby budov a jednotek nebo kdy vyslat jednotky do útoku, ale konkrétní ovládání jednotek na mapě už nikoliv.

Tímto způsobem se sice zredukuje větvení, ale pořád je modelování hry v extenzivní formě nevhodné, a to hned z několika důvodů.

1. Přetrvává problém s hloubkou stromu, kvůli velkému množství možností, kdy volit akce. Tento problém je řešitelný zredukováním množství příležitostí volby akcí, například na jednu za sekundu.

2. Hráč má možnost provádět libovolné množství akcí najednou (například poslat jednotky do útoku, dát stavět nějaké nové jednotky atd.). Řešením by v tomto případě bylo vytvoření složených akcí, které jsou všemi možnými kombinacemi původních akcí. Ve výsledku by ale akcí bylo příliš velké množství. Lepším řešením by nejspíše bylo nevyužít této možnosti a modelovat pouze případ, kdy hráč může dělat jen jednu z definovaných akcí najednou.

3. Hry v OpenRA implementují mechaniku mlhy války, která znemožňuje hráči vidět, co dělají soupeři. To znamená, že značnou část hry hráč nemůže volit své akce podle soupeřových. Hra se v tomto případě více podobá hrám v normální formě.

Ve výsledku modelování RTS her pomocí teorie her není příliš vhodné a v práci je zvolen alternativní přístup vysvětlený v následující sekci 2.3.

2.3 Dynamické prostředí

Na konci sekce týkající se herních agentů (2.1) bylo v podsekci nazvané „Herní agenti v RTS“ (2.1.4) řečeno, že vzhledem k tomu, že se jedná o mutiagentní problém, kde agenti soupeří, tak je vhodná teorie her. Existuje ale alternativní způsob, jak situaci vnímat, a to jako problém s jediným agentem v dynamickém prostředí (tedy soupeře vnímat jen jako součást dynamického prostředí; nikoliv jako jiné agenty). Vzhledem k tomu, že hráči nemají moc informací o chování soupeřů, tak není tento předpoklad příliš problematický (jedná se obecně o vhodný způsob řešení problému částečné pozorovatelnosti pracovního prostředí).

V tomto případě lze jako součást programu agenta použít automatizované plánování. To je schopné nalézt sekvence akcí vedoucí ke zvolenému cíli i v případech, kdy některé akce mají různé požadavky (stavba většiny budov je podmíněna postavením jiných budov). Vzhledem k tomu, že jsou soupeři považováni jen za součást dynamického prostředí, bude nutné více dbát na kontrolu a případnou opravu plánu například pomocí kompletního přeplánování nebo opravy plánu.

2.4 Klasické plánování

Plánování je abstraktní proces uvažování, který vybírá a organizuje akce předvídáním jejich očekávaných výsledků. Snahou je co nejlépe dosáhnout předem stanovených cílů. Automatizované plánování je oblastí umělé inteligence (AI), která tento proces uvažování studuje. [3, 4]

Tato kapitola klasického plánování bude vycházet, co se týče definice a představení problému plánování, především z [4], ale původním zdrojem pro plánování a jazyk, definující řešené problémy, je program STRIPS (STanford Research Institute Problem Solver) [3].

Praktickou motivací je potřeba rychlého a spolehlivého informačního systému schopného vytvářet plány automatizovaně. Příkladem může být nutnost řešit problémy zahrnující bezpečnostní rizika, které je nutné řešit rychle a efektivně, jako jsou záchranné operace po živelných katastrofách. Takovéto situace vyžadují precizní nasazení a koordinaci velkého množství účastníků. V těchto případech je rychlost a spolehlivost nutnou součástí plánovacího systému. [4]

Existuje celá řada specifických druhů plánování, například: [4]

- Plánování cesty a pohybu se věnuje vytváření konkrétních sekvencí pohybů vycházejících z počáteční pozice do cíle při určité dané specifikaci prostoru a možností pohybu.

- Plánování vnímání se věnuje sbírání informací. Cílem je vyřešit otázky týkající se kdy, jak a čím zjistit potřebné informace o aktuální situaci.

- Navigační plánování je kombinace předchozích dvou typů s cílem přesunu agenta s nutností prozkoumání terénu. Příkladem může být automatizovaný pohyb po silnici robotem využívajícím vizuální senzory.

- Plánování manipulace se věnuje manipulaci předmětů při výrobě. Jedná se o akce vyžadující detekci a manipulaci předmětů.

- Plánování komunikace se věnuje nutnosti kooperace mezi agenty. Cílem je vyřešit otázky týkající se toho, kdy a kde získat nutné informace.

Přirozeným řešením zmíněných značně odlišných plánovacích problémů je vytvoření konkrétního doménově specifického systému pro daný problém. Výhodou je možnost využít tyto doménové specifikace k vytvoření efektivního systému řešícího daný problém. Nevýhodou je náročnost a cena vytváření těchto systémů. [4]

Z tohoto důvodu je vhodné využít doménově nezávislé automatizované plánování. Při řešení problému je vstupem popis domény a konkrétní problém. [4]

Doménově nezávislé přístupy závisí na modelech akcí. Tyto modely sahají od jednoduchých, které umožňují pouze omezené formy uvažování, až po modely s bohatšími prediktivními schopnostmi. Jedná se zejména o následující formy modelů a plánovacích schopností. [4]

- Plánování projektů, ve kterém jsou modely akcí redukovány hlavně na časová a prioritní omezení, například nejdřívější a nejpozdější čas zahájení akce nebo její časová závislost vůči jiné akci. Plánování projektu se používá pro interaktivní editaci a ověřování plánu.

- Rozvrhování a alokace zdrojů, ve kterých modely akcí zahrnují výše uvedené typy omezení rozšířené o omezení zdrojů, které mají být použity každou akcí.

- Syntéza plánů, ve které jsou předchozí zmíněné akční modely rozšířené o podmínky nutné pro využití akce a účinky akce na stav světa.

Automatizované plánování se zaměřuje na syntézu plánů. [4]

2.4.1 Konceptuální model plánování

Konceptuální model je jednoduchý teoretický prostředek pro popis hlavních prvků problému automatizovaného plánování. [4]

Protože plánování se zabývá výběrem a organizováním akcí měnících stav systému, jako konceptuální model plánování je nutný obecný model pro dynamický systém. Tím je přechodový systém, formálně se jedná o čtveřici $\Sigma = (S, A, E, \gamma)$, kde [4]

- $S = \{s_1, s_2, \ldots, s_n\}$ je konečná nebo rekurzivně spočetná množina stavů;

- $A = \{a_1, a_2, \ldots, a_n\}$ je konečná nebo rekurzivně spočetná množina akcí;

- $E = \{e_1, e_2, \ldots, e_n\}$ je konečná nebo rekurzivně spočetná množina událostí;

- $\gamma : S \times A \times E \to 2^S$ je stavová přechodová funkce.

Přechodový systém může být reprezentován orientovaným grafem, jehož vrcholy jsou stavy v S. Jestliže $s' \in \gamma(s, a, e)$, kde $s, s' \in S$, $a \in A$ a $e \in E$, pak graf obsahuje hranu z s do s' označenou pomocí (a, e). Tyto hrany reprezentují přechody mezi stavy. Pro zjednodušení může být zavedena neutrální událost ϵ pro situace, kdy přechod je způsoben pouze akcí, a neutrální akce *no-op* pro situace, kdy přechod je způsoben pouze událostí. Zápis $\gamma(s, a, \epsilon)$ bude zjednodušován na $\gamma(s, a)$ a zápis $\gamma(s, \textit{no-op}, e)$ bude zjednodušován na $\gamma(s, e)$. [4]

Události i akce mění stav systému s tím rozdílem, zda nad nimi má plánovač kontrolu. Akce jsou přechody, nad kterými plánovač má kontrolu. Jestliže $a \in A$ a $\gamma(s, a) \neq \emptyset$, pak akci a lze aplikovat ve stavu s, čímž se systém přesune ze stavu s do nějakého stavu v $\gamma(s, a)$. Události jsou přechody, které jsou nahodilé. Místo toho, aby byly řízeny plánovačem, tak nastávají podle vnitřní dynamiky systému. Měly by být zohledněny při plánování, ale nelze je vynutit. Jestliže

$e \in E$ a $\gamma(s, e) \neq \emptyset$, pak e může nastat v s a přesunout systém ze stavu s do nějakého stavu z $\gamma(s, e)$. [4]

Co se týče přechodové funkce, tak je možné model dále specifikovat následujícími možnostmi: [4]

- Model, ve kterém neexistuje žádná událost přecházející ze stavu, ze kterého je nějaká akce, a naopak. Množinu stavů lze tedy rozdělit na stavy s událostmi a stavy s akcemi.
- Model, ve kterém z jednoho stavu může být jak akce, tak událost. To znamená, že pokud aplikujeme akci a ve stavu s a $\gamma(s, e) \neq \emptyset$, pak výsledným stavem může být kterýkoli stav z $\gamma(s, a, e)$.

Cílem plánování je nalézt, které akce zvolit v konkrétních stavech, aby se dosáhlo zadaného cíle z počátečního stavu. Plán je struktura, která obsahuje tyto akce. Cíle mohou být specifikovány několika způsoby:

- Nejjednodušší specifikaci tvoří cílový stav s_g nebo množina cílových stavů S_g. V tomto případě je cíle dosaženo jakoukoliv sekvencí stavových přechodů, která končí v jednom z cílových stavů.
- Obecněji může být cílem splnit určitou podmínku posloupnosti stavů, které systém prochází. Například může být požadováno určit stavy, které systém nesmí projít, stavy, které mají být systémem v určitém bodě dosaženy, a stavy, ve kterých by měl systém zůstat.
- Alternativní specifikace je s pomocnou funkcí oceňující stavy hodnotami. Cílem pak je optimalizovat nějakou funkci těchto hodnot (například minimalizaci jejich součtu) ve všech stavech, které systém prošel.
- Další alternativou je specifikovat cíl jako úlohy, které má systém provést. Tyto úlohy lze rekurzivně definovat jako množiny akcí a dalších úloh. Tato varianta bude podrobněji probrána v kapitole 2.5.

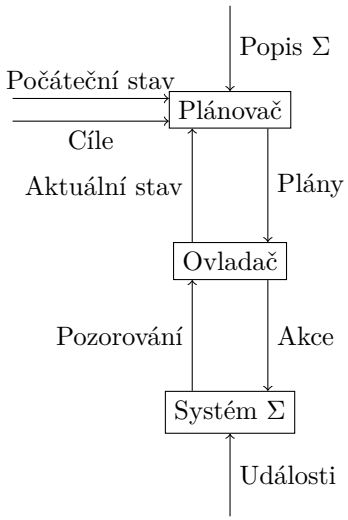
Tento model je vhodné znázornit prostřednictvím interakce mezi třemi komponentami na obrázku 2.3 s následujícím popisem: [4]

1. Přechodový systém Σ se vyvíjí podle stavové přechodové funkce γ a obdržených událostí a akcí.
2. Ovladač poskytuje jako výstup akce podle aktuálního stavu systému Σ a plánovačem poskytnutého plánu.
3. Plánovač, kterému je jako vstup poskytnut popis systému Σ, výchozí stav a nějaký cíl, vytváří plán pro ovladač za účelem dosažení cíle.

V určitých nasazeních automatizovaného plánování má ovladač limitovanou znalost aktuálního stavu systému Σ. Částečnou znalost stavu systému lze modelovat pozorovací funkcí $\eta : S \to O$, která mapuje S na nějakou množinu $O = \{o_1, o_2, \ldots, o_n\}$ možných pozorování. Vstupem ovladače je pak pozorování $o = \eta(s)$ odpovídající aktuálnímu stavu. [4]

Většinou existují rozdíly mezi fyzickým systémem, který má být řízen, a jeho modelem, který je popsaný v Σ. Obecně je plánování omezeno na model formálně popsaný systémem Σ. Ovladač musí být z toho důvodu dostatečně robustní, aby se vyrovnal s rozdíly mezi systémem Σ a reálným světem. Vypořádávání se s pozorováními, která se odchylují od toho, co se očekává v plánu, vyžadují složitější kontrolní mechanismy, než jaké jsou potřeba pro pouhé sledování stavu a aplikaci odpovídajících akcí. Z tohoto důvodu je běžně potřeba prokládat plánování a jednání dohlížením nad plánem, revizí plánů a mechanismy přeplánování. Toto je v obrázku 2.3 znázorněno šipkou z ovladače do plánovače, která tak tvoří mezi těmito prvky smyčku. Výsledkem je dynamické plánování. [4]

Neočekává se, že tento konceptuální model bude v této formě funkční, ale je považován za výchozí bod pro posouzení různých restriktivních předpokladů, zejména následujících: [4]



Obrázek 2.3 Konceptuální model pro dynamické plánování. [4]

1. Systém Σ má konečnou množinu stavů.
2. Systém Σ je plně pozorovatelný, tj. ovladač má přesné znalosti o stavu Σ. V tomto případě je pozorovací funkce η identitou.
3. Systém Σ je deterministický, tj. pro každý stav s a pro každou událost u platí, že $|\gamma(s,u)| \leq 1$. Jestliže je tedy akce aplikovatelná na stav, tak její aplikace přináší deterministický systém do jednoho jiného stavu.
4. Systém Σ je statický, tj. množina událostí E je prázdná. Σ nemá žádnou vnitřní dynamiku.
5. Plánovač zpracovává pouze omezené cíle, které jsou specifikovány jako jeden cílový stav s_g nebo množina cílových stavů S_g.
6. Plán řešící plánovací problém je lineárně uspořádaná konečná posloupnost akcí.
7. Akce a události nemají trvání. Jsou to okamžité, stavové přechody. Tyto předpoklady vychází přímo z přechodového systému, který čas explicitně nemodeluje.
8. Plánovač se nezabývá žádnou změnou, která může nastat v Σ během plánování. Plánuje pro daný počáteční a cílový stav.

V nejjednodušším případě, nazývaném jako restriktivní model (nebo restriktivní přechodový systém), je zkombinovaných všech osm restriktivních předpokladů. V takovém případě se problém redukuje na trojici $\Sigma = (S, A, \gamma)$, počáteční stav s_0 a množinu cílových stavů S_g, pro které má plánovač nalézt posloupnost akcí $\langle a_1, a_2, \ldots, a_k \rangle$ odpovídající posloupnosti stavových přechodů $(s_0, s_1, \ldots, s_k)$ takových, že $s_1 \in \gamma(s_0, a_1), s_2 \in \gamma(s_1, a_2), \ldots, s_k \in \gamma(s_{k-1}, a_k)$, a $s_k \in S_g$. [4]

Protože je zmíněný systém deterministický, tak jestliže je γ aplikovatelné ve stavu s, pak $\gamma(s,a)$ obsahuje pouze jeden stav s'. Pro zjednodušení zápisu bude používáno $\gamma(s,a) = s'$ spíše než $\gamma(s,a) = \{s'\}$. V tomto restriktivním systému je plánem posloupnost $\langle a_1, a_2, \ldots, a_k \rangle$ taková, že $\gamma(\gamma(\ldots\gamma(\gamma(s_0, a_1), a_2), \ldots, a_{k-1}), a_k)$ je cílový stav. [4]

Problém v případě restriktivního modelu se zjednodušil na pouhé nalezení nejkratší cesty v grafu, což je známý a dobře vyřešený problém. Ve skutečnosti, pokud je graf Σ dostupný explicitně, pak skutečně není potřeba nic víc. Ale i pro velmi jednoduchou aplikační doménu může být graf tak velký, že jeho explicitní vyjádření není možné. [4]

2.4.2 Reprezentace klasického plánování

Nezbytnou součástí vstupu každého plánovacího algoritmu je popis řešeného problému. V praxi obvykle není nemožné, aby tento popis problému obsahoval explicitní výčet všech možných stavů a přechodů mezi nimi. Takový popis problému by byl mimořádně rozsáhlý a jeho generování by obvykle vyžadovalo více práce než řešení daného plánovacího problému. Místo toho je zapotřebí reprezentace problému, která explicitně nevyjmenovává stavy a přechody mezi nimi, ale usnadňuje jejich výpočet za běhu. [4]

V následujícím seznamu budou zmíněny tři způsoby reprezentace problému klasického plánování, přičemž dále v této podsekci bude podrobněji vysvětlena pouze ta druhá. Každá z nich je ekvivalentní ve vyjadřovací síle v tom smyslu, že plánovací doménu reprezentovanou v jedné z nich lze také reprezentovat pomocí kterékoli z ostatních. Následuje jejich stručný popis: [4]

1. Reprezentace teorií množin má každý stav světa vyjádřený množinou výroků a každá akce je syntaktickým výrazem určujícím, které výroky musejí být ve stavu, aby byla akce aplikovatelná, a které výroky akce přidá nebo odstraní při vytváření nového stavu.

2. V klasické reprezentaci jsou stavy a akce podobné těm, které jsou popsány u reprezentace teorií množin, s tím rozdílem, že namísto výroků se používají literály predikátové logiky prvního řádu a logické spojky.

3. V reprezentaci stavovými proměnnými je každý stav reprezentován n-ticí hodnot stavových proměnných $\{x_1, x_2, \ldots, x_n\}$ a každá akce je reprezentována funkcí, která mapuje tuto n-tici do nějaké jiné n-tice hodnot n stavových proměnných.

2.4.2.1 Stavy

Jazyk pro klasické plánování bude vycházet z jazyka predikátové logiky prvního řádu $\mathcal{L}$, ve kterém je ale konečně mnoho predikátových a konstantních symbolů a žádné funkční symboly. Každý term jazyka $\mathcal{L}$ je tedy buď proměnná nebo konstantní symbol. Později bude rozšířen o nějaké další symboly a výrazy. [4]

▶ **Definice 2.1.** *Stav je množina plně určených atomických formulí jazyka $\mathcal{L}$ (tj. atomických formulí, kde nejsou žádné proměnné). Protože $\mathcal{L}$ nemá žádné funkční symboly, množina S všech možných stavů je zaručeně konečná. Atomická formule p platí v s jen tehdy, když $p \in s$. Jestliže je g množina literálů (pro logiku prvního řádu definované jako atomické formule a negované atomické formule), tak bude používáno, že s splňuje g (označené jako $s \models g$), pokud existuje substituce σ taková, že každý kladný literál ze $\sigma(g)$ je v s a žádný negovaný literál ze $\sigma(g)$ není v s. [4]*

V klasickém plánování je používán předpoklad uzavřeného světa (anglicky „Closed-world assumption“). To znamená, že atomická formule, která není výslovně v konkrétním stavu specifikovaná, v tomto stavu neplatí. [4]

2.4.2.2 Operátory a akce

Přechodová funkce γ je specifikována prostřednictvím množiny plánovacích operátorů, které jsou konkretizovány do akcí. [4]

▶ **Definice 2.2.** *V klasickém plánování je plánovacím operátorem uspořádaná trojice o = (name(o), precond(o), effects(o)), jejíž prvky jsou následující: [4]*

- *name(o), jméno operátoru je výraz ve tvaru $n(x_1, x_2, \ldots, x_k)$, kde n je nazývaný symbol operátoru, $x_1, x_2, \ldots, x_k$ jsou všechno proměnné, které se vyskytují kdekoli v o, a n je unikátní (tj. žádné dva operátory v $\mathcal{L}$ nemají stejný symbol operátoru).*

- *precond(o) a effects(o), předpoklady a efekty operátoru o jsou množiny literálů.*

▶ **Definice 2.3.** *Pro libovolnou množinu literálů L je L^+ množina všech atomických formulí v L a L^- je množina všech atomických formulí, jejichž negace jsou v L. Konkrétně, pokud o je operátor nebo instance operátoru, pak $precond^+(o)$ a $precond^-(o)$ jsou pozitivní a negativní předpoklady o a $effects^+(o)$ a $effects^-(o)$ jsou pozitivní a negativní efekty o. [4]*

▶ **Definice 2.4.** *Akce je jakákoli plně určená instance operátoru plánování. Jestliže a je akce a s je stav takový, že $precond^+(a) \subseteq s$ a $precond^-(a) \cap s = \emptyset$, pak akce a platí pro s a výsledkem aplikace a na s je stav: [4]*

$$\gamma(s, a) = (s - effects^-(a)) \cup effects^+(a)$$

2.4.2.3 Domény, plány, problémy a řešení

▶ **Definice 2.5.** *Nechť $\mathcal{L}$ je jazyk prvního řádu, který má konečně mnoho predikátových a konstantních symbolů. Doména klasického plánování v $\mathcal{L}$ je restriktivní model $\Sigma = (S, A, \gamma)$ takový, že: [4]*

- $S \subseteq 2^{\{\text{všechny plně určené atomické formule v } L\}}$;
- $A = \{$*všechny plně určené instance operátorů v* $O\}$, *kde O je množina všech operátorů;*
- $\gamma(s, a) = (s - effects^-(a)) \cup effects^+(a)$, *pokud $a \in A$ je aplikovatelné v $s \in S$, jinak $\gamma(s, a)$ není definované;*
- *S je uzavřené pro γ (tj. pokud $s \in S$, tak pro každou akci a aplikovatelnou v s platí $\gamma(s, a) \in S$).*

▶ **Definice 2.6.** *Problém klasického plánování je trojicí $\mathcal{P} = (\Sigma, s_0, g)$, kde: [4]*

- *s_0, počáteční stav, je kterýkoliv stav v množině S;*
- *g, cíl, je jakákoliv množina kompletně určených literálů;*
- *$S_g = \{s \in S | s$ splňuje $g\}$.*

Ustanovení plánovacího problému $\mathcal{P} = (\Sigma, s_0, g)$ je $P = (O, s_0, g)$.

Ustanovení plánovacího problému je syntaktická specifikace, která by šla použít například k popisu $\mathcal{P}$ počítačovému programu. Také platí, že pro více problémů klasického plánování může být stejné ustanovení. Ale v takovém případě, pokud dva problémy klasického plánování mají stejné ustanovení, pak budou mít stejnou množinu dosažitelných stavů a stejnou množinu řešení. [4]

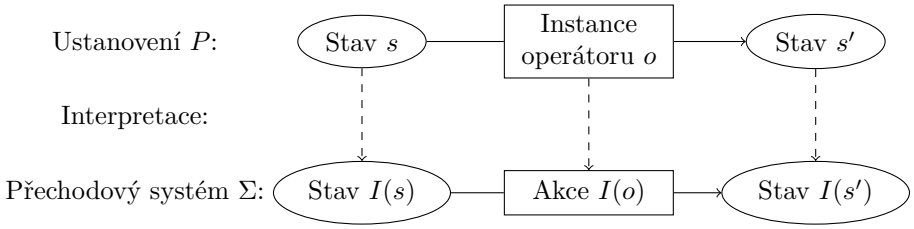
▶ **Definice 2.7.** *Plán je libovolná posloupnost akcí $\pi = \langle a_1, a_2, \ldots, a_k \rangle$, kde $k \geq 0$. Délka plánu je $|\pi| = k$, tedy počet akcí. [4]*

Stav vytvořený aplikováním π na stav s je stav, který je vytvořen aplikováním akcí π v daném pořadí. Toto bude popsáno rozšířením stavové přechodové funkce γ následovně: [4]

$$\gamma(s, \pi) = \begin{cases} s & \text{pokud } k = 0 \ (\pi \text{ je prázdné}) \\ \gamma(\gamma(s, a_1), \langle a_2, \ldots, a_k \rangle) & \text{pokud } k > 0 \text{ a } a_1 \text{ je aplikovatelné v } s \\ \text{nedefinované} & \text{jinak} \end{cases}$$

▶ **Definice 2.8.** *Nechť $\mathcal{P} = (\Sigma, s_0, g)$ je plánovací problém. Plán π je řešením pro $\mathcal{P}$, pokud $g \subseteq \gamma(s_0, \pi)$. Řešení π je redundantní, pokud existuje podposloupnost π, která je také řešením pro $\mathcal{P}$; π je minimální, pokud žádné jiné řešení pro $\mathcal{P}$ neobsahuje méně akcí než π. [4]*

Z předchozí definice vyplývá, že minimální řešení nemůže být redundantní. [4]



Obrázek 2.4 Interpretace domény klasického plánování jako přechodového systému. [4]

2.4.2.4 Sémantika a syntaxe

Dosud byl stírán rozdíl mezi syntaktickou specifikací klasického plánovacího problému a tím, co znamená. Intuitivně je sémantika dána Σ a syntaxe je specifikována pomocí P. [4]

Předpokládejme, že $P = (O, s_0, g)$ je ustanovením klasického plánovacího problému $\mathcal{P}$. Nechť Σ je restriktivní přechodový systém pro $\mathcal{P}$ a I je funkce, nazývaná interpretace, taková, že (viz obrázek 2.4): [4]

- Pro každý stav t ustanovení P je $I(t)$ stavem Σ.
- Pro každou instanci operátoru o z P je $I(o)$ akcí Σ.

Pro specifikaci sémantiky bude definován pár (Σ, I), označovaný jako model daného ustanovení P, jestliže pro každý stav t ustanovení P a každou instanci operátoru o ustanovení P platí: [4]

$$\gamma(I(s), I(o)) = I((s - \text{effects}^-(o)) \cup \text{effects}^+(o))$$

2.4.2.5 Rozšíření reprezentace klasického plánování

Vzhledem k tomu, že formalismus klasického plánování je velmi omezený, jsou k popisu složitějších domén potřeba jeho rozšíření. Například se může jednat o následující: [4]

- Rozšíření umožňující použití typů proměnných. Ustanovení problému může obsahovat kromě O, s_0, a g také typy proměnných definovaných množinami konstant a jménem typu.
- Rozšíření pomocí podmíněných efektů operátorů. Efekty operátorů neobsahují přímo literály, ale dvojice předpokladů a efektů. Efekty jsou pak aplikované na stav pouze tehdy, pokud je předpoklad splněn.
- Rozšíření umožňující použít kvantifikované výrazy. Například v efektu operátoru může být kromě literálů také logická formule, ve které jsou literály vyjadřující předpoklady a literály vyjadřující efekt, pokud nějaké konstanty tyto předpoklady splňují.
- Dalším rozšířením můžou být disjunktivní předpoklady. V operátoru může být v předpokladech použita disjunkce.
- Rozšíření přidávající axiomy, které umožňují odvodit předpoklady, které nejsou přímo součástí stavu světa.
- Rozšíření přidávající funkční symboly.

2.4.3 Složitost

Automatizované klasické plánování je velice časově náročný proces. Pokud je dovoleno, aby v předpokladech a efektech operátorů byly negativní atomické formule (tedy $\text{precond}^-(o) \neq \emptyset$ pro alespoň jeden operátor o a $\text{effects}^-(o) \neq \emptyset$ pro alespoň jeden operátor o), pak podle [5] je složitost PSPACE. [5, 4]

2.5 HTN

Plánování pomocí hierarchických sítí úloh, anglicky Hierarchical Task Network (HTN), je stejné jako klasické plánování v tom, že každý stav světa je reprezentován sadou atomických formulí a každá akce odpovídá deterministickému přechodu mezi nimi. HTN plánovače se však od klasických plánovačů liší v tom, co je cílem a jak toho dosáhnout. [6, 4]

HTN plánovač nemá dosáhnout splnění množiny cílů, ale místo toho provést nějaký soubor úloh. Vstup do plánovače obsahuje množinu operátorů podobných těm u klasického plánování a také množinu metod, z nichž každá je předpisem, jak rozložit nějakou úlohu na nějakou sadu dílčích úloh. Plánování probíhá rekurzivním rozkladem neprimitivních úloh na menší a menší úlohy, dokud nejsou dosaženy primitivní úlohy, které lze provádět přímo pomocí operátorů. HTN plánovač použije operátor pouze tehdy, když je to definované v nějaké metodě. [6, 4]

Plánování HTN se pro praktické aplikace používá více než kterákoli z jiných plánovacích technik. Částečně je to proto, že metody HTN poskytují pohodlný způsob, jak napsat „recepty“ na řešení problémů, které odpovídají tomu, jak by odborník v nějakém oboru mohl přemýšlet o řešení problému plánování. [6, 4]

Většina HTN plánovačů je „ručně přizpůsobitelná“: jejich plánování je doménově nezávislé, ale jejich metody jsou doménově specifické. Plánovač lze tedy přizpůsobit tak, aby pracoval v různých doménách odlišně tím, že mu jsou poskytnuty různé množiny metod. Schopnost využívat znalosti specifické pro určitou doménu může výrazně zlepšit výkon plánovače a někdy může znamenat rozdíl mezi řešením problému v exponenciálním a polynomiálním čase [7, 8]. V experimentálních studiích [9, 10, 11] ručně přizpůsobené plánovače vyřešily problémy plánování řádově složitější než ty, které jsou typicky řešeny „plně automatizovanými“ klasickými plánovacími systémy, ve kterých se doménově specifické znalosti skládají pouze z plánovacích operátorů. [6, 4]

V [4] se rozlišuje mezi Hierarchical Task Network (HTN) a zjednodušenou variantou Simple Task Network (STN). V této práci bude popsáno pouze STN, protože bude využit plánovací systém SHOP2. Ten bude více popsán v podsekci 2.5.4. Zároveň ale bude v celé práci používáno pojmenování HTN i přesto, že by se podle [4] mělo jednat o STN.

Definice pojmů, literálů, operátorů, akcí a plánů jsou stejné jako v klasickém plánování. Definice $\gamma(s, a)$, výsledku aplikace akce a na stav s, je rovněž stejná jako v klasickém plánování. Jazyk však zahrnuje také úlohy, metody a sítě úloh, které se používají při definicích plánovacího problému a jeho řešení. [6, 4]

2.5.1 Úlohy a metody

Jedním z nových druhů symbolů je symbol úloh. Každý symbol operátoru je také symbolem úlohy, ale existují další symboly úloh, které se nazývají neprimitivní symboly úloh. Úloha je výraz ve tvaru $t(r_1, r_2, \ldots, r_k)$, kde t je symbol úlohy a $r_1, r_2, \ldots, r_k$ jsou termy. Jestliže je t také symbolem operátoru, pak je úloha primitivní; jinak je úloha neprimitivní. Úloha je plně určená, pokud jsou všechny termy plně určené; jinak není plně určená. Akce $a = (\text{name}(a), \text{precond}(a), \text{effects}(a))$ vykoná plně určenou primitivní úlohu t ve stavu s, pokud $\text{name}(a) = t$ a akci a lze aplikovat v s. [4, 6]

▶ **Definice 2.9.** *Síť úloh je acyklický orientovaný graf $w = (U, E)$, ve kterém je U množina vrcholů, E je množina hran a každý vrchol $u \in U$ obsahuje úlohu t_u. Graf w je plně určený, pokud všechny úlohy $\{t_u | u \in U\}$ jsou plně určené; jinak w není plně určený. Graf w je primitivní, pokud všechny úlohy $\{t_u | u \in U\}$ jsou primitivní; jinak je w neprimitivní. [4, 6]*

Hrany w určují uspořádání vrcholů U a tedy i uspořádání úloh v daných vrcholech. Tím je určena vzájemná závislost úloh (tedy, které úlohy musejí být vykonány, aby daná úloha mohla být vyřešena). [4, 6]

Do plánovacího jazyka budou také zahrnuty nové symboly nazývané symboly metod. [4, 6]

▶ **Definice 2.10.** *Metoda je čtveřice* $m = (name(m), task(m), precond(m), network(m))$*, ve které jsou prvky popsány následovně: [4, 6]*

- *name*(m)*, název metody, je syntaktický výraz ve tvaru* $n(x_1, x_2, \ldots, x_k)$*, kde* n *je unikátní symbol metody (tj. žádné dvě metody nemají stejnou hodnotu pro* n*), a* $x_1, x_2, \ldots, x_k$ *jsou proměnné, které se vyskytují kdekoli v* m*.*
- *task*(m) *je neprimitivní úloha.*
- *precond*(m) *je množina literálů nazývaných předpoklady metody*
- *network*(m) *je síť úloh, jejíž prvky se nazývají dílčí úlohy* m*.*

Prvek task(m) metody m říká, na jakou úlohu lze m použít, precond(m) specifikuje, jaké podmínky musí aktuální stav splňovat, aby bylo m možné použít, a network(m) specifikuje dílčí úlohy, které je třeba splnit pro splnění úlohy task(m) pomocí této metody m. [4, 6]

Obdobně jako pro operátory je definované $\text{precond}^+(o)$ a $\text{precond}^-(o)$, tak pro metody je analogicky definované $\text{precond}^+(m)$ a $\text{precond}^-(m)$. To je využité v následující definici. [4, 6]

▶ **Definice 2.11.** *Instance metody* m *je aplikovatelná ve stavu* s*, jestliže* $precond^+(m) \subseteq s$ *a* $precond^-(m) \cap s = \emptyset$*. [4, 6]*

▶ **Definice 2.12.** *Nechť* t *je úloha a* m *je instance metody. Pokud existuje substituce* σ *taková, že* $\sigma(t) = task(m)$*, pak* m *je relevantní pro* t *a rozklad* t *podle* m *s pomocí* σ *je* $\delta(t, m, \sigma) = network(m)$*. [4, 6]*

Když je k rozkladu úlohy t použita metoda m, bude tato úloha obvykle součástí vrcholu u v síti úloh w, v takovém případě dostaneme novou síť úloh $\delta(w, u, m, \sigma)$, podle definice 2.13. Formální definice $\delta(w, u, m, \sigma)$ je poněkud komplikovaná, ale intuitivní myšlenka je jednoduchá: u je odstraněno z w, kopie dílčích úloh network(m) je vložena do w místo u a každé omezení týkající se uspořádání, které dříve platilo pro u, nyní platí pro každý vrchol v kopii network(m) v nově síti úloh. [4, 6]

▶ **Definice 2.13.** *Nechť* $w = (U, E)$ *je síť úloh,* u *je vrchol ve* w*, který nemá žádné předchůdce ve* w*, a* m *je metoda, která je relevantní pro* t_u *s pomocí nějaké substituce* σ*. Nechť* $succ(u)$ *je množina všech bezprostředních následníků* u*, tj.* $succ(u) = \{u' \in U | (u, u') \in E\}$*. Nechť* $succ_1(u)$ *je množina všech bezprostředních následníků* u*, pro které je* u *jediným předchůdcem. Nechť* (U', E') *je výsledkem odstranění* u *a všech hran, které* u *obsahují. Nechť* (U_m, E_m) *je kopie sítě network*(m)*. Jestliže* (U_m, E_m) *není prázdné, pak výsledkem rozkladu* u *ve* w *podle* m *s pomocí* σ *je tato množina sítí úloh: [4, 6]*

$$\delta(w, u, m, \sigma) = \{(\sigma(U' \cup U_m), \sigma(E_v)) | v \in subtasks(m)\},$$

kde

$$E_v = E_m \cup (U_m \times succ(u)) \cup \{(v, u') | u' \in succ_1(u)\}.$$

Jinak $\delta(w, u, m, \sigma) = \{(\sigma(U'), \sigma(E'))\}$*. [4, 6]*

2.5.2 Domény, problémy a řešení

▶ **Definice 2.14.** *Doména HTN plánování je pár* $\mathcal{D} = (O, M)$*, kde* O *je množina operátorů a* M *je množina metod. [4, 6]*

▶ **Definice 2.15.** *Problém HTN plánování je čtveřice* $\mathcal{P} = (s_0, w, O, M)$*, kde* s_0 *je počáteční stav,* w *je síť úloh nazývaná síť počátečních úloh a* $\mathcal{D} = (O, M)$ *je doména HTN plánování. [4, 6]*

Nyní bude definované, co znamená, že plán $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ je řešením plánovacího problému $\mathcal{P} = (s_0, w, O, M)$. Intuitivně to znamená, že existuje způsob, jak rozložit w na π takovým způsobem, že π lze použít v počátečním stavu a každý rozklad úlohy je aplikovatelný v příslušném stavu. Formální definice je rekurzivní a má tři případy. [4, 6]

▶ **Definice 2.16.** *Nechť $\mathcal{P} = (s_0, w, O, M)$ je problém HTN plánování. Následují případy, kdy plán $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ je řešením pro $\mathcal{P}$. [4, 6]*

1. *Případ, kde w je prázdné. Pak π je řešením pro $\mathcal{P}$, pokud je π prázdné (tj. $n = 0$).*

2. *Existuje primitivní vrchol úlohy $u \in w$, který nemá ve w žádné předchůdce. Pak π je řešení pro $\mathcal{P}$, pokud a_1 lze aplikovat pro t_u v s_0 a plán $\pi = \langle a_2, \ldots, a_n \rangle$ je řešením tohoto plánovacího problému:*

$$\mathcal{P}' = (\gamma(s_0, a_1), w - \{u\}, O, M)$$

   Intuitivně je $\mathcal{P}'$ problém HTN plánování vytvořený aplikací první akce π a odstraněním odpovídajícího vrcholu úlohy z w.

