VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

ROZHRANÍ PRO PROPOJENÍ STRATEGICKÝCH HER S MULTIAGENTNÍMI SYSTÉMY

INTERCONNECTION OF RECENT STRATEGIC GAMES WITH MULTI-AGENT FRAMEWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. LUKÁŠ VÁLEK**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Válek Lukáš, Bc.**
Program: Informační technologie Obor: Inteligentní systémy
Název: **Rozhraní pro propojení strategických her s multiagentními systémy**
Interconnection of Recent Strategic Games with Multi-Agent Frameworks
Kategorie: Umělá inteligence
Zadání:

1. Seznamte se s architekturami současných strategických her a možnostmi jejich propojení s agentními systémy za účelem využití agentních systémů pro kontrolu chování entit v těchto hrách.
2. Zvolte takové metody z oblasti agentních a multiagentních systémů, které podporují plánování v reálném čase, strojové učení a práci v multiagentních skupinách.
3. Navrhněte rozhraní, které by umožnilo propojení zvolených systémů a příslušné komunikační protokoly.
4. Pro vhodně zvolenou strategickou hru vytvořte jednoduchý multiagentní systém a tyto systémy propojte skrz vaše řešení. Vyhodnoťte jeho použitelnost a případné nedostatky.
5. Vytvořte dokumentaci pro vývojáře her popisující způsob použití vytvořeného rozhraní.

Literatura:

- Wooldridge, M.: An Introduction to MultiAgent Systems, 2nd Edition, Willey, 2009
- Caire, G.: JADE Tutorial, Jade Programming for Beginners, TILAB, 2009

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 12. května 2019

Abstrakt

Tato práce se zaměřuje na návrh frameworku, který usnadní tvorbu počítačem řízených protivníků ve strategických hrách. V práci se soustředíme na analýzu typů strategických her a systémů umělé inteligence používaných v současných hrách. Budou vysvětleny problémy, jež se u těchto systémů vyskytují a jak je agentní systémy řeší. Dále je s využitím těchto poznatků navržen a implementován framework, který slouží jako podpora pro tvorbu inteligentních systémů ve strategických hrách.

Abstract

This thesis is focused on design of framework for creation an articial opponents in strategy games. We will analyze different types of strategy games and artificial intelligence systems used in these types of games. Next we will describe problems, which can occur in these systems and why agent-based systems makes better artificial opponents. Next we will use knowledge from this research to design and implement framework, which will act as support for creating an artificial intelligence in strategy games.

Klíčová slova

Strategická hry, RTS, Multi-agentní systémy, UI, agent, umělá inteligence, real-time strategy

Keywords

Strategy games, RTS, Multi-agent systems, AI, agent, artificial intelligence, real-time strategy

Citace

VÁLEK, Lukáš. *Rozhraní pro propojení strategických her s multiagentními systémy.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. František Zbořil, Ph.D.

Rozhraní pro propojení strategických her s multiagentními systémy

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Doc. Ing. Františka Zbořila, Ph.D.

Další informace poskytl Bc. Petr Knapek, který pracuje na diplomové práci, která je na tuto práci navázána.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Válek
21. května 2019

Poděkování

Rád bych poděkoval vedoucímu diplomové práce, Doc. Ing. Františku Zbořilovi, Ph.D za rady a tipy, které mně věnoval při tvorbě této práce. Také bych rád poděkoval Bc. Petru Knapkovi za profesionální a bezproblémovou spolupráci na diplomové práci.

Obsah

Kapitola 1

Úvod

Hraní her bylo součástí výzkumu umělé inteligence již od jejího raného počátku. Jedny z prvních počítačových programů vykazujících znaky umělé inteligence byly právě programy pro hraní dámy a šachu, které vznikly v roce 1951. Výzkum umělé inteligence, jež byla schopna nejen hrát, ale i konzistentně vyhrávat proti lidským oponentům, vyvrcholila v roce 1997, kdy byl světový šampión v šachu Garry Kasparov poražen počítačem Deep Blue vyvinutého firmou IBM. [17]

Přestože hraní her bylo součástí vývoje umělé inteligence již od počátků, to samé nemůže být řečeno o vývoji počítačových her. První hry, jako např. Pong od společnosti Atari, byly založeny interakci dvou hráčů. Až rozvoj arkádových automatů a domácích konzolí způsobil rostoucí zájem herních vývojářů o umělou inteligenci. Nové technologie umožňovaly provádět potřebné výpočty a implementovat systémy, které do hry vnášely jistou formu nepředvídatelnosti, což přinášelo větší výzvu pro hráče. Obtížnost hry byla v té době jedním z nejdůležitějších faktorů, a to jak na arkádových automatech, tak i domácích konzolích. Arkádové automaty si účtovaly peníze za každý hráčův pokus dohrát hru, takže čím více pokusů hráč potřeboval, tím víc peněz automat vydělal. U domácích konzolí byla vyšší obtížnost vyžadována z jiného důvodu. Hry neměly příliš mnoho obsahu a byly drahé, takže bylo potřeba co nejvíce natáhnout herní dobu. Jak se herní průmysl rozvíjel, tak důvodů pro používání systémů umělé inteligence ve vývoji her jen přibývalo a tyto systémy se staly nedílnou součástí herního vývoje.[1]

Tato práce se zabývá převážně strategickými hrami a agentními systémy. Strategické hry jsou žánrem počítačových her, které jsou založeny na přemýšlení o svých akcích dopředu a plánování svých kroků pro dosažení vítězství. Cílem hry je většinou podniknout sérii akcí, jenž vedou k oslabení protivníka natolik, že již není hrozbou, čímž hráč docílí vítězství. Náhoda v těchto hrách hraje malou roli a většinou vyhraje hráč s lepší strategií a plánováním. Z tohoto popisu lze odvodit, že cíl strategických počítačových her a způsob, jakým je dosaženo vítězství, se v základu příliš neliší od klasických stolních her jako jsou šachy nebo shogi[1]. Největším rozdílem mezi těmito hrami je, že počítačové strategické hry bývají komplexnější. Tyto hry lze jednoduše rozdělit na tahové strategie a strategie hrané v reálném čase. Jednotlivé žánry se liší nejen svou formou, ale i tím, jak se v nich přistupuje k UI.

Centralizované UI pro strategie obvykle vyžadují relativně velké množství výpočetních zdrojů, které si herní vývojáři nemohou dovolit postrádat. Výzkum umělé inteligence ve strategických hrách významně ovlivnily multiagentní systémy. Multiagentní sys-

[1] Verze šachů, která vznikla v Japonsku.

témy (zkr. MAS) umožňují rozdělit velmi složitý problém na několik menších, jednodušších úkolů. To umožní nejen efektivnější využití zdrojů, ale také umožňuje lepší přizpůsobitelnost. V této práci se budeme zabývat vytvořením frameworku, který by měl umožnit vytvořit efektivní MAS pro libovolnou strategickou hru hranou v reálném čase, a který bude schopný hrát tak, aby byl pro hráče důstojným soupeřem.

Práce je rozdělena na několik částí. První kapitolu tvoří úvod práce. Druhá kapitola se věnuje popisu počítačových strategických her a jejich herních mechanik. Jejím účelem je seznámit čtenáře s problematikou a zasvětit ho do základních problémů, s nimiž se hráč potýká. Ve této kapitole budou uvedeny i základní informace o multiagentních systémech, které budou požívány v dalších částech práce a také je zde rozebráno použití agentních systému ve strategických hrách. Třetí kapitola obsahuje popis návrhu, jenž bude použit jako základ frameworku, jehož implementace je popsána ve čtvrté kapitole. Jsou zde popsány použité technologie a implementace všech hlavních částí frameworku. V páté kapitole je detailněji popsáno propojení frameworku s konkrétní hrou, vytvoření testovacího modelu, který byl použit k validaci implementovaného frameworku a je zde proveden rozbor poznatků získaných z vytvořeného modelu. Šestou kapitolu tvoří závěrečné zhodnocení práce.

Kapitola 2

Umělá inteligence ve strategických hrách

Umělá inteligence může být v strategických hrách rozdělena do dvou kategorií.

- Vytváření počítačově řízených protivníků.
- Automatická odezva a chování.