3. *Případ, kdy je neprimitivní vrchol úlohy $u \in w$, který nemá žádné předchůdce ve w. Existuje-li instance m nějaké metody v M taková, že m je relevantní pro t_u a aplikovatelné v s_0, pak π je řešením pro $\mathcal{P}$, pokud existuje síť úloh $w \in \delta(w', u, m, \sigma)$ taková, že π je řešením pro (s_0, w', O, M).*

Síť w může obsahovat více než jeden vrchol, který nemá žádné předchůdce. Případy 2 a 3 se tedy nemusí nutně vzájemně vylučovat. [4, 6]

Je-li π řešením pro $\mathcal{P} = (s_0, w, O, M)$, pak pro každý vrchol úlohy $u \in U$ existuje rozklad v podobě stromu, jehož listy jsou akcemi π. [4, 6]

2.5.3 Procedura HTN plánování

Pro úplnost HTN plánování je, zde na obrázku 2.5, uveden pseudokód příslušné PFD (Partial-order Forward Decomposition) procedury. Ta zadané úlohy ve w řeší pomocí operací v O a metod v M s počátečními atomickými formulemi v s. Následující seznam bude obsahovat některé anglické varianty termínů s jejich překlady používanými v této práci.

- anglický výraz „ground“ byl překládán jako „plně určená“;
- anglický výraz „is applicable to“ byl překládán jako „je aplikovatelná ve“.

2.5.4 SHOP2

SHOP2, Simple Hierarchical Ordered Planner 2, je doménově nezávislý plánovací systém založený na HTN plánování. V roce 2002 na International Planning Competition získal SHOP2 jedno ze čtyř nejlepších ocenění a jedno ze dvou ocenění za vynikající výkon. [6, 12, 13]

V práci bude využit systém JSHOP2, ten je implementací systému SHOP2 v jazyce Java. Následující podsekce stručně popisují syntaxi systému JSHOP2 (ta je stejná i pro SHOP2), konkrétně pouze části, které budou v této práci využity. Pokud je část definice v hranatých závorkách, tak je nepovinná. Komentáře jsou v kódu označeny středníkem. [14, 6, 12]

Procedure PFD(s, w, O, M)**:**
  if $w = \emptyset$ **then return** *the empty plan*
  nondeterministically choose any $u \in w$ that has no predecessors in w
  if t_u *is a primitive task* **then**
    $active \leftarrow \{(a, \sigma) \mid a$ is a ground instance of an operator in O, σ is a substitution such that $name(a) = \sigma(t_u)$, and a is applicable to $s\}$
    if $active = \emptyset$ **then return** *failure*
    nondeterministically choose any $(a, \sigma) \in active$
    $\pi \leftarrow$ PFD$(\gamma(s, a), \sigma(w - \{u\}), O, M))$
    if $\pi = failure$ **then return** *failure*
    else return $a.\pi$
  else
    $active \leftarrow \{(m, \sigma) \mid m$ is a ground instance of a method in M, σ is a substitution such that $name(m) = \sigma(t_u)$, and m is applicable to $s\}$
    if $active = \emptyset$ **then return** *failure*
    nondeterministically choose any $(m, \sigma) \in active$
    nondeterministically choose any task network $w' \in \delta(w, u, m, \sigma)$
  end
return PFD(s, w', O, M)

Obrázek 2.5 PFD procedura pro HTN plánování.

2.5.4.1 Symboly

Ve strukturách popsaných níže je pět druhů symbolů: proměnné, konstantní symboly, primitivní symboly úloh, neprimitivní symboly úloh a symboly funkcí. K rozlišení mezi těmito symboly používá JSHOP2 následující konvence: [14, 6, 12]

- Proměnná může být jakýkoli symbol, jehož název začíná otazníkem (např. ?x nebo ?hello-there).
- Primitivním symbolem úloh může být jakýkoli symbol, jehož název začíná vykřičníkem (například !unstack nebo !putdown).
- Konstantní symbol, predikátový symbol nebo neprimitivní symbol úlohy může být jakýkoli symbol, jehož název začíná písmenem nebo podtržením.
- Symbolem funkce může být jakýkoli platný Java identifikátor.

Jakákoli ze struktur definovaných ve zbývajících částech se považuje za plně určenou, pokud neobsahuje žádné proměnné. [14, 6, 12]

2.5.4.2 Termy

Term je cokoli z následujícího: [14, 6, 12]

- proměnná,
- konstantní symbol,
- číslo,
- seznam termů,
- volací term.

Seznam termů je term, který má následnou podobu:

$$(t_1\ t_2\ \ldots\ t_n),$$

kde každé t_i je term. To určuje, že $t_1, t_2, \ldots, t_n$ jsou položky seznamu. [14, 6, 12]

Volací term je term, který má následnou podobu:

$$(call\ f\ t_1\ t_2\ \ldots\ t_n),$$

kde f je buď symbol funkce nebo vestavěná funkce JSHOP2 (jako je například +) a každé t_i je term. Volací term má zvláštní význam pro JSHOP2, protože říká systému JSHOP2, že f je připojená procedura, tj. kdykoli JSHOP2 potřebuje vyhodnotit jakoukoli strukturu, kde se objeví volací term, JSHOP2 nahradí term výsledkem použití funkce f na argumentech $t_1, t_2, \ldots, t_n$. Například následující volací term by měl hodnotu 9: [14, 6, 12]

$$(call\ *\ (call\ +\ 1\ 2)\ 3)$$

Některé z nejběžnějších funkcí jsou v JSHOP2 již přítomny, ale uživatel může definovat funkce nové. [14, 6, 12]

2.5.4.3 Atomické formule

Atomická formule má tvar:

$$(p\ t_1\ t_2\ \ldots\ t_n),$$

kde p je predikátový symbol a každé t_i je term. [14, 6, 12]

2.5.4.4 Logické výrazy

Logický výraz je atomická formule nebo kterýkoli z následujících komplexních výrazů: konjunkce, disjunkce, negace, výrazy univerzálních kvantifikátorů nebo volací výrazy. [14, 6, 12]

Konjunkce má tvar:

$$([and]\ [L_1\ L_2\ \ldots\ L_n]),$$

kde každé L_i je logický výraz. [14, 6, 12]

Disjunkce má tvar:

$$(or\ L_1\ L_2\ \ldots\ L_n),$$

kde každé L_i je logický výraz. [14, 6, 12]

Negace má tvar:

$$(not\ L),$$

kde L je logický výraz. [14, 6, 12]

Výraz univerzálního kvantifikátoru má tvar:

$$(\text{forall}\ V\ Y\ Z),$$

kde Y a Z jsou logické výrazy a V je seznam proměnných v Y. Záměrem výrazu univerzálního kvantifikátoru je splnit pro každou možnou substituci u pro proměnné ve V, že pokud je splněno Y^u, musí být splněno i Z^u v aktuálním stavu. Toto použití klíčového slova forall se liší od jeho použití v seznamech přidání a odstranění v operátorech (viz podsekce 2.5.4.8); to se používá k vyjádření množiny efektů spíše než logického výrazu, a proto má odlišnou syntaxi. [14, 6, 12]

Volací výraz má stejný tvar jako volací term, ale sémanticky je jeho hodnota interpretována jako true nebo false. [14, 6, 12]

Výpis kódu 2.1 Ukázka axiomu systému SHOP2 (axiom použitý v novém AI pro Dune 2000).

```
(:- (enough_money ?necMon)
  ;tail - seznam
  ((money ?mon) (call >= ?mon ?necMon))
)
```

2.5.4.5 Axiomy

Axiom má tvar:

$$(\text{:- } a\ [name_1]\ L_1\ [name_2]\ L_2\ \ldots\ [name_n]\ L_n),$$

kde hlavička a je atomická formule a zbytek je seznam $[name_1]\ L_1\ [name_2]\ L_2\ \ldots\ [name_n]\ L_n$, kde každé L_i je logickým výrazem a každé $name_i$ je symbol zvaný název L_i. Názvy výrazů nejsou povinné. Když se definice domény načte do JSHOP2, bude pro každou větev vygenerován jedinečný název, pokud žádný nebyl zadán. Tyto názvy nemají pro JSHOP2 žádný sémantický význam, ale jsou poskytovány, aby pomohly uživateli ladit systém. Význam axiomu je ten, že a je pravdivé, pokud L_1 je pravdivé, nebo pokud L_1 je nepravdivé, ale L_2 je pravdivé, ..., nebo pokud všechny $L_1, L_2, \ldots, L_{n-1}$ jsou nepravdivé, ale L_n je pravdivé. Například axiom ve výpisu kódu 2.1 říká, zda aktuální množství zdrojů ?mon stačí na objekt s cenou ?necMon. Tato ukázka má hlavičku s jednou proměnnou a seznam má jediný prvek, který nemá definovaný název. [14, 6, 12]

2.5.4.6 Atomické formule úloh

Atomická formule úlohy má tvar:

$$(s\ t_1\ t_2\ \ldots\ t_n),$$

kde s je symbol úlohy a argumenty $t_1\ t_2\ \ldots\ t_n$ jsou termy. Atomická formule úlohy je primitivní, pokud s je primitivní symbol úlohy, a je neprimitivní, pokud s je neprimitivní symbol úlohy. [14, 6, 12]

2.5.4.7 Seznamy úloh

Seznam úloh je buď atomická formule úlohy, nebo výraz ve tvaru:

$$([\text{:unordered}]\ [tasklist_1\ tasklist_2\ \ldots\ tasklist_n]),$$

kde $tasklist_1\ tasklist_2\ \ldots\ tasklist_n$ jsou seznamy úloh. Platí, že n může být nula, což má za následek prázdný seznam úloh. [14, 6, 12]

Pokud v seznamu úloh není klíčové slovo :unordered, znamená to, že JSHOP2 musí provádět seznamy úloh v pořadí, v jakém jsou uvedeny. Klíčové slovo :unordered určuje, že mezi seznamy úloh $tasklist_1\ tasklist_2\ \ldots\ tasklist_n$ není žádné definované pořadí. S použitím klíčového slova :unordered může JSHOP2 prokládat úlohy mezi různými seznamy úloh. Předpokládejme, že jsou dva následující seznamy úloh:

$$T = (t_1\ t_2\ \ldots\ t_m)$$

$$U = (u_1\ u_2\ \ldots\ u_n)$$

a že je hlavní seznam úloh

$$M = (\text{:unordered } T\ U).$$

Úlohy v T a U budou provedeny v daném pořadí (od jedné do m, respektive n), ale JSHOP2 může libovolně prokládat úlohy z T a U mezi sebou.

Výpis kódu 2.2 Ukázka operátoru systému SHOP2 (zjednodušená varianta operátoru použitého v nové AI pro Dune 2000).

```
(:operator
  ;head - hlavicka
  (!o_wait ?harvester_income)
  ;precondition - predpoklady
  (
    (money ?mon)
    (wait_time ?wt)
    (cnt harvester ?harcnt)
  )
  ;delete list - seznam odstraneni
  (
    (money ?mon)
  )
  ;add list - seznam pridani
  (
    (money (call + ?mon (call * ?harcnt ?harvester_income ?wt)))
  )
  ;cost - cena
  ?wt
)
```

2.5.4.8 Operátory

Operátor má tvar:

$$(\text{:operator } h\ P\ D\ A\ [c]),$$

kde: [14, 6, 12]

- h (hlavička operátoru) je primitivní atomická formule úlohy.
- P (předpoklad operátoru) je logický výraz. P může obsahovat libovolné proměnné (tedy mohou, ale nemusí, být v h).
- D (seznam odstranění operátoru) je seznam, jehož každý z prvků může být kterýkoli z následujících:
  - Atomická formule, která může obsahovat pouze proměnné, které se objevují v h nebo P.
  - Výraz ve tvaru (forall V Y Z), kde V je seznam proměnných v Y, Y je logický výraz a Z je seznam atomických formulí, které neobsahují žádné jiné proměnné než ty v h, P nebo V.
- A (seznam přidání operátoru) je seznam prvků, které mají podobu jako prvky D.
- c (cena operátoru) je term. Pokud je c vynecháno, jeho výchozí hodnota je 1.

Jak bylo uvedeno výše, hlavička operátoru je primitivní atomická formule úlohy, takže musí začínat primitivním symbolem úlohy, tj. symbolem, který začíná vykřičníkem. [14, 6, 12]

Oproti operátoru definovanému dříve pro klasické plánování v podsekci 2.4.2.2, je zde množina effects(o) rozdělena na pozitivní a negativní efekty o, tedy effects$^+$(o) a effects$^-$(o), pro JSHOP2 označené jako D a A. Hlavička operátoru h přímo odpovídá name(o). [14, 6, 12, 4]

Jména operátorů, která začínají dvěma vykřičníky, mají v JSHOP2 zvláštní význam; operátory tohoto druhu jsou označované jako interní operátory. Interní operátory jsou operátory, které se používají pro interní účely při plánování (např. aby provedly nějaké výpočty, které budou později užitečné při rozhodování o tom, jaké akce provést). Kromě požadavku na dva vykřičníky na

Výpis kódu 2.3 Ukázka metody systému SHOP2 (upravená varianta metody použité v nové AI pro Dune 2000).

```
(:method (m_build_X_light_inf ?x)
  branch1
  ((call <= ?x 0))
  ()
  branch2
  ((barracks_built)  (cost_light_inf ?costb) (enough_money ?costb))
  ((!o_build_infantry_light_inf) (m_build_X_light_inf (call - ?x 1)))
  branch3
  ((not (barracks_built)))
  ((m_build_barracks) (m_build_X_light_inf ?x))
  branch4
  ((cost_light_inf ?costb) (not (enough_money ?costb)))
  ((m_wait) (m_build_X_light_inf ?x))
)
```

začátku názvu je syntaxe a sémantika interních operátorů totožná se syntaxí a sémantikou ostatních operátorů. JSHOP2 při plánování pracuje s interními operátory úplně stejně jako s běžnými operátory. JSHOP2 zahrnuje tyto operátory do plánů, které vrací na konci provádění. Primárním důvodem, proč v JSHOP2 existuje syntaxe interního operátoru, je to, že automatizované systémy, které používají plány JSHOP2 jako vstup, mohou snadno rozlišovat mezi operátory, které zahrnují akci, a těmi, které byly pouze interní v procesu plánování. [14, 6, 12]

Příklad operátoru je v ukázce kódu 2.2, kde je předveden operátor reprezentující čekání plánovače. Operátor má v hlavičce jednu proměnou udávající odhad výdělku za sekundu jedním harvesterem ?harvester_income, přičemž další proměnné jsou v předpokladech operátoru. Konkrétně je to množství zdrojů, délka úseku čekání a počet harvesterů. V seznamu odstranění je jediná atomická formule, která určuje množství zdrojů. Přičemž následně se přidá atomická formule určující nové množství zdrojů navýšené podle počtu harvesterů, jejich výdělku a délky čekání.

2.5.4.9 Metody

Metoda má tvar:

$$(\text{:method } h\ [name_1]\ L_1\ T_1\ [name_2]\ L_2\ T_2\ \ldots\ [name_n]\ L_n\ T_n),$$

kde: [14, 6, 12]

- h (hlavička metody) je neprimitivní atomická formule úlohy.
- Každé L_i (předpoklad metody) je logický výraz.
- Každý T_i (úlohy metody) je seznam úloh.
- Každé $name_i$ je název pro následující pár $(L_i\ T_i)$. Tyto názvy jsou volitelné a pokud jsou vynechány, bude každému páru automaticky přiřazen unikátní název. Tyto názvy nemají pro JSHOP2 žádný sémantický význam, ale jsou poskytovány proto, aby uživatelům pomohly ladit popisy domén.

Metoda označuje, že úlohu zadanou v hlavičce h lze provést vykonáním všech úloh v jednom ze seznamů úloh T_i, když je splněn předpoklad příslušný tomuto seznamu. Předpoklady jsou zpracovávány v daném pořadí a pozdější předpoklad je použit pouze v případě, že nelze splnit všechny dřívější předpoklady. Pokud je v určitém okamžiku k dispozici více metod pro danou úlohu, budou všechny tyto metody eventuálně vyzkoušeny. [14, 6, 12]

Oproti metodě definované dříve pro HTN plánování v podsekci 2.5.1 je v syntaxi JSHOP2 sjednoceno jméno metody (n z výrazu $n(x_1, x_2, \ldots, x_k)$, což je první prvek v metodě označovaný jako name(m)) a task(m) (což byla neprimitivní úloha, na kterou šlo metodu při splnění předpokladů aplikovat). Síť úloh je zde reprezentována seznamem úloh. [14, 6, 12, 4]

Příklad metody je v ukázce kódu 2.3, kde je předvedena metoda pro stavbu lehké pěchoty. Její množství je v proměnné v hlavičce metody. Následují čtyři dvojice předpokladů a seznamů úloh řešící různé situace.

2.5.4.10 Domény, problémy a plány

Doména JSHOP2 plánování má tvar:

$$(\text{defdomain domain-name } (d_1\ d_2\ \ldots\ d_n)),$$

kde domain-name je symbol a každá položka d_i je jedním z následujících: operátor, metoda nebo axiom. [14, 6, 12]

Problém JSHOP2 plánování má tvar:

$$(\text{defproblem problem-name domain-name } ([a_{1,1}\ a_{1,2}\ \ldots\ a_{1,n}])\ T_1\ \ldots\ ([a_{m,1}\ a_{m,2}\ \ldots a_{m,n}])\ T_m),$$

kde problem-name a domain-name jsou symboly, každé $a_{i,j}$ je plně určená atomická formule a každé T_i je seznam úloh. Tento výraz definuje m plánovacích problémů v doméně domain-name, z nichž každý může být vyřešen řešením úloh v T_i s počátečním stavem definovaným atomickými formulemi $a_{i,1}$ až $a_{i,n}$. [14, 6, 12]

Plán v JSHOP2 je seznam hlaviček operátorů v dané doméně. Jestliže $p = (h_1\ h_2 \ldots\ h_n)$ je plán a s je stav, pak $p(s)$ je stav vytvořený aplikováním operací $o_1, o_2, \ldots, o_n$ v uvedeném pořadí ve stavu s. Cena plánu p je součtem cen každé operace o_i. [14, 6, 12]

2.6 Real-time strategické hry

Real-time strategické hry (RTS) jsou žánrem počítačových her zaměřených na strategii. Jedná se o variantu strategických her společně s tahovými strategickými hrami. Prvotní použití pojmenování žánru patří Brettu Sperrymu, kreativnímu řediteli a spoluzakladateli společnosti Westwood Studios. Ten toto pojmenování vymyslel při vývoji hry Dune II, která je považována za první hru tohoto žánru. [15]

Při hraní RTS her uživatel opakovaně začíná a odehrává konkrétní mise, ve kterých ovládá objekty vlastněné jednou zúčastněnou stranou. Každá tato strana je ovládaná jedním hráčem (uživatelem nebo počítačem). Hráči mohou mít mezi sebou různé vztahy, například válku, mír nebo alianci. Jednotlivé mise jsou definované určitou mapou (terénem i objekty) a cíli, které musí hráč splnit. Ty jsou většinou známé předem, ale například v misích s příběhem se mohou v průběhu mise změnit.

V RTS každá mise probíhá na předdefinované, ve většině případů navíc ohraničené mapě. Aby mohl hráč vyhrát, musí obvykle získávat suroviny, většinou je k tomu potřeba kontrolovat určité části mapy, za suroviny stavět budovy a jednotky, které ovládá a plní s nimi zadané úkoly. Ty se obvykle týkají zničení soupeřových budov a jednotek. Častou součástí RTS her je nutnost mapu nejdříve prozkoumat, protože ta je zpočátku hry neznámá.

Typicky je ve hře uživatelova obrazovka rozdělena na prostor zobrazující zvolenou část mapy a její obsah, a uživatelské rozhraní, přes které není mapa vidět. To uživateli poskytuje některé možnosti ovládání hry (například stavbu nových budov a jednotek) a výpis důležitých informací ve hře. Mezi ně zpravidla patří uživatelovy zdroje, které jsou potřebné ke stavbě nových budov a jednotek, nebo třeba minimapa ukazující zmenšenou a značně zjednodušenou celou mapu hry (díky tomu má uživatel přehled o dění na celém území).

Pro ovládání jednotek na mapě jsou obvykle k dispozici různé funkcionality, jako například možnost označit větší množství jednotek najednou kliknutím a táhnutím, čímž se určí oblast, ve které jsou všechny jednotky označeny. Pak jim má uživatel možnost dát najednou nějaký rozkaz, například k pohybu na jinou část mapy nebo k útoku na soupeřovy jednotky nebo budovy.

Důležitou součástí RTS her je soupeř a jeho chování. Pokud uživatel nechce nebo nemůže hrát proti jinému člověkem ovládanému hráči, má většinou možnost hrát proti počítačem ovládanému hráči. Ten je obvykle označován jako umělá inteligence nebo AI (Artificial Intelligence).

Hry také obvykle obsahují sekvenci příběhových misí nazývaných „Campaign“, česky většinou „Kampaň“, které mají úkoly spojeny s příběhem a soupeř je vždy řízen počítačem s velice pevně nastavenými pravidly. Další možností jsou pro uživatele potyčky, uživatelem nastavené mise s cílem poražení všech soupeřů, kde všichni hráči začínají hru vyrovnaně silní a hrají podle stejných pravidel (v některých hrách je možné dát libovolným hráčům ekonomické nebo bojové bonusy).

2.6.1 Analýza cílů AI v RTS

Umělá inteligence je součástí počítačové hry a jejím cílem je, stejně jako celé hry, být pro hráče zábavná. V jejím případě být zábavným soupeřem.

Jedním z aspektů, zda je hra zábavná, je i její obtížnost, která nesmí být příliš nízká, ani příliš vysoká. V případě potyček v real-time strategických hrách je obtížnost přímo závislá na umělé inteligenci. Uživatel by ji v ideálním případě měl být schopen s dostatečně velkým úsilím porazit. Vzhledem k tomu, že uživatelé jsou různě dobří v dané hře, tak může být nemožné vytvořit umělou inteligenci vhodnou pro všechny. Z tohoto důvodu jsou často dostupné její různě obtížné verze. Tyto verze mohou být rozdílné jak svou inteligencí, tak ale zároveň mohou mít určité herní bonusy. [16]

Vzhledem k tomu, že pro uživatele je zábavnější soupeře porážet, tak může být nevhodné mít umělou inteligenci příliš schopnou. To se týká všech aspektů RTS her, tedy jak dlouhodobých strategických rozhodnutí, tak ale především řízení samotných jednotek, kde pro hráče může být rychlost, s jakou je AI schopná ovládat jednotky, nedosažitelná. Umělá inteligence by tedy v nejlepším případě neměla příliš zneužívat své schopnosti ovládat všechny jednotky simultánně.

Potenciálním problémem je také to, že uživatel se při hraní proti umělé inteligenci učí kromě toho, jak lépe hrát hru (což je vhodné), také jak hrát konkrétně proti ní. To může vést k tomu, že uživatel nalezne strategii efektivní proti tomu, jak daná AI hraje, a bude ji vždy používat. Ve výsledku může uživatele hra přestat bavit dříve právě proto, že začne příliš opakovat jednu konkrétní strategii a hra mu tak připadá jednoduchá a neměnná. Tento problém může být způsoben tím, že AI hraje příliš předvídatelně a vždy opakuje určitá rozhodnutí. Z tohoto důvodu je vhodné, aby umělá inteligence náhodně volila strategie, jak bude hrát. Tím se může prodloužit doba, po kterou uživatele hra baví, protože zdokonalení se vyžaduje více úsilí. [16]

Ve výsledku umělá inteligence nesmí hrát hru nejlépe, jak je možné (vzhledem ke složitosti hry a tomu, že hra musí běžet plynule v reálném čase, to ani není většinou možné), ale tak silně jako uživatel nebo hůře, a její chování by mělo obsahovat různé náhodně volené strategie, aby se předešlo monotónnosti.

2.6.2 OpenRA

OpenRA je open source projekt licencovaný pod GPLv3. Jedná se o herní engine, tedy softwarový framework vytvořený pro usnadnění vývoje počítačových her, vhodný k tvorbě real-time strategických her. [17, 18, 19]

Herní engine je psaný v programovacím jazyce C#. Samotné hry mohou být psány pouze pomocí souborů YAML konfigurujících fungování hry. Vytvořené hry jsou oficiálně nazývány jako módy, tohoto značení ve zbytku práce nebude používáno, protože by vedlo ke zbytečnému zvýšení složitosti terminologie. [18]

V OpenRA jsou již vytvořeny tři RTS hry a čtvrtá je ve vývoji (poslední popsaná níže). Ve všech případech se jedná o hry vytvářené společností Westwood Studios. Všechny tyto hry jsou si herními mechanikami podobné. [18]

Command & Conquer: Tiberian Dawn Hra původně známá jen pod názvem Command & Conquer. Tiberian Dawn bylo později přidáno, aby se lépe odlišilo mezi touto hrou a pojmenováním celé série her Command & Conquer. Hra byla vydaná divizí Virgin Interactive Entertainment (tj. vydavatelská divize videoher britského konglomerátu Virgin Group) roku 1995. Hru vytvořila společnost Westwood Studios.

Command & Conquer: Red Alert Hra byla vydaná divizí Virgin Interactive Entertainment roku 1996. Hru vytvořila společnost Westwood Studios.

Dune 2000 Hra byla vydaná divizí Virgin Interactive Entertainment pro počítačovou verzi roku 1998 a společností Electronic Arts pro PlayStation verzi roku 1999. Hru vytvořily společnosti Westwood Studios a Intelligent Games.

Command & Conquer: Tiberian Sun Hra byla vydaná společností Electronic Arts roku 1999. Hru vytvořila společnost Westwood Studios.

2.6.3 Dune 2000

Dune 2000 je real-time strategická hra, vydaná divizí Virgin Interactive Entertainment pro počítačovou verzi roku 1998 a společností Electronic Arts pro PlayStation verzi roku 1999. Hru společně vytvořily společnosti Westwood Studios a Intelligent Games. [20]

Dune 2000 je vylepšenou verzí hry Dune II. Hra se změnila ve všech směrech, zlepšila se grafika i hudba, nejvíce se ale změnilo ovládání hry, které umožňuje rychleji a lépe ovládat větší množství jednotek. Hra, co se týče hratelnosti, se nyní více podobá real-time strategickým hrám, které byly studii stojícími za touto hrou vytvořeny v podobné době, jako je například Command & Conquer: Red Alert vydané 1996. [20, 18]

Hra obsahuje tři kampaně za tři různé rody (rod Atreidů, rod Harkonnenů a rod Ordosů). Každá je složená z devíti po sobě jdoucích misí s tím, že má uživatel často možnost výběru mezi dvěma misemi. Jednotlivé mise mají různé cíle, ty jsou vysvětleny příběhem. [20, 18]

Následující podsekce popisující hru Dune 2000 bude vysvětlovat fungování OpenRA verze hry, nikoliv původní verzi. Rozdíly jsou rozepsány na konci této podsekce v části 2.6.3.6.

2.6.3.1 Mechaniky hry

Mise probíhají na mapě s čtverečkovou mřížkou, která má v průběhu neměnnou velikost. Každé pole mapy má určitý typ terénu. Každý typ terénu má různé vlastnosti, například zda na terén lze vstoupit jednotkou nebo zda na něj lze postavit budovu. Následuje jejich konkrétní popis: [18]

Písek Obvykle nejběžnější terén na mapě. Ve hře má světle žlutou až bílou barvu. Průchozí je všemi typy jednotek. Nelze na něj stavět budovy.

Koření Jedná se o písek, na kterém je koření, to má oranžovou barvu. Má stejné vlastnosti jako písek.

Duny Vzhledem podobné písku. Vlastnostmi stejné jako písek s tím rozdílem, že pozemní jednotky se na něm pohybují pomaleji (pěchota se pohybuje 80% a vozidla 50% rychlostí).

Skála Ve hře znázorněna hnědou barvou. Průchozí je všemi typy jednotek. Lze na ni stavět budovy, ty jsou ale po výstavbě zraněny až do poloviny svých životů.

Betonové desky Hráč má možnost postavit na skálu betonové desky, pokud tak udělá, je typ terénu změněn. Betonové desky mají vlastnosti stejné jako skála s tím rozdílem, že budovy na nich postavené nejsou zraněny. Pokud je budova postavena na betonových deskách jen částečně, je zraněna méně, proporcionálně k části budovy umístěné na skále.

Kamenitý terén Terén průchozí pouze pro pěší jednotky. Ty se na něm pohybují pomaleji (80 % běžné rychlosti) a dostávají méně zranění (80 % běžného zranění). Nelze na něj stavět budovy.

Neprůchozí terén Jedná se o terén neprůchozí pro všechny pozemní jednotky a nelze na něm stavět budovy. Na mapě může vypadat například jako vrak nějakého vozidla nebo letadla. Nejčastěji se ale jedná o útes nacházející se mezi skálou a pískem.

Všechny popsané typy terénu, s výjimkou dun, jsou vidět na obrázku 2.6.

Na mapě se mohou nacházet různé herní objekty. Některé tyto objekty lze rozdělit do skupin, které sdílí určité vlastnosti. Následuje jejich popis: [18]

Budovy Jedná se o nepohyblivé herní objekty, které většinou leží na několika polích mapy současně. Jejich výčet i popis je v podsekci 2.6.3.2.

Jednotky Objekty schopné pohybu. Konkrétní popis je v podsekci 2.6.3.3. Jednotky lze dále dělit na:

Pozemní jednotky Jedná se o objekty, které vždy okupují jedno pole mapy, to pro ostatní jednotky udělají neprůchodné. Výjimkou je pěchota, až pět jednotek tohoto typu může být na jednom poli současně. Pozemní jednotky se mohou dále dělit na:

- Pěchota
- Lehká vozidla
- Těžká vozidla

Letecké jednotky Speciální typ jednotky pohybující se nezávisle na mřížce mapy a jejím terénu.

Dalším pohyblivým objektem na mapě je písečný červ. Ten je téměř nezničitelný, ale je schopen se pohybovat pouze přes písek, koření a duny. Pokud se dostane do kontaktu s jakýmkoli vozidlem, tak ho může zničit. [18]

Kopec koření, anglicky Spice mound nebo Spice Bloom, je objekt, který se objevuje na určitých polích písku na mapě. Těchto míst je obvykle málo. Tento objekt po určité době, nebo pokud na něj nějaká jednotka vstoupí, exploduje a rozptýlí do okolí nové koření. Jedná se o jediný způsob, jak se může objevit nové koření na mapě. [18]

V práci bude používán pojem základna. Ten označuje veškeré hráčovy budovy, které jsou vůči sobě na mapě v relativní blízkosti. Budovy lze stavět jen do určité vzdálenosti od ostatních vlastních budov. Základna pak vzniká jako přirozený důsledek tohoto omezení. (Stavba bude vysvětlena dále v této podsekci.)

Mapa je na začátku hry pro hráče neprozkoumaná, to je znázorněno černou oblastí, tu je pro ilustraci možné vidět na obrázku 2.6. Pro hráče je nutné ji odhalit, toho může dosáhnout jakákoli jím vlastněná jednotka, přičemž každá jednotka má definovaný dohled. Část mapy, která byla alespoň jednou v dosahu dohledu, je již po zbytek hry odhalena. [18]

Části mapy, které hráč odhalil, ale nejsou aktuálně v dohledu žádné hráčovy jednotky, jsou zahaleny mlhou války (anglicky fog of war). Oblast mapy takto zahalená hráči neposkytuje žádné nové informace o aktuálním dění, například pokud hráč projde soupeřovou základnou, tak po jejím opuštění na mapě uvidí pouze budovy, které tam v tu dobu byly (zobrazen zůstane samozřejmě i terén). [18]

Ve hře jsou tři frakce reprezentované třemi různými rody (rod Atreidů, rod Harkonnenů a rod Ordosů). Frakce se liší jen mírně. Některé jednotky a budovy fungují odlišně. Případně mají dostupné dodatečné možnosti stavby nebo naopak některé jednotky stavět nemohou. [18]

Jak již bylo zmíněno, tak se všechny jednotky mohou pohybovat. Každá jednotka má definovanou rychlost. Pozemní jednotky se pohybují po polích mapy, přitom s výjimkou pěchoty nesmí být více jednotek na stejném poli. Každá pozemní jednotka se může přesunout na libovolné z osmi sousedních polí, pokud jim to terén dovolí a nejsou obsazené. Letecké jednotky se nepohybují v mřížce a ignorují pozemní terén. Zároveň mezi sebou nemají žádné kolize. [18]

Boj je nedílnou součástí hry a tedy většina jednotek (a i pár budov) má zbraně. Všechno jsou střelné zbraně, které mají rozdílné vlastnosti, například dostřel, rychlost střelby, přesnost a poškození (různé podle typu odolnosti cíle). Některé zbraně explodují na místě dopadu, v tom případě může být zraněna i vlastní jednotka. Každá jednotka má navíc určitý počet životů (číslo, které u jednotky pod útokem klesá, a pokud dosáhne nuly, tak jednotka umírá) a určitý typ odolnosti (různé zbraně ubírají životy různě odolným cílům odlišně rychle). Těžká vozidla mají navíc možnost zabít soupeřovu pěchotu tím, že ji přejedou (tj. vjedou na stejné pole). Výsledkem je, že každá jednotka má své silné i slabé stránky. Konkrétní jednotky a to, k čemu jsou vhodné, bude popsáno v podsekci 2.6.3.3. [18]

Další podstatnou mechanikou hry je stavba. Určité budovy umožňují stavět nové budovy nebo jednotky. Některé budovy i jednotky vyžadují kromě budovy, ve které je lze stavět, i postavení některé další budovy. Stavba všech budov i jednotek stojí hráče zdroje. Ty je možné rozlišit na peníze a koření. Koření je nutné skladovat a je získáváno pomocí harvesterů. Peníze jsou zdroj, který nelze běžně získat, ale začíná s nimi mise. Jinak se koření i peníze chovají ve všech případech stejně. Budovy lze stavět jen do určité vzdálenosti od ostatních vlastních budov. Stavba může být kdykoli v průběhu pozastavena nebo zrušena. Budova může být zrušena i po jejím dokončení, před jejím umístění na mapu. Zrušení stavby navrátí spotřebované zdroje ve formě peněz. [18]

Stavět může hráč současně jen jednu jednotku nebo budovu v každé z šesti front, které jsou umožněny konkrétní budovou. Tyto fronty a potřebné budovy jsou: [18]

- budovy umožněné základnou
- vylepšení umožněné základnou
- pěchota umožněná kasárnami
- lehká vozidla umožněná lehkou továrnou
- těžká vozidla umožněná těžkou továrnou
- letecké jednotky umožněné leteckou továrnou
- vesmírné letiště (umožňuje některá vozidla a letecké jednotky)

Pokud je nějaká z budov produkujících jednotky postavena vícekrát, tak se rychlost stavby dané fronty zrychlí, to neplatí pro frontu vylepšení a vesmírného letiště. Při postavení druhé budovy se doba stavby zkrátí na dvě třetiny původních časů a při postavení třetí se zkrátí na polovinu. Další budovy již nemají efekt. [18]

Všechny stavby, s výjimkou zdí a betonových desek, spotřebovávají nebo vyrábějí určité množství energie. Pokud má hráč nedostatek energie (větší spotřeba než produkce), tak některé budovy fungují pomaleji nebo nefungují vůbec. Konkrétní způsob ovlivnění nedostatkem energie je popsán v podsekci 2.6.3.2. [18]

Hráč má vždy možnost udělat kteroukoliv z následujících akcí pro jeho postavené budovy: [18]

Oprava Zahájí se oprava zvolené budovy. Životy se postupně začnou doplňovat. Hráč může opravovat libovolné množství budov najednou, ale stojí ho to zdroje.

Prodej Prodá vybranou budovu a navrátí polovinu její ceny. Pokud je budova zraněná, tak je navrácena jen část úměrná zbylým životům (například s polovinou životů je navrácena čtvrtina ceny budovy). Na místě prodané budovy se objeví lehká pěchota v počtu závislém na ceně prodané budovy.

Deaktivace Tuto akci lze použít jen na budovy, které při nedostatku energie přestávají fungovat (radar, raketová věž, palác). Ty pak nebudou spotřebovávat žádnou energii, ale stále nebudou fungovat (díky tomu ale mohou ostatní budovy mít dostatek energie). Budovu lze znova kdykoli aktivovat.

Ve hře jsou celkem čtyři speciální schopnosti. Ty pro aktivaci nic nestojí, ale lze je používat jen, když jsou připravené. Každá je dostupná pouze jedné frakci za určitých podmínek. Jejich popis a komu jsou dostupné, je popsáno níže: [18]

Recruit Fremen Rekrutování Fremenů je schopnost dostupná pouze Atreidům. Použití schopnosti vytvoří dva Fremeny. Dobíjí se 90 sekund.

Air Strike Letecký úder je schopnost dostupná pouze Atreidům. Použití schopnosti vyšle z kraje mapy ze zvoleného směru (osm možností stejných jako jsou směry pohybu po mapě, tedy vodorovné, svislé a úhlopříčné směry) letku ornitoptér, které na zvolený cíl shodí bomby. Cestou na místo mohou být zničeny. Dobíjí se 300 sekund.

Death Hand Schopnost je dostupná pouze Harkonnenům. Použití schopnosti vyšle raketu s atomovou bombou na zvolené místo, kde exploduje. Dobijí se 300 sekund.

Recruit Saboteur Rekrutování sabotéra je schopnost dostupná pouze Ordosům. Použití schopnosti vytvoří jednoho sabotéra. Dobíjí se 90 sekund.

Jednotky po zničení libovolné soupeřovy jednotky dostávají zkušenosti v závislosti na její ceně. Pokud jich mají určité množství, tak se jim zvýší úroveň a tím se zesílí (jsou odolnější, rychlejší, přesnější a střílí častěji). [18]

Hráči jsou na začátku mise stanoveny primární cíle, kterých musí dosáhnout, aby ji vyhrál. Může mít také nějaké sekundární cíle, které k výhře nutné nejsou, ale mohou mu k ní dopomoci. [18]

V misi může být různý počet hráčů a ti vůči sobě mohou být buď spojenci (nemohou se napadnout a spojenecké jednotky odhalují mapu stejně jako vlastní) nebo soupeři (mohou se vzájemně napadnout). [18]

Uživatel má při hraní Dune 2000 možnost výběru jedné ze tří kampaní nebo potyčky. Kampaň je sekvencí příběhových misí s různými primárními a sekundárními cíli. V těchto misích má soupeř již postavenou základnu, kterou dále nerozšiřuje, a jeho chování je speciálně konfigurováno pro danou misi (například jak často útočí, kudy útočí a co staví za jednotky), případně se mu objevují posily (ty jsou přiváženy transportními letadly). Uživatel má navíc možnost volby mezi třemi obtížnostmi (lehká, střední a těžká) a pěti stupni rychlosti hry. [18]

Potyčka je mise bez příběhu, kde jediným primárním cílem je poražení všech soupeřů (zničení všech budov a jednotek) a nejsou zde žádné sekundární cíle. (V práci je používán pojem mise, i když se jedná pouze o potyčku.) Uživatel má mnoho nastavení, kterými může určit podobu dané mise. Následuje popis těch důležitějších: [18]

Map Uživatel má možnost nastavit mapu, na které se bude hrát. Mapy jsou různě velké a mají různý počet počátečních pozic pro hráče.

Players Uživatel má možnost nastavit počet hráčů a jejich barvu, frakci, tým (hráči stejného týmu jsou spojenci, jinak jsou soupeři), handicap (snižuje sílu jednotek), startovní pozici na mapě, a pokud se jedná o AI, tak její typ (ve hře jsou v základu tři rozdílně se chovající AI, které jsou více popsané v podsekci 2.6.3.5).

Starting Cash Počáteční finance umožňují nastavit, s kolika penězi všichni hráči začínají.

Starting Units Počáteční jednotky umožňují začít hru již s nějakými jednotkami.

Game Speed Rychlost hry mění, jak rychle hra běží. Veškeré zmíněné doby budou pro normální nastavení rychlosti, které je výchozí (při nejrychlejším nastavení je hra dvakrát rychlejší než při normálním).

Explored Map Odhalená mapa nastavuje, zda je pro hráče na začátku hry celá mapa odhalena (ale stále může být zakryta mlhou války).

Fog of War Mlha války umožňuje vypnout mlhu války (stále může být neprozkoumaná).

Crates Umožňuje, aby se na mapě objevovaly bedny, na které může vstoupit jakákoli jednotka a stane se nějaká náhodná událost (například hráč dostane finance nebo jednotky, jednotka v kontaktu dostane úroveň, bedna exploduje).

Build off Allies Nastavuje, zda hráči ve stejném týmu mohou stavět jen ke svým budovám nebo i k budovám ostatních členů týmu.

Automatic Concrete Automatické betonové desky nastavují, že není nutné ani možné stavět betonové desky, ale kdykoli je umístěna nová budova, tak se pod ní změní terén na beton.

Short Game Krátká hra umožňuje nastavit, že hráč prohraje, pokud ztratí všechny své budovy.

Worms Umožňuje rozhodnout, zda ve hře budou píseční červi.

2.6.3.2 Budovy

Tato podsekce obsahuje popis všech budov ve hře.

Budovy mají rozdílné tvary (okupují různé množství polí v různých tvarech). Tato pole musí být volná, aby mohla být budova na zvolené místo postavena. Některá tato pole jsou po postavení volně průchozí. Pro zjednodušení budou u budov zmíněny jen rozměry obdélníku, do kterého se celá budova vejde a je nejmenší možný. Rozměry budou uváděny dvěma čísly oddělenými znakem „×“, kde první číslo určuje velikost budovy ve vodorovném směru a druhé ve svislém. [18]

Následuje popis budov dostupných všech hráčům: [18]

Construction Yard Konstrukční základna je velice důležitá budova, bez které není možné stavět další budovy. Produkuje 20 energie. Nemůže být postavena přímo, ale vzniká přeměnou z MCV (toto vozidlo je popsáno v následující podsekci 2.6.3.3). Tuto budovu lze vylepšit s cenou 1000 a trváním 25 sekund. Při nedostatku energie se doba stavby nových budov ztrojnásobí. Rozměry budovy jsou 3×3.

Concrete Slab Pokud budovy nejsou postaveny na betonové desky, tak jim chybí polovina životů. V případě, že je budova postavena na betonových deskách jen částečně, tak je zraněna méně, proporcionálně k části budovy umístěné na skále. Pokud hráč budovu opraví, tak je v průběhu času poškozována, dokud životy neklesnou na původní úroveň. Cena budovy je 20 a doba stavby je 2,48 sekund. Rozměry budovy jsou 2×2.

Large Concrete Slab Větší varianta betonové desky. Cena budovy je 50 a doba stavby je 3,76 sekund. Pro postavení je nutné mít vylepšení konstrukční základny. Rozměry budovy jsou 3×3.

Wind Trap Elektrárna produkující 200 energie. Produkce proporcionálně klesá s počtem životů budovy. Cena budovy je 225 a doba stavby je 8,32 sekund. Rozměry budovy jsou 2×3.

Barracks Kasárny umožňují stavbu pěchoty a spotřebovávají 30 energie. Cena budovy je 225 a doba stavby je 10,72 sekundy. Tuto budovu lze vylepšit s cenou 500 a trváním 8,32 sekund. Při nedostatku energie se doba stavby nových jednotek ztrojnásobí. Pro postavení je nutné mít elektrárnu. Rozměry budovy jsou 2×3.

Spice Refinery V rafinérii se mohou vykládat harvestery. Při jejím postavení hráč dostane navíc jeden harvester. Budova spotřebovává 75 energie, její cena je 1500 a doba stavby je 25 sekund. Pokud hráč přijde o poslední harvester a má alespoň jednu rafinérii, tak je mu transportován nový zdarma. Budova má kapacitu na koření v ceně 2000. Pro postavení je nutné mít elektrárnu. Rozměry budovy jsou 3×3.

Silo Silo má kapacitu koření v ceně 1500 a spotřebu 15 energie. Koření je automaticky rovnoměrně rozděleno mezi všechny budovy, které můžou koření skladovat (rafinérie a sila). Cena budovy je 120 a doba stavby je 6,24 sekund. Pro postavení je nutné mít rafinérii. Rozměry budovy jsou 1×1.

Light Factory Lehká továrna produkuje lehká vozidla a spotřebovává 125 energie. Cena budovy je 500 a doba stavby je 12,84 sekund. Tuto budovu lze vylepšit s cenou 400 a trváním 10,72 sekund. Při nedostatku energie se doba stavby nových jednotek ztrojnásobí. Pro postavení je nutné mít rafinérii. Rozměry budovy jsou 3×3.

Heavy Factory Těžká továrna produkuje těžká vozidla a spotřebovává 150 energie. Cena budovy je 1000 a doba stavby je 30 sekund. Tuto budovu lze vylepšit s cenou 800 a trváním 18,72 sekund. Při nedostatku energie se doba stavby nových jednotek ztrojnásobí. Pro postavení je nutné mít rafinérii. Rozměry budovy jsou 3×4.

Outpost Radar umožňuje hráči používat minimapu. Budova spotřebovává 125 energie, její cena je 750 a doba stavby je 12,48 sekund. Při nedostatku energie budova nefunguje. Pro postavení je nutné mít kasárny. Rozměry budovy jsou 3×3.

Starport Vesmírné letiště je schopné produkovat velké množství jednotek běžně vytvářených v jiných produkčních budovách, v tomto případě ale bez nutnosti splňovat jejich požadavky na postavení. Jejich produkce je 2,12krát delší a ceny jsou zvýšeny (ty jsou vypsány jednotlivě u jednotek popsaných v podsekci 2.6.3.3). Budova spotřebovává 150 energie, její cena je 1500 a doba stavby je 25 sekund. Pro postavení je nutné mít těžkou továrnu a radar. Rozměry budovy jsou 3×3.

Concrete Wall Zeď znemožňuje průchod jednotek a blokuje soupeřovu střelbu. Cena budovy je 20 a doba stavby je 2,48 sekund. Pro postavení je nutné mít kasárny. Rozměry budovy jsou 1×1.

Gun Turret Dělová věž střílí po soupeři dělem efektivním především proti vozidlům. Budova spotřebovává 50 energie, její cena je 550 a doba stavby je 10,72 sekund. Pro postavení je nutné mít kasárny. Rozměry budovy jsou 1×1.

Rocket Turret Raketová věž střílí po soupeři raketami schopnými zasáhnout i letecké jednotky. Budova spotřebovává 60 energie, její cena je 750 a doba stavby je 12,48 sekund. Při nedostatku energie budova nefunguje. Pro postavení je nutné mít radar a vylepšenou konstrukční základnu. Rozměry budovy jsou 1×1.

Repair Pad Opravovací plošina umožňuje opravu všech vozidel, pokud se na ní zastaví. Budova spotřebovává 50 energie, její cena je 800 a doba stavby je 15 sekund. Pro postavení je nutné mít těžkou továrnu a její vylepšení. Rozměry budovy jsou 3×3.

High Tech Factory Letecká továrna produkuje transportní letadla a umožňuje stavbu pokročilých jednotek. Budova spotřebovává 75 energie, její cena je 1150 a doba stavby je 18,72 sekund.

Frakce Atreidů může tuto budovu vylepšit s cenou 1500 a trváním 37,48 sekund, to umožní použít letecký úder (speciální schopnost). Při nedostatku energie se doba stavby nových jednotek ztrojnásobí. Pro postavení je nutné mít radar. Rozměry budovy jsou 3×4.

IX Research Center Výzkumné středisko umožňuje stavbu pokročilých budov a jednotek. Budova spotřebovává 175 energie, její cena je 1000 a doba stavby je 12,48 sekund. Pro postavení je nutné mít radar, těžkou továrnu a její vylepšení. Rozměry budovy jsou 3×4.

Palace Palác umožňuje používání speciálních schopností (s výjimkou leteckého úderu) v závislosti na frakci hráče. Budova spotřebovává 200 energie, její cena je 1600 a doba stavby je 37,48 sekund. Při nedostatku energie budova nefunguje (zastaví se odpočítávání času do možnosti použití schopnosti). Pro postavení je nutné mít výzkumné středisko. Rozměry budovy jsou 3×3.

2.6.3.3 Jednotky

Tato podsekce obsahuje popis všech jednotek ve hře rozdělených do čtyř částí podle kategorií vyjmenovaných v části popisující herní objekty.

Následuje popis pěchoty: [18]

Light Infantry Lehká pěchota je vybavena kulomety silnými proti pěchotě, ale slabými proti vozidlům. Jednotku lze stavět v kasárnách s cenou 50 a dobou stavby 2,48 sekund.

Trooper Těžká pěchota je vybavena raketami silnými proti vozidlům, ale slabými proti pěchotě. Jednotku lze stavět v kasárnách s cenou 90 a dobou stavby 3,4 sekund. Pro postavení je nutné mít vylepšené kasárny.

Engineer Inženýr nemá zbraně, ale je možné ho poslat do soupeřovy budovy, kterou vstupem do ní obsadí. Jednotku lze stavět v kasárnách s cenou 400 a dobou stavby 5 sekund. Pro postavení je nutné mít vylepšené kasárny.

Thumper Infantry Úderník nemá zbraně, ale je schopen přilákat písečného červa. Tato jednotka není dostupná v kampaňových misích. Jednotku lze stavět v kasárnách s cenou 200 a dobou stavby 5 sekund. Pro postavení je nutné mít vylepšené kasárny.

Grenadier Granátník je vybaven granáty silnými proti pěchotě a budovám, ale slabými proti vozidlům. Pokud jednotka zemře, tak je šance, že vybuchne. Tato jednotka není dostupná v kampaňových misích. Jednotku lze stavět v kasárnách s cenou 80 a dobou stavby 3,76 sekund. Pro postavení je nutné mít leteckou továrnu a vylepšené kasárny. Jednotka je dostupná pouze Atreidům.

Fremen Fremen je vybaven útočnou puškou a raketami silnými proti pěchotě a lehce obrněným vozidlům. Jednotka je mimo boj neviditelná (lze ji vidět, pokud střílí nebo ji někdo zranil). Jednotku lze stavět pouze v paláci pomocí speciální schopnosti rekrutující fremeny. Jednotka je dostupná pouze Atreidům.

Sardaukar Sardaukar je vybaven kulometem a raketami silnými proti pěchotě a lehce obrněným vozidlům. Tato jednotka není dostupná v kampaňových misích, s výjimkou poslední mise za Harkonneny po obsazení paláce. Jednotku lze stavět v kasárnách s cenou 200 a dobou stavby 6,4 sekund. Pro postavení je nutné mít leteckou továrnu a vylepšené kasárny. Jednotka je dostupná pouze Harkonnenům.

Saboteur Sabotér nemá zbraně, ale je možné ho poslat do soupeřovy budovy, kterou vstupem do ní spolu se sebou zničí. Jednotka je při nečinnosti neviditelná. Jednotku lze stavět pouze v paláci pomocí speciální schopnosti rekrutující sabotéra. Jednotka je dostupná pouze Ordosům.

Následuje popis lehkých vozidel: [18]

Trike Trojkolka je neobrněné (jsou proti němu užitečnější protipěchotní zbraně než protitankové) rychlé průzkumné vozidlo vybavené kulometem silným proti pěchotě, ale slabým proti vozidlům. Jednotku lze stavět v lehké továrně s cenou 300 a dobou stavby 9 sekund a na vesmírném letišti s cenou 315. Jednotka je v lehké továrně dostupná pouze Atreidům a Harkonnenům, ale na vesmírném letišti je dostupná všem.

Missile Quad Čtyřkolka je slabě obrněné rychlé průzkumné vozidlo vybavené raketami silnými proti vozidlům, ale slabými proti pěchotě. Jednotku lze stavět v lehké továrně s cenou 400 a dobou stavby 12,84 sekund a na vesmírném letišti s cenou 500. Pro postavení je nutné mít vylepšenou lehkou továrnu.

Raider Trike Nájezdnická trojkolka je, co do poškození, rychlosti i počtu životů, vylepšená varianta trojkolky. Silná je proti pěchotě a lehce obrněným vozidlům, ale slabá proti těžce obrněným vozidlům. Jednotku lze stavět v lehké továrně s cenou 350 a dobou stavby 9 sekund. Jednotka je dostupná pouze Ordosům.

Stealth Raider Trike Neviditelná nájezdnická trojkolka je vylepšená varianta nájezdnické trojkolky o neviditelnost působící mimo boj, ale její cena je 400. Pro postavení je nutné mít leteckou továrnu a vylepšenou lehkou továrnu. Jednotka je dostupná pouze Ordosům.

Následuje popis těžkých vozidel: [18]

Mobile Construction Vehicle (MCV) Mobilní konstrukční základna (MCV) je slabě obrněné vozidlo schopné jednosměrné přeměněny na konstrukční základnu, pokud je na skále, kde je prostor alespoň 3x3. Betonové desky se vytvoří pod budovou automaticky. Jednotku lze stavět v těžké továrně s cenou 2000 a dobou stavby 30 sekund a na vesmírném letišti s cenou 2500. Pro postavení je nutné mít opravovací plošinu a vylepšenou těžkou továrnu.

Spice Harvester Harvester je neozbrojené těžce obrněné vozidlo schopné sběru koření z písku a jeho vyložení v rafinérii. Jednotku lze stavět v těžké továrně s cenou 1200 a dobou stavby 25 sekund a na vesmírném letišti s cenou 1500. Pro postavení je nutné mít rafinérii.

Combat Tank Bojový tank je těžce obrněné vozidlo vybavené kanónem silným proti vozidlům, ale slabým proti pěchotě. Jednotku lze stavět v těžké továrně s cenou 700 a dobou stavby 17,28 sekund a na vesmírném letišti s cenou 875. Jednotka je dostupná všem frakcím, ale má u nich mírně odlišné vlastnosti. Varianta Harkonennů má více životů, ale je pomalejší, varianta Ordosů má méně životů, ale je rychlejší a varianta Atreidů je vlastnostmi přibližně mezi předchozími variantami.

Siege Tank Obléhací tank je slabě obrněné vozidlo s velkým dostřelem vybavené dělem silným proti pěchotě a budovám, ale slabým proti vozidlům. Jednotku lze stavět v těžké továrně s cenou 700 a dobou stavby 15 sekund a na vesmírném letišti s cenou 1075. Pro postavení je nutné mít vylepšenou těžkou továrnu.

Missile Tank Raketový tank je neobrněné vozidlo s velkým dostřelem vybavené raketami silnými proti vozidlům, budovám a letadlům, ale slabými proti pěchotě. Jednotku lze stavět v těžké továrně s cenou 900 a dobou stavby 20,48 sekund a na vesmírném letišti s cenou 1250. Pro postavení je nutné mít výzkumné středisko a vylepšenou těžkou továrnu. Jednotka je v těžké továrně dostupná pouze Atreidům a Harkonnenům, ale na vesmírném letišti je dostupná všem.

Sonic Tank Zvukový tank je slabě obrněné vozidlo vybavené zvukovým dělem silným proti pěchotě a slabě obrněným vozidlům, ale slabým proti těžce obrněným vozidlům. Jednotku lze stavět v těžké továrně s cenou 1000 a dobou stavby 22,48 sekund. Pro postavení je nutné mít výzkumné středisko. Jednotka je dostupná pouze Atreidům.

Obrázek 2.6 OpenRA verze hry Dune 2000 v rozlišení 1366 × 768 [18]

Devastator Devastátor je těžce obrněné vozidlo vybavené plasmovými děly silnými proti většině jednotek. Vozidlo je velice pomalé, dlouho nabíjí, ale je velice silné a má hodně životů. Zároveň má možnost se přetížit, čímž se mu znemožní pohyb a po chvíli vybuchne, což zraní okolní jednotky. Jednotku lze stavět v těžké továrně s cenou 1050 a dobou stavby 25 sekund. Pro postavení je nutné mít výzkumné středisko. Jednotka je dostupná pouze Harkonnenům.

Deviator Deviátor je neobrněné vozidlo s velkým dostřelem vybavené raketami schopnými dočasně změnit hráče ovládajícího zasažené vozidlo na vlastníka deviátoru. Jednotku lze stavět v těžké továrně s cenou 1000 a dobou stavby 22,48 sekund. Pro postavení je nutné mít výzkumné středisko. Jednotka je dostupná pouze Ordosům.

Následuje popis leteckých jednotek: [18]

Carryall Transportní letadlo je letecká jednotka schopná transportu harvesterů podle jejich potřeby (od rafinérie ke koření nebo naopak) a poškozených vozidel, pokud je jim rozkázáno dojet na opravovací plošinu a opravit se. Jednotku lze stavět v letecké továrně s cenou 1100 a dobou stavby 30 sekund.

Ornithopter Ornitoptéra je letecká jednotka nesoucí výbušniny při použití schopnosti leteckého náletu. Jednotka je dostupná pouze Atreidům.

2.6.3.4 Ovládání hry

Uživatel hru ovládá především pomocí myši, případně pro urychlení lze některé akce dělat klávesnicí. Obrazovka, kterou uživatel vidí (viz obrázek 2.6, je rozdělena na část, kde je vidět herní mapa, a na část zobrazující uživatelské rozhraní, která je z větší části napravo. [18]

Levé tlačítko se používá k vybírání objektů na mapě nebo naopak ke zrušení jejich vybrání, a k umisťování budov (pokud uživatel předtím klikl na ikonu s dostavěnou budovou). Pravým tlačítkem se vybraným objektům dávají rozkazy. Výchozím rozkazem pro jednotky je pohyb, pokud

na pole lze vstoupit a je prázdné, útok, pokud je pole obsazené soupeřovým objektem, nebo použití speciální akce, pokud je pole obsazené jednotkou, která nějakou takovou akci má dostupnou (jsou to přeměna MCV na konstrukční základnu, aktivace úderníka a přetížení devastátoru). [18]

Uživatel má možnost vybrat jeden herní objekt tím, že na něj klikne na mapě, a následně mu dávat rozkazy, nebo může vybrat celou skupinu herních objektů najednou a dávat jim rozkazy. Způsobů, jak toho dosáhnout, je několik, například může kliknutím a tažením určit oblast, ve které vybere veškeré herní objekty s nejvyšší prioritou označení (každý herní objekt má tuto hodnotu předdefinovanou pro usnadnění ovládání, například pokud jsou v oblasti jednotky schopné boje a harvester, tak se vyberou dané jednotky bez harvesteru), nebo dvojklikem na herní objekt se vyberou veškeré objekty jeho typu na obrazovce. Pokud má již uživatel nějaké objekty vybrané a při dalším vybírání je stisklá klávesa „Shift“, tak se přidají mezi vybrané. [18]

Pravá část uživatelského rozhraní ukazuje nad minimapou zleva doprava zdroje (součet peněz i koření), odehraný čas a aktuální stav energie (rozdíl výroby a spotřeby všech budov). Minimapa je hráči dostupná pouze, pokud má postavený radar a má dostatek energie. Hned pod minimapou je pět tlačítek, zleva doprava jsou to oprava budov, prodání budov, zobrazení menu hry (kulaté tlačítko uprostřed), které zároveň pozastaví hru, signalizace spoluhráčům a deaktivace budov. Tlačítka pro opravu, prodání a deaktivaci budov po stisknutí umožní následným klikáním na budovy na mapě provést vybranou akci. Tyto akce byly popsány v podsekci 2.6.3.1. Tlačítko pro signalizaci spoluhráčům umožní následně kliknout někam na mapu a zobrazit tam dočasný ukazatel viditelný pouze hráčům ve stejném týmu. [18]

Následuje část uživatelského rozhraní umožňující hráči stavbu. Levý sloupec malých ikon umožňuje výběr fronty (ikony jsou ve stejném pořadí jako byly fronty popsány v kapitole 2.6.3.1). Podle výběru fronty jsou zobrazeny ikony budov, vylepšení nebo jednotek v řadách po třech vyplňující daný prostor. Kliknutím levým tlačítkem myši na ikonu se stavba zařadí do fronty, pravým tlačítkem lze stavbu pozastavit a dalším kliknutím zrušit, v případě, že se jednalo jen o stavbu ve frontě, tak se z fronty odstraní. [18]

Speciální schopnosti, pokud jsou hráči nějaké dostupné, jsou vlevo nahoře. [18]

Vlevo dole je panel s možnostmi pro ovládání jednotek. Tyto možnosti vždy ovlivňují pouze aktuálně vybrané jednotky. Větší ikony, začínající u kraje obrazovky, jsou rozkazy jednotkám, při jejich zvolení se rozkaz okamžitě provede, nebo je nutné kliknout tlačítkem myši udávajícím rozkazy na mapu a jednotky ho teprve pak začnou provádět. Zleva doprava jsou to: [18]

Attack Move Útočný pohyb umožňuje hráči dát jednotkám rozkaz pohybu, při kterém napadnou jakoukoli soupeřovu jednotku, která je při cestě v dosahu střelby. Po jejím zničení pokračují dál v cestě.

Force Move Vynucený pohyb umožňuje hráči dát jednotkám rozkaz pohybu i na místo, kde je soupeřova jednotka (vhodné pro přejíždění pěchoty).

Force Attack Vynucený útok umožňuje hráči dát jednotkám rozkaz útočit na zvolené pole na mapě.

Guard Chránit umožňuje hráči dát jednotkám rozkaz následovat zvolenou jednotku.

Deploy Aktivovat umožňuje hráči dát jednotkám rozkaz použít svou speciální akci, pokud mají nějakou k dispozici.

Scatter Rozptýlit umožňuje hráči dát jednotkám rozkaz se přesunout na nějaké sousední pole na mapě (vhodné pro zachránění pěchoty před přejetím).

Stop Stop umožňuje hráči jednotkám zrušit všechny rozkazy (aktuální i naplánované).

Waypoint Mode Režim navigačních bodů umožňuje hráči řetězit jednotkám rozkazy do fronty. Jednotky začnou provádět následující rozkaz až po dokončení předchozího.

Následující čtyři menší ikony nastavují jednotkám jejich chování, pokud nemají žádný rozkaz: [18]

Attack Anything Stance Jednotky začnou útočit na soupeřovy jednotky v dosahu a budou je i případně pronásledovat.

Defend Stance Jednotky začnou útočit na soupeřovy jednotky v dosahu, ale nebudou se pohybovat.

Return Fire Stance Jednotky začnou útočit na soupeřovy jednotky v dosahu pouze, pokud jsou samy napadeny. Zároveň se nebudou pohybovat.

Hold Fire Stance Jednotky nebudou nikdy bez přímého rozkazu útočit na soupeře, ani se hýbat.