Druhá kategorie obsahuje mechaniky, které jsou přítomny jak ve hře hráče, tak i ve hře řízené umělým protivníkem. Jedná se o mechaniky, které jsou ve hře kvůli usnadnění jednoduchých, opakujících se akcí. Jedná se například o nalezení nejlepší cesty terénem nebo shlukování jednotek do formací. Tento typ umělých inteligencí je momentálně hlavním zaměřením žánru. Za dlouhé roky vývoje strategických her se algoritmy pro hledání nejlepších cest, reaktivní odezvy na útoky a automatického dokončování jednoduchých úkolů výrazně zlepšily.

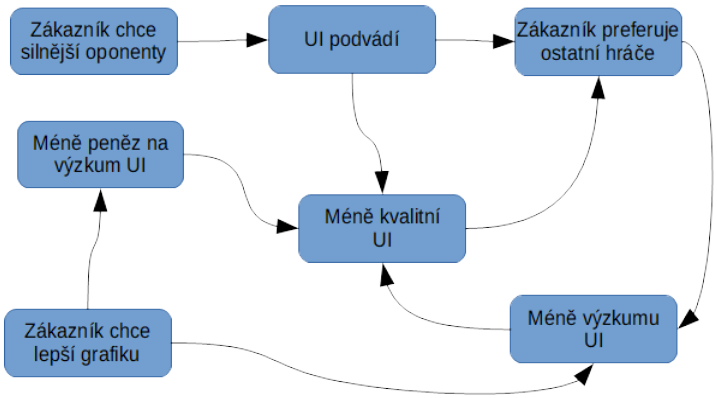
I přesto, že samotné hry se staly chytřejší a jsou schopny hráčům velmi pomáhat s ovládáním hry, tak to samé nemůže být řečeno o jejich schopnosti produkovat schopné, počítačem řízené protivníky pro hráče. Z pozorování a hraní[1] nových strategických her nelze říct, že by se UI řídící vaše protivníky příliš zlepšila.

V tahových hrách je možno vytvořit silnou a efektivní umělou inteligenci, protože tyto hry mají omezený počet akcí za kolo a tyto akce mají snadno předvídatelné následky. Z toho důvodu je možné vyčíslit všechny proveditelné tahy hráčů a vybírat nejlepší možné varianty.

U strategií hraných v reálném čase (RTS) je situace jiná. Počet možných akcí je téměř nevyčíslitelný. V jednu chvíli zde interagují stovky, nebo i tisíce objektů. Probíhá zde obrovské množství mikro-rozhodnutí, kterým většinou není možné zcela jasně přiřadit nějakou váhu. Navíc RTS si příliš nemohou dovolit obětovat tolik výpočetní síly na řízení protivníků hráče.

[1] Společnosti vyvíjející hry nezveřejňují své algoritmy pro řízení umělých protivníků.

V současnosti se většina firem zdráhá investovat zdroje do vývoje UI pro protivníky ve strategických hrách, protože považují za dominantní herní mód ve strategiích multiplayer. Což je skutečně pravda, ale ne protože by hráči neměli zájem hrát proti umělým protivníkům. Současná UI používaná ve strategiích zkrátka neposkytuje to, co hráči vyžadují, a proto se o tento mód hraní nezajímají. Jejím největším problémem je předvídatelnost a nedostatek realistických odpovědí na akce hráče. Jinými slovy je hra pro hráče příliš jednoduchá. Ve snaze zvýšit úroveň obtížnosti se herní vývojáři často uchylují ke zkratkám a mění pravidla hry ve prospěch UI, což činí hru těžší, ale také pro hráče méně uspokojivou.[14] Schéma těchto závislostí lze vidět na obrázku 2.1.



Obrázek 2.1: Schéma relace zájmu a výzkumu UI ve strategických hrách

2.1 Strategické počítačové hry

Strategické počítačové hry jsou zaměřeny na schopnost hráče vymýšlet nové strategie a plánovat. Ve strategických hrách dostane hráč kompletní kontrolu nad budovami a jednotkami, kterým může dávat příkazy. Úkolem hráče je obvykle oslabit své nepřátele a zničit všechny jejich jednotky a budovy, čímž dosáhne vítězství. Tento cíl se může v některých hrách drobně lišit, ale prakticky bude vždy cílem hráče mít více zdrojů a silnější armádu. V počítačových strategických hrách je třeba mnoha kroků, abychom tohoto cíle dosáhli. Hráč se musí starat o infrastrukturu své civilizace, produkovat jednotky, sbírat suroviny, ovládat svou armádu a aktivně zjišťovat stav svého protivníka. Tato práce je zaměřena na Real-Time strategie, neboli zkráceně RTS.

Jedná se vojenské strategické hry, ve kterých hra probíhá v reálném čase. Dva a více hráčů ovládají jednotky a budovy na herní mapě s cílem shromáždit suroviny a zničit jednotky nepřítele společně s jeho prostředky na jejich produkci. Pomocí nashromážděných zdrojů, lze produkovat další jednotky a budovy. Každá budova, jednotka či vylepšení stojí určité zdroje, které hráč získává kontrolou nějaké oblasti nebo těžbou.

Obrázek 2.2: Ukázka uživatelského rozhraní ze hry Starcraft.

2.1.1 Prvky a mechaniky

V typické RTS je obrazovka rozdělena na mapu, jednotky, terén a uživatelské rozhraní, které dává hráči kontrolu nad herními prvky.

Hra je velmi dynamická, takže vyžaduje rychlé rozhodování a reflexy. Aby byl hráč efektivní, je potřeba, aby v jednu chvíli řídil všechny své jednotky na různých místech mapy a byl připravený reagovat na několik událostí najednou. Hráč má na provedení akcí jen malý zlomek času a jeho rozhodnutí bývají zpravidla spíše taktická, než strategická. To znamená že hráč si většinou stanovuje několik malých cílů splnitelných v krátké době, které směřují k vítězství, ale během hry často mění pořadí v jakém tyto cíle plní, nebo je vyměňuje za jiné cíle.

Hra většinou začíná umístěním hráče na náhodné místo mapy poblíž zdrojů surovin, které potřebuje na své první budovy. Hráč dostane pár základních jednotek a jeho úkolem je začít rozšiřovat své síly. Zdroje surovin jsou většinou omezené, a tudíž nutí hráče objevovat mapu a hledat nové zdroje, o které musí často také soupeřit s ostatními hráči.

Mapy se liší velikostí, počtem surovin a terénem. První dvě věci se většinou odvíjí od počtu hráčů, pro které je mapa určena. Strategické hry zavádějí mechaniku známou jako „Fog of War“, která je pro ně typická. Hráč zpravidla vidí jen to, co vidí jeho jednotky. Zbytek mapy je pro něj zahalen mlhou. Ve strategických hrách existují dva druhy Fog of War. Oblasti, které hráč ještě vůbec nenavštívil, jsou zahaleny černou, nepropustnou tmou, takže hráč nemá představu o tom, co se zde nachází. Oblasti, které hráč sice navštívil, ale nemá na nich žádné jednotky, jsou hráči zobrazeny, ale nevidí jejich aktuální stav. To znamená, že vidí terén nebo nepřátelské budovy, které tam byly, když tu oblast navštívil, ale nevidí už jednotky nebo nově postavené budovy. Z pohledu agentních systémů se tedy jedná o nedostupné prostředí, protože nelze získat úplnou informaci o stavu hry.

2.1.2 Vedení boje

V RTS hrách je většinou cílem hry buď zničení základny nepřítele nebo kompletní vyhlazení všech jeho jednotek, budov a prostředků k jejich produkci. Z toho vyplývá, že postavení velké a silné armády, která následně zaútočí na nepřátelskou základnu, je klíčem k vítězství. Ovšem obrana vlastních lokací je stejně důležitá. Ztráta předsunutých základen vede ke snížení rychlosti získávání zdrojů a produkce jednotek, což může způsobit, že nebude schopný doplňovat padlé jednotky nebo také bránit jeho vlastní základny.

Ve většině her jednotky zahájí útok na nepřítele ve chvíli, kdy jim vstoupí do zorného pole. Hráč tedy nemusí přímo řídit boj, ale jen rozhodovat kam a kdy dané vojáky poslat. Jednotky se vyskytují v různých variantách, s různými silami a slabinami. Většinou platí že určité jednotky mají výhodu proti konkrétnímu druhu nepřátelských jednotek, ale jsou zase naopak slabé proti jiným jednotkám nepřítele.