Harvesterům lze dávat rozkazy manuálně, ale pracovat dokáží i samy (tedy těžit koření a vykládat ho do rafinerií). Transportní letadla nemůže hráč přímo ovládat, pracují vždy automaticky. To nastává, pokud se harvester potřebuje přesunout na větší vzdálenost nebo pokud hráč dá rozkaz poškozenému vozidlu se jet opravit. [18]

2.6.3.5 Výchozí AI

Tato podsekce se zaměřuje pouze na počítačem řízené hráče v potyčkách, kde jsou vyrovnané počáteční podmínky. Tento počítačem řízený hráč bude označován jako umělá inteligence (AI). Nevztahuje se na hráče v kampaňových misích, který se řídí speciálními pravidly určujícími například kdy, kudy a čím má útočit. [18]

Umělá inteligence je pro všechny hry, vytvořené v OpenRA, univerzální (je řízená stejným kódem). Liší se pouze její konfigurace ve dříve zmíněných YAML souborech popisující pravidla hry. Tato konfigurace obsahuje především informace potřebné pro její fungování v dané hře, například jména určitých budov sdílející nějaké vlastnosti jako třeba rafinérie, sila, obranné budovy, budovy produkující jednotky atd., nebo třeba jednotky, které nemají být posílány do útoku nebo mají být chráněny (MCV, harvester atd.). [18]

Umělá inteligence má jen velice omezené možnosti úpravy jejího chování. Jedná se například o: [18]

- nastavení minimální požadované hodnoty energie (AI nepostaví budovu tak, aby klesla pod tuto hodnotu)
- nastavení velikosti útočných skupin
- nastavení limitu počtu budov a jednotek (pokud limit není nastaven, pak počet není omezen)
- nastavení časového limitu, tedy doby hry, která musí uběhnout, aby mohla být daná budova nebo jednotka postavena (pokud limit není nastaven, pak může být budova nebo jednotka postavena kdykoli)
- nastavení poměru limitujícího počet konkrétních budov a jednotek vůči celkovému počtu budov a jednotek (například může být nastaveno, že určitá budova nesmí být postavena, pokud jí hráč má poměrně vůči všem jeho budovám vícekrát než je tímto zadáno, stejným způsobem platí i pro jednotky, hodnoty jsou nastavovány jako procenta)

Řízení stavby se liší pro fronty na budovy a vylepšení, a pro fronty produkující jednotky. Následující popis chování je popsán konkrétně pro AI v Dune 2000. [18]

Používání fronty na budovy a vylepšení se řídí následujícími pravidly v tomto pořadí (pokud zmíněná podmínka není splněna, zkouší se další pravidlo, pokud nemůže být nic postaveno, tak se stavba zkusí později znovu): [18]

1. Postav elektrárnu, pokud máš méně energie než definovaná hodnota a tato budova existuje v dané frontě.

2. Postav rafinérii, pokud jich má hráč méně než jednu, tato budova existuje v dané frontě a po postavení budovy bude mít hráč více energie než definované minimum (pokud toto je jediná nesplněná podmínka, tak se postaví elektrárna).

3. Postav budovu produkující jednotky, pokud máš více koření než definovaná hodnota, nějaká taková budova existuje v dané frontě a po postavení budovy bude mít hráč více energie než definované minimum (pokud toto je jediná nesplněná podmínka, tak se postaví elektrárna).

4. Postav silo, pokud máš z 80 % zaplněnou kapacitu, tato budova existuje v dané frontě a po postavení budovy bude mít hráč více energie než definované minimum (pokud toto je jediná nesplněná podmínka, tak se postaví elektrárna).

5. Postav náhodnou budovu, která splňuje následující podmínky:

   - má definovaný limitující poměr počtu budov a ten není překročen
   - hráč ji může postavit
   - je splněn časový limit
   - po postavení budovy bude mít hráč více energie než definované minimum (pokud toto je jediná nesplněná podmínka, tak se postaví elektrárna)

Po dostavění budovy je vybráno místo na mapě, kam bude budova umístěna. Pokud se jedná o rafinérii, tak je vybráno místo nejblíže k některému z polí obsahujících koření v definovaném dosahu. Pokud se jedná o definované obranné budovy, tak co nejblíže k nejbližší soupeřově budově. Pokud cokoli jiného, tak náhodně. Vybrané místo musí mít terén, na kterém může být budova postavena, a musí toto místo být volné (nesmí na něm tedy stát žádná budova ani jednotka), pokud neexistuje žádné takové místo, stavba je zrušena a vybrána nějaká znovu (může se vybrat i ta samá). [18]

Při vybírání místa pro budovu není kontrolováno, zda vznikne uzavřená oblast. Může se tedy stát, že nějaká budova produkující jednotky bude v této uzavřené oblasti a postavené jednotky tedy nebudou moci odejít. Další věc, kterou umělá inteligence nedělá při stavbě budov, je používání betonových desek. Pro většinu budov to znamená pouze poloviční počet životů důležitý jen při napadení soupeřem, ale v případě elektráren to způsobí poloviční výrobu energie a tedy nutnost jich stavět dvakrát tolik. [18]

Používání front na jednotky se řídí následujícími pravidly v tomto pořadí (pokud zmíněná podmínka není splněna, zkouší se další pravidlo, pokud nemůže být nic postaveno, tak se stavba zkusí později znovu):

1. Postav harvester, pokud jich je méně než rafinérií a tato jednotka existuje v dané frontě a hráč ji může postavit.

2. Postav MCV, pokud hráč nemá žádnou konstrukční základnu ani MCV a tato jednotka existuje v dané frontě a hráč ji může postavit.

3. Postav náhodnou jednotku, která splňuje následující podmínky:

   - má definovaný limitující poměr počtu jednotek (ale nezávisí na jeho hodnotě)
   - hráč ji může postavit
   - je splněn časový limit

Nově postavené jednotky jsou považovány za čekající. Pokud jejich počet dosáhne definovaného množství, tak je z nich vytvořena útočící skupina. Všechny tyto skupiny jsou vždy v nějakém stavu a jsou pravidelně aktualizované. Při tom může být stav změněn, může se změnit cíl nebo se mohou dát jednotkám v dané skupině nové rozkazy. [18]

Útočící skupiny jsou po vytvoření ve stavu nečinnosti. Pokud skupina nemá zvolený cíl, tak si jako nový cíl zvolí nejbližší soupeřův objekt. Skupina se ze stavu nečinnosti přepne do stavu přesunu, pokud je skupina dostatečně zdravá nebo cíl a soupeřovy jednotky okolo něj nejsou silnější. Při této změně se všem jednotkám ve skupině dá rozkaz útočného pohybu na místo cíle. Pokud se při přesunu jednotky od sebe příliš vzdálí, tak je jednotka ve skupině nejblíže k cílovému poli zastavena a ostatním je dán rozkaz útočného pohybu k ní. Cílem je, aby se jednotky ve skupině od sebe příliš nevzdálily, pokud mají například příliš odlišnou rychlost pohybu. Zároveň pokud se nějakou dobu jednotka nejbližší k cílovému poli nezměnila ani nepřesunula, tak je skupina přesunuta zpět do stavu nečinnosti. Díky tomu se skupina po chvíli dá znovu do pohybu. [18]

Pokud je skupina ve stavu přesouvání, tak ve chvíli, kdy potká libovolný soupeřův objekt, se přepne do stavu bojování a je změněn cíl na aktuálně nejbližší soupeřův objekt. [18]

Skupina v bojovém stavu zvolí jako cíl nejbližší soupeřův objekt vždy, když předchozí již není živý. Každé jednotce ve skupině je dán rozkaz zaútočit na cíl, kromě jednotek, které aktuálně útočí nebo útok mají naplánovaný. [18]

Pokud jsou v kteroukoli chvíli jednotky ve skupině příliš zraněné a cíl a okolní soupeřovy jednotky jsou silnější, tak se skupina přepne do stavu útěku. V tomto stavu se všem jednotkám dá rozkaz pohybu k náhodné vlastní budově a skupina se zruší. [18]

Pokud je soupeřem napadena některá z definovaných jednotek, které mají být chráněny, nebo budova, tak se vytvoří obranná skupina ve stavu nečinnosti. Ta se následně přepne do stavu bojového, ve kterém se vybere nejbližší soupeřova jednotka jako nový cíl, pokud předchozí již není živý, a na ten se dá rozkaz útočného pohybu, pokud je hráčem vidět. Pokud vidět není, tak skupina chvíli nic nedělá, a pokud nezačne být vidět, tak je zrušena. [18]

Pokud má umělá inteligence dostupné a připravené nějaké speciální schopnosti, tak se vyhodnotí, jak je nejlépe použít. Schopnosti vytvářející jednotky jsou okamžitě použity. V případě schopností způsobujících škody na zvolené místo je nejdříve nalezeno nejlepší místo pro použití a následně rozhodnuto, zda bude schopnost použita či nikoliv, podle potenciálních způsobených škod. [18]

Pro Dune 2000 jsou vytvořené tři různé umělé inteligence: [18]

- Omnius
- Vidious
- Gladius

Jejich chování se liší jen nepatrně. Jejich rozdílnosti jsou pouze v definovaných konfiguracích: [18]

- velikost útočné skupiny
- limitující poměr počtu dělových a raketových věží
- limitující poměr počtu jednotek

2.6.3.6 Rozdíly mezi originální a OpenRA verzí

Jak již bylo dříve několikrát zmíněno, tak původní verze hry se mírně liší od OpenRA verze. Změny ve většině případů neovlivňují pravidla samotné hry, tedy vlastnosti a chování objektů na mapě, ale mění pouze ovládání hry. Tyto změny hru zjednodušují a zavádí standardní chování novějších RTS her. Následuje výčet nejdůležitějších rozdílů týkající se především ovládání hry:

Obrázek 2.7 Originální Dune 2000 v rozlišení 640 × 400 [20]

Attack-move Nyní hráč může, zvolením první možnosti na levém dolním panelu nebo držením klávesy „A“ a rozkazem pro pohyb, dát jednotkám rozkaz útočného pohybu. Ten byl popsán v podsekci 2.6.3.4.

Fronty stavby Hráč může nastavit stavbu do fronty. Pokud se jedná o budovu, tak se nová začne stavět až po umístění předchozí. V případě jednotek se začnou stavět okamžitě po dokončení předchozí.

Rally Point Pro každý typ budovy produkující jednotky je možné samostatně nastavit tlačítkem myši nastavujícím rozkazy bod shromáždění (anglicky Rally Point). Pokud je tento bod nastaven, tak se na něj jednotka po dostavení přesune, pokud ho hráč nenastavil, tak se zastaví před budovou.

Řetězení příkazů Pokud je při dávání rozkazu jednotce aktivní režim navigačních bodů na levém dolním panelu nebo stisknuta klávesa „Shift“, tak se rozkaz přidá do fronty rozkazů jednotky. Tedy tyto rozkazy se začnou provádět až poté, co je aktivní rozkaz dokončen.

Selektivní výběr jednotek Při výběru objektů zónou vzniklou kliknutím a tažením nelze vybrat bojové jednotky a harvestery dohromady. Primárně jsou vybírány bojové jednotky, tedy pokud ve výběru jsou bojové jednotky i harvestery, tak jsou chyceny pouze bojové jednotky.

Změna uživatelského rozhraní Původní uživatelské rozhraní je možné vidět na obrázku 2.7. Umístění nejvýraznějších prvků, jako je minimapa nebo místo obrazovky, kde se zahajuje stavba budov i jednotek, zůstalo v zásadě nezměněno. Změnila se ale konkrétní podoba. V originální hře je v pravém horním rohu minimapa, pod ní čtyři ovládací prvky (zleva jsou to tlačítka pro opravu budovy, prodání budovy, nastavení označených jednotek do Gaurd módu a stáhnutí se označených jednotek). Níže jsou dva sloupce, levý je pro stavbu budov a pravý pro stavbu jednotek. Pod každým sloupcem jsou tlačítka umožňující posun zobrazených budov v daném sloupci. Nad sloupcem stavby budov je tlačítko, které místo zmíněných dvou sloupců

zobrazí dostupná vylepšení budov, tlačítko je viditelné pouze za předpokladu, že jsou nějaká vylepšení dostupná. Nad sloupcem pro stavbu jednotek je tlačítko pro stavbu jednotek pomocí vesmírného letiště, tlačítko je dostupné pouze, pokud je budova postavena. Vlevo od sloupců pro stavbu je sloupec znázorňující aktuální stav energie.

Následují změny, které mění pravidla hry, tedy vlastnosti a chování objektů na mapě:

Deaktivace budov Hráč má možnost deaktivovat některé budovy, aby nespotřebovávaly energii. (Vysvětleno v podsekci 2.6.3.1.)

Úrovně jednotek Jednotky mohou získávat úrovně ničením soupeřových jednotek a budov. (Vysvětleno v podsekci 2.6.3.1.)

Transoprní letadla Při letu jsou transportní letadla pomalejší, ale nemusí při příletu k vozidlu, které má být přesunuto, zpomalovat ani po jeho zvednutí zrychlovat.

Mlha války Originální Dune 2000 nemá mechaniku mlhy války. (Veškerý prozkoumaný terén je již po celou misi vidět.)

Sabotér Tato jednotka frakce Ordos nebyla neviditelná jen tehdy, když stála na místě, ale byla neviditelná až po aktivování její akce. Jednotka byla neviditelná i při pohybu, ale jen po omezenou dobu.

Vesmírné letiště V originální Dune 2000 byly ceny jednotek ve vesmírném letišti proměnlivé, ale vždy levnější (cena se v průběhu času hry měnila). Získání jednotek probíhalo tak, že hráč zvolil, jaké jednotky a kolik jich chce a pak stiskem tlačítka „Purchase“ je nakoupil. Množství takto získatelných jednotek bylo počtem omezené, ale po koupení opět postupně přibývalo do určitého limitu. Hráč při nákupu musel mít dostatek zdrojů na zaplacení.

Sjednocení cen stavby Ceny budov i jednotek měly v originální Dune 2000 v potyčce a kampani jiné ceny. Ceny v OpenRA verzi (potyčce i kampani) jsou stejné jako byly původně v kampani. (V originální verzi měly budovy i jednotky v potyčce třičtvrtinovou cenu toho, co v kampani.)

Kapitola 3
Plánování v RTS

Jako první bude v této kapitole hra zanalyzovaná jako celek a budou vysvětleny některé strategické aspekty hry. Následně bude hra vnímána z hlediska uživatele. Nakonec bude popsán způsob využití plánovače pro strategické hry.

3.1 Analýza hry

Na konci minulé kapitoly byla popsána hra Dune 2000, do které bude v práci implementovaná nová AI pro hraní v potyčkách využívající HTN plánování. Tato část se bude věnovat tomu, co je náplní hry a jaké jsou vhodné strategie.

Hráčovým cílem je porazit všechny soupeře. Toho dosáhne tím, že zničí všem soupeřům veškeré jejich budovy, pokud je při spuštění hry zapnutá možnost rychlé hry, nebo zničí veškeré jejich budovy a jednotky. Z tohoto důvodu je nejdůležitější částí hry získávání zdrojů a stavba jednotek. Hráč s větší armádou má větší šanci na výhru. Vzhledem k tomu, že různé jednotky vyžadují různé postavené budovy pro svou stavbu, tak je především na počátku hry důležitá jejich stavba.

Náplň hry lze rozdělit na tři hlavní části:

1. stavba budov;
2. stavba jednotek;
3. ovládání jednotek.

První část se týká toho, jaké budovy postavit, kdy je postavit a v jakém pořadí je postavit. Tato rozhodnutí jsou prováděna především v počátcích mise a vyžadují určitou formu plánování, protože existuje velké množství požadavků pro stavbu určitých jednotek i budov. Například pokud chce hráč postavit palác, tak potřebuje výzkumné středisko, které vyžaduje radar, těžkou továrnu a její vylepšení. Tyto budovy potřebují kasárny a rafinérii, ty lze ale postavit jen, pokud je postavena elektrárna. Navíc vyjmenované budovy spotřebovávají nějaké množství energie a je tedy nutné průběžně stavět další elektrárny.

Druhá část týkající se stavby jednotek je provázaná s rozhodnutími týkajícími se stavby budov. Plánování pro stavbu budov musí zohledňovat stavbu jednotek. Ve výsledku velké množství plánování stavby budov se týká umožnění stavby nějakých vhodných kombinací jednotek. Tomu, jaké jednotky je vhodné stavět, se věnuje jedna z následujících podsekcí, konkrétně podsekce 3.1.2.

Třetí část náplně hry týkající se ovládání jednotek je vůči předchozím dvou výrazně oddělenější. Tato část obsahuje dávání rozkazů jednotkám, například k pohybu na určené místo nebo k útoku na zvolený soupeřův objekt.

Existují úkony a rozhodnutí, které nespadají ani do jedné ze zmíněných částí. Může se jednat například o používání speciálních schopností nebo třeba zahájení opravy budovy.

V následujících podsekcích budou rozebrány určité strategické aspekty hry.

3.1.1 Využívání zdrojů

Jedním z hlavních strategických rozhodnutí je poměr rozdělení zdrojů mezi stavbu bojových jednotek a rozvojem. Rozvoj může být ekonomický, tedy zrychlení získávání dalších zdrojů například v podobě postavení nového harvesteru, tak technologický, umožňující stavbu pokročilých budov a jednotek například prostřednictvím postavení výzkumného střediska. Důležité u ekonomického rozvoje je to, že nově navýšený výdělek zdrojů může být dále použit k ještě rychlejšímu ekonomickému rozvoji. Z tohoto hlediska je možné ekonomický rozvoj považovat za exponenciální.

Odpověď na otázku, jaký je nejlepší poměr rozdělení zdrojů, není jednoznačná, protože se tento poměr vyplatí v průběhu hry měnit, a to navíc v závislosti na rozhodnutí soupeře. Obecně je výhodné investovat více zdrojů do ekonomického rozvoje než soupeř, protože tím může rychleji růst výdělek a hráč tak získá větší potenciální produkci jednotek (produkci jednotek, kterou by měl, kdyby do ní investoval veškeré zdroje).

Vzhledem k tomu, že hráč o soupeřích nemá příliš velké množství informací, tak toto rozhodnutí je velice obtížné, a pokud hráč podcení množství financí, které soupeř investuje do stavby jednotek, tak se nemusí případnému útoku ubránit, a to ani díky tomu, že má více času, protože soupeřovy jednotky k němu nejdříve musí dorazit, a má možnost využít obranných věží.

Hráč tedy při tomto rozhodnutí musí buď riskovat a doufat, že soupeř nebude stavět příliš velké množství jednotek, nebo jednou za čas poslat do soupeřovy základny nějakou průzkumnou jednotku (například trojkolku), aby zjistil, zda vlastní investování do jednotek je adekvátní.

Poslední důležitou součástí rozhodování o nastavení poměru rozdělování zdrojů je možnost přestat s rozvojem a investovat veškeré zdroje do stavby jednotek (a případného technologického rozvoje) a díky tomu postavit výrazně větší množství jednotek, než má soupeř. Toto rozhodnutí může rozhodnout hru, protože buď tím bude soupeř poražen, nebo nikoliv a pak je pravděpodobné, že na tom bude ekonomicky lépe.

3.1.2 Skladba armády

Dalším důležitým strategickým rozhodnutím je, jaké jednotky se vyplatí stavět. Stejně jako u rozhodování o poměru rozdělování zdrojů, tak i v tomto případě záleží na rozhodnutích soupeře, protože některé jednotky jsou silnější proti určitým typům jednotek. Konkrétní vlastnosti jednotek byly popsány v sekci 2.6.3.3 v předešlé kapitole.

Zjednodušeně řečeno platí, že většinu bojových jednotek lze zařadit do jedné ze dvou kategorií:

- jednotky silné proti pěchotě a neobrněným vozidlům, dále souhrnně nazývané jako protipěchotní;
- jednotky silné proti lehce a těžce obrněným vozidlům, dále souhrnně nazývané jako protitankové.

Zároveň všechny tyto jednotky lze také zařadit do některé ze zmíněných cílových kategorií (pěchota, neobrněná vozidla, lehce a těžce obrněná vozidla). Ty budou dále nazývány následujícím způsobem:

- pěchota a neobrněná vozidla budou souhrnně nazývány jako neobrněné jednotky;
- lehce a těžce obrněná vozidla budou souhrnně nazývány jako obrněné jednotky.

Co se týče stavby jednotek z první kategorie, tak záleží na tom, co staví soupeř. Například pokud staví jen obrněné jednotky, vyplatí se zaměřit stavbu na protitankové jednotky. Ve výsledku

se vyplatí stavět jednotky z první kategorie v odpovídajícím poměru soupeřovým jednotkám z druhé kategorie. To je samozřejmě obtížné, protože hráč nemá mnoho informací o tom, co soupeř staví za jednotky. Pokud ale tato informace hráči nebude dostupná a sám postaví jednotky rovnoměrně z první kategorie a soupeř postaví například pouze obrněné jednotky, tak část jeho jednotek bude vysoce neefektivní a jejich stavba se mu nevyplatí. Z tohoto důvodu při volbě stavby jednotek podle první kategorie hráč musí dbát na to, co staví soupeři.

V případě volby stavby jednotek z druhé kategorii také záleží na tom, jaké jednotky staví soupeř. Tedy pokud například staví pouze protitankové jednotky, pak se vyplatí stavět neobrněné jednotky. Takováto situace ale není příliš pravděpodobná, protože je nelogické, aby soupeř stavěl armádu zaměřenou proti obrněným nebo neobrněným jednotkám, pokud by se mu to nevyplatí. Na rozdíl od rozhodování, které jednotky stavět v první kategorii, tak při stavbě jednotek z druhé může hráč zvolit libovolný poměr a je prací soupeře se tomu efektivně přizpůsobit, není tedy většinou nutné mít znalosti o soupeři. To ve výsledku znamená, že hráč může bezpečně stavět armádu smíšenou nebo i nějakým způsobem nevyváženou, pokud o tom soupeř nemá dopředu informace.

3.1.3 Ovládání jednotek

Jedním z důležitých aspektů hry je ovládání jednotek. Jak bylo popsáno v sekci 2.6.3.4 vysvětlující možnosti ovládání hry na konci minulé kapitoly, tak má hráč možnost s jednotkami libovolně hýbat. Vzhledem k tomu, že některé jednotky jsou odolnější a mají menší dostřel, zatímco jiné jsou zranitelné, ale mají velký dostřel (dále označované jako artilerie), tak je výhodnější mít vpředu odolné jednotky s malým dostřelem a za nimi artilerii.

Při střetu armád je, kromě pozice různých jednotek v armádě, také důležité, na kterou soupeřovu jednotku hráčovy jednotky střílí. Pokud soupeřova armáda obsahuje neobrněné i obrněné jednotky, pak je vhodné, aby protipěchotní jednotky stříleli na neobrněné cíle a protitankové jednotky stříleli na obrněné cíle.

Jednotky neumí automaticky vybírat pro sebe vhodné cíle. Hráč musí případně tyto cíle střelby sám upravit.

3.1.4 Analýza výchozí AI

Výchozí AI hry Dune 2000 bylo popsáno v sekci 2.6.3.5. V této sekci bude její chování shrnuto a některé části budou zanalyzovány.

Co se týče stavby budov, tak umělá inteligence postaví rafinerii a pro ni potřebné množství elektráren, a následně začne stavět budovy relativně náhodně. Jednotky staví náhodně z těch, které má dostupné na stavbu.

Ohledně používání jednotek platí, že AI vyšle jednotky do útoku, pokud se jich vyskytne v základně dostatečné, konkrétně definované množství. Tyto jednotky mají daný rozkaz útočného pohybu na nejbližší soupeřův objekt s tím, že pokud se jednotky od sebe příliš vzdálí kvůli různým rychlostem pohybu, tak se srocují na jednom místě a po chvíli dostanou nový rozkaz útoku.

Zároveň platí, že AI nijak nevyužívá minimapy a tedy radar je pro ni zbytečné stavět. Přitom ho ale může postavit, protože splňuje všechny nutné předpoklady pro jeho stavbu a tedy v určitou chvíli ho nakonec postaví. Toto chování je rozumné z hlediska uživatele, protože i když ho AI nepotřebuje, tak ho postaví a nemá tak výhodu, co se týče utracených zdrojů.

Výchozí umělá inteligence má řadu problémů, kvůli kterým není příliš efektivní. Její nejdůležitější problémy jsou uvedeny v následujícím seznamu:

- Kromě postavení jedné rafinerie na začátku hry, AI nemá žádnou vyšší prioritu stavby rafinérií a harvesterů. Jak již bylo několikrát zmíněno, tak stavba budov i jednotek je volena náhodně. Často tak nastane situace, kdy AI spotřebuje veškeré počáteční zdroje stavbou bojových jednotek a budov umožňující stavbu dalších jednotek.

- Jedním z největších problémů AI jsou stále stejně velké útočné skupiny jednotek. Lepší by bylo vysílat čím dál větší skupiny, protože pokud nějaká skupina soupeře neporazila, tak stejně velká skupina to nejspíše příště také nezvládne.
- AI neumí stavět betonové desky. Ve výsledku to znamená, že všechny jeho budovy kromě konstrukční základny mají polovinu životů. To ve většině případů není takový problém, protože jsou dané budovy akorát náchylnější na zničení při útoku. Problém ale dělají elektrárny, které kvůli tomu produkují polovinu energie, takže jich AI musí ve výsledku stavět dvakrát tolik.
- Další věcí, kterou AI nedělá, je ovládání jednotek způsobem popsaným v sekci 3.1.3. Umělá inteligence ovládá jednotky pouze tak, jak bylo vysvětleno v sekci 2.6.3.5 popisující výchozí AI v Dune 2000. Tedy pouze rozkazem útočného pohybu na soupeřovy jednotky.
- Vzhledem k tomu, jak málo často probíhá aktualizace skupin jednotek, tak když je AI napadena soupeřem (nějaká budova, MCV nebo harvester byly zraněny), reakce pomocí poslání jednotek na obranu je velice pomalá.

3.2 Reflexe lidského hráče

V minulé části byly vysvětleny problémy výchozí umělé inteligence. V této části bude zohledněn uživatel (lidský hráč), přičemž tato část bude částečně navazovat na sekci 2.6.1 analyzující cíle AI v RTS.

Ve výsledku většina zmíněných nedostatků AI není takovým problémem, protože pro uživatele je zábavné soupeře porážet. Z tohoto hlediska jsou například časté útoky relativně vhodným přístupem. Bohužel po určité době přichází monotónnost, protože takováto AI nepřináší přílišnou výzvu a lze snadno vytvořit efektivní strategii proti ní.

Pro umělou inteligenci je tedy jednou z nejdůležitějších vlastností různorodost strategií a ekonomická síla alespoň podobná uživateli, aby nebylo příliš snadné ji porazit.

Co se týče pomalé reakce na napadení a žádnému podrobnému ovládání jednotek, tak se nejedná o žádný zásadní problém, protože sám uživatel tyto úkony nemusí stíhat efektivně řešit. Například pokud by AI efektivně volila cíle pro všechny bojující jednotky, tak by při srážce většího množství jednotek bylo pro uživatele nemožné dosáhnout stejného výsledku.

Další důležitou vlastností umělé inteligence je to, že zvládá dělat rozhodnutí souběžně se hrou. V RTS hrách není možné, aby její rozhodování zastavilo hru. Vzhledem k tomu, že plánování není dostatečně rychlé, aby bylo zahájeno a ukončeno mezi dvěma aktualizacemi hry, tak bude muset být spouštěno asynchronně se hrou.

3.3 Aplikace plánování pro OpenRA

V této sekci bude rozebráno, na co bude plánování v OpenRA hrách použité. V sekci 3.1 byla náplň hry rozdělena na tři hlavní části (stavba budov, stavba jednotek, ovládaní jednotek). Toto rozdělení bude v této sekci využité.

Jak již bylo zmíněno, tak stavba budov a jednotek vyžaduje určité množství plánování kvůli relativně složitým předpokladům na jejich stavbu. Navíc při stavbě jednotek je plánování vhodné i kvůli tomu, že hráč by měl stavět jednotky v určitém poměru vůči sobě, jak bylo vysvětleno v sekci 3.1.2. Toho lze pomocí plánování dosáhnout. Z těchto důvodů bude plánování aplikováno na stavbu budov i jednotek.

Co se týče ovládání jednotek, tak je v určitých případech nutná relativně rychlá reakce, která při použití plánovacího systému v podobě externího programu nemusí být dostatečná. Z tohoto důvodu plánovač nebude mít na starosti konkrétní ovládání jednotek, ale bude mít při ovládání jednotek na starosti jen nejdůležitější rozhodnutí týkající se vyslání jednotek do útoku.

Konkrétně kdy zaútočit a s jakými jednotkami. Toto rozhodnutí je navíc značně provázané se stavbou jednotek.

V práci bylo několikrát upozorněno na to, aby AI používala různé strategie, aby uživateli nestačilo vymyslet jednu vhodnou protistrategii, kterou by mohl opakovaně využívat. Plánování ale normálně žádnou náhodnost neobsahuje, a je tedy nutné, aby plánovač obsahoval určitou množinu strategií, mezi kterými si na začátku hry náhodně vybere pomocí určitého externě dodaného náhodného rozhodnutí.

Tyto zadefinované strategie by měly obsahovat rozhodnutí týkající se stavby budov, stavby jednotek a kdy vyslat jednotky do útoku. Stavba jednotek by ideálně neměla být v plánu konkrétní, protože to, které jednotky stavět, by mělo záviset na soupeřových jednotkách.

Vzhledem k tomu, že AI soupeře nepovažuje za jiné agenty, ale pouze za součást dynamického prostředí, tak, jak již bylo dříve zmíněno, bude třeba plán opravovat. Například ztracený harvester by měl být znovu postaven. Plánovací systém JSHOP2 žádný způsob opravy plánu nemá a tedy bude opakovaně znovu plánováno, ale s jinými počátečními atomickými formulemi (určujícími počáteční stav plánování).

Konkrétnější způsob propojení plánovače se hrou bude vysvětlen v následující kapitole. Části nové AI, které nebudou používat plánování, budou využívat výchozí AI. Díky tomu bude možné relativně dobře porovnávat AI používající plánování vůči výchozí AI.

Kapitola 4
Návrh herních AI

V této kapitole budou jako první popsány obecné vlastnosti nových AI. Následně bude popsána první z nich, označována jako RBAI. Pro ni bude vysvětlena doména a problém plánování následované příslušným ovladačem a možnými rozšířeními. Poté bude vysvětlena druhá AI, označovaná jako IBAI, obdobným způsobem jako první zmíněná.

4.1 Obecné vlastnosti nových AI

V této práci jsou vytvořeny dvě nové AI, označované jako RBAI a IBAI. Obě využívají výchozí AI a modifikují její chování jen do té míry, jaká je potřeba. Obecně je změněno, kdy se začne stavět, co se začne stavět, kdy se pošlou jednotky do útoku a které to budou.

Obě nové AI jsou vytvořené tak, aby je bylo potenciálně možné využít pro kteroukoliv hru v OpenRA. Toho je dosaženo tak, že ovladač dané AI dokáže pracovat s různými doménami. Ty je následně potřeba vytvořit unikátní pro každou hru.

Vzhledem k tomu, že plánování nějakou dobu trvá a běží asynchronně s během hry, tak aby zpočátku neztrácely čas, obě nové AI vždy začnou na začátku mise stavbou elektrárny následované rafinérií (toto je nezávislé na plánovači).

4.2 Návrh nové AI pro Dune 2000

Tato sekce a následující podsekce budou popisovat fungování nové AI pro Dune 2000 označované jako zdrojově založená AI (Resourced Based AI, RBAI). Cílem této AI bude relativně přesné modelování hry tak, aby plánování bylo co nejpodobnější s tím, jak hra funguje.

Jak již bylo řečeno, tak bude využité HTN plánování. To znamená, že plánovač musí mít zadanou úlohu, kterou má řešit. Také byl dříve zvolen předpoklad použití dynamického prostředí místo multiagentního. Ve výsledku zadaná úloha v problému HTN plánování musí obsahovat všechny důležité úkony k vyhrání hry. Které to jsou, bylo diskutováno na konci kapitoly analyzující plánování v RTS v podsekci 3.3 řešící aplikaci plánování pro OpenRA. Jak bylo také v minulé kapitole zmíněno, tak hru a tedy i části, na které je AI aplikovaná, lze rozdělit následovně:

- stavba budov popsaná v podsekci 4.2.5;
- stavba jednotek popsaná v podsekci 4.2.6;
- ovládání jednotek reprezentované rozhodnutími o zahájení útoku popsanými v podsekci 4.2.7.

Výpis kódu 4.1 Axiomy RBAI

```
(:- (same ?x ?x) nil)

(:- (enough_power ?necPow)
  (
    (cur_power ?pow) (min_power ?mpow)
    (call >= ?pow (call + ?necPow ?mpow))
  )
)

(:- (can_wait)
  ((wait_current ?wc) (wait_limit ?wl) (call < ?wc ?wl))
)
```

Spojení těchto částí bude realizováno strategiemi popsanými v podsekci 4.2.8, následované podsekcí 4.2.9 týkající se toho, jak ovladač aplikuje plán ve hře. Popis RBAI bude zakončen v podsekci 4.2.10 řešící možná rozšíření některých nedokončených funkcionalit.

Tato zdrojově založená AI bude relativně přesně modelovat hru při plánování. Znamená to, že operátory i metody budou pracovat s některými herními mechanikami explicitně, jedná se o:

- zdroje;
- energii;
- počty herních objektů;
- čas.

4.2.1 Zdroje

RBAI bude přímo pracovat s hodnotou zdrojů. To znamená, že například operátory pro stavbu budov i jednotek budou snižovat zdroje v aktuálním stavu. Přičemž pokud jich není dostatek, tak se provede operace čekání. (Ta bude vysvětlena v podsekci 4.2.4.)

Plně určená atomická formule určující zdroje má tvar `(money <num>)`, kde `<num>` je číslo určující zdroje v aktuálním stavu. V předpokladech operátorů a metod vyžadujících informaci o množství zdrojů má atomická formule tvar `(money ?mon)`, kde `?mon` je proměnná. Tato atomická formule je při potřebě plánovačem použít daný operátor nebo metodu nahrazena zmíněnou plně určenou atomickou formulí.

Pro zjednodušení předpokladů operátorů a metod je zaveden axiom se jménem `enough_money`. Ten byl již ukázán jako příklad axiomu v podsekci 2.5.4.5 vysvětlující axiomy v JSHOP2 ve výpisu kódu 2.1. Tento axiom říká, zda aktuální množství zdrojů stačí na objekt s cenou předanou při použití axiomu v jeho hlavičce.

4.2.2 Energie

RBAI bude pracovat s hodnotu energie a bude zařizovat, aby se její hodnota nesnížila pod definovanou hodnotu tím, že v případě potřeby rozhodne o postavení další elektrárny. Energie je reprezentována plně určenou atomickou formulí, podobnou té pro zdroje, `(cur_power <num>)`, kde `<num>` je číslo určující energii v aktuálním stavu. V předpokladech se pak vyskytuje jako `(cur_power ?pow)`, kde `?pow` je proměnná.

Pro zjednodušení předpokladů operátorů a metod je zaveden axiom se jménem `enough_power`, který určuje, jestli je dostatečně velký přebytek energie pro postavení požadované budovy. Tento

Výpis kódu 4.2 Externí funkce pro výpočet délky fronty po aplikaci operátoru čekání.

```
import JSHOP2.*;

public class WaitQueue implements Calculate {
  public Term call(List l)
  {
    double queueTime = ((TermNumber)l.getHead()).getNumber();
    l = l.getRest();
    double waitLength = ((TermNumber)l.getHead()).getNumber();
    l = l.getRest();
    double buildingCount = ((TermNumber)l.getHead()).getNumber();

    double modifier = 1;
    if (buildingCount == 0)
      modifier = 0;
    else if (buildingCount == 2)
      modifier = 3.0/2;
    else if (buildingCount >= 3)
      modifier = 2;

    queueTime -= waitLength * modifier;

    if (queueTime < 0)
      queueTime = 0;

    return new TermNumber(queueTime);
  }
}
```

axiom je spolu s dalšími dvěma ve výpisu kódu 4.1. Požadované množství energie je reprezentované proměnou `?necPow` v hlavičce axiomu a aktuální množství energie je v proměnné `?pow`. Zároveň axiom zaručuje, že hodnota energie nikdy neklesne pod stanovenou mez v proměnné `?mpow`, z příslušné plně určené atomické formule `(min_power 20)` zadané v problému JSHOP2 plánování. Ve výsledku plánovač při používání tohoto axiomu nikdy nenechá energii klesnout pod 20. (Hodnota 20 byla použita při veškerém testování.)

4.2.3 Počty herních objektů

RBAI bude pracovat s konkrétními počty budov i jednotek. Operátory pro jejich stavbu budou tyto hodnoty zvyšovat. Všechny tyto počty objektů budou reprezentované pomocí plně určené atomické formule ve tvaru:

```
(cnt <objectName> <num>),
```

kde `<objectName>` je jméno konkrétního typu herního objektu a `<num>` je číslo určující, kolikrát je tento objekt hráčem vlastněný. Výjimku tvoří jednotky poslané do útoku, ty se již nepočítají.

Vzhledem k tomu, jak OpenRA pracuje s vylepšeními budov, tak je tímto způsobem také reprezentované provedení daného vylepšení, i přestože je ho možné udělat pouze jednou. (Například vylepšení kasáren je reprezentované atomickou formulí `(cnt upgrade_barracks 1)`.) Hodnota v těchto případech nabývá pouze hodnoty 1, a to v případě, kdy je vylepšení provedené. Pokud ještě není, tak tato atomická formule ve stavu není vůbec přítomna. Z tohoto důvodu je ve zbytku práce vylepšení považované za objekt, který lze postavit.

Výpis kódu 4.3 Metoda čekání v RBAI.

```
;;Wait method
(:method (m_wait)
  ;;Branch with enough carryalls
  (
    (can_wait) (cnt carryall ?cntca) (cnt harvester ?cnth)
    (harvester_income_with_carryall ?hin)
    (call >= ?cntca (call * ?cnth 0.75))
  )
  ((!o_wait ?hin))
  ;;Branch without enough carryalls
  ((can_wait) (harvester_income_basic ?hin))
  ((!o_wait ?hin))
)
```

4.2.4 Čas při plánování

RBAI modeluje čas pomocí provedení operace čekání reprezentující uplynulý herní čas. Zjednodušená varianta této operace je jako ukázka operace v podsekci vysvětlující fungování JSHOP2 ve výpisu kódu 2.2. Skutečná varianta obsahuje, kromě zvýšení zdrojů, navíc dvě funkcionality.

První se týká front stavby. RBAI modeluje délku stavby budov a jednotek pomocí atomických formulí:

```
(queue <queueName> <num>),
```

kde `<queueName>` je anglický název fronty (jako například `building` nebo `infantry`) a `<num>` je číslo reprezentující, jak dlouho se v dané frontě ještě něco produkuje. Operace čekání pak tyto hodnoty snižuje podle definované délky čekání (při veškerém testovaní nastavené na 20 sekund při normální rychlosti), dále modifikované podle počtu postavených produkčních budov. Ve výsledku se nová hodnota jednotlivých front získá voláním externí funkce. Následující řádka ukazuje příklad atomické formule v seznamu formulí přidaných operátorem pro čekání (jedná se o frontu stavby budov):

```
(queue building (call WaitQueue ?queue_b ?wt ?yc)),
```

kde `?queue_b` je číslo reprezentující, jak dlouho se v dané frontě ještě něco produkuje, `?wt` je číslo udávající délku čekání a `?yc` je počet základen (tedy počet budov produkujících pro danou frontu). Funkce `WaitQueue` je v ukázce kódu 4.2. Jak bylo v sekci 2.5.4 řešící JSHOP2 zmíněné, tak tyto funkce jsou psané v jazyce Java. Tato funkce zkrátí délku fronty o čas čekání vynásobený konstantou, určenou podle počtu produkčních budov dané fronty. Pokud výsledná hodnota je menší než nula, tak je výsledek nastaven na nulu.

Druhá dodatečná funkcionalita v operátoru čekání je limitace, kolikrát může být tato operace provedena. Limit je dán atomickou formulí `(wait_limit 60)` (při veškerém testovaní nastavené na 60). Důvodem tohoto limitu je, aby v případě, kdy AI nemá žádný výdělek a nemá dostatek zdrojů na postavení nového harvesteru, se operace nevolala donekonečna. (Ve výsledku se místo chyby kvůli přetečení zásobníku vrátí prázdný plán.) Pro kontrolu, zda tato hodnota není překročena, je použitý axiom `can_wait` v ukázce kódu 4.1. Zároveň tento operátor hodnotu `<num>` v atomické formuli `(wait_current <num>)` zvýší o jedna.

Operátor čekání je využit pouze v metodě čekání ukázané v kódu 4.3. Tato metoda volá operátor čekání s různými hodnotami výdělku jednoho harvesteru za sekundu, podle počtu vlastněných transportních letadel. Pokud jich je alespoň tři čtvrtiny počtu harvesterů, tak se použije první větev (kde výdělek je při veškerém testovaní nastavený na 17,5), jinak se použije druhá větev (kde je výdělek nastavený na 14).

4.2.5 Stavba budov

Jedna z nejpodstatnějších částí strategie je pořadí stavby budov. V této podsekci budou nejdříve popsány využité axiomy, následně operátory stavby budov a nakonec metody využívající tyto operátory.

4.2.5.1 Axiomy postavených budov

Doména plánování obsahuje axiomy udávající, zda je daná budova alespoň jednou postavená. (Tyto axiomy existují také pro vylepšení, vzhledem k tomu, že provedené vylepšení je reprezentované také počtem, jak bylo zmíněno v podsekci 4.2.3.) Axiom říkající, že byla postavena alespoň jedna elektrárna, je v ukázce kódu 4.4.

Všechny tyto axiomy obsahují proměnnou `?cnt` udávající počet daných budov, která pokud je větší než nula, tak je axiom platný.

4.2.5.2 Operátory stavby budov

Doména plánování obsahuje operátory pro stavbu všech budov a jejich případných vylepšení. Jako ukázka operátorů je zvolen operátor pro stavbu kasáren ukázaný v kódu 4.4.

Operátory obsahují v předpokladech následující:

- Axiomy pro kontrolu splněných požadavků pro postavení daného objektu (kasárny potřebují pouze postavenou elektrárnu, je tedy použit axiom `wind_trap_built`).
- Atomickou formuli obsahující množství daného objektu.
- Pokud se jedná o budovu, tak atomické formule pro aktuální množství energie a kolik daná budova energie spotřebovává nebo produkuje. (Kasárny spotřebovávají energii podle atomické formule `(power_con_barracks 30)`. Pro ostatní budovy jsou vytvořené obdobné atomické formule.)
- Atomické formule pro aktuální množství zdrojů a kolik daný objekt vyžaduje zdrojů k postavení. (Cena kasáren je v atomické formuli `(cost_barracks 225)`. Pro ostatní objekty jsou vytvořené obdobné atomické formule.)
- Atomické formule s aktuální délkou fronty příslušející danému objektu a jak dlouho se objekt produkuje v jedné produkční budově. (Délka stavby kasáren je v atomické formuli `(building_time_barracks 10.72)`. Pro ostatní objekty jsou vytvořené obdobné atomické formule.)

Seznam odstraněných atomických formulí obsahuje:

- původní počet objektů;
- původní množství energie, pokud se jedná o budovu;
- původní množství zdrojů;
- původní délku konkrétní fronty.

Seznam přidaných atomických formulí obsahuje:

- počet objektů zvýšený o jedna;
- množství energie snížené o energii spotřebovávanou danou budovou, pokud se jedná o budovu;
- množství zdrojů snížené o cenu daného objektu;
- prodlouženou délku konkrétní fronty o délku stavby objektu.

Výpis kódu 4.4 Příklad axiomu určujícího, zda byla nějaká budova postavena nebo nějaké vylepšení provedeno, operátoru stavby budov a metody stavby budov v RBAI. Konkrétně se jedná o axiom postavení elektrárny, operátor pro postavení kasáren a dvě metody pro postavení kasáren.

```
(:- (wind_trap_built)
  ((cnt wind_trap ?cnt) (call > ?cnt 0))
)

(:operator (!o_build_building_barracks)
  (
    (wind_trap_built) (cnt barracks ?cnt)
    (cur_power ?pow) (power_con_barracks ?pc)
    (money ?mon) (cost_barracks ?costb)
    (queue building ?queue) (building_time_barracks ?buildtime)
  )
  (
    (cnt barracks ?cnt)  (cur_power ?pow) (money ?mon)
    (queue building ?queue)
  )
  (
    (cnt barracks (call + ?cnt 1))  (cur_power (call - ?pow ?pc))
    (money (call - ?mon ?costb))
    (queue building (call + ?queue ?buildtime))
  )
0)

(:method (m_build_barracks)
  branch1
  (
    (wind_trap_built)  (power_con_barracks ?pc) (enough_power ?pc)
    (cost_barracks ?costb) (enough_money ?costb)
  )
  ((!o_build_building_barracks))
  branch2
  ((not (wind_trap_built)))
  ((m_build_wind_trap) (m_build_barracks))
  branch3
  ((power_con_barracks ?pc) (not (enough_power ?pc)))
  ((m_build_wind_trap) (m_build_barracks))
  branch4
  ((cost_barracks ?costb) (not (enough_money ?costb)))
  ((m_wait) (m_build_barracks))
)
(:method (m_build_barracks_if_below ?limit)
  branch1
  ((cnt barracks ?cnta) (call <= (call - ?limit ?cnta) 0))
  ()
  branch2
  ()
  ((m_build_barracks))
)
```

4.2.5.3 Metody stavby budov

Doména plánování obsahuje metody pro stavbu všech budov a jejich případných vylepšení. Pro ukázání používaných metod jsou v kódu 4.4 příklady dvou souvisejících metod stavby kasáren. Tato souvislost je vždy znázorněna ve jménech těchto metod, a to tak, že jedna z metod se jmenuje stejně jako ta druhá, ale její jméno je rozšířené o `_if_below` a proměnnou `?limit`. Tato metoda s rozšířeným názvem využívá první zmíněné základní stavební metody a rozšiřuje její fungování tak, aby byla budova postavena, jen pokud její množství je nižší než tento limit.

Základní stavební metody mají různý počet větví. Pořadí a jejich obsah lze shrnout následujícím způsobem:

- První větev je vždy ideálním případem, kdy jsou splněny všechny předpoklady reprezentované axiomy v následujícím seznamu. Pokud jsou splněny všechny, tak se výsledný seznam úloh skládá pouze z operace pro postavení daného objektu.
  - Axiomy kontrolující postavení všech požadovaných budov pro stavbu. (V případně kasáren se jedná jen o postavení elektrárny kontrolované axiomem `wind_trap_built`.)
  - Axiom kontrolující dostatek energie, pokud se jedná o budovu. (Název axiomu zmíněného již v podsekci 4.2.2 je `enough_power`.)
  - Axiom kontrolující dostatek zdrojů. (Název axiomu zmíněného již v podsekci 4.2.1 je `enough_money`.)
- Následují větve v takovém počtu, kolik je různých požadovaných budov v předpokladech. (V případě kasáren se jedná o jednu větev nazvanou jako `branch2`.) Každá takováto větev obsahuje jako předpoklad to, že nějaký axiom vyžadující postavení určitého objektu není splněn. (V případě kasáren se jedná o nepostavení žádné elektrárny reprezentované logickým výrazem `(not (wind_trap_built))`.) Úlohy metody jsou pak dvě metody. První stavějící potřebnou budovu a druhá je tato metoda sama. (V případě kasáren se jedná o metody stavějící elektrárnu a následně kasárny.)
- Další větev přítomná u všech objektů, které spotřebovávají energii, řeší situaci, kdy je nedostatek energie pro postavení dané budovy tím, že úlohy metody jsou postavení elektrárny následované touto metodou. (Úlohy metody jsou pro tuto i předchozí větev stejné. To platí ale pouze pro kasárny, protože jejich požadavkem je elektrárna. Tyto větve by v tomto případě mohly být sjednoceny pomocí disjunkce předpokladů.)
- Poslední větev je vždy ta, která řeší nedostatek zdrojů pro stavbu. Úlohy metody jsou pak metoda čekání následovaná touto metodou. (Při stavbě nějakého drahého objektu může být tímto způsobem třeba čekat vícekrát před tím, než bude dostatek zdrojů.)

Tyto základní metody jsou udělané tak, aby pokud je dostupný alespoň nějaký výdělek, tak požadovaný objekt pro stavbu kompletně naplánují takovým způsobem, že výsledkem je pořadí operátorů stavby všech potřebných budov proložené potřebným čekáním na zdroje.

V ukázce kódu 4.4 je na konci rozšířená metoda využívající základní metody stavby elektrárny. Jedná se o metodu stavby podmíněnou počtem již vlastněných objektů daného typu. Tyto metody při jejich použití musí mít zadanou jednu hodnotu označenou proměnnou se jménem `?limit`. Tato hodnota určuje, zda bude metoda pro stavbu daného objektu použita nebo nikoliv. Pokud je objektů stejně nebo více než limit, tak se použije první větev, kde nejsou žádné úlohy metody. Pokud je objektů méně než limit, tak úlohy metody obsahují příslušnou základní metodu, tedy metodu, která staví daný objekt.

4.2.6 Stavba jednotek

Tato podsekce se zaměří na stavbu jednotek od operátorů udávajících postavení jedné jednotky až po metody stavějící celé armády.

Výpis kódu 4.5 Příklad operátoru stavby obléhacího tanku, metody jeho stavby a metody stavby transportních letadel v RBAI.

```
(:operator (!o_build_armor_siege_tank)
  (
    (heavy_factory_built) (upgrade_heavy_built) (cnt siege_tank ?cnt)
    (money ?mon) (cost_siege_tank ?costb)
    (queue armor ?queue) (building_time_siege_tank ?buildtime)
  )
  (
    (cnt siege_tank ?cnt) (money ?mon)  (queue armor ?queue)
  )
  (
    (cnt siege_tank (call + ?cnt 1)) (money (call - ?mon ?costb))
    (queue armor (call + ?queue ?buildtime))
  )
0)

(:method (m_build_X_siege_tank ?x)
  branch1
  ((call <= ?x 0))
  ()
  branch2
  (
    (heavy_factory_built) (upgrade_heavy_built)
    (cost_siege_tank ?costb) (enough_money ?costb)
  )
  ((!o_build_armor_siege_tank) (m_build_X_siege_tank (call - ?x 1)))
  branch3
  ((not (heavy_factory_built)))
  ((m_build_heavy_factory) (m_build_X_siege_tank ?x))
  branch4
  ((not (upgrade_heavy_built)))
  ((m_build_upgrade_heavy) (m_build_X_siege_tank ?x))
  branch5
  ((cost_siege_tank ?costb) (not (enough_money ?costb)))
  ((m_wait) (m_build_X_siege_tank ?x))
)

(:method (m_build_X_carryall_target ?target)
  ((cnt carryall ?cnta) (call <= (call - ?target ?cnta) 0))
  ()
  ((cnt carryall ?cnta))
  ((m_build_X_carryall (call - ?target ?cnta)))
)
```

Výpis kódu 4.6 Pomocné operátory v RBAI.

```
(:operator (!!assert ?g)
  ()
  ()
  (?g)
0)
(:operator (!!remove ?g)
  ()
  (?g)
  ()
0)
```

4.2.6.1 Operátory stavby jednotek

Doména plánování obsahuje operátory pro stavbu všech jednotek. Jako ukázka operátorů je zvolen operátor pro stavbu obléhacího tanku ukázaný v kódu 4.5. Jak je z ukázky vidět, tak jsou operátory stavby jednotek téměř stejné jako ty stavějící budovy v ukázce kódu 4.4. Jediným rozdílem je absence energie. Z tohoto důvodu zde nebudou operátory pro stavbu jednotek více vysvětleny.

4.2.6.2 Základní metody stavby jednotek

Doména plánování obsahuje metody pro stavbu zadaného množství konkrétní jednotky. Jak je vidět ve výpisu kódu 4.5, tak tato hodnota je v proměnné `?x`. Stavba jednotek probíhá rekurzivně, tedy pokud může, tak je postavena jedna jednotka, a pak je metoda použita znovu s `?x` o jedna nižším. Metody stavby jednotek jsou podobné těm pro stavbu budov, liší se jen v několika následujících bodech upravující fungování tak, aby bylo v případě potřeby rekurzivně postaveno větší množství dané jednotky:

- První větev je kompletně nová. Ostatní původní větve jsou odsunuty za ní. Tato větev ukončuje rekurzi v případě, kdy `?x` je nula nebo méně.
- Druhá větev (původně první) řešící ideální případ, kdy jsou splněny všechny předpoklady, má úlohy metody rozšířené o rekurzivní volání s hodnotou `?x` sníženou o jedna.
- Zbylé větve jsou obdobné těm u stavby budov. Jedná se tedy o případy, kdy není splněn nějaký předpoklad a daná větev pak zařizuje jeho splnění.

Tyto metody budou dále používány v metodách stavby armády popsané v další podsekci. Pro stavbu jednotek, které nejsou součástí armády, jako je harvester a transportní letadlo, jsou dostupné rozšiřující metody pro jejich stavbu podobné těm v případě budov. Pro stavbu harvesteru existuje metoda `m_build_harvester_if_below ?target`, která funguje obdobně jako ta pro stavbu budov (metoda pro jejich stavbu je použita s hodnotou jedna v proměnné `?x`). V případě transportních letadel je dostupná metoda s hlavičkou `m_build_X_carryall_target ?target`. Ta funguje také podobně, ale v tomto případě je v proměnné `?x` určující počet postavených transportních letadel hodnota rovná rozdílu mezi `?target` a počtem transportních letadel v současném stavu. Tato metoda je ve výpisu kódu 4.4.

4.2.6.3 Stavba armád

Cílem této podsekce je popsání metod, které umožní stavbu jednotek do zvolené cenové hranice rovnoměrně ve všech dostupných frontách, ve kterých lze stavět bojové jednotky. Tato hromadná rovnoměrná stavba potenciálně velkého množství jednotek je v práci často označována jako stavba

Výpis kódu 4.7 Metody a operátory pro stavbu armád v RBAI. (Poslední vypsaná metoda je zjednodušená pouze na pěchotu.)

```
(:method (m_build_army_target_budget ?targetBudget ?squadCodeDep)
  branch1
  ((squad_code ?sc) (same ?sc ?squadCodeDep))
  ()
  branch2
  ()
  (
    (m_calculate_army_cost)
    (m_build_army_target_budget_calculate_budget ?targetBudget)
  )
)

(:method (m_build_army_target_budget_calculate_budget ?targetBudget)
  branch1
  ((army_cost ?ac) (call < (call - ?targetBudget ?ac) 0))
  ()
  branch2
  ((army_cost ?ac))
  ((m_build_army_with_budget (call - ?targetBudget ?ac)))
)

(:method (m_build_army_with_budget ?budget)
  (
    (army_budget ?oldBudget)
    (army_infantry_iter ?iteri) ;...
  )
  (
    (!!assert (something_was_built))
    (!!remove (army_budget ?oldBudget))
    (!!assert (army_budget ?budget))

    (!!remove (army_infantry_iter ?iteri)) ;...
    (!!assert (army_infantry_iter 0)) ;...

    (!!o_remove_actor_fractions)
    (m_build_army_with_budget_set_infantry_fractions) ;...
    (m_build_army_with_budget_iter)
  )
)

(:operator (!!o_remove_actor_fractions)
  ()
  (
    (forall (?actor1)
      ((actor_fraction ?actor1) (actor_fraction ?actor1 ?fr))
      ((actor_fraction ?actor1 ?fr)))
  )
  ()
0)
```

armád. Příslušná metoda pro postavení armády se jmenuje `m_build_army_target_budget` a má v hlavičce dvě proměnné (`?targetBudget` a `?squadCodeDep`).

Vzhledem k tomu, jak složitá stavba armády je, zde bude nejdříve shrnuta pro snadnější pochopení. Popis začne od zmíněné metody `m_build_army_target_budget`. Nejdříve je pomocí cílové částky `?targetBudget` a aktuální ceny všech bojových jednotek vypočítáno, za kolik zdrojů má být armáda postavena (tato hodnota je označována jako rozpočet a je s ní vytvořena samostatná pomocná atomická formule; označení pomocná referuje na způsob jejího používání, tato formule se využívá pro dočasné uložení nějaké informace). Následně je definováno několik dalších pomocných atomických formulí pro stavbu armády. Mezi ně patří formule určující, v jakém poměru mají být jednotky stavěny (tyto poměry závisí mimo jiné na soupeřových jednotkách). Následně se použije metoda, která opakovaně rozhoduje, ve které frontě budou jednotky stavěny, podle toho, která je zrovna nejkratší. Přičemž skončí, pokud to, co mělo být postaveno, nebylo postaveno, protože hodnota rozpočtu nebyla dostatečná. Samotná stavba v jednotlivých frontách je volena podle nastavených poměrů stavby jednotek v již zmíněných pomocných atomických formulích. Samotnou stavbu konkrétní jednotky pak zařizují příslušné speciální metody pro stavbu jednotek s příslušným rozpočtem, ty mimo jiné snižují hodnotu tohoto rozpočtu.

Hlavní metoda pro stavbu armády, později používaná ve strategiích vysvětlených v podsekci 4.2.8, se jmenuje `m_build_army_target_budget` a je ukázaná ve výpisu kódu 4.7. Metoda má v hlavičce dvě proměnné. První `?targetBudget` určuje, jaká je horní hranice ceny pro stavbu jednotek. Druhá proměnná `?squadCodeDep` umožňuje podmínit stavbu jednotek tak, aby při pozdějším přeplánování poté, co byl již proveden nějaký útok, byla tato stavba ignorována. Cílem je umožnit ve strategiích postavení silného útoku, který ale nebude při každém přeplánování opakován (nebude opakována stavba velkého množství jednotek). Tohoto je dosaženo tak, že při přeplánování, pokud již byl proveden nějaký útok, je na začátku přítomná atomická formule `(squad_code <code>)`, kde `<code>` je řetězec znaků používaný k identifikaci daného útoku. Jak je ve výpisu 4.7 vidět, pokud existuje atomická formule značící provedení útoku s kódem stejným s tím, který je v `?squadCodeDep`, tak se použije první větev, která neobsahuje žádně úlohy. K tomuto se využívá axiom `same` ukázaný ve výpisu kódu 4.1. Ten je pravdivý jen tehdy, pokud jsou obě jeho proměnné v hlavičce stejné řetězce znaků. Pokud neexistuje taková atomická formule, tak se použije druhá větev obsahující dvě úlohy popsané v následujících dvou odstavcích.

První úloha `m_calculate_army_cost` je řešena metodou počítající celkovou cenu všech bojových jednotek v aktuálním stavu plánování. Tato metoda má v předpokladech atomickou formuli `(army_cost ?army_cost)` znázorňující poslední spočítanou cenu všech jednotek a atomické formule obsahující počty a ceny všech bojových jednotek. Úlohy této metody jsou řešeny dvěma pomocnými operátory `!!remove`, který má jednu proměnnou v hlavičce, podle jejíž hodnoty odstraní existující atomickou formuli se stejným jménem, a `!!assert`, který má taktéž jednu proměnnou v hlavičce, podle jejíž hodnoty přidá novou atomickou formuli. Oba operátory jsou ve výpisu kódu 4.6. Tyto metody jsou využité k odstranění původní atomické formule `(army_cost ?army_cost)` se starou cenou všech bojových jednotek a vytvoření nové formule s touto aktuální cenou. Samotná celá tato metoda není v této práci uváděna, protože je velice dlouhá kvůli výpočtu zmíněné ceny, ale je možné ji nalézt v přiloženém souboru domény plánování pojmenovaném „dune2000adv“.

Druhá úloha metody `m_build_army_target_budget` je řešena metodou počítající cenu, za kterou má být armáda postavena. Pokud je tato cena již překročena (jsou tedy vlastněny bojové jednotky s cenou vyšší než je požadováno), tak se využije první větev, která neobsahuje žádné úlohy a stavba armády tak skončí. Jinak se využije druhá větev obsahující úlohu řešenou metodou `m_build_army_with_budget`, která má jednu proměnnou v hlavičce obsahující částku, za kterou mají být postaveny nové jednotky.

Nyní, co se týče metody `m_build_army_with_budget`, tak ta má v hlavičce proměnnou `?budget` obsahující hodnotu určující cenu, za kterou mají být jednotky postaveny. Také má jen jednu neoznačenou větev. Tato metoda je ve výpisu kódu 4.7, ale je zjednodušená jen na stavbu pěchoty kvůli zkrácení délky kódu, místa odstranění jsou znázorněna komentářem s třemi

tečkami. Odstraněné jsou úlohy týkající se stavby vozidel.

V předpokladech metody jsou v nezkrácené formě čtyři atomické formule. První je rozpočet `(army_budget ?oldBudget)`. Pomocná plně určená atomická formule `(army_budget <num>)`, kde `<num>` je číslo určující, kolik zbývá zdrojů pro stavbu jednotek (nazývané jako rozpočet), je v této metodě nejdříve odstraněna s původní hodnotou v proměnné `?oldBudget` a následně přidána s novou hodnotou v proměnné `?budget` z proměnné v hlavičce. Další tři pomocné atomické formule v předpokladech jsou formule s hodnotami reprezentující číslo iterace metod stavějící daný typ jednotek (konkrétní využití bude vysvětleno později). Obdobně jako pro `army_budget` je i pro tyto atomické formule nejdříve odstraněna původní hodnota a následně přidána atomická formule v tomto případě s hodnotou nula. (Tato odstraňování jsou přítomna z toho důvody, že po dokončení stavby armády se neodstraňují pomocné formule a tedy zůstávají ve stavech s původními starými hodnotami.) Všechny tyto zmíněné atomické formule jsou pomocné k procesu stavby armády.

Kromě již zmíněných úloh jsou přítomny další, jedná se o:

- Přidání atomické formule `something_was_built` vysvětlené a použité později. (Také se jedná o pomocnou formuli.)
- Operace `!!o_remove_actor_fractions` odstraňuje všechny pomocné atomické formule ve tvaru `(actor_fraction <unit> <fraction>)`, kde `<unit>` je nějaká jednotka a `<fraction>` je pro ni nastavený požadovaný poměr při stavbě. Tedy jednotka by měla být v tomto poměru vůči ostatním jednotkám z dané fronty.
- Následují tři úlohy řešené příslušnými metodami přidávající pomocné atomické formule určující poměry stavby jednotek zmíněné v minulém bodě. Metody zde nejsou uvedeny, protože obsahují velké množství větví, které závisí na předpokladech vyjmenovaných v následujícím seznamu. Také jsou dlouhé, protože vytvářejí relativně hodně pomocných atomických formulí obsahující zmiňovaný poměr stavby.

  1. Za kterou frakci hráč hraje.
  2. Jaké budovy a vylepšení má hráč postavené.
  3. Kolik zdrojů soupeři investovali do pěchoty a kolik naopak do ostatních bojových jednotek. V podsekci 3.1.2 v minulé kapitole bylo řečeno, že by hráč měl stavět jednotky podle toho, co staví jeho soupeř. Tento bod je v kódu reprezentován dvěma axiomy `should_build_more_anti_infantry` a `should_build_more_anti_armor`. Ty jsou vysvětleny dále v textu.
- Poslední úloha `m_build_army_with_budget_iter` je hlavní, co se týče stavby jednotek, a také je vysvětlena až dále.

Axiomy zmíněné v bodě 3 v minulém seznamu jsou si podobné. První ze zmíněných je zde v práci ukázaný ve výpisu kódu 4.8. Tento axiom platí, jestliže jsou splněny všechny obsažené podmínky:

1. Podmínka `(call > ?eaci (call * ?eac ?const))` je splněna, pokud má soupeř více zdrojů v pěchotě, než ve všech jednotkách dohromady vynásobeným definovaným číslem (při veškerém testovaní nastaveným na 0,5). Toto je hlavní podmínka, při které má být stavba bojových jednotek speciálně zaměřena. (V případě veškerého testování je tato podmínka splněna, pokud má soupeř alespoň polovinu zdrojů v pěchotě.)
2. Podmínka `(call > ?eac ?const2)` je splněna, pokud je celková cena soupeřových bojových jednotek větší než definovaná hodnota (při veškerém testovaní nastaveným na 500). Důvodem této podmínky je, aby při přeplánování, při kterém soupeř nemá téměř žádné bojové jednotky, nebylo toto zaměření stavby jednotek používáno.

Výpis kódu 4.8 Metody a axiomy pro stavbu armád v RBAI. (První metoda je zjednodušena pouze na pěchotu.)

```
(:method (m_build_army_with_budget_iter)
  branch1
  ((not (something_was_built)))
  ()
  branch2
  (
    (something_was_built)  (barracks_built)
    (queue infantry ?iq) (queue vehicle ?vq) (queue armor ?aq)
    (or (call <= ?iq ?vq) (not (light_factory_built)))
    (or (call <= ?iq ?aq) (not (heavy_factory_built)))
  )
  (
    (!!remove (something_was_built))
    (m_build_army_with_budget_iter_infantry)
    (m_build_army_with_budget_iter)
  )
  ;branch3 vehicles
  ;branch4 armor
  branch5
  ()
  ()
)

(:- (should_build_more_anti_infantry)
  (
    (army_cost ?ac) (enemy_army_cost ?eac)
    (enemy_army_cost_infantry ?eaci)
    (enemy_infantry_fraction_to_change_focus ?const)
    (enemy_minimal_army_cost_to_change_focus ?const2)
    (enemy_army_fraction_to_stop_focus_change ?const3)
    (call > ?eaci (call * ?eac ?const))
    (call > ?eac ?const2)
    (call < ?ac (call * ?eac ?const3))
  )
)
```

3. Podmínka `(call < ?ac (call * ?eac ?const3))` je splněna, pokud cena hráčových bojových jednotek je nižší než cena soupeřových bojových jednotek vynásobených definovanou konstantou (při veškerém testovaní nastavené na 1,5). Tato podmínka je zde kvůli tomu, aby při dlouhodobém plánování nebyly bojové jednotky stavěny zaměřeně, i pokud je hráčova cena bojových jednotek značně větší než soupeřova.