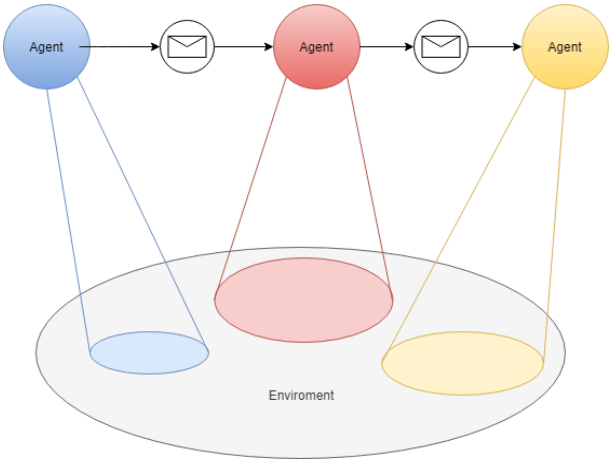
2.1.3 Ekonomika

V RTS hrách je produkce jednotek a budov omezená zdroji, které hráč vlastní. Každá jednotka a budova stojí určité množství zdrojů, které musí hráč vynaložit, aby mohl daný objekt vyprodukovat. Zdroje hráč zpravidla získává těžením určitých objektů na mapě. Někdy vyžadují, aby hráč na daném zdroji postavil budovu nebo k němu vyslal jednotky, které ho budou těžit. Čím více takových „dolů“ hráč vlastní, tím rychleji může produkovat jednotky a budovy. Z toho lze jasně vydedukovat, že obsazení a bránění těchto „dolů“ je důležitým klíčem k vítězství.

Budovy jsou hlavním zdrojem jednotek. Obvykle je v jednu chvíli každá budova schopna produkovat jednu jednotku. Různé budovy jsou schopny cvičit různé jednotky, kde platí, že čím lepší jednotka je, tím více zdrojů stojí. Rychlost, s jakou je hráč schopný produkovat jednotky, je přímo úměrná počtu tréninkových budov, které vlastní. Jelikož silnější ekonomika vede k větší produkci jednotek, jedná se o velmi důležitý aspekt hry. Je však třeba vyrovnávat rozložení sil mezi ekonomickými a válečnými aspekty hry, protože množství surovin na mapě je omezené.

2.2 Multiagentní systémy

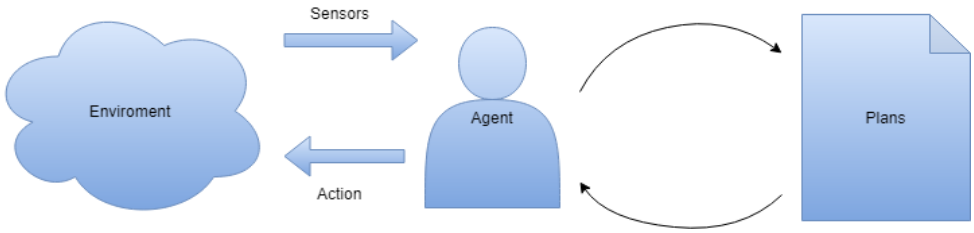
Multiagentní systém (MAS) je systém složený z několika autonomních a interagujících softwarových částí, které jsou umístěny v prostředí, jenž mohou ovlivňovat. Tyto softwarové části se nazývají agenti. Agenti v MAS se vyznačují tím, že mohou samostatně operovat jen ve své vlastní sféře vlivu. Ta omezuje jaké informace mají jednotlivý agenti k dispozici a s čím mohou interagovat. Další důležitou vlastností agentů v MAS je možnost mezi sebou vzájemně komunikovat. Vzhledem k tomu, že agenti většinou vidí jen své okolí a starají se jen o jistou část prostředí, je pro ně důležité předávat si informace a delegovat cíle ostatním agentům[22]. Tyto relace jsou naznačeny v obrázku 2.3.



Obrázek 2.3: Schéma MAS. Šipky naznačují komunikaci mezi agenty a barevné elipsy, oblast se kterou agent pracuje.

2.2.1 Agent

Agent je systém, který je umístěn do prostředí, které může monitorovat pomocí senzorů, a má určitý repertoár akcí, kterými může toto prostředí ovlivňovat. Rozhodování co a jak bude agent dělat probíhá na základě interních plánů agenta (Obrázek 2.4).[20]



Obrázek 2.4: Schéma relace agenta a prostředí.

Agent je definován následujícími vlastnostmi:

- **Samostatnost** – označuje schopnost agenta jednat samostatně na základě cílů, které mu byly dány programátorem. Po přijmutí cíle je agent schopný sestavit dílčí cíle, které musí uskutečnit, aby dosáhl požadovaného výsledku. Tyto dílčí cíle jsou sestavovány na základě plánů, daných agentovi programátorem.
- **Proaktivita** — je schopnost vykazovat cílem řízené chování. Agent se bude snažit dosáhnout cílů, které jsou mu delegovány, vytvořením a vykonáním dílčích podcílů vedoucích k požadovanému výsledku.
- **Reaktivita** — reprezentuje schopnost agenta pohotově reagovat na změny v prostředí. Reaktivní chování je série akcí, které mají být provedeny, pokud nastane určitá změna v prostředí a je potřeba na ni reagovat.
- **Sociální schopnosti** -– jedná se o schopnost agenta komunikovat s ostatními agenty v systému a koordinovat s nimi své akce.

2.2.2 JADE

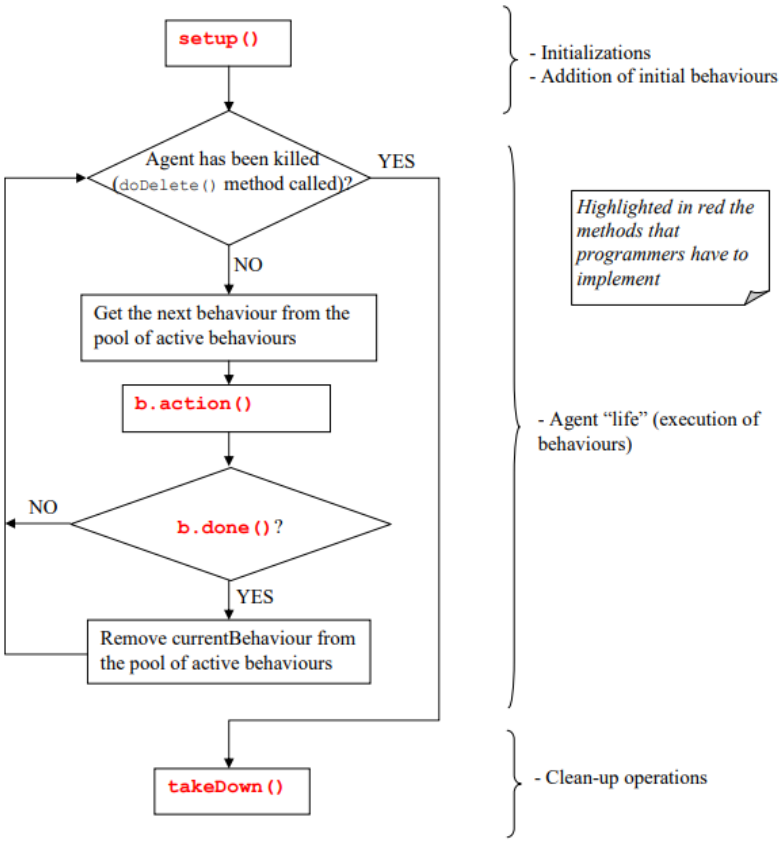
V této práci jsou využiti agenti frameworku JADE[6]. JADE je softwarový framework v jazyce Java, který usnadňuje implementaci multiagentních systémů, které komunikují s využitím FIPA specifikace zpráv[6].