Hlavní metoda stavějící jednotky `m_build_army_with_budget_iter` ve výpisu kódu 4.8 se rekurzivně volá, dokud nenastane situace, kdy nic nebylo a ani nemohlo být postaveno. To je indikované nepřítomností pomocné atomické formule `(something_was_built)`. V takovém případě je použita první větev s prázdným seznamem úloh. Následující hlavní tři větve (dvě z nich nejsou ve výpisu obsažené kvůli celkové délce metody) jsou používané podle dostupných produkčních budov a jejich aktuálního zaplnění. Přesněji řečeno každá ze tří větví řeší stavbu v jedné z produkčních budov (kasárny, lehká továrna a těžká továrna) a je zvolena větev odpovídající nejprázdnější frontě, přičemž jsou ignorovány fronty produkčních budov, které nejsou postaveny. Pokud je více front se stejným zaplněním, tak je zvoleno podle pořadí větví metody. Seznamy úloh těchto větví obsahují následovné:

- Odstranění atomické formule `(something_was_built)` pomocí operace `!!remove`.
- Úlohu řešící stavbu jednotek konkrétní fronty v poměru uloženém v příslušných pomocných atomických formulích. Tyto metody budou řešeny dále v textu.
- Poslední úloha je rekurzivní volání této metody.

Poslední pátá větev je vždy splněna a má prázdný seznam úloh. Tato větev je přítomna jen kvůli případům, kdy by bylo omylem ve strategii rozhodnuto o stavbě armády před postavením jakékoliv budovy produkující jednotky. Díky ní bude plán nalezen s tím, že tato stavba nebude mít žádný efekt.

Metody řešící stavbu jednotek konkrétní fronty v poměrech uložených v příslušných pomocných atomických formulích nejsou v žádném výpisu kódu v práci ze stejného důvodu, jako metody pro nastavení zmíněných poměrů (mají hodně větví a ty jsou dlouhé). Každá jejich větev řeší stavbu jednotek konkrétního typu. Stavba probíhá tak, že se vybere první větev, která splňuje to, že počet jednotek je menší než číslo získané vynásobením zmíněného poměru s číslem iterace. Toto číslo iterace se zvyšuje, pokud nebylo nic postaveno. Při tomto zvyšování iterace se tato metoda rekurzivně volá znovu. Ve výsledku je volána do té doby, než není něco postaveno. Samotná stavba jednotky, zvolené podle použité větve, probíhá pomocí speciální funkce stavby jednotky (ty budou rozebrány dále). Předpoklady zmiňovaných větví obsahují následující:

- Frakci, za kterou hráč hraje v případech, kdy jednotka není dostupná všem frakcím.
- Počet jednotek daného typu.
- Poměr `actor_fraction`, který udává, kolik dané jednotky má být postaveno v poměru s ostatními jednotkami přístupnými v dané frontě.
- Kolikátá je to iterace příslušné fronty.
- Podmínka určující, zda je číslo `(call - (call * ?actor_fraction ?iter) ?cnta)` větší než definovaná hodnota. (Při veškerém testovaní nastavené na 1 pro vozidla a na 3 pro pěchotu. Důvodem je snížení zanoření při stavbě pěchoty, které dosahovalo situací, kdy program zhavaroval na vyčerpání zásobníku.) Proměnná `?actor_fraction` udává zmiňovaný poměr požadovaných jednotek, `?iter` je číslo iterace a `?cnta` je počet jednotek daného typu.

Příkladem dříve zmíněné specifické metody stavby jednotek je metoda pro stavbu obléhacího tanku ve výpisu kódu 4.9. Tyto metody mají vždy dvě větve. První větev kontroluje, kromě potřebných budov a vylepšení, také dostatečnou velikost rozpočtu, tedy zda je hodnota rozpočtu `?b` z atomické formule `(army_budget ?b)` větší nebo rovná než cena jednotek, které mají být postaveny. Úlohy metody obsahují:

Výpis kódu 4.9 Metoda pro stavbu armád v RBAI.

```
(:method (m_build_X_siege_tank_target_with_budget ?target)
  branch1
  (
    (army_budget ?b) (cost_siege_tank ?costb) (call >= ?b ?costb)
    (heavy_factory_built) (upgrade_heavy_built)
  )
  (
    (m_build_X_siege_tank 1) (!!remove (army_budget ?b))
    (!!assert (army_budget (call - ?b ?costb)))
    (!!assert (something_was_built))
  )
  branch2
  ()
  ()
)
```

- Postavení definovaného množství jednotek. (Při veškerém testovaní nastavené na 1 pro vozidla a na 3 pro pěchotu.)
- Odstranění atomické formule rozpočtu `(army_budget ?b)` a následné opětovné přidání s novou hodnotou sníženou o cenu postavených jednotek. (Díky tomu je neustále známo, za kolik zdrojů mají být ještě jednotky potenciálně postaveny.)
- Přidání atomické formule `(something_was_built)`. (Díky této atomické formuli plánovač ví, že má pokračovat ve stavbě armády.)

Druhá větev této metody je vždy splněna a nemá žádné úlohy. Primární využití je, že poté, co nastane situace, kdy rozpočet nebude stačit na postavení dané jednotky, se využije tato větev a díky tomu nebude přidána atomická formule `(something_was_built)`. To povede k přerušení stavby armády (metoda `m_build_army_with_budget_iter` projde první větví a skončí).

Na konci této podsekce budou vysvětleny některé důvody, proč byla stavba armády vytvořená tak složitě, jak je.

Jednou z částí stavby armády bylo řešení, které jednotky mají být stavěny. To bylo vyřešeno pomocí atomických formulí určujících, v jakém poměru mají být dané jednotky stavěny. Hlavní motivací tohoto přístupu byla možnost reakce na soupeře pomocí stavby různých typů jednotek. Například RBAI tak bude stavět protipěchotní jednotky, pokud soupeř staví hodně pěchoty. Zároveň je tento přístup snadno rozšiřitelný, pokud by bylo vhodné reagovat nějak specificky na různé protihráčem zvolené strategie.

Dalším důvodem komplikované stavby armády byl požadavek na rovnoměrnou stavbu v dostupných frontách. Hlavním důvodem je, aby se při stavbě velkého množství jednotek nečekalo na nějakou frontu, zatímco jiné by byly prázdné. Zároveň zvolený způsob řeší situace, kdy se například staví harvester, ale je již požadováno začít stavět armádu. V takovém případě je fronta pro těžká vozidla již částečně zaplněna a bylo by vhodné začít stavět jednotky nejdříve v ostatních frontách.

4.2.7 Útoky

Doména plánování obsahuje metody a operátor pro vyslání útoku na soupeře. Jednotky vyslané do útoku se odečtou od jednotek, se kterými plánovač pracuje (sníží se hodnota `<num>` v `(cnt <unit> <num>)` pro daný typ jednotky `<unit>`). Hlavní metoda používaná ve strategiích se jmenuje `m_form_squad` a je ve výpisu kódu 4.10 jako první. Metoda má tři proměnné

Výpis kódu 4.10 Metody pro útok v RBAI.

```
(:method (m_form_squad ?type ?repeatCode ?fractionGame)
  ((not (squad_code ?repeatCode))  (can_attack ?actors))
  (
    (m_choose_attack_army_fraction) (m_form_squad_supp ?actors ())
    (m_form_squad_fin ?type ?repeatCode ?fractionGame)
  )
  ()
  ()
)

(:method (m_choose_attack_army_fraction)
  ((attack_army_fraction ?aaf) (enemy_cnt ?ecnt) (call > ?ecnt 1))
  (
    (!!remove (attack_army_fraction ?aaf))
    (!!assert (attack_army_fraction 0.7))
  )
  ((attack_army_fraction ?aaf) (enemy_cnt ?ecnt) (call = ?ecnt 1))
  (
    (!!remove (attack_army_fraction ?aaf))
    (!!assert (attack_army_fraction 0.9))
  )
)
```

v hlavičce. První `?type` a poslední `?fractionGame` jsou informacemi pro ovladač a nemají pro plánovač žádný význam (jsou postupně mezi metodami předávány až k příslušnému operátoru). Druhá proměnná `?repeatCode` v hlavičce je již v minulé podsekci 4.2.6.3 zmíněný řetězec znaků určující, které útoky již byly provedeny a při novém přeplánování nemají být opakovány. Toho je dosaženo tak, že metoda má dvě větve.

První větev vyžaduje, aby nebyla přítomna atomická formule `(squad_code ?repeatCode)`, kde `?repeatCode` je řetězec znaků z hlavičky této metody (tento řetězec je používaný k identifikaci daného útoku). Tímto je zajištěno, aby se při přeplánování neopakovali již provedené útoky. Zároveň je v předpokladech atomická formule `(can_attack ?actors)`. Příslušná plně určená atomická formule má na druhé pozici, která je v tomto případě v proměnné `?actors`, seznam termů. Ten obsahuje názvy všech jednotek, které mohou být vysílány do útoku. Úlohy této větve jsou tři a týkají se následujícího:

- Rozhodnutí, kolik procent jednotek má být vysláno do útoku. Rozhodnutí je určeno podle počtu soupeřů. Pokud existuje jediný soupeř, plánovač zajistí atomickou formuli s hodnotou 0,9. Pokud jich je více, hodnota je 0,7. Cílem má být, aby si hráč nechával při útoku s více oponenty větší množství jednotek na potenciální obranu. Metoda, která toto řeší, je jako druhá ve výpisu kódu 4.10.
- Rozhodnutí, kolik jednotek a které jednotky mají být poslány do útoku. Tento proces bude vysvětlen v textu dále.
- Zavolání operátoru vysílajícího útok a zajišťujícího pomocné operace. Konkrétní metoda bude také později vysvětlena.

4.2.7.1 Metody pro vybrání jednotek pro útok

Pro rozhodnutí, kolik kterých jednotek má být posláno do útoku, se využijí dvě pomocné metody ve výpisu kódu 4.11. Metody mají stejné jméno a jejich použití tedy závisí v tomto případě na

Výpis kódu 4.11 Podpůrné metody pro útok v RBAI.

```
(:method (m_form_squad_supp ((?actorCanAttack) . ?actors) ?attack_units)
  (
    (cnt ?actor ?cntHave) (same ?actor ?actorCanAttack)
    (attack_army_fraction ?aaf)
  )
  (
    (!!remove (cnt ?actor ?cntHave))
    (!!assert (cnt ?actor (call - ?cntHave
      (call MyRound (call * ?cntHave ?aaf)))))
    (m_form_squad_supp ?actors ((?actor
      (call MyRound (call * ?cntHave ?aaf))) . ?attack_units))
  )
)
(:method (m_form_squad_supp nil ?attack_units)
  ()
  ((!!assert (attack_units (?attack_units))))
)
```

plánovači. Ten ve výsledku opakovaně volá první ukázanou metodu a pouze jednou jako poslední zavolá druhou metodu ukázanou ve výpisu. Důvodem je, že při každém volání první metody je rekurzivně zavolaná metoda se stejným jménem se seznamem termů v první proměnné metody zkrácené o první prvek. (Toto oddělení prvního prvku je provedené již v hlavičce metody pomocí `((?actorCanAttack) . ?actors)`. V první proměnné bude jen první prvek seznamu a v druhé proměnné `?actors` zůstanou ostatní.) Druhá metoda je použita až ve chvíli, kdy je tento seznam termů prázdný. Zjednodušeně lze říct, že první metoda se rekurzivně používá, dokud je v seznam termů alespoň jeden prvek, a poté se použije druhá metoda.

Jako seznam termů, vstupující do této metody, je použit již dříve jmenovaný seznam všech jednotek, které mohou být vyslány do útoku. Při iterování přes tento seznam je adekvátně snížen počet jednotek daného typu (pomocí operátorů `!!remove` a `!!assert`) a stejný počet je vložen do postupně rostoucího seznamu termů obsahujícího jednotky, které půjdou do útoku, a jejich počty. Nakonec, když je seznam jednotek, které mohou být vyslány do útoku, prázdný, je pomocí druhé závěrečné metody vytvořen nový seznam jednotek s jejich konkrétními počty jako atomická formule `(attack_units (?attack_units))`, kde v proměnné `(?attack_units)` je seznam termů, kde každý term je pár jména jednotky a jejího počtu vyslaného do útoku.

Nyní podrobně k první rekurzivně volané metodě. Metoda při použití vyžaduje dva seznamy termů. První je hned v hlavičce rozdělen na první prvek obsažený v `?actorCanAttack` a zbytek v `?actors`. Druhý seznam termů (zpočátku prázdný) je v `?attack_units`. V předpokladech je jako první atomická formule `(cnt ?actor ?cntHave)`. Příslušná plně určená atomická formule s konkrétní jednotkou v `?actor` a jejím počtem v `?cntHave` se rozhodne pomocí axiomu `same`. Ten zařídí, že `?actor` a `?actorCanAttack` jsou stejné typy jednotek (stejné řetězce znaků). Díky tomu je získán konkrétní počet jednotek v `?cntHave` pro jednotku v `?actorCanAttack`. Zároveň je v předpokladech získáno do proměnné `?aaf`, jak velká část jednotek daného typu má být do útoku odeslána.

Úlohy dané metody obsahují dva pomocné operátory `!!remove` a `!!assert` pro snížení počtu konkrétní jednotky a poslední úloha má stejné jméno jako tato metoda. Jedná se o již zmíněné rekurzivní volání se zkráceným seznamem jednotek, které mohou být vyslány do útoku, a prodlouženým druhým seznamem jednotek, které půjdou do útoku, a jejich počtů.

Zjednodušeně řečeno je tedy z prvního seznamu odebrána první jednotka a je vložena do druhého seznamu tentokrát v páru s číslem reprezentujícím počet jednotek, které mají jít do útoku. Zároveň jsou tyto hodnoty odečteny z jednotek, se kterými plánovač pracuje. Toto se opakuje, dokud je v prvním seznamu alespoň jedna jednotka. Poté se použije druhá metoda,

Výpis kódu 4.12 Metoda a operátor pro útok v RBAI.

```
(:method (m_form_squad_fin ?type ?repeatCode ?fractionGame)
  ((attack_units (?actors)))
  (
    (!o_form_squad ?type ?repeatCode ?fractionGame ?actors)
    (!!remove (attack_units (?actors)))
  )
)

(:operator (!o_form_squad ?type ?repeatCode ?fractionGame ?army)
  ()
  ()
  ()
0)
```

která vytvoří atomickou formuli s druhým zmíněným seznamem.

4.2.7.2 Finální metoda útoku a příslušná operace

Co se týče poslední úlohy v hlavní útočné metodě `m_form_squad`, tak adekvátní metoda se jménem `m_form_squad_fin` je ve výpisu kódu 4.12. Tato metoda zařizuje použití operátoru `!o_form_squad` s adekvátními proměnnými tak, aby ovladač věděl vše potřebné o útoku, a zároveň odstraní atomickou formuli obsahující seznam jednotek, které půjdou do útoku, a jejich počtů.

Zmíněný operátor je také ve výpisu kódu 4.12. Tento operátor, jak je vidět, nic v plánovači nedělá. Jeho jediná funkce je předání informace ovladači o útoku spolu s potřebnými parametry. Ty jsou:

- Proměnná `?type` říká, jaký typ útočné skupiny má být vytvořen. V použité doméně byla pouze varianta útočné skupiny `assault`. Druhý existující typ skupiny `rush` nebyl pro svou nevhodnost nakonec použit a v práci nebyl obecně zmiňován vzhledem k tomu, jak nelogicky byl výchozí AI používán. Dalším důvodem této možnosti je způsob ovládání vodních a bojových leteckých jednotek přítomných v jiných hrách v OpenRA, ty mají vlastní typ útočných skupin.
- Proměnná `?repeatCode` dává řetězec znaků jedinečný pro daný útok. Důvodem je, aby ovladač mohl plánovači říct, které útoky již byly provedeny.
- Proměnná `?fractionGame` říká ovladači, jak velká část ceny jednotek daného útoku musí být postavena, aby mohl být útok vyslán (při veškerém testování nastavenou na 0,9).
- Proměnná `?army` obsahuje seznam všech jednotek a jejich počtů, které mají jít do útoku.

4.2.8 Strategie

Strategie jsou metody obsahující velké množství metod udávající chování daného umělou inteligencí ovládaného hráče. Pro testování byly vytvořeny pouze dvě strategie. Která z nich bude použita, je ovladačem rozhodnuto náhodně, a po zbytek mise je toto rozhodnutí neměnné. Jedna ze strategií je ve výpisu kódu 4.13. V následující části bude popsána. Na konci podsekce budou zmíněné rozdíly vůči druhé strategii a obecně, v čem se mohou strategie lišit.

Strategie začíná postavením jedné rafinérie následované lehkou továrnou, v té se hned začínají stavět trojkolky (s touto částkou budou tři, protože každá stojí 300; v případě ordosů to

Výpis kódu 4.13 Jedna z větví určující strategii pro RBAI.

```
  (
    (random_strategy ?rs) (call >= ?rs 0.5)
  )
  (
    (m_build_refinery_if_below 1)
    (m_build_light_factory_if_below 1)
    (m_build_army_target_budget 1000 none)
    (m_build_refinery_if_below 2)
    (m_build_upgrade_light)
    (m_build_army_target_budget 1500 basic1)
    (m_form_squad assault basic1 0.9)
    (m_build_barracks_if_below 1)
    (m_build_army_target_budget 1000 basic1)
    (m_build_outpost_if_below 1)
    (m_build_heavy_factory_if_below 1)
    (m_build_harvester_if_below 2) (m_build_harvester_if_below 3)
    (m_build_upgrade_barracks)
    (m_build_army_target_budget 2000 none)
    (m_build_harvester_if_below 4)
    (m_build_medium_gun_turret_if_below 1)
    (m_build_upgrade_heavy)
    (m_build_army_target_budget 4000 basic2)
    (m_build_harvester_if_below 5)
    (m_build_army_target_budget 6000 basic2)
    (m_form_squad assault basic2 0.9)
    (m_build_X_carryall_target 1)
    (m_build_upgrade_hightech)
    (m_build_army_target_budget 2000 basic2)
    (m_build_harvester_if_below 6)
    (m_build_large_gun_turret_if_below 1)
    (m_build_X_carryall_target 2)
    (m_build_army_target_budget 6000 late1)
    (m_build_large_gun_turret_if_below 2)
    (m_build_X_carryall_target 4)
    (m_build_research_centre_if_below 1)
    (m_build_army_target_budget 8000 late1)
    (m_form_squad assault late1 0.9)
    (m_build_army_target_budget 2500 none)
    (m_build_heavy_factory_if_below 2)
    (m_build_harvester_if_below 7)
    (m_build_X_carryall_target 5)
    (m_build_palace_if_below 1)
    (m_build_X_carryall_target 6)
    (m_build_army_target_budget 5000 none)
    (m_build_large_gun_turret_if_below 3)
    (m_build_harvester_if_below 8)
    (m_build_X_carryall_target 7)
    (m_build_heavy_factory_if_below 3)
    (m_build_army_target_budget 10000 none)
    (m_form_squad assault repeat 0.9)
    (m_build_army_target_budget 3000 none)
  )
```

budou nájezdnické trojkolky a budou pouze dvě, protože jsou dražší). Než se dostaví, tak se souběžně začne stavět druhá rafinérie a začne se vylepšovat lehká továrna. Následně se postaví další jednotky (v tomto případě se bude jednat o jednu čtyřkolku vzhledem k nastavené ceně armády; v případě ordosů budou dvě). Následuje první vyslání jednotek do útoku s řetězcem znaků `basic1` (ten umožní ovladači zařídit, že při přeplánování nebude tento útok opakován). (Pokud je jediný soupeř a hráč není Ordos, tak útok bude obsahovat všechny jednotky, jinak jen dvě trojkolky a jednu čtyřkolku. V případě Ordosů bude v každém případě do útoku vysláno vše. Důvodem je zaokrouhlení po vynásobení, pro tři jednotky se jedná buď o $3 \times 0,9 = 2,7$ zaokrouhlené na všechny tři nebo o $3 \times 0,7 = 2,1$ zaokrouhlené na dvě.)

Po prvním útoku jsou postaveny kasárny a nové jednotky pro případnou obranu. (Pěchota je pomalá, ale relativně silná, je tedy vhodná pro obranu.) Následně se postaví radar, umělá inteligence ho sice nepotřebuje, ale hráči by se při útoku nejspíše hodil. Z tohoto důvodu ho začne stavět. Následně se postaví těžká továrna a v ní harvestery, aby byly tři (AI by v této době pořád měla mít dva harvestery, protože dostala jeden ke každé rafinérii, ale mohla o nějaký přijít, například kvůli písečnému červu, a tak při přeplánování se případně dostaví nejdříve druhý.)

Následně se vylepší kasárny a postaví se armáda, která v tuto chvíli již může obsahovat bojové tanky, trojkolky, čtyřkolky a lehkou i těžkou pěchotu (v případě Ordosů nájezdnické trojkolky místo trojkolek). Potom se postaví již čtvrtý harvester, dělová věž a vylepší se těžká továrna (to umožní stavět obléhací tanky). Následně se začnou stavět jednotky pro druhý útok s řetězcem znaků `basic2`, proložený postavením pátého harvesteru. Poté je vyslán tento útok.

Zbytek plánu již bude jen stručně popsán. Nyní se staví, kromě posledních několika harvesterů, co se týče ekonomické stránky, i transportní letadla. Pokud se jedná o frakci Atreidů, tak se co nejdříve vylepší letecká továrna, aby se umožnilo používání leteckého úderu. Postaví se výzkumné středisko, další těžké továrny (až do počtu tří), raketové věže (také až tři) a později také palác. Celkově se postaví osm harvesterů a sedm transportních letadel. Průběžně se staví jednotky a je vyslán další útok. Na konci plánu je vysílán útok s cenou až 10.000, tento útok je jako jediný vysílán opakovaně.

Druhá strategie vůči této má několik rozdílů. Ty nejdůležitější jsou následující:

- zpočátku se více zaměřuje na ekonomický rozvoj;
- na obranu používá zpočátku hry především pěchotu;
- poprvé útočí značně později;
- nikdy nestaví lehkou továrnu a místo toho dříve staví další těžké továrny.

Strategie umožňují, aby umělá inteligence hrála různorodě, ale zároveň rozumně a její chování dávalo smysl. Toho lze dosáhnout pomocí používání různých metod a konkrétní volbě jejich pořadí. Zároveň platí, že strategie je aplikována tak rychle, jak se AI ekonomicky daří (pokud například brzy ztratí jeden harvester, tak se její hra zpomalí). Co se týče možností, čeho lze dosáhnout, tak následující seznam některé shrnuje.

- Možnost zaměření se na konkrétní jednotky, co se týče produkčních budov.
- Možnost měnit zaměření využití zdrojů, co se týče množství zdrojů investovaných do ekonomického rozvoje, technologického rozvoje a stavby jednotek.
- Četnost útoků a jejich síla, při které budou provedeny.

4.2.9 Ovladač

Jak již dříve v podsekci 2.4.1 řešící konceptuální model plánování bylo řečeno, tak lze daný model rozdělit na tři komponenty (zobrazené na obrázku 2.3):

1. plánovač, v aktuálním případě se jedná o JSHOP2, kterému je poskytnuta popsaná doména;

2. ovladač, jedná se o část aplikující plán získaný od plánovače, zároveň také opakovaně požaduje po plánovači nový plán z aktuálního stavu hry;

3. systém Σ, ten je v tomto případě reprezentován hrou Dune 2000.

Tato podsekce se bude zaměřovat na ovladač. Ten je vytvořen tak, aby byl univerzální pro všechny hry v OpenRA. Jeho chování lze rozdělit na dvě části. Jedna část se stará o aplikaci plánu ve hře a druhá o komunikaci s plánovačem kvůli nutnosti přeplánování. Jako první bude vysvětlena v následující podsekci druhá zmíněná část.

4.2.9.1 Komunikace s plánovačem

Plán je plánovačem ukládán jako fronta objektů (zde fronta referuje na datovou strukturu), přičemž každý z nich obsahuje všechny operátory mezi dvěma konkrétními sousedními čekáními, začátkem plánu a čekáním nebo čekáním a koncem plánu. (Hned po zpracování textového plánu od plánovače první objekt ve frontě obsahuje všechny operátory od začátku plánu do prvního čekání. Druhý objekt obsahuje všechny operátory mezi prvním čekáním a druhým čekáním a tak dále. Poslední objekt obsahuje všechny operátory mezi posledním čekáním a koncem plánu.) Zmíněný objekt je nazýván jako krok plánu a obsahuje dvě datové struktury, první obsahující frontu objektů, které mají být postaveny, pro každou stavební frontu. Druhá datová struktura je seznam akcí obsahující případné rozhodnutí o útoku.

Ovladač musí plán v textové podobě získaný od plánovače převést do datové struktury popsané v minulém odstavci. Tento strukturovaný plán je získán iterativně, sekvenčním čtením jednotlivých operátorů v textovém plánu. Ovladač reaguje na operátory různými způsoby podle následujícího seznamu pravidel:

1. Pokud se jedná o operátor stavějící nějaký objekt, tak je do posledního kroku plánu vložen k příslušné stavební frontě herní objekt, který má být postaven (přesněji řečeno se jedná o typ daného objektu).

2. Pokud se jedná o operátor určující vyslání útoku, tak je do posledního kroku plánu vložena akce obsahující všechny příslušné informace od plánovače (typ skupiny, řetězec znaků jednoznačně určující útok, jak velká část armády postačuje k vyslání útoku a seznam všech jednotek, které mají být vyslány do útoku, a jejich počtů).

3. Pokud se jedná o operátor čekání, tak je za poslední krok plánu vložen nový krok (ten v tu chvíli neobsahuje žádné objekty na stavbu ani rozkazy k útoku).

Ovladač nechává vytvořit nový plán každou minutu. Důvodem jsou především situace, kdy hráč ztratí nějaké vlastněné herní objekty. Nejčastěji harvester kvůli písečnému červu nebo jednotky čekající v základně kvůli soupeřovu útoku.

Ve chvíli, kdy má nastat přeplánování, je celý plán smazán až na objekty pro postavení v prvním kroku. Ty jsou zachovány, aby umělá inteligence alespoň trochu hrála i v průběhu přeplánování. Plánovači jsou pak předány počáteční atomické formule již počítající s tím, že tyto objekty jsou stavěné (aby nedošlo k duplikovanému postavení z předešlého plánu a následujícího plánu). Plánovač tak může začínat plánování s částečně zaplněnými frontami stavby a v některých případech i zápornými zdroji (důvod vyplyne až v následující podsekci vysvětlující, jak ovladač plán používá).

Zmíněné počáteční atomické formule obsahují všechny potřebné informace o aktuálním stavu hry. Například frakci hráče, zdroje, energii, počet všech vlastněných herních objektů (s výjimkou jednotek, které jsou vyslány do útoku), počet soupeřů, zvolenou strategii, zaplnění stavebních front, všechny řetězce znaků definující již provedené útoky (výjimku tvoří řetězec `repeat`, ten je ignorován, aby mohl být případně tento útok opakován), cenu soupeřových bojových jednotek a jejich část v pěchotě a ve vozidlech. Zároveň jsou plánovači předávány konstantní informace definující hru, například ceny všech objektů, délky jejich stavby, produkce a spotřeby energie

všech budov, jaké jednotky mají být posílány do útoků a určité pomocné atomické formule používané při stavbě armád.

4.2.9.2 Aplikace plánu

Vzhledem k tomu, že RBAI pouze upravuje chování výchozí AI, tak je používání plánu rozdělené mezi příslušné části výchozí AI, v programu nazývané jako moduly (toto pojmenování bude zde v práci využito). Hlavní část ovladače je v novém RBAI modulu obsahující veškerou komunikaci s plánovačem popsanou v předešlé podsekci a potřebné funkcionality k aplikování daného plánu, které budou popsány v této podsekci.

Tento nový RBAI modul vystavuje metody (metody jsou myšleny z programátorského hlediska, nikoliv z hlediska plánování) pro jiné moduly umožňující získání objektů, které mají být postaveny, z prvního kroku strukturovaného plánu pro konkrétní stavební frontu (ta je předána v parametru metody při volání jiným modulem). Také umožňuje získání akce využívané pro vysílání útoků.

Ze strukturovaného plánu je jednou za definovaný čas (při veškerém testování nastaveném na pět sekund) odebrán první krok plánu a jeho případný obsah je přesunut do následujícího kroku tak, aby se zachovalo pořadí stavby (všechny prvky v původním prvním kroku musí zůstat ve stejném pořadí a musí být před prvky z nového prvního kroku). Zároveň musí být splněna alespoň jedna z následujících podmínek, aby odebrání prvního kroku nastalo.

1. První krok strukturovaného plánu je prázdný. Důvod je v tomto případě přímočarý, nejspíše se čekalo již dost času a může se přesunout k dalšímu kroku.

2. Hráč má více než definované množství koření (při veškerém testování nastavené na 800). Důvodem je obecně situace, kdy hráč nestíhá spotřebovávat své zdroje. (Může se jednat o situaci, kdy hráč má lepší těžbu, než se kterou plánovač počítá.)

Plánovač počítá s čekáním při veškerém testování nastaveným na dvacet sekund, ale jak bylo zmíněno, tak ovladač kontroluje přesun mezi kroky pouze každých pět sekund. Důvodem této nesourodosti je nepřesnost mezi skutečným systémem Σ a doménou v plánovači. Navíc vzhledem k dynamickému prostředí může nastat situace, kdy hráč přijde o harvester, a bude tak plánem postupovat pomaleji, než s čím počítal plánovač.

Souvisejícím důvodem je také to, že plánovač při plánování staví objekty jen ve chvíli, když na ně má dost zdrojů, jinak čeká. To ale není ve hrách v OpenRA vhodné, protože stavba nějakou dobu trvá. I když aktuálně není dost zdrojů, tak je vhodné se stavbou začít, protože během ní je hráč možná získá, a i pokud ne, tak se objekt dostaví, jakmile jich bude dost.

Co se týče vysílání útoků, tak příslušný model pravidelně kontroluje, zda je v posledním kroku plánu přítomný rozkaz k útoku (ten vždy obsahuje všechny potřebné informace k vyslání útoku). Pokud není, tak nedělá nic. Pokud je, tak zkontroluje, zda je dostupná dostatečně velká část jednotek, které mají být vyslány v útoku definovaném plánovačem. O jak velkou část jednotek se jedná, je také zadáno plánovačem (ve všech případech se jednalo o 90 %). Pokud je jednotek dost, tak je vytvořena útočná skupina konkrétního typu. Zmíněný typ (dříve v plánovači v `?type`), jak velká část jednotek musí být přítomná (dříve v `?fractionGame`) i které jednotky to jsou a kolik jich má být (dříve v `?army`), je vše zapsané ve zmíněné akci a je nyní využito. Pokud je útok vyslán, tak je o tom RBAI modul informován a je uložena informace, že tento útok, unikátně identifikovaný konkrétním řetězcem znaků, byl již vyslán.

4.2.10 Možná rozšíření

RBAI je relativně složitý již v tuto chvíli. Stále má ale nějaké nedostatky. Následující seznam obsahuje ty nejdůležitější.

- RBAI nepoužívá vesmírné letiště. Důvodem je to, že jednotky jsou v něm dražší a staví se pomaleji. Navíc hra jednotky z vesmírného letiště považuje za jiný typ (nejspíše kvůli jejich ceně), což znamená, že mají jiné jméno. Ve výsledku by bylo nutné to nějak řešit v plánovači nebo v ovladači.
- Používat skutečný výdělek harvesterů měřený při hře, nikoliv konstantu závislou pouze na počtu transportních letadel.
- RBAI neumí reagovat na nedostatek stavebního prostoru. V Dune 2000, jak bylo popsáno v podsekci 2.6.3.1 o mechanikách hry, bylo řečeno, že lze stavět budovy pouze na skálu a nikoli na písek. Pokud mapa obsahuje malé množství skály, tak RBAI se může zaseknout v situaci, kdy má postavit nějakou budovu umožňující stavbu dalších jednotek (například výzkumné středisko), protože plánuje s jejím postavením a následným postavením nově umožněných jednotek. Následné útoky pak mají tyto jednotky obsahovat. Ve výsledku RBAI nemusí nikdy zaútočit. (Útoky mají sice toleranci 10 %, co se týče aktuálně dostupných jednotek, ale pokud cena těchto nedostupných jednotek tuto hodnotu přesáhne, tak RBAI nikdy nezaútočí.)
- Více strategií pro BRAI. V tuto chvíli jsou jen dvě a jsou voleny náhodně. Vhodnější by bylo, aby tato volba strategií byla ovlivněna určitými nastaveními mise. Například hráčovou frakcí, soupeřovou frakcí, velikostí mapy, velikostí prostoru pro budovy, počátečními financemi, množstvím dostupného koření, vzdáleností od nejbližšího koření atd.
- Umožnit ve strategiích nepoužívat nějakou produkční budovu při stavbě armády. (V tuto chvíli jsou využívány vždy všechny dostupné, ale bylo by zajímavé umožnit ve strategii přestat stavět jednotky například v lehké továrně.)
- RBAI nepoužívá opravovací plošinu. Důvodem je to, že výchozí AI ji neumí používat (nikdy žádnou jednotku nepošle na opravu). V tomto případě by bylo nutné kromě přidání její stavby do plánu vylepšit výchozí AI tak, aby ji umělo používat.
- RBAI svou stavbu armád zaměřuje specifickým způsobem, pokud soupeř staví velké množství pěchoty nebo naopak vozidel. Podle toho, co bylo zmíněno v sekci 3.1.2 analyzující skladbu armády v předešlé kapitole, by ale bylo vhodnější toto rozhodnutí dělat podle toho, které jednotky jsou neobrněné a které naopak obrněné.

4.3 Alternativní návrh AI

Tato sekce a následující podsekce budou popisovat fungování druhé nové AI pro Dune 2000 označované jako výdělkově založená AI (Income Based AI, IBAI). Cílem této AI bude naopak plánovat v doméně výrazně méně přesně, co do podobnosti se hrou, vůči RBAI. Tato AI nemodeluje zdroje a pracuje pouze s výdělkem reprezentovaným počtem harvesterů. Zároveň nepracuje s časem reprezentovaným čekáním na zdroje a ani délkou front. Také neřeší stavbu bojových jednotek. Spousta funkcionalit je podobná těm v RBAI, ale zároveň výrazně zjednodušená. Cílem je vyzkoušet, zda jednodušší přístup není vhodnější (zda ztráta komplikovanosti domény vede ke ztrátě možností strategií, pokud nikoliv, tak by mohl být tento přístup vhodnější).

4.3.1 Plánovač

Doména a problém plánování v IBAI jsou podobné těm v RBAI. Postupně v této podsekci budou popsány rozdíly ve stejném pořadí, jako byly vysvětleny části domény RBAI. Nejdříve byly popsány základní modelované vlastnosti hry, jednalo se o následující.

- Zdroje jsou kompletně odstraněny. Jedná se například o počet zdrojů v atomické formuli `(money <num>)`, ceny jednotlivých objektů a axiom `enough_money`.

- Energie je zachována v nezměněné podobě.
- Počty herních objektů jsou zkrácené, co do počtu řešených herních objektů. IBAI neplánuje stavbu bojových jednotek.
- Čas je z větší části odstraněn. Metoda čekání je kompletně odstraněna a operátor čekání je výrazně zjednodušen. Jeho nové fungování bude popsáno dále v textu. Modelování doby stavby ve frontách pomocí atomických formulí `(queue <queueName> <num>)` je také kompletně odstraněno.

Operátor čekání zůstal, ale vzhledem k tomu, že byly odstraněny zdroje i fronty, tak již nic nedělá. Použit je pouze ve strategiích a důvod k tomu je popsán v následující sekci 4.3.2 vysvětlující ovladač.

Operátory stavby budov a vylepšení jsou stejné až na to, že v nich není nic týkající se ceny, zdrojů hráče, doby stavby ani front. Konkrétně, pro ukázaný operátor pro stavbu kasáren ve výpisu kódu 4.4, jsou odstraněny v předpokladech následující atomické formule:

- `(money ?mon)`;
- `(cost_barracks ?costb)`;
- `(queue building ?queue)`;
- `(building_time_barracks ?buildtime)`.

Mezi odstraněnými atomickými formulemi tohoto operátoru již není:

- `(money ?mon)`;
- `(queue building ?queue)`.

Mezi přidanými atomickými formulemi tohoto operátoru již není:

- `(money (call - ?mon ?costb ))`;
- `(queue building (call + ?queue ? buildtime ))`.

Co se týče metod stavby budov, tak jsou odstraněny všechny příslušné atomické formule týkající se zdrojů a času. Také jsou odstraněny poslední větve řešící nedostatek zdrojů čekáním. Metody se jménem prodlouženým o `_if_below` rozšiřující chování základních metod stavby jsou úplně stejné jako v RBAI.

Stavba jednotek je až na harvestery a transportní letadla kompletně odstraněna. To znamená, že jsou odstraněny operátory i metody stavby samostatných jednotek i celých armád. Zachované metody pro harvestery a transportní letadla jsou stejné, ale jsou odstraněny všechny příslušné atomické formule týkající se zdrojů a času. U základní metody stavby těchto nebojových jednotek je opět odstraněna poslední větev řešící nedostatek zdrojů, podobně jako u stavby budov.

V IBAI je výrazně zjednodušen operátor týkající se vysílání útoků na soupeře. Všechny příslušné metody jsou odstraněny. To znamená, že ve strategii je přímo operátor vyslání útoku `!o_form_squad`. Přičemž v hlavičce je pouze jedna proměnná, a to `?type`, ta má stejný význam jako v případě RBAI. (IBAI při útoku vysílá vždy všechny bojové jednotky.)

Kromě zjednodušování a odstraňování jsou v IBAI dva nové operátory, první se jmenuje `!o_activate` a druhý je `!o_deactivate`. Oba jsou ve výpisu kódu 4.14. Operátory v plánovači nic nedělají. Oba mají proměnnou `?queue` v hlavičce. Tyto operátory informují ovladač, ve kterých frontách mají být automaticky stavěny bojové jednotky. Podrobněji bude fungování vysvětleno v následující podsekci 4.3.2.

Strategie po veškerém zjednodušení domény obsahují pouze metody stavby budov, harvesterů a transportních letadel. Také obsahují operátory čekání, vysílání útoků a nově také aktivace

Výpis kódu 4.14 Operátory aktivace a deaktivace fronty stavby v IBAI.

```
(:operator (!o_activate ?queue)
  ()
  ()
  ()
0)

(:operator (!o_deactivate ?queue)
  ()
  ()
  ()
0)
```

a deaktivace front stavby. V IBAI je aktuálně vytvořena jediná strategie. Její úvodní část je ve výpisu kódu 4.15. Jak je vidět, tak operátory `!o_wait` jsou ve strategii opakovaně používány. Jejich konkrétní efekt bude popsán v následující podsekci 4.3.2, ale pro tuto chvíli stačí říct, že sousední dvojice těchto operátorů ohraničují minutové úseky plánu obsahující, co se v nich má stát.

Konkrétně se v této strategii mají udělat během každé minuty určité úkony. V následujícím seznamu jsou k jednotlivým minutám sepsány dané úkony. Jedná se jen o prvních šest minut. Elektrárny jsou stavěny ve výsledném plánu, tak jak jsou potřeba, stejným způsobem jako v RBAI.

1. Postavení rafinérie a kasáren.
2. Aktivace kasáren a postavení druhé rafinérie.
3. Postavení těžké továrny a případně druhého harvesteru, pokud o něj hráč přišel.
4. Postavení třetího harvesteru a vyslání prvního útoku.
5. Postavení čtvrtého harvesteru a radaru.
6. Postavení pátého harvesteru a dělové věže. Vyslání druhého útoku.

4.3.2 Ovladač

Ovladač IBAI je podobný ovladači RBAI, ale s několika rozdíly. Plán je ukládán úplně stejným způsobem co do struktury datových typů (fronta kroků plánu). Také platí to, že při zahájení přeplánování je smazán celý plán, až na aktuálně první krok plánu. Ovladač v IBAI navíc ukládá samostatně informace o tom, které fronty stavby jednotek jsou aktivní (mají se v nich stavět bojové jednotky). Samotná stavba jednotek probíhá tak, že pokud se nic ve frontě nestaví a je v prvním kroku plánu přítomné postavení nějaké jednotky, tak se začne stavět (týká se harvesterů a transportních letadel). Jinak, pokud je fronta aktivní, se postaví náhodná jednotka stejným způsobem, jako je staví výchozí AI (pouze s tím rozdílem, že se takto nikdy nepostaví harvester).

Co se týče komunikace s plánovačem, tak to je zjednodušené snížením počtu potřebných počátečních atomických formulí při přeplánování. Naopak zpracování textového plánu od plánovače musí fungovat složitěji. Vrácený textový plán od plánovače vždy obsahuje stejné všechny operátory čekání, vysílání útoků a aktivací/deaktivací front. Mění se pouze operátory stavby budov, harvesterů a transportních letadel, a to tak, že zůstávají jen ty, co je ještě nutné udělat. Ovladač pak při každém zpracování textového plánu ignoruje všechny nestavební operátory v aktuálně probíhajícím časovém kroku a ve všech předešlých (ty byly již aplikované). Stavební operátory (budovy, harvestery a transportní letadla) nacházející se v aktuálním časovém kroku nebo dříve

Výpis kódu 4.15 Operátory aktivace a deaktivace fronty stavby v IBAI.

```
(
  (m_build_refinery_if_below 1)
  (m_build_barracks_if_below 1)
  (m_build_upgrade_barracks)
  (!o_wait)
  (!o_activate Infantry) ;activate
  (m_build_refinery_if_below 2)
  (!o_wait)
  (m_build_heavy_factory_if_below 1)
  (m_build_harvester_if_below 2)
  (!o_wait)
  (m_build_harvester_if_below 3)
  (!o_form_squad assault) ;attack
  (!o_wait);5. min begins
  (m_build_harvester_if_below 4)
  (m_build_outpost_if_below 1)
  (!o_wait)
  (!o_form_squad assault) ;attack
  (m_build_harvester_if_below 5)
  (m_build_medium_gun_turret_if_below 1)
  (!o_wait)
  (m_build_X_carryall_target 1)
  (m_build_upgrade_hightech)
  (m_build_upgrade_heavy)
  (m_build_harvester_if_below 6)
  (!o_wait)
  (!o_form_squad assault) ;attack
  (m_build_X_carryall_target 2)
  (!o_activate Armor) ;activate
  (!o_wait)
  (m_build_X_carryall_target 3)
  (m_build_light_factory_if_below 1)
  (m_build_upgrade_light)
  (!o_wait);10. min begins
  (!o_form_squad assault) ;attack
  (!o_activate Vehicle) ;activate
  (m_build_research_centre_if_below 1)
  (m_build_X_carryall_target 4)
  (!o_wait)
  (!o_form_squad assault) ;attack
  (m_build_heavy_factory_if_below 2)
  (m_build_harvester_if_below 7)
  (!o_wait)
  (m_build_X_carryall_target 5)
  (!o_form_squad assault) ;attack
  (!o_wait)
  (m_build_X_carryall_target 6)
  (m_build_large_gun_turret_if_below 1)
  (!o_wait)
  (!o_form_squad assault) ;attack
  (m_build_harvester_if_below 8)
  (!o_wait);15. min begins
  (m_build_palace_if_below 1)
  ...
```

jsou vloženy do aktuálního časového kroku. Následující časové kroky (v textovém plánu v podobě operací mezi operátory čekání) jsou zpracovány stejným způsobem jako v RBAI. To znamená, že každý operátor čekání přidá na konec fronty nový krok plánu. Ostatní operátory přidávají stavbu nebo akci do posledního kroku.

Část zpracovávající operátory textového plánu musí navíc obsahovat část říkající, že pokud se jedná o operátor aktivace nebo deaktivace fronty, tak je do aktuálně posledního kroku plánu přidána aktivita obsahující dané informace (zda se jedná o aktivaci nebo deaktivaci a jaké fronty se to týká).

Co se týče aplikace plánu, tak to je podobné jako v RBAI, ale modul této AI navíc vystavuje metodu umožňující zjistit, které fronty jsou aktivní (týká se stavby bojových jednotek). Tato metoda je využívána v modulu stavějící jednotky.

Útoky jsou také řešeny jednodušším způsobem. Ve chvíli, kdy má být vyslán útok, tak je vytvořena útočná skupina ze všech aktuálně dostupných bojových jednotek.

4.3.3 Vhodná rozšíření

IBAI má v tuto chvíli jeden důležitý nedostatek, který by bylo nanejvýš vhodné v budoucnu vyřešit. Strategie obsahuje napevno operátory čekání určující konkrétní časové úseky hry. Těchto operátorů čekání je ale jen tolik, kolik jich člověk, který strategii navrhl, udělal. Například strategie používaná při testování je na 40 minut. Poté IBAI už nikdy nezaútočí (a ani neaktivuje nebo nedeaktivuje žádnou frontu; pokud ale nějakou budovu ztratí, tak ji po přeplánování znova postaví). Další nedostatky a jejich případná řešení jsou v následujícím seznamu:

- Na rozdíl od RBAI, tak IBAI nereaguje na složení soupeřovy armády (poměr pěších jednotek vůči vozidlům).
- IBAI má v tuto chvíli jedinou strategii. Důležité je jich udělat více. Vhodné by také bylo, aby tato volba strategií byla ovlivněna určitými nastaveními mise. Například hráčovou frakcí, soupeřovou frakcí, velikostí mapy, velikostí prostoru pro budovy, počátečními financemi, množstvím dostupného koření, vzdáleností od nejbližšího koření atd.
- IBAI neumí, stejně jako RBAI, reagovat na nedostatek stavebního prostoru. V případě IBAI ale kvůli tomu nemůže přestat útočit (protože neřeší, které jednotky mají být do útoku poslány).
- IBAI, stejně jako RBAI, nepoužívá opravovací plošinu.

Kapitola 5
Experimentální výsledky

V této kapitole budou jako první popsány společné vlastnosti provedených experimentů, následované jednotlivými testovanými scénáři s různými počátečními nastaveními. Nakonec budou samostatně zanalyzovány výsledky mezi RBAI a IBAI týkající se počtu výher.

5.1 Experimenty, jejich výsledky a jejich analýza

Pro otestování nových AI bude provedeno několik experimentů s různými počátečními nastaveními. Primárním cílem těchto testů bude popsat a otestovat vlastnosti obou nových umělých inteligencí. Zároveň ale bude také řešeno, které jsou schopnější vyhrát. V následující části práce bude používán pojem scénář. Tím jsou myšleny všechny společné počáteční vlastnosti misí používaných při testování. Jedná se o veškeré nastavení mise s výjimkou pozice hráčů a jejich frakcí.

Všech pět testovaných scénářů mělo odehráno 12 misí, přičemž vždy byly konkrétní počáteční pozice hráčů zastoupeny rovnoměrně. Mise probíhaly vždy mezi dvěma stejnými frakcemi, přičemž ty byly zastoupeny také rovnoměrně. (Proto je počet testovaných scénářů zvolen tak, aby byl dělitelný třemi.) Pokud žádný z hráčů neprohrál během 60 minut, tak mise byly předčasně ukončeny a považovány za remízu.

Všechny scénáře probíhaly na jedné z map na obrázku 5.1. V případě mapy 5.1b v polovině měření byl první hráč na pozici „A“ a druhý hráč na pozici „B“. Zbylá měření měla pozice hráčů prohozené. V případě mapy 5.1a jsou obdobným způsobem využity pozice „B“ a „E“. Hnědá barva na obrázcích znázorňuje skálu, bílá barva je písek a duny, oranžová barva znamená koření a tmavě hnědá až černá barva je neprůchozí terén.

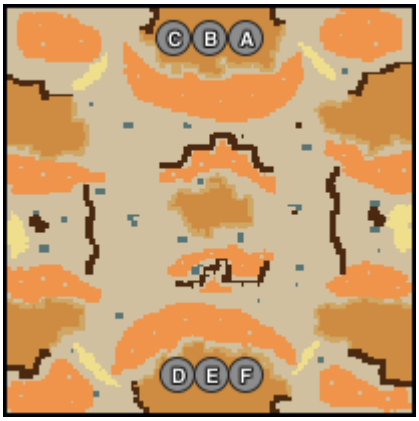
Mapa na obrázku 5.1a je čtvercová a má délku 128 polí. Mapa na obrázku 5.1b je také čtvercová, ale má pouze 64 polí. Mapy byly vybrány pro svůj velký rozdíl ve velikosti (čtvrtina herních polí) a dostatečný prostor pro stavbu.

Při veškerém testování byl jako zástupce výchozí AI zvolen Omnius. Jak bylo dříve v podsekci 2.6.3.5 řečeno, tak se chování různých variant (Omnius, Vidious a Gladius) liší jen nepatrně. Jedním z nejpodstatnějších rozdílů je velikost útočné skupiny. Ta je v případě Omnius průměrem hodnot zbylých zástupců.

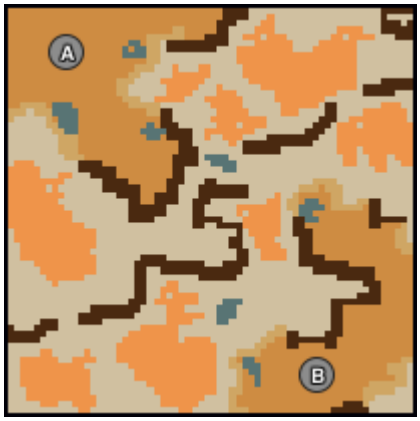
Co se týče počátečních nastavení vyjmenovaných v podsekci 2.6.3.1, tak jsou nastaveny následujícím způsobem:

Starting Units Nejsou přítomny žádné počáteční jednotky.

Game Speed Nastavení rychlosti hry je nastaveno na normální rychlost. (Vyšší rychlost není využita, protože plánovač běží se hrou asynchronně a výsledky nových AI by tak mohly být odlišné.)



(a) Mapa s názvem Spice Mesa je čtvercová se stranou délky 128 polí.



(b) Mapa s názvem Mount Idaho je čtvercová se stranou délky 64 polí.

Obrázek 5.1 Mapy používané při testování.

Explored Map Mapa je zpočátku zahalena.

Fog of War Mlha války je používána.

Crates Bedny se neobjevují. (Žádná AI je neumí aktivně sbírat.)

Build off Allies Toto nastavení určující, zda hráči ve stejném týmu mohou stavět jen ke svým budovám nebo i k budovám ostatních členům týmu, nemá vliv, protože v žádném scénáři nejsou použity týmy.

Automatic Concrete Nastavení automatické stavby betonových desek je povoleno. Vzhledem k tomu, že výchozí AI neumí stavět betonové desky a nové AI z ní vychází, tak to také neumí. Ve výsledku tedy žádná AI neumí stavět betonové desky a z tohoto důvodu je toto nastavení aktivní.

Short Game Nastavení krátké hry je aktivní a tedy kterýkoli hráč prohrává po ztrátě všech svých budov.

Worms Píseční červi jsou v misích přítomní.

V grafech je ukázáno vždy jen prvních 10 minut hry vzorkovaných po 30 sekundách. Důvod pro pouhých 10 minut je ten, že ve chvíli, kdy nějaký hráč prohraje, se mise zastaví. V některých případech při testování mise skončily velice brzy. Doba 10 minut byla zvolena právě kvůli tomu, že ve většině případů v té době ještě neskončily. V situacích, kdy ale dříve skončily, bylo postupováno tak, že se poslední naměřená hodnota opakuje. Důvodem tohoto rozhodnutí je to, že ostatní řešení jsou méně vhodná. Například kompletní vynechání daného měření by mohlo vést k výraznému ovlivnění výsledků. Zároveň je nevhodné měnit v průběhu osy x počet vzorků, ze kterých byl graf tvořen.

Následující čtyři podsekce provádí dříve zmíněných 12 měření pro konkrétní scénáře. Měřené hodnoty a jejich příslušné grafy jsou ve všech případech stejné. Jedná se o sedm grafů ve dvou obrázcích. První obrázek obsahuje čtyři grafy měřící ekonomickou stránku hry a druhý obrázek obsahuje zbylé tři grafy řešící vojenskou stránku hry. Všechny výsledky měření, ze kterých byly jednotlivé grafy vytvořeny, jsou v přiložených souborech.

5.1.1 RBAI vs. Omnius

Scénář tohoto testování probíhal na velké mapě na obrázku 5.1a s počátečními zdroji nastavenými na 5000, přičemž mezi sebou soupeřili RBAI a Omnius. Grafy ukazující výsledky testování jsou na obrázcích 5.2 a 5.3.

Na obrázku 5.3c je znázorněn počet výher účastníků scénáře a počet jejich remíz. Jak je vidět, RBAI vyhrává častěji než Omnius. Důvodem je především lepší ekonomická situace dále popsaná v následující části.

Obrázek 5.2a ukazuje graf s výdělkem. Výdělek je vyložené koření harvestery za posledních 60 sekund. Obrázek 5.2b obsahuje graf se vším získaným kořením do daného okamžiku hry. Graf má exponenciální podobu, protože rychlost získávání koření roste. Na obrázku 5.2c je graf s počty hráčem vlastněných harvesterů. Všechny tyto grafy znázorňují ekonomický stav. Přičemž ten je pro RBAI výrazně lepší. Do 10. minuty se jí, jak je vidět na obrázku 5.2b, podařilo získat v průměru přibližně dvakrát více zdrojů, než získal Omnius. Zároveň byl ekonomický stav značně stabilnější mezi jednotlivými měřeními, to je vidět ze směrodatných odchylek znázorněných rozpětím kolem každého datového bodu. Například v 10. minutě je pro Omnius směrodatná odchylka téměř čtyřnásobná než pro RBAI. Důvodem nízké ekonomické stability je značná náhodnost stavby budov i jednotek, týkající se i stavby rafinérií a harvesterů.

Nejlépe lze ekonomický stav vidět na počtu vlastněných harvesterů, protože není kumulativní jako získané koření a je méně zašuměný než výdělek, ten hodně závisí na tom, kolik harvesterů se stihne vyložit během 60 sekund.

Graf na obrázku 5.2d ukazuje průměrné nevyužité zdroje. Jak je vidět, tak Omnius na rozdíl od RBAI žádné takové nemá. To je způsobeno tím, že v zásadě ve všech dostupných frontách staví bez přestání a při tom nemá dostatečný výdělek, který by to pokryl. RBAI nějaké přebytečné zdroje má, to je způsobené především tím, jak již bylo dříve zmíněno, že v průběhu přeplánování je původní plán, až na jeho první krok, smazán. Pokud obsah tohoto jediného zachovaného kroku není dostatečně velký, tak RBAI už jen čeká.

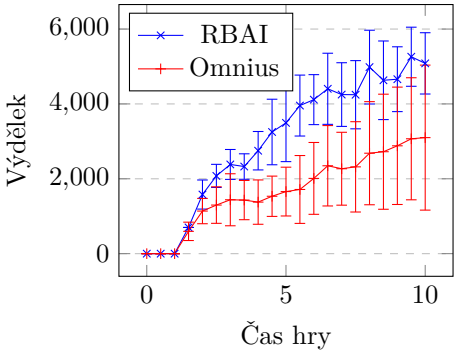
Na obrázku 5.3 jsou grafy týkající se vojenské stránky hry. Graf na obrázku 5.3a obsahuje průměrnou cenu armády. Jak je vidět, tak v prvních čtyřech minutách je RBAI mírně slabší než Omnius. To je způsobené tím, že RBAI investuje značně velké množství zdrojů do ekonomického rozvoje (postaví druhou rafinérii a další harvestery). Díky tomu má později výrazně větší výdělek (viz obrázek 5.2a). Ten pak může použít ke stavbě velkého množství nových bojových jednotek. Důvodem větší směrodatné odchylky a stagnujícího růstu armády od osmé minuty je podoba používaných strategií. Obě mají metodu vysílají útok po metodě stavějící armádu do ceny 6000 (poté, ještě než útok narazí na soupeře, jsou postaveny nějaké další jednotky). Tento útok vede ke ztrátě různě velké části jednotek (v některých případech je celý útok zničen a v jiných případech z větší části přežije a zničí soupeře). Efekt útoku je vidět i na grafu popsaném v následujícím odstavci.

Na obrázku 5.3b je graf ukazující cenu zničených herních objektů každým hráčem. Okolo osmé minuty je vidět efekt zmiňovaného útoku (značný nárůst v grafu na obou stranách). Do té doby, okolo páté minuty, je jen malý nárůst. Ten způsobí Omnius svými malými útoky. Ty jsou provedeny brzo, často postavené ještě za počáteční finance.

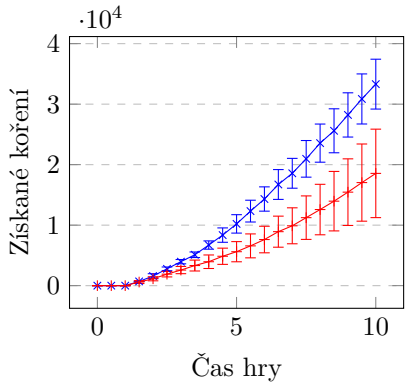
5.1.2 IBAI vs. Omnius

Scénář tohoto testování probíhal na velké mapě na obrázku 5.1a s počátečními zdroji nastavenými na 5000, přičemž mezi sebou soupeřili IBAI a Omnius. Grafy ukazující výsledky testování jsou na obrázcích 5.4 a 5.5.

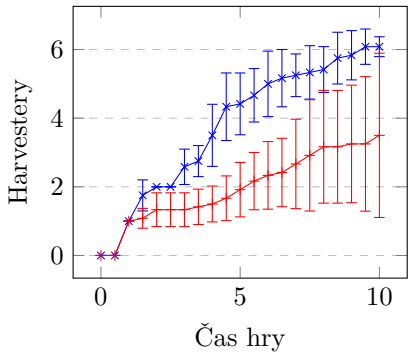
Na obrázku 5.5c je znázorněn počet výher účastníků scénáře a počet jejich remíz. Jak je vidět, tak podobně jako RBAI, i IBAI vyhrává častěji než Omnius, dokonce je ještě schopnější a Omnius tak vyhrál jen v jediném případě.



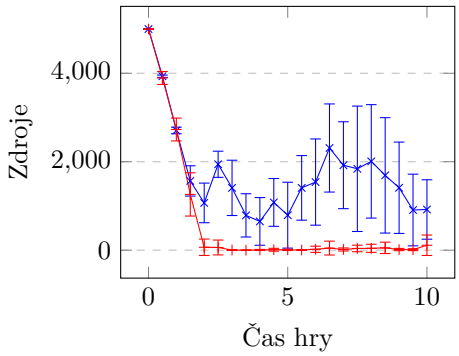
(a) Graf zobrazující průměrný výdělek a příslušné směrodatné odchylky.



(b) Graf zobrazující průměrné od počátku hry získané zdroje a příslušné směrodatné odchylky.

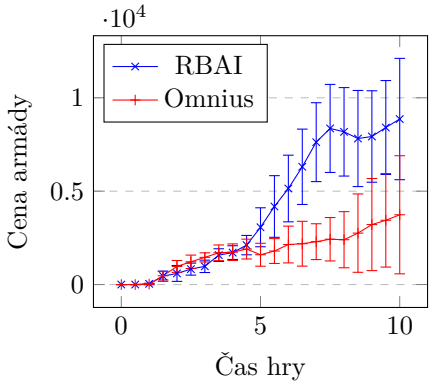


(c) Graf zobrazující průměrný počet harvesterů a příslušné směrodatné odchylky.

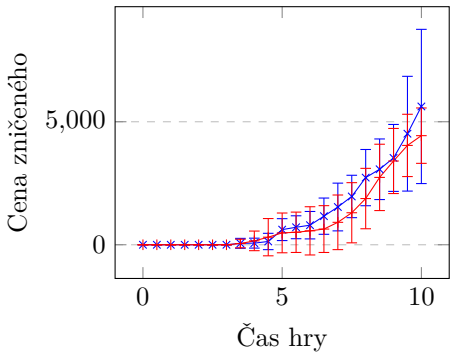


(d) Graf zobrazující průměrné nevyužité zdroje a příslušné směrodatné odchylky.

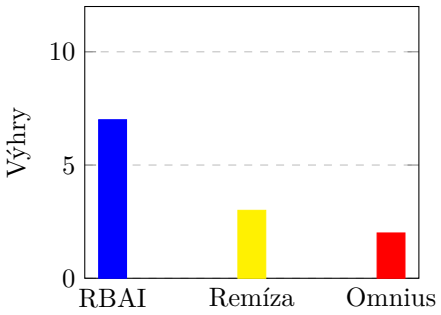
Obrázek 5.2 Grafy týkající se měření ekonomické stránky hry na velké mapě při porovnávání RBAI a Omnius. Legenda v grafu (a) platí i pro zbylé grafy.



(a) Graf zobrazující průměrnou cenu všech bojových jednotek hráče a příslušné směrodatné odchylky.

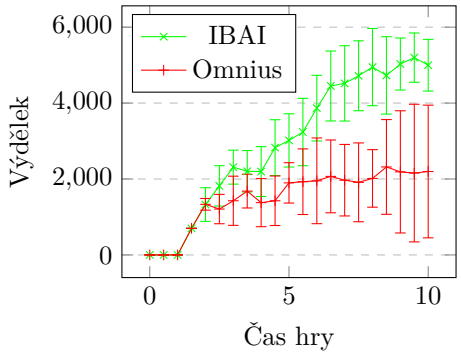


(b) Graf zobrazující průměrnou cenu všech zničených objektů a příslušné směrodatné odchylky.

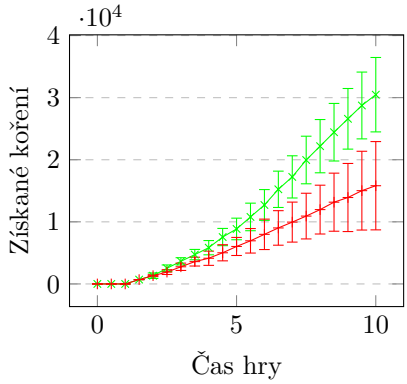


(c) Graf zobrazující počet výher RBAI a Omnius a počet vzájemných remíz.

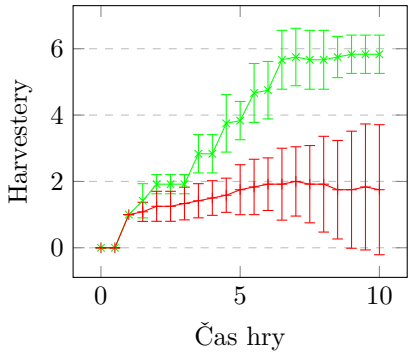
Obrázek 5.3 Grafy týkající se měření vojenské stránky hry na velké mapě při porovnávání RBAI a Omnius. Legenda v grafu (a) platí i pro (b).



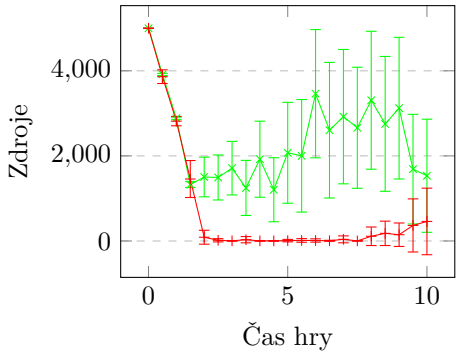
(a) Graf zobrazující průměrný výdělek a příslušné směrodatné odchylky.



(b) Graf zobrazující průměrné od počátku hry získané zdroje a příslušné směrodatné odchylky.



(c) Graf zobrazující průměrný počet harvesterů a příslušné směrodatné odchylky.



(d) Graf zobrazující průměrné nevyužité zdroje a příslušné směrodatné odchylky.

Obrázek 5.4 Grafy týkající se měření ekonomické stránky hry na velké mapě při porovnávání IBAI a Omnius. Legenda v grafu (a) platí i pro zbylé grafy.

Co se týče ekonomicky zaměřených grafů na obrázcích 5.4a, 5.4b a 5.4c, tak IBAI je také silnější než Omnius v podobné míře jako RBAI.

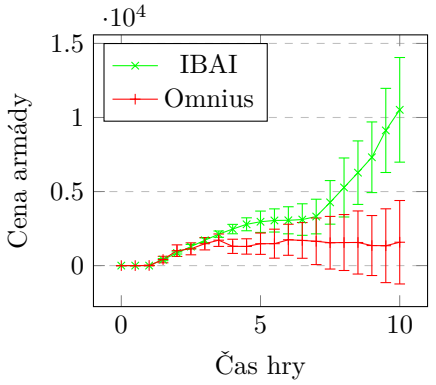
Na obrázku 5.4d je graf s průměrnými nevyužitými zdroji. Jak je vidět, tak IBAI nemá dobře vytvořenou strategii pro tuto mapu. V době mezi 7. a 10. minutou jí chvílemi přebývá tolik zdrojů, že se zaplní kapacita koření. Ta je 4000, vzhledem k tomu, že má v té době postavené dvě rafinérie. Způsob její implementace jí ale neumožňuje plánem procházet rychleji.

Co se týče grafů týkajících se vojenské stránky hry na obrázcích 5.5a a 5.5b, tak IBAI zvládá mít neustále cenu armády vyšší. V grafu je dobře vidět zlomový bod, kdy začíná stavět jednotky v těžké (8. minuta) a následně i lehké továrně (10. minuta). Cena zničeného je výrazně vyšší, to je spojené s větším množstvím výher a také s tím, že doba výher je v průměru nižší než v případě minulého měřeného scénáře s RBAI. IBAI dokázala soupeře při několika měřeních porazit během prvních 10 minut (díky tomu zničila budovy v součtu za hodně zdrojů).

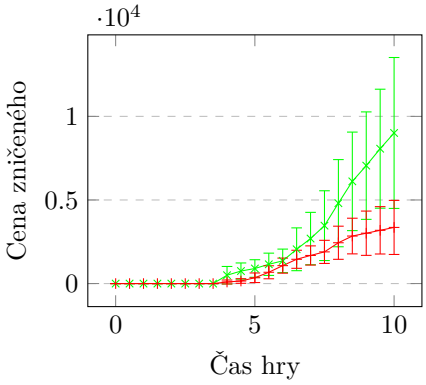
5.1.3 RBAI vs. IBAI

Scénář tohoto testování probíhal na velké mapě na obrázku 5.1a s počátečními zdroji nastavenými na 5000, přičemž mezi sebou tentokrát soupeřili RBAI a IBAI. Grafy ukazující výsledky testování jsou na obrázcích 5.6 a 5.7.