Jedná se o formát zprávy definovaný organizací The Foundation for Intelligent Physical Agents(FIPA), což je mezinárodní organizace určená k šíření a podpoře inteligentních agentů[12]. ACL Message je zpráva, která obsahuje množinu jednoho a více předem definovaných parametrů. Které parametry budou použity se liší podle konkrétní situace a ne všechny musejí být validní pro konkrétní systém. Formát zpráv je definován tak, aby byl schopný obstarat libovolně složitou komunikaci a tvořil univerzální "jazyk"pro komunikaci mezi libovolnými dvěma agenty. Kompletní seznam parametrů ACL zprávy je zobrazen v tabulce v tabulce 2.6. Jediným povinným parametrem zprávy je položka *performative*, avšak přepokládá se, že ve většině případů budou obsaženy i položky *sender*, *receiver* a *content*.

JADE dále obsahuje komunikační architekturu umožňující flexibilní a efektivní zasílání zpráv, kde JADE vytváří a spravuje příchozí ACL zprávy. Dále obsahuje abstraktní třídy pro implementaci agentů a jejich chování. JADE také obstarává i paralelizaci jednotlivých agentů a jejich chování. Každý agent tak vykonává své akce nezávisle na ostatních. To ve spojení s možností neblokující komunikace vytváří distribuovaný systém, který umožňuje řešit několik úkolů najednou. Celý MAS tedy také díky tomu může běžet mimo hlavní vlákno, na kterém běží samotná hra. To umožňuje provádět řízení hry na pozadí a příkazy do hry zadávat ve chvíli, kdy se volá synchronizační funkce pro vykreslení obrazu.

2.2.3 JADE Agent

Agent je v JADE implementován jako běžná třída, tudíž může být děděn a rozšířen. Agent vytvořený pomocí JADE je vložen do speciální kontrolní třídy, která se stará o to, aby agent periodicky prováděl své chování. Chování agentů se definuje v tzv. *Behaviour* třídách. V těchto třídách jsou definovány akce, jež agent vykonává buď neustále, nebo na základě přijaté zprávy. Díky tomu lze snadno rozdělit chování agenta pro konkrétní situace. Lze tak vytvořit např. chování pro příjímání zpráv, nebo *Behaviour* modul, který bude periodicky počítat a vyhodnocovat informace z databáze. Životní cyklus agenta je zobrazen v obrázku 2.5. Komunikace mezi agenty probíhá pomocí ACL[3] zpráv. I pro ty JADE poskytuje vlastní třídu a metody pro jejich zasílání.



Obrázek 2.5: Životní cyklus agenta[5]

Parameter	Category of Parameters
performative	Type of communicative acts
sender	Participant in communication
receiver	Participant in communication
reply-to	Participant in communication
content	Content of message
language	Description of Content
encoding	Description of Content
ontology	Description of Content
protocol	Control of conversation
conversation-id	Control of conversation
reply-with	Control of conversation
in-reply-to	Control of conversation
reply-by	Control of conversation

Obrázek 2.6: FIPA ACL Message parametry[3]

2.3 Agentní systémy jako UI ve strategiích

Zatímco v tahových hrách je možné dosáhnout velmi dobrých výsledků s použitím nejrůznějších centralizovaných UI, tak v RTS existují oblasti, na které nestačí. Pro řízení RTS jsou mnohem vhodnější multiagentní systémy umožňuje pomocí několika kroků vytvořit velmi robustní a efektivní systém pro ovládání hry.

Těmito kroky jsou:

- Namapováním všech objektů ve hře jednotlivým agentům.
- Poskytnutí prostředků ke sledování herního světa.
- Vytvořením komunikačních prostředků, pomocí nichž si budou agenti předávat informace.
- Definování vzorů chování.
- Přidělení podmnožin těchto vzorů jednotlivým agentům.
- Navržení plánů agentů tak, aby podporovali kooperaci a řídili je směrem ke společnému cíli.

Provedením těchto kroků získáme agentní systém schopný nahradit konečné automaty a naskriptované chování, které se v současné době ve strategických hrách používá. V následujících kapitolách budou popsány hlavní výhody agentních systémů ve strategických hrách.

2.3.1 Rozdělení rozhodovacího procesu

Každý agent má svůj vlastní seznam cílů, které se samostatně snaží splnit. Na základě dostupných informací agent volí akce, které jej dovedou k cíli. Všechny akce, které agent zvažuje, by měli vést ke stejnému cíli, ale mohou si lišit tím, jak daného cíle dosáhnou. Výsledné chování systému je tedy založeno na souhrnu všech akcí, ke kterým se jednotlivý agenti rozhodnou. Každý agent musí být schopný zvážit externí vstupy, jako jsou vjemy nebo zprávy od ostatních agentů. Dále musí obsahovat nějakou databázi znalostí, ke které může libovolně přistupovat. Také musí být schopný se rozhodovat, kterou akci uskuteční na základě všech informací, které má k dispozici.

2.3.2 Realistické chování

Je mnoho charakteristik chování, které mohou rozlišovat umělého a skutečného protivníka. Jedná se o vlastnosti, které je třeba emulovat. Protivník by se tedy měl snažit provádět to, co by udělal skutečný hráč za daných okolností.

Jedná se například o posílání průzkumníků za účelem získání informací, což identifikuje hráče s nedokonalou znalostí svého okolí. Běžný umělý protivník tuto potřebu nemá, protože přesně zná pozice všech jednotek na mapě. Dalším znakem může být například stahování svých armád z nevýhodných situací, nebo také častá změna taktiky může být správným krokem k tomu, aby UI více připomínala hráče.

Všechny tyto vlastnosti, lze relativně snadno emulovat v agentních systémech.

2.3.3 Lepší výkon

Centralizované UI je složeno z komplexního výpočetního cyklu, které se provádí v jednom vlákně. Rozložení rozhodovacího procesu do různých částí umožňuje vytvořit vlákno pro každého agenta a využít plný potenciál paralelních systémů.

Komplexní algoritmus centralizované UI může být značně vytěžující, a z toho důvodu není prováděn po celou dobu hry, aby se nevytvářela přebytečná zátěž. Místo toho je prováděn periodicky po uplynutí určité doby, což vede k nerovnoměrnému rozdělení zátěže, a to může způsobovat výkyvy v kvalitě, s jakou hra běží.

V distribuovaných systémech může tato zátěž být rozdělena na menší a méně zatěžující výpočetní cykly, které je možno rovnoměrněji rozložit a zvýšit tím stabilitu hry.

2.4 Druhy agentních systémů pro ovládání strategických her

Agentní systémy sloužící ke kontrole počítačem řízených protivníků můžeme rozdělit do tří hlavních kategorií. Tyto kategorie byly vytvořeny podle dostupných materiálů, jež se zabývají použitím agentních systémů ve strategických hrách.[15][18][13] V následujících kapitolách budou tyto přístupy popsány společně s výhodami a nevýhodami spojenými s jejich užitím.

2.4.1 Každá entita jako agent

Při použití tohoto přístupu jsou všechny objekty ve hře modelovány jako samostatný agent. To znamená, že každá jednotka a budova ve hře má vlastní rozhodovací proces a vlastní sadu cílů. Velkou výhodou tohoto přístupu je, že vede k obrovské nepředvídatelnosti chování umělého protivníka. Vzhledem k tomu, že ve hře je velké množství agentů, kteří se nezávisle rozhodují na základě vnějších podnětů a zpráv od ostatních agentů, tak je velká pravděpodobnost, že každá hra bude probíhat jiným způsobem i při použití stejného počátečního nastavení. V ideálním případě by takový systém na každou strategii hráče reagoval jiným způsobem. Ovšem tento přístup sebou přináší i několik úskalí.

Tyto systémy vyžadují velmi složitý komunikační protokol, díky kterému by si agenti mohli efektivně předávat informace tak, aby byli schopni jednotně pracovat směrem ke společným cílům. Další podmínkou k jejich efektivní spolupráci je správné nastavení plánů jednotlivých agentů. To se může ukázat značně problematické ve hrách, kde je velké množství různých jednotek, kde každá potřebuje agenta s jinými plány. Samozřejmě je zde ještě velká náročnost na výpočetní výkon. V běžné strategické hře, kde spolu interagují stovky až tisíce agentů, kteří pokaždé musí procházet svůj rozhodovací proces, se může tento přístup ukázat jako značně neefektivní. Obzvláště proto, že velmi často se bude tento rozhodovací proces provádět zbytečně.

2.4.2 Centrálně řízené skupiny

Tento přístup předpokládá použití malého počtu agentů kontrolujících jednotlivé důležité aspekty strategických her, tzv. Manažerů. Může se jednat třeba o manažery kontrolující ekonomiku, bojové jednotky, a nebo třeba stavbu nových budov a produkci jednotek. Manažeři pak kontrolují hru jen na nejvyšší úrovni a nechávají některé triviální akce na zabudované umělé inteligenci samotné hry. Tyto systémy poskytují mnohem větší univerzálnost. Malé množství agentů, jejichž plány jsou z pravidla velmi obecné umožňuje znovupoužitelnost v několika různých hrách. Další výhodou je menší náročnost na výpočetní výkon. Nevýhodou tohoto systému je obětování decentralizace. Zobecnění kontrolních mechanismů a centralizace kontroly do menšího počtu agentů může velmi snadno vést k větší předvídatelnosti chování, což je aspekt, který není u umělého protivníka ve strategické hře vítán.

2.4.3 Hybridní přístup

Zde se spojují obě předchozí metody do jedné. Každý objekt ve hře je reprezentován velmi jednoduchým agentem, který je řízen některým z manažerů. Manažeři provádí složitá a důležitá rozhodnutí a předávají tento příkaz svým podřízeným agentům, kteří již samostatně pracují na jeho vykonání. Teoreticky se jedná o nejsilnější systém, protože odstraňuje většinu nevýhod předchozích variant s tím, že zachovává jejich výhody, ale zároveň se také jedná o systém, jenž je nejsložitější implementovat a při nesprávném rozložení zátěže mezi manažery a agenty může systém být i velmi náročný na výpočetní výkon. Jako nejideálnější rozložení zátěže se jeví ponechat tzv. makromanagment na manažerech a jednotkovým agentům přenechat tzv. mikromanagment. Příkladem může být, že manažer vybere jednotku a pošle ji těžit konkrétní surovinu, ale jednotka se již sama rozhodne kde ji bude těžit.

Kapitola 3

Návrh frameworku pro řízení agentního systému ve strategické hře

V této kapitole je popsán návrh frameworku určeného k řízení jednotek ve strategických hrách. Tento návrh je založen na agentním systému s hybridním přístupem. Systém byl zvolen díky své univerzálnosti a robustnosti, což by teoreticky mělo snížit náročnost napojení frameworku na libovolnou hru a zároveň zachovat efektivitu agentního přístupu.

Nejprve bude popsána struktura frameworku. Dále budou popsány jednotlivé komponenty, jejich význam a komunikace s jinými komponentami.

3.1 Struktura frameworku

Framework je softwarová struktura, která slouží jako podpora při programování a vývoji jiných softwarových projektů. Framework obsahuje určitou obecnou funkcionalitu pro řešení konkrétních problémů a jeho části mohou být selektivně rozšířeny, či přepsány. Framework popisovaný v této práci je multiagentní systém, ve kterém se vyskytují dva typy agentů. Jedná se o "manažery" a "jednotky".

Prvním typem jsou tzv. Manažeři. Tito agenti mají na starost řízení jednotlivých částí strategické hry na té nejvyšší úrovni. Jejich úkolem je tzv. makromanagment, to jest určovat cíle akcí, rozhodovat o nejlepším možném přístupu k problému a rozdávat úkoly "jednotkám". V systému se vyskytuje celkem pět manažerů a každý z nich zastupuje jeden z pěti hlavních cílů vedoucích k vítězství ve strategické hře. Těmito cíli jsou:

- Výběr vhodné strategie proti soupeři
- Zajištění a udržení stabilního příjmu surovin
- Nepřerušovaná stavba budov a produkce jednotek
- Vedení boje
- Průzkum mapy

Počet manažerů a jejich povinnosti byly částečně odvozeny z práce Joshe McCoye a Michaela Matease[18], kteří použily podobně dělený systém pro ovládání strategické hry Wargus. Úkoly manažerů však byly zobecněny a upraveny, aby byly použitelné pro co největší množinu her.

Druhým typem agentů jsou tzv. Jednotky, nebo jednotkoví agenti. Jednotky se zase starají o řízení hry na té nejnižší úrovni. Tento agent má minimální rozhodovací schopnosti. Jeho úkolem je převážně komunikace s konkrétní hrou a převádění příkazů od manažerů do podoby jimž hra rozumí. Jednotkoví agenti obstarávají tzv. mikromanagment. Tento přístup byl inspirován frameworkem Atlantis [2], který vytváří systém řízený manažery pro hru Starcraft: Brood War. V tomto frameworku jsou vytvářeny obaly nad herními reprezentacemi jednotek, které zobecňují příkazy pro tyto entity a zařizují mikromanagment.

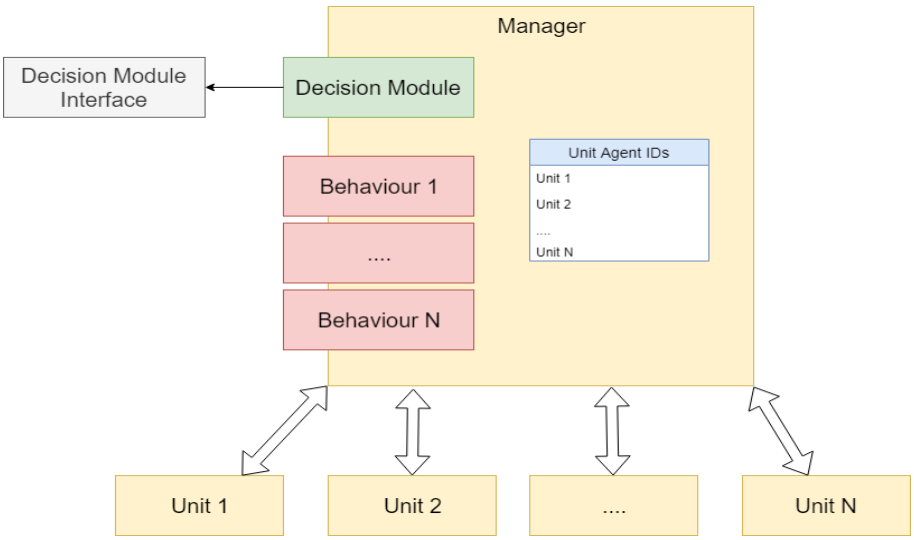
3.1.1 Manažeři

Všichni manažeři mají jednotnou strukturu (viz. obrázek 3.1). Každý manažer obsahuje rozhodovací modul (Decision Module), seznam svých podřízených jednotek a několik chování (Behaviour).

Rozhodovací modul je třída implementující frameworkem definované rozhraní pro konkrétního manažera. Manažeři volají metody modulu, když potřebují rozhodnout nějaký netriviální problém. Může se jednat například o uspořádání fronty budov a jednotek čekajících na produkci, nebo výběr nepřátelské základny na kterou je třeba zaútočit. Uživatel frameworku si může buď implementovat vlastní rozhodovací modul, nebo použít výchozí implementaci frameworku.

Každý manažer obsahuje jedno chování, které se stará o příjem a vyhodnocování zpráv. Dále mohou obsahovat další chování, které mohou sloužit například pro analýzu herních dat, nebo provádění nějaké periodické činnosti, jako je třeba produkce jednotek.

Poslední částí manažera je seznam jeho podřízených jednotek. Jedná se o jednotkové agenty, kteří mu byly přiděleni a které může žádat o vykonání nějaké akce. Jedná se o jeho primární spojení se hrou a skrz tyto agenty manažer zadává příkazy ke stavbě budov, produkci jednotek a samozřejmě samotného pohybu a útoku jednotky.

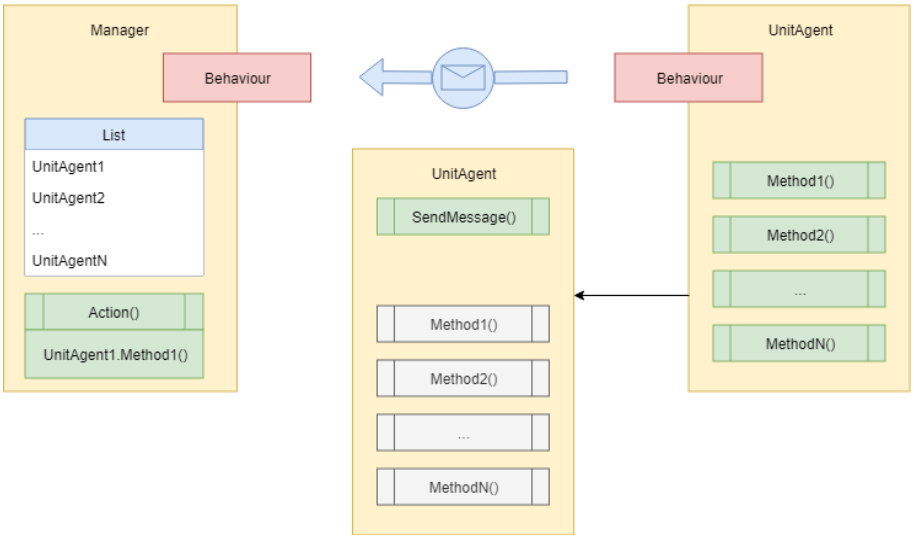


Obrázek 3.1: Struktura agenta typu "manažer"

3.1.2 Jednotkový agent

Jednotkoví agenti slouží jako propojení frameworku a hry. Jedná se o abstraktní třídu, která má implementovány funkce pro přijímání a zasílání ACL zpráv. Každá jednotka je přiřazena nějakému manažerovi a ten pak skrze ni může komunikovat s konkrétní hrou. Pokud vznikne potřeba, aby s touto jednotkou pracoval jiný manažer, tak její vlastník může s daným manažerem domluvit její předání. Dále mohou být skrze jednotkové agenty předávány znalosti o hře do databází znalostí jednotlivých manažerů.

Vzhledem k tomu, že agent komunikuje přímo s hrou, tak zasílání jednotlivých zpráv musí být implementovány uživatelem. Framework definuje jen zprávy pro předávání a mazání znalostí z agentů, které mohou být ve vhodných příležitostech zaslány uživatelem, aby měli manažeři aktuální informace o hře. Je však poskytnuto rozhraní pro zasílání a přijímání vlastních zpráv, pokud by uživatel chtěl tímto způsobem jednoduše rozšířit funkcionalitu celého agentního systému. Dále je definován seznam metod, které jsou volány manažery pro vykonání určité akce ve hře. Tyto metody zajišťují pohyb jednotky po mapě, stavbu budovy, útok na cíl a podobně. Struktura tohoto agenta je naznačena v obrázku 3.2. Kompletní seznam a popis metod jednotkového agenta je uveden v dokumentaci, kterou lze nalézt v příloze B.



Obrázek 3.2: Struktura agenta typu "Jednotka"

3.2 Strategy manager

Úkolem Strategy managera je volit vhodnou strategii pro každou fázi hry a kontrolovat, zda je zvolená strategie stále nejlepší volbou. Může se jednat o obecnou strategii hry, nebo strategii stavění budov. Tyto strategie ovlivňují cíle a plánování všech manažerů. Strategie jsou rozděleny podle fáze hry, pro kterou jsou určeny, a Strategy manager mezi nimi vhodně přepíná. Aby mohl tuto činnost efektivně provádět, potřebuje mít informace o stavu manažerů, existujících budovách, jednotkách a dalších objektech na mapě. Může se jednat například o pozici nepřátel na mapě nebo potencionální místa k expanzi.

Strategy manager byl navržen tak, aby podporoval rozpoznávání nepřátelských strategií, tak jak je popsáno například v práci Geert L. J. Pingena[19], která se zabývá dolováním strategií ze hry Starcraft: Brood War.

3.2.1 Povinnosti Strategy managera

Strategy manager potřebuje pro svou činnost komunikovat se všemi manažery v agentním systému. Tento agent na základě svých představ o herním světě vybírá nejlepší vhodnou strategii a informuje ostatní manažery o krocích, které musí podniknout pro naplnění této strategie. Jedná se například o nastavení stavebních plánů nebo určení nepřátelských cílů. Komunikace, jež probíhá mezi jednotlivými manažery, má následující podobu:

- Economic manager - Pro splnění dané strategie v co nejlepším čase je nutný minimální příjem surovin. Strategy manager o této hodnotě informuje Economic managera, který tuto hodnotu vyhodnocuje a vyjedná s Infrastructure managerem prostředky, aby tohoto cíle dosáhl.

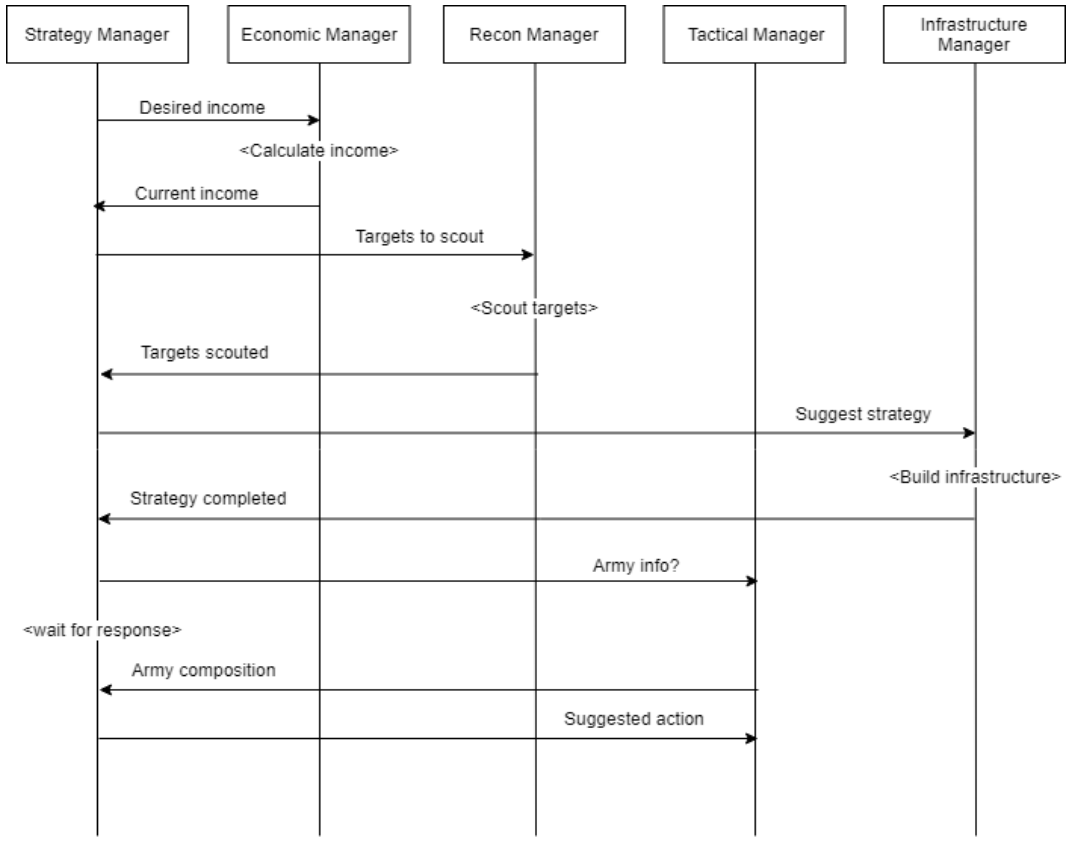
- Infrastructure manager - Každá strategie, se kterou se pracuje, je složena z několika akcí, které musí být provedeny Infrastructure managerem. Strategy manager tento plán nalezne a zašle mu ho. Strategie mají uveden cíl, jenž by měl být jejich výsledkem. Ve chvíli, kdy je tento cíl dosažen, je vyhodnocena nová strategie a Infrastructure managerovi je zaslán nový plán.

- Tactical manager - Strategy manager vyhodnocuje svou databázi znalostí a je z ní schopen zjistit, které základny je třeba bránit a na které je třeba útočit. Díky tomu může zaslat Tactical managerovi pozice těchto základen, společně s příkazem k obraně, či útoku. Také je schopný od něj získat složení armády, kterou Tactical manager momentálně disponuje.

- Recon manager - Strategy manager je schopen získat informace o potencionálních startovních lokací nepřítele a místech, kde může vzniknout expanze. Pozici těchto míst zasílá Recon managerovi, který pak nalezne nejlepší cestu k jejich prozkoumání.

3.2.2 Návrh komunikace

Pro plnění těchto povinností byla navržena následující komunikace mezi agenty:



Obrázek 3.3: Komunikační schéma Strategy managera

Komunikace s Infrastructure managerem

Po zvolení vhodné strategie je o této strategii informován Infrastructure manager pomocí zprávy obsahující jméno strategie, podle které si musí naplánovat své akce tak, aby se dostal k cíli co nejefektivněji. Mimo to je také možné nastavit Infrastructure managerovi konkrétní stavební strategii pro danou fázi hry, kterou si pak může tento manažer pomocí rozhodovacího modulu upravovat.

Komunikace s Economic managerem

Strategy manager má v rozhodovacím modulu schopnost vypočítat potřebný příjem surovin, aby byla nově zvolená strategie splněna co nejrychleji a informuje Economic managera o této hodnotě definovanou zprávou. Tato hodnota je pak uložena v Economic managerovi a ten může její hodnotu používat při rozesílání dělníků k surovinám a rozhodování, kolik dělnických jednotek potřebuje pro svou činnost. Economic manager poté pravidelně zasílá aktuální příjem surovin Strategy Managerovi, který tuto informaci může využít ke změně strategie.

Komunikace s Tactical managerem

Strategy manager se pravidelně ptá Tactical managera na složení jeho armády. Poté podle své databáze znalostí vyhodnotí zda je čas spíše bránit, či útočit a informuje o svém rozhodnutí Tactical managera. Ten potom podle svých znalostí o hře rozhodne jakým způsobem vybranou akci uskuteční.

Komunikace s Recon managerem

Recon manager od Strategy managera dostává vhodné cíle k průzkumu. Potom co takový cíl získá, tak si zajistí vhodné jednotky a provede průzkum daného místa. Jak je určené místo prozkoumáno informuje o tom Strategy managera a pokračuje ve své činnosti.

3.3 Economic manager

Úkolem tohoto manažera je zajistit, aby hráč měl dostatek zdrojů k budování infrastruktury a vytváření nových jednotek. K této činnosti vyžaduje kontrolu nad jednotkami určenými k těžbě surovin, stejně jako nad budovami, které tyto zdroje produkují. Zajišťuje, aby tyto jednotky a budovy byly efektivně využity tím, že jim přiřazuje práci tak, aby byl přírůstek jednotlivých zdrojů úměrný rychlosti, s jakou jsou spotřebovávány. Hlavním cílem Economic mangera je rovnoměrné rozložení pracovní síly takovým způsobem, aby nebyly zbytečně produkovány suroviny, které momentálně nejsou prioritní na úkor surovin, které jsou aktivně využívány.

Návrh Economic managera byl inspirován existujícími agenty pro těžení surovin ve strategických hrách, jako je například SC2_HarvesterAgent[4], nebo ResourceGather[9].

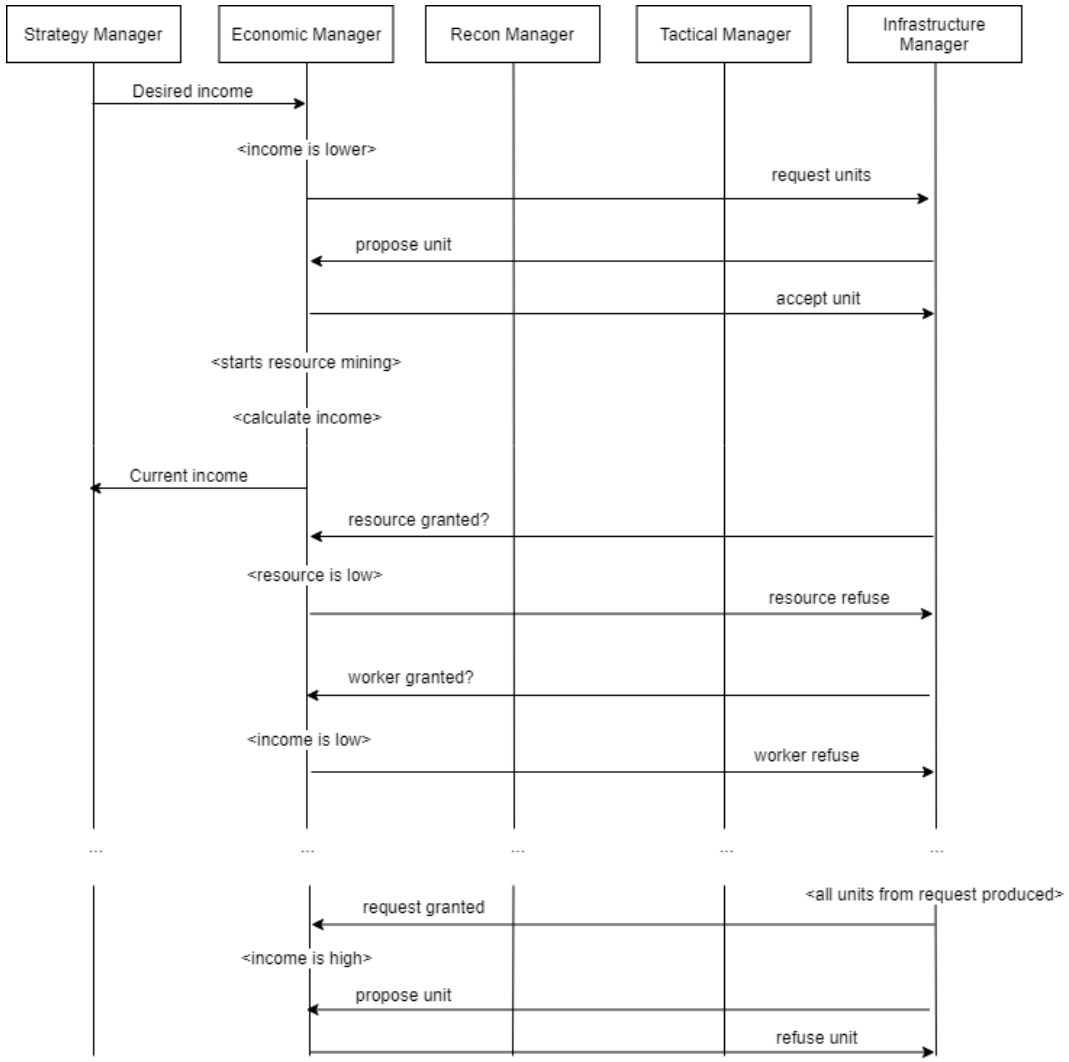
3.3.1 Povinnosti Economic managera

Aby mohl Economic manager správně plnit svou roli v systému, musí komunikovat se Strategy managerem a Infrastructure managerem. Strategy manager by měl být, na základě svých představ o herním světě, schopen určit pravděpodobnou míru příjmů, která je potřeba k naplnění ním aktuálně zvolené strategie. Pokud nelze tuto hodnotu z nějakého důvodu získat, je nutné, aby si požadovaný příjem surovin určil podle komunikace, která probíhá s Infrastructure managerem.

Economic manager po celou dobu hry dostává všechny vytvořené pracovní jednotky. K tomuto předání dochází pomocí příchozích zpráv od Infrastructere managera oznamujících jeho ochotu vzdát se nově vytvořených jednotek v jeho prospěch. Vzhledem k tomu, že ve většině her jsou tyto jednotky určené jak k těžení surovin, tak i ke stavbě budov, je nutná existence komunikačního kanálu zařizujícího zapůjčení pracovní jednotky ke stavbě a její následné vrácení. Dále Infrastructure manager musí být schopen požádat o přidělení zdrojů, která provede rezervaci daných surovin do té doby, než je akce, pro kterou byly určeny, započata a zdroje skutečně utraceny.

3.3.2 Návrh komunikace

Pro plnění daných povinností byla pro Economic managera navržena následující komunikace s ostatními agenty:



Obrázek 3.4: Komunikační schéma Economic managera

Získávání jednotek

Předávání jednotek probíhá tak, že pokud má Infrastructure manager vyrobené dělnické jednotky, tak je nepřetržitě nabízí Economic Managerovi, který je přijímá, nebo odmítá zasláním zprávy. Dělník předaný Economic managerovi je poslán těžit prioritní surovinu. To je realizováno zavoláním vybrané metody nad jednotkovým agentem. Tato metoda by měla zajistit, aby byl dělník poslán těžit požadovaný zdroj. Ve chvíli, kdy zjistí, že surovina dochází, je jí nadbytek nebo ji není schopen produkovat dostatečně rychle, může zažádat o expanzi, přidělit dělníka na těžbu jiné suroviny nebo poslat žádost o vytvoření více dělníků pomocí zprávy.

Rezervace surovin a dělníků

Dalším druhem komunikace, která probíhá mezi Infrastructure a Economic managerem, je příjem a odesílání odpovědí na zprávy Infrastructure managera, které žádají o přidělení zdrojů, nebo dělníků. Po přijetí zprávy o zdroje vyhodnotí zda má dostatek surovin, tyto suroviny zarezervuje a nedovolí, aby byly použity, dokud nedostane zprávu o jejich uvolnění. Následně odešle příslušnou zprávu v případě, že zdroje bylo možné zarezervovat, nebo odmítnutí pokud není dostatek zdrojů na vyřízení této žádosti. Obdobná komunikace probíhá i při žádosti o dělníka, kde se Economic manager vzdává jednoho ze svého dělníků, pokud uzná, že tím nebude výrazně narušen jeho přísun prioritních surovin.

3.4 Infrastructure manager

Tento manažer se stará o plynulou produkci jednotek a budov. Jeho úkolem je na základě dané strategie stavět budovy takovým způsobem, aby byly co nejefektivněji využity dostupné zdroje v co možno nejkratším čase. Toho je docíleno jak neustálou optimalizací pořadí naplánovaných akcí, tak i úzkou komunikací s Economic managerem. Mimo to je jeho úkolem shromažďovat vytvořené jednotky a zasílat je příslušným manažerům. Pokud nedostane specifickou žádost, je jeho úkolem zjistit, komu se má vyrobená jednotka přiřadit. Toto rozhodování je prováděno na základě agentových současných informací o hře.

Infrastructure manager byl navržen tak, aby mimo jiné podporoval učení posilováním, jak je popsáno například v práci Jieverson Maissiata a Felipe Meneguzziho[16].

3.4.1 Povinnosti Infrastructure managera

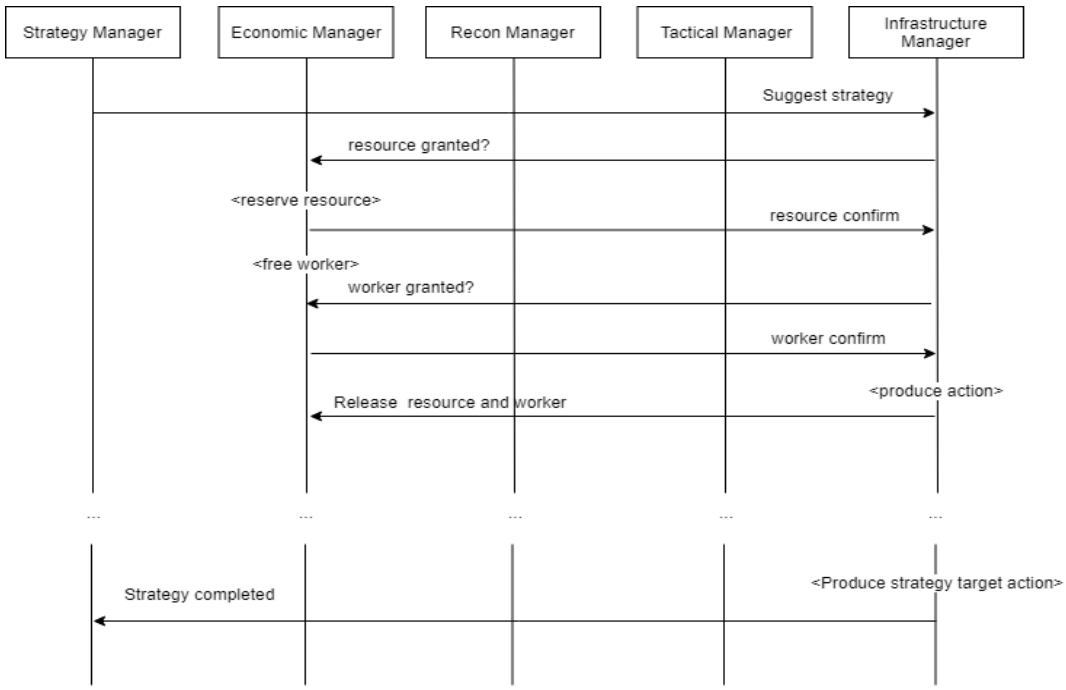
Infrastructure manager má nejsložitější komunikační síť v celém agentním systému. Musí být schopen oboustranně komunikovat téměř se všemi agenty v systému. Během celého svého životního cyklu musí Infrastructure manager jednat s Economic managerem o přidělení zdrojů na produkci jednotek a stavbu budov. Mimo to také musí se všemi managery vyjednávat předání kontroly nad nově vytvořenými jednotkami. V neposlední řadě musí být schopen od ostatních manažerů přijímat mimo plánové žádosti o specifické jednotky. Tyto komunikace vypadají následovně:

- Economic manager - Infrastructure manager má s Economic managerem několik akcí na kterých se musí domluvit. Prvním z nich je žádost o přidělení zdrojů k produkční akci. Následně si od něj musí vyžádat pracovníka, pokud je používána stejná jednotka k produkčním i těžebním akcím. Posledním typem komunikace jsou žádosti o produkci těžařských jednotek a předání jejich kontroly.
- Tactical manager - Od Tactical managera jsou zasílány žádosti o specifické jednotky a informace o tom, zda je třeba vůbec bojové jednotky v danou chvíli produkovat. I zde probíhá komunikace o předání kontroly nad jednotkami.
- Recon manager - Tato komunikace je téměř totožná jako s Tactical managerem. Jediným rozdílem je reakce na žádost o průzkumné jednotky. Vyplnění této žádosti má menší prioritu a ta je splněna až ve chvíli, kdy je Economic managerovi a Infrastructure managerovi přiděleno minimum jednotek, které potřebují pro svou činnost.

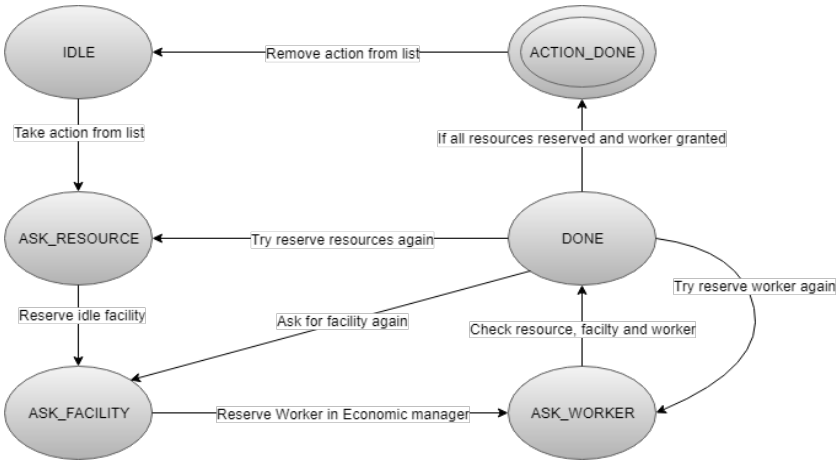
- Strategy manager - Od Strategy managera tento agent přijímá stavební plány v podobě strategií, jejichž cílem je většinou určitý počet budov, nebo jednotek. Následně zasílá Strategy managerovi informaci o splnění zadané strategie.

3.4.2 Návrh komunikace

První navržená komunikace mezi agenty je stanovení strategie a produkce potřebných jednotek. K vykonávání této činnosti Infrastructure manager používá konečný automat z obrázku 3.6 a vyjednávání zobrazené v komunikačním grafu na obrázku 3.5.



Obrázek 3.5: Komunikační schéma Infrastructure managera pro plnění strategie



Obrázek 3.6: Stavový automat zajišťující provádění akcí v Infrastructure managerovi

Provádění plánovaných akcí