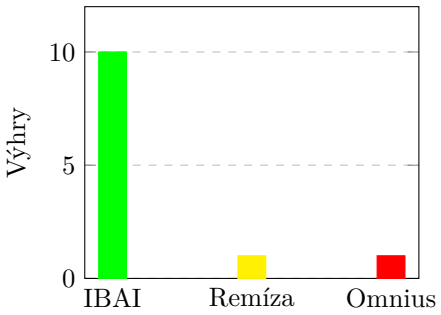
Na obrázku 5.7c je znázorněn počet výher účastníků scénáře a počet jejich remíz. Jak je vidět,



(a) Graf zobrazující průměrnou cenu všech bojových jednotek hráče a příslušné směrodatné odchylky.

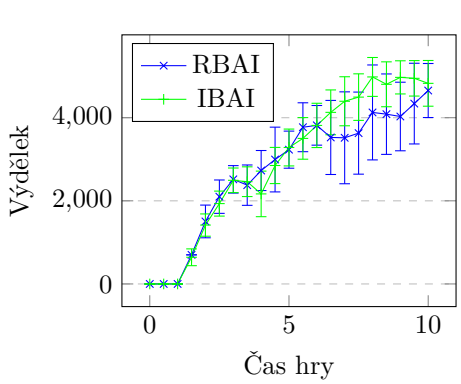


(b) Graf zobrazující průměrnou cenu všech zničených objektů a příslušné směrodatné odchylky.

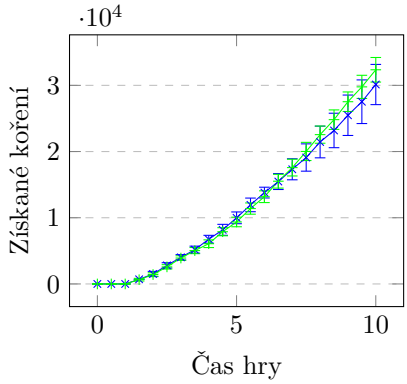


(c) Graf zobrazující počet výher IBAI a Omnius a počet vzájemných remíz.

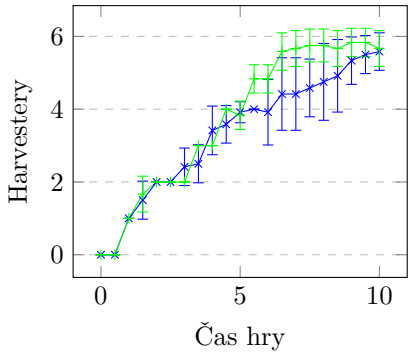
Obrázek 5.5 Grafy týkající se měření vojenské stránky hry na velké mapě při porovnávání IBAI a Omnius. Legenda v grafu (a) platí i pro (b).



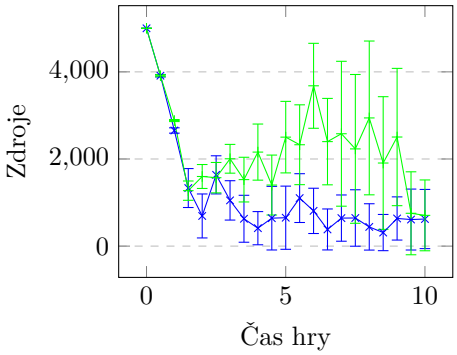
(a) Graf zobrazující průměrný výdělek a příslušné směrodatné odchylky.



(b) Graf zobrazující průměrné od počátku hry získané zdroje a příslušné směrodatné odchylky.



(c) Graf zobrazující průměrný počet harvesterů a příslušné směrodatné odchylky.



(d) Graf zobrazující průměrné nevyužité zdroje a příslušné směrodatné odchylky.

Obrázek 5.6 Grafy týkající se měření ekonomické stránky hry na velké mapě při porovnávání RBAI a IBAI. Legenda v grafu (a) platí i pro zbylé grafy.

tak IBAI vyhrálo osmkrát a RBAI ani jednou.

Co se týče ekonomicky zaměřených grafů na obrázcích 5.6a, 5.6b a 5.6c, tak IBAI je ekonomicky mírně silnější. Větší směrodatné odchylky v případě RBAI jsou způsobené tím, že při určitých měřeních přišel brzy o některé harvestery kvůli útoku soupeře a tím také o příslušný výdělek.

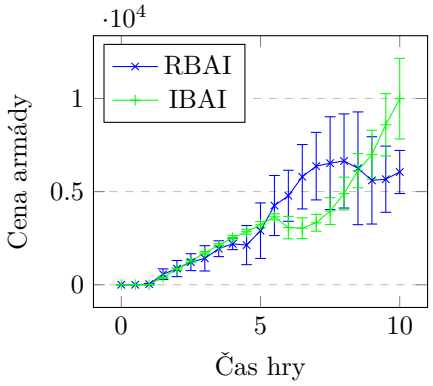
Graf s nevyužitými zdroji na obrázku 5.6d má pro oba účastníky scénáře podobný průběh jako v případě scénářů, ve kterých byl jako oponent Omnius.

Grafy týkající se vojenské stránky hry na obrázcích 5.7a a 5.7b ukazují, že IBAI má většinu doby hry větší celkovou cenu svých bojových jednotek, až na čas v průběhu sedmé a osmé minuty těsně před tím, než začne stavět jednotky v dalších frontách. Jak lze očekávat z počtu výher účastníků tohoto scénáře na obrázku 5.7c, tak IBAI vykazuje větší cenu zničených soupeřových objektů.

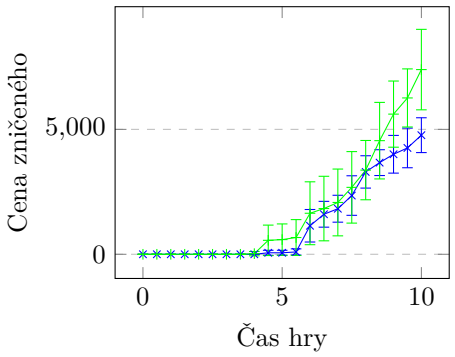
5.1.4 Malá mapa

Scénář tohoto testování probíhal na malé mapě na obrázku 5.1b s počátečními zdroji nastavenými na 5000, přičemž mezi sebou opět soupeřili RBAI a IBAI. Grafy ukazující výsledky testování jsou na obrázcích 5.8 a 5.9.

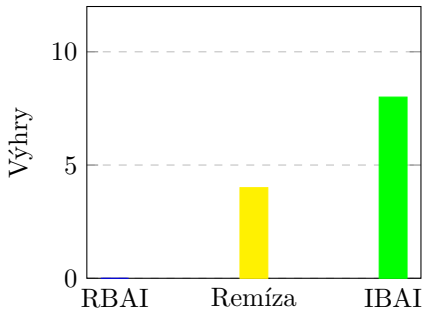
Na obrázku 5.9c je znázorněn počet výher účastníků scénáře a počet jejich remíz. Jak je



(a) Graf zobrazující průměrnou cenu všech bojových jednotek hráče a příslušné směrodatné odchylky.

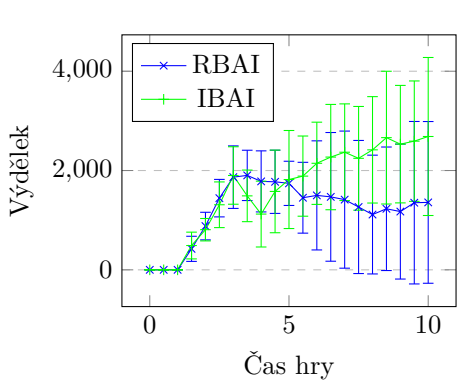


(b) Graf zobrazující průměrnou cenu všech zničených objektů a příslušné směrodatné odchylky.

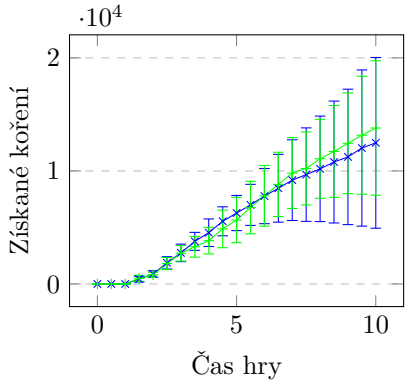


(c) Graf zobrazující počet výher RBAI a IBAI, a počet vzájemných remíz.

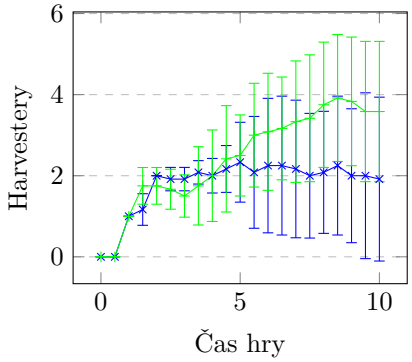
Obrázek 5.7 Grafy týkající se měření vojenské stránky hry na velké mapě při porovnávání RBAI a IBAI. Legenda v grafu (a) platí i pro (b).



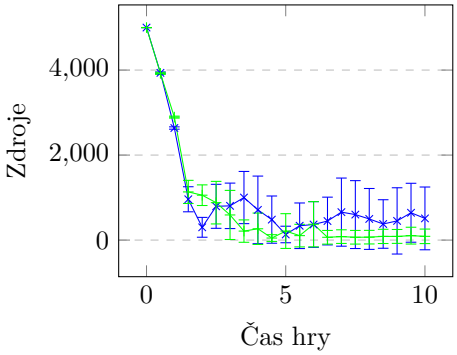
(a) Graf zobrazující průměrný výdělek a příslušné směrodatné odchylky.



(b) Graf zobrazující průměrné od počátku hry získané zdroje a příslušné směrodatné odchylky.



(c) Graf zobrazující průměrný počet harvesterů a příslušné směrodatné odchylky.



(d) Graf zobrazující průměrné nevyužité zdroje a příslušné směrodatné odchylky.

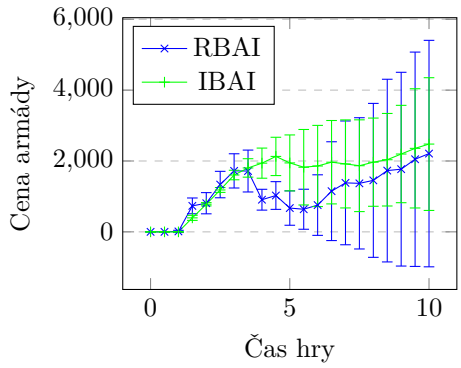
Obrázek 5.8 Grafy týkající se měření ekonomické stránky hry na malé mapě při porovnávání RBAI a IBAI. Legenda v grafu (a) platí i pro zbylé grafy.

vidět, IBAI vyhrálo vícekrát než RBAI, ale oproti scénáři s velkou mapou je výsledek tentokrát vyrovnanější. Může za to vyšší aktivita písečného červa, způsobená menším prostorem mapy, ten tak zničil výrazně více harvesterů. Díky tomu mohl jeden hráč náhodně získat velkou výhodu tím, že soupeř přišel v brzké fázi o harvester (nebo i o více harvesterů).

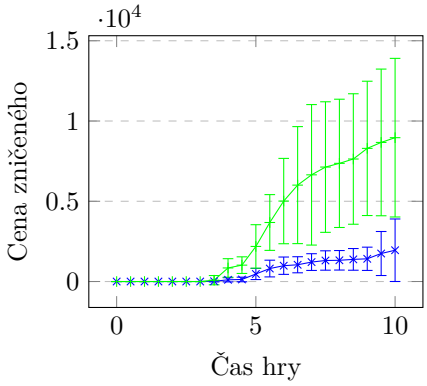
Co se týče ekonomicky zaměřených grafů na obrázcích 5.8a, 5.8b a 5.8c, tak jsou výrazně větší směrodatné odchylky a obecně nižší průměrné hodnoty. To je opět tím, jak již bylo v předešlém odstavci řečeno, že písečný červ byl aktivnější a zničil výrazně více harvesterů. Tímto byl také ovlivněn graf na obrázku 5.8d zobrazující nevyužité zdroje. Těch je výrazně méně, především kvůli sníženému výdělku a drahým dostavováním harvesterů. IBAI se stále snaží dodržet časový rozvrh stavby a akcí, což kvůli nedostatku zdrojů vede ke stejnému problému, kterým trpí Omnius. To je stavba ve větším množství front, než na které má výdělek, a to vede k tomu, že nemá dostatek zdrojů na stavbu podstatných věcí (například nových harvesterů).

Co se týče grafů týkajících se vojenské stránky hry na obrázcích 5.9a a 5.9b, tak cena armády je pro RBAI výrazně méně stabilní. To je způsobeno tím, že v některých případech při měření prohrála relativně brzy a na konci tohoto grafu již tedy neměla žádné jednotky. V jiných případech se jí naopak dařilo a měla podobné množství jednotek jako v předchozích testovaných scénářích. Ze stejného důvodu má také cena zničeného pro IBAI tak velkou směrodatnou odchylku.

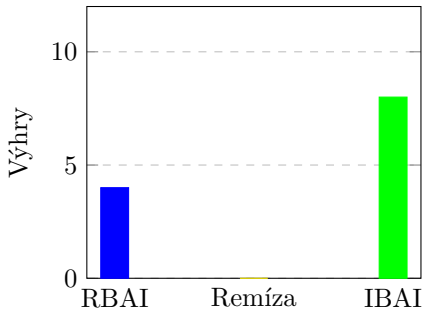
Ve výsledku menší mapa způsobila nižší stabilitu výsledků nových AI, především kvůli vyšší aktivitě písečného červa.



(a) Graf zobrazující průměrnou cenu všech bojových jednotek hráče a příslušné směrodatné odchylky.



(b) Graf zobrazující průměrnou cenu všech zničených objektů a příslušné směrodatné odchylky.



(c) Graf zobrazující počet výher RBAI a IBAI a počet vzájemných remíz.

Obrázek 5.9 Grafy týkající se měření vojenské stránky hry na malé mapě při porovnávání RBAI a IBAI. Legenda v grafu (a) platí i pro (b).

5.1.5 Velké počáteční zdroje

Scénář tohoto testování probíhal na malé mapě na obrázku 5.1b s počátečními zdroji tentokrát nastavenými na 20000 (nejvyšší nastavitelná částka), přičemž mezi sebou opět soupeřili RBAI a IBAI. Grafy ukazující výsledky testování jsou na obrázcích 5.10 a 5.11.

Na obrázku 5.11c je znázorněn počet výher účastníků scénáře a počet jejich remíz. Jak je vidět, tak i v tomto případě je IBAI schopnější výhry než RBAI.

Co se týče ekonomicky zaměřených grafů na obrázcích 5.10a, 5.10b a 5.10c, tak jsou na tom RBAI i IBAI téměř stejně. To je způsobené tím, že na stavbu mají dostatečně velké množství zdrojů a mohou tak stavět harvestery průběžně, jak potřebují. Zároveň mají ale grafy velkou směrodatnou odchylku, protože písečný červ stále často ničí harvestery (na jejich doplnění ale v je tomto případě dost zdrojů).

Na obrázku 5.10d jsou vidět nevyužité zdroje. RBAI je rychlejší v jejich spotřebě než IBAI. To je způsobené rozdílností jejich fungování. RBAI bude svou strategii díky velkým zdrojům aplikovat rychleji. Oproti tomu IBAI následuje svůj plán jen tak rychle, jak to je specifikované ve strategii.

Co se týče grafů týkajících se vojenské stránky hry na obrázcích 5.11a a 5.11b, tak je na ceně armády dobře vidět lepší schopnost využívání velkého množství zdrojů. RBAI tak má okolo páté minuty v průměru dvojnásobné množství bojových jednotek než IBAI. Cena zničeného a pozdější hodnoty ceny bojových jednotek odpovídají tomu, že IBAI více vyhrává.

Ve výsledku vysoké počáteční finance způsobily stabilnější průběh měření na malé mapě a ukázaly některé nedostatky IBAI.

5.2 Analýza výsledků mezi RBAI a IBAI

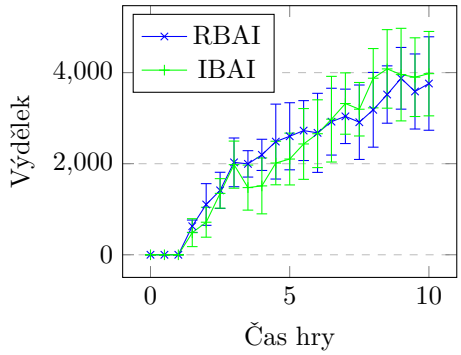
V minulých podsekcích věnujících se testování jednotlivých scénářů byly vysvětleny důvody různých naměřených výsledku. Záměrně tam nebyly vysvětleny důvody, proč IBAI vyhrává nad RBAI i přes to, že například v případě posledního scénáře měl v určitou chvíli bojové jednotky za výrazně více zdrojů než RBAI. Jedná se především o následující důvody:

1. Strategie, kterou IBAI používá, je hodně efektivní. Pěchota je relativně levná a velice silná. Také je ale velmi zranitelná vůči obléhacím tankům. Těch ale RBAI obvykle nestihne postavit dost. Navíc je není schopna ovládat tak, aby efektivně pěchotu ničily.

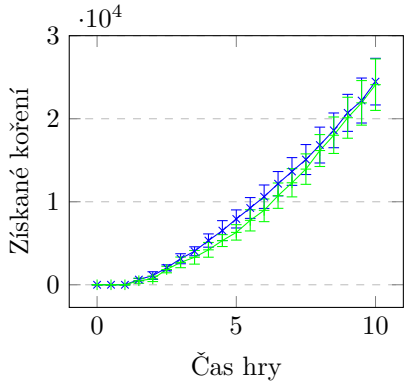
2. IBAI útočí častěji. Velké méně časté útoky by měly být lepší (protože množstevní převaha vede k menším ztrátám), ale výchozí AI není příliš dobrá v ovládání jednotek, především pak velkých skupin jednotek. Ve výsledku se ukázalo, že četnost útoků IBAI je vhodnější než ta v případě RBAI.

3. RBAI přeplánovává útoky nevhodným způsobem. Metody plánovače pro vytvoření armády používají vždy všechny v daném stavu dostupné jednotky. Ve výsledku tedy mají být použity všechny rozestavěné jednotky a i ty, které se mají postavit v zachovaném kroku plánu. Útoky, kvůli nevhodně načasovanému přeplánování, pak nakonec mohou být ještě větší, než bylo ve strategii zamýšleno. Jedná se o nevhodnost návrhu příslušné části domény.

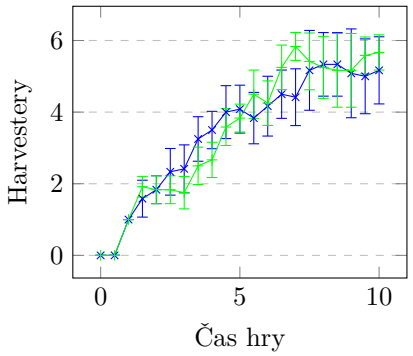
4. Dalším problémem RBAI je to, že při přeplánování si zachová pouze jediný krok plánu. Ten nemusí obsahovat dostatek úkonů pro celou dobu běhu plánovače, takže nakonec nemusí chvíli nic dělat. Navíc potom, co plánovač doběhne, výsledný plán reaguje na situaci při jeho spuštění. To například znamená, že RBAI v době jeho dokončení ale bude mít ve skutečnosti více zdrojů, než s kterými plán počítal.



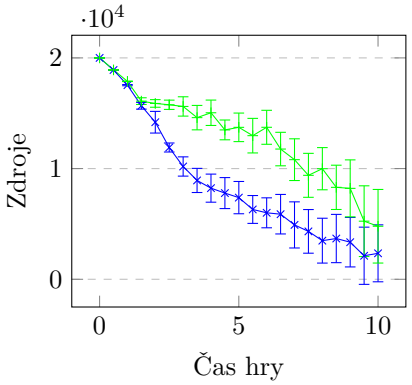
(a) Graf zobrazující průměrný výdělek a příslušné směrodatné odchylky.



(b) Graf zobrazující průměrné od počátku hry získané zdroje a příslušné směrodatné odchylky.

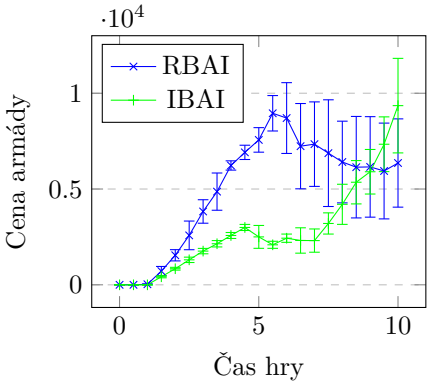


(c) Graf zobrazující průměrný počet harvesterů a příslušné směrodatné odchylky.

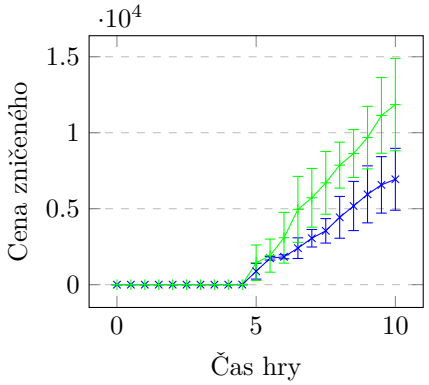


(d) Graf zobrazující průměrné nevyužité zdroje a příslušné směrodatné odchylky.

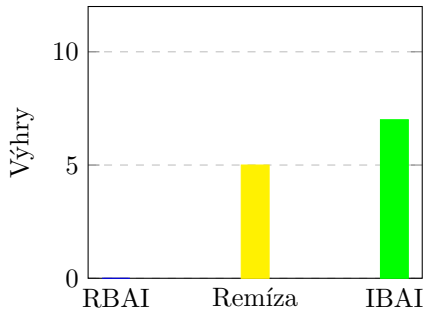
Obrázek 5.10 Grafy týkající se měření ekonomické stránky hry na malé mapě s velkými zdroji při porovnávání RBAI a IBAI. Legenda v grafu (a) platí i pro zbylé grafy.



(a) Graf zobrazující průměrnou cenu všech bojových jednotek hráče a příslušné směrodatné odchylky.



(b) Graf zobrazující průměrnou cenu všech zničených objektů a příslušné směrodatné odchylky.



(c) Graf zobrazující počet výher RBAI a IBAI a počet vzájemných remíz.

Obrázek 5.11 Grafy týkající se měření vojenské stránky hry na malé mapě s velkými zdroji při porovnávání RBAI a IBAI. Legenda v grafu (a) platí i pro (b).

Kapitola 6
Závěr

Cíli teoretické části práce bylo vysvětlit potřebné pojmy. Dalším cílem bylo odůvodnit použití HTN plánování. Cíli kapitoly plánování v RTS bylo zanalyzovat hru Dune 2000 použitou v práci a způsob aplikace plánování pro OpenRA hry. Cílem praktické části práce byl návrh nové herní AI využívající HTN plánování ke hraní hry. Dalším cílem praktické části práce byla implementace navržené nové AI. V neposlední řadě bylo cílem experimentálně otestovat novou AI a porovnat ji s již existující výchozí AI. Nakonec výsledky zanalyzovat. Cíli práce jako celku bylo vyzkoušet vhodnost aplikace HTN plánování pro RTS hry.

V teoretické části práce byly vysvětleny potřebné pojmy týkající se agentů, automatizovaného plánování a hry Dune 2000 (například definice agenta, definice klasického a HTN plánování nebo popis hry Dune 2000). Dále byla provedena analýza hry, výchozí AI a způsobů aplikace plánování pro RTS hry v OpenRA. V praktické části práce byly navrženy dvě nové AI, jmenovitě Resourced Based AI (RBAI) a Income Based AI (IBAI), lišící se především komplexností plánovací domény.

Nové AI byly následně experimentálně testovány ve scénářích proti výchozí AI a mezi sebou s různými počátečními nastaveními. Z analýzy těchto měření bylo zjištěno značné zlepšení vůči původní AI, především co se týče ekonomické stránky hry. Nové AI měly mezi jednotlivými měřeními malé rozdíly mezi výdělky a obecně jej měly v počáteční fázi hry výrazně vyšší než původní AI. Při porovnávání RBAI a IBAI byly zjištěny některé problémy návrhu domény RBAI, jejích strategií a částí výchozí AI, které byly novými AI využívány.

Co se týče obecného cíle vyzkoušet vhodnost aplikace HTN plánování pro RTS hry, tak se ukázalo plánování jako částečně zbytečné, protože HTN plánování nebylo použito pro komplikované nalezení vyhovující sekvence operátorů k dosažení cílů, ale pouze pro dodržování předdefinované strategie. HTN plánování toho je dobře schopné (umí velice dobře řešit závislosti, jako je udržování dostatku energie nebo konkrétní požadavky pro stavbu), ale je možná až zbytečně složité a jeho potenciální aplikace v RTS hrách je pracná. Ve výsledku by pro zlepšení AI, a tak i ze zážitku hraní proti ní, bylo vhodnější řešit její nedostatky v samotném kódu hry (například problémy s velikostí útočných skupin, s náhodnou stavbou nebo s ovládáním skupin jednotek) nikoliv používáním HTN plánování.

Příloha A

Implementační detaily

V této příloze budou sepsány některé implementační detaily vynechané v textu práce, například kompilace a spuštění hry. V závěru bude také popsána struktura souborů s výsledky měření. Podrobnější informace ke kompilaci a spuštění lze nalézt také na stránkách s repositářem OpenRA:

- `https://github.com/OpenRA/OpenRA/blob/bleed/INSTALL.md`
- `https://github.com/OpenRA/OpenRA/wiki/Compiling`

Hru lze spustit pouze na Windows a pro její zapnutí je nutné spustit „launch-game.cmd“. Poté je nutné specifikovat požadovanou hru (ra, cnc, d2k, ts). Nové AI je funkční pouze v Dune 2000, tedy volba „d2k“. Pro spuštění je potřeba Windows 7 nebo vyšší systém podporující DirectX 11 nebo OpenGL 3+.

Pro kompilaci je třeba Windows PowerShell verze 4.0 nebo vyšší a .NET 6 SDK. Samotnou kompilaci lze spustit ve Visual Studiu po otevření „sln“ souboru z adresáře OpenRA, pomocí příkazové řádky s použitím „dotnet“ příkazu nebo využitím souboru Makefile použitím příkazu „make all“ v příkazové řádce.

Co se týče obsahu adresáře „OpenRA“ v přiložených souborech, tak obsahuje velké množství adresářů a souborů herního enginu a samotných her. Upravené a nové soubory herního enginu jsou, spolu i s dalšími neupravenými, v adresáři „OpenRA.Mods.Common\Traits\BotModules“ (soubor „BaseBuilderQueueManager.cs“ je v podadresáři „BotModuleLogic“). Upravený soubor hry „ai.yaml“ je v adresáři „mods\d2k\rules“. Plánovací domény a problémy pro RBAI a IBAI jsou v samostatných adresářích „RBAI“ a „IBAI“. Jejich spouštění při hře zajišťují soubory příkazové řádky „run-jshop2.bat“ a „run-jshop2-IB.bat“.

Měření jsou automaticky ukládána při hře do složky „results“ v adresáři „OpenRA“.

A.1 Plánování

Ve složce JSHOP2 je kompletní plánovací systém JSHOP2. Pro jeho zprovoznění je dostupný originální návod v souboru „README“ a manuál v souboru „JSHOP2.pdf“. Plánovací domény a problémy obou AI jsou zařazeny mezi příklady ve složce „examples“. Lze je spustit pomocí příkazu „make.bat 15“ pro RBAI a „make.bat 16“ pro IBAI. Problémy plánovaní obsahují stav podobný prvnímu volání plánovače při hře.

A.2 Soubory výsledků měření

Soubory s výsledky měření v adresáři „results“ jsou uložené v samostatných adresářích podle testovaného scénáře. V každém tomto adresáři se nachází textové soubory, jejichž názvy se skládají

ze tří částí. První část textu určuje měřenou hodnotu, například soubory s „armyValue“ obsahují cenu armády. Druhá část textu obsahuje číslo určující jednotlivé hráče (například číslo jedna určuje prvního z dvojce účastníků měření). Poslední část názvu souboru je písmeno definující, o které z dvanácti měření se jedná. (Například soubor „harvesterCnt2b.txt“ v adresáři „IBAIvsOmnius largeMap“ obsahuje počet harvesterů hráče Omnius v druhém měření při testování IBAI vs. Omnius.) Obsahy souborů jsou vždy jen čísla na samostatných řádcích.

Bibliografie

1. RUSSELL, Stuart J.; NORVIG, Peter. *Artificial intelligence: a modern approach*. 3rd. Upper Saddle River: Prentice Hall, 2013. ISBN 1292024208.
2. VON NEUMANN, John. On the theory of games of strategy. *Contributions to the Theory of Games*. 1959, roč. 4, s. 13–42.
3. FIKES, Richard E; NILSSON, Nils J. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*. 1971, roč. 2, č. 3, s. 189–208. ISSN 0004-3702. Dostupné z DOI: https://doi.org/10.1016/0004-3702(71)90010-5.
4. GHALLAB, Malik; NAU, Dana S.; TRAVERSO, Paolo. *Automated planning: theory and practice*. Amsterdam: Elsevier, 2004. ISBN 1558608567.
5. EROL, Kutluhan; NAU, Dana S.; SUBRAHMANIAN, V.S. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*. 1995, roč. 76, č. 1, s. 75–88. ISSN 0004-3702. Dostupné z DOI: https://doi.org/10.1016/0004-3702(94)00080-K. Planning and Scheduling.
6. NAU, Dana S; AU, Tsz-Chiu; ILGHAMI, Okhtay; KUTER, Ugur; MURDOCK, J William; WU, Dan; YAMAN, Fusun. SHOP2: An HTN planning system. *Journal of artificial intelligence research*. 2003, roč. 20, s. 379–404. Dostupné z DOI: https://doi.org/10.1613/jair.1141.
7. GUPTA, Naresh; NAU, Dana S. On the complexity of blocks-world planning. *Artificial Intelligence*. 1992, roč. 56, č. 2, s. 223–254. ISSN 0004-3702. Dostupné z DOI: https://doi.org/10.1016/0004-3702(92)90028-V.
8. SLANEY, John; THIÉBAUX, Sylvie. Blocks World revisited. *Artificial Intelligence*. 2001, roč. 125, č. 1, s. 119–153. ISSN 0004-3702. Dostupné z DOI: https://doi.org/10.1016/S0004-3702(00)00079-5.
9. NAU, Dana; CAO, Yue; LOTEM, Amnon; MUNOZ-AVILA, Hector. SHOP: Simple hierarchical ordered planner. In: *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*. 1999, s. 968–973.
10. NAU, Dana; MUNOZ-AVILA, Héctor; CAO, Yue; LOTEM, Amnon; MITCHELL, Steven. Total-order planning with partially ordered subtasks. In: *IJCAI*. 2001, sv. 1, s. 425–430.
11. BACCHUS, Fahiem; KABANZA, Froduald. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*. 2000, roč. 116, č. 1, s. 123–191. ISSN 0004-3702. Dostupné z DOI: https://doi.org/10.1016/S0004-3702(99)00071-5.
12. NAU, Dana; MUNOZ-AVILA, Héctor; CAO, Yue; LOTEM, Amnon; MITCHELL, Steven. Total-order planning with partially ordered subtasks. In: *IJCAI*. 2001, sv. 1, s. 425–430.

13. *International Planning Competition 2002*. 2002. Dostupné také z: https://ipc02.icaps-conference.org/main.html.
14. ILGHAMI, Okhtay. 2006.
15. GERYK, Bruce. A History of Real-Time Strategy Games. In: *GameSpot*. [B.r.]. Dostupné také z: https://web.archive.org/web/20110427052656/http://gamespot.com/gamespot/features/all/real_time/.
16. KOSTET, Raph. *Theory of fun for Game Design*. 2. vyd. O'Reilly Media, 2013.
17. *GNU General Public License, version 3* [http://www.gnu.org/licenses/gpl.html]. 2007.
18. *OpenRA* [Computer software]. 2007.
19. LAGOS-ORTIZ, Katty. *Technologies and Innovation, Second International Conference, CITI 2016, Guayaquil, Ecuador, November 23-25, 2016, Proceedings, Communications in Computer and Information Science 658*. 2016. ISBN 978-3-319-48024-4. Dostupné z DOI: 10.1007/978-3-319-48024-4.
20. WESTWOOD STUDIOS, Intelligent Games. *Dune 2000* [Computer software]. 1998.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `results` ..... adresář s výsledky měření
- `src`
  - `OpenRA` ..... zdrojové kódy a spustitelná forma implementace
  - `changed` ..... zdrojové kódy pouze změněných tříd implementace
  - `JSHOP2` ..... adresář se soubory pro plánovač
    - `examples`
      - `dune2000RBAI` ..... adresář se soubory pro plánování týkající se RBAI
      - `dune2000IBAI` ..... adresář se soubory pro plánování týkající se IBAI
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF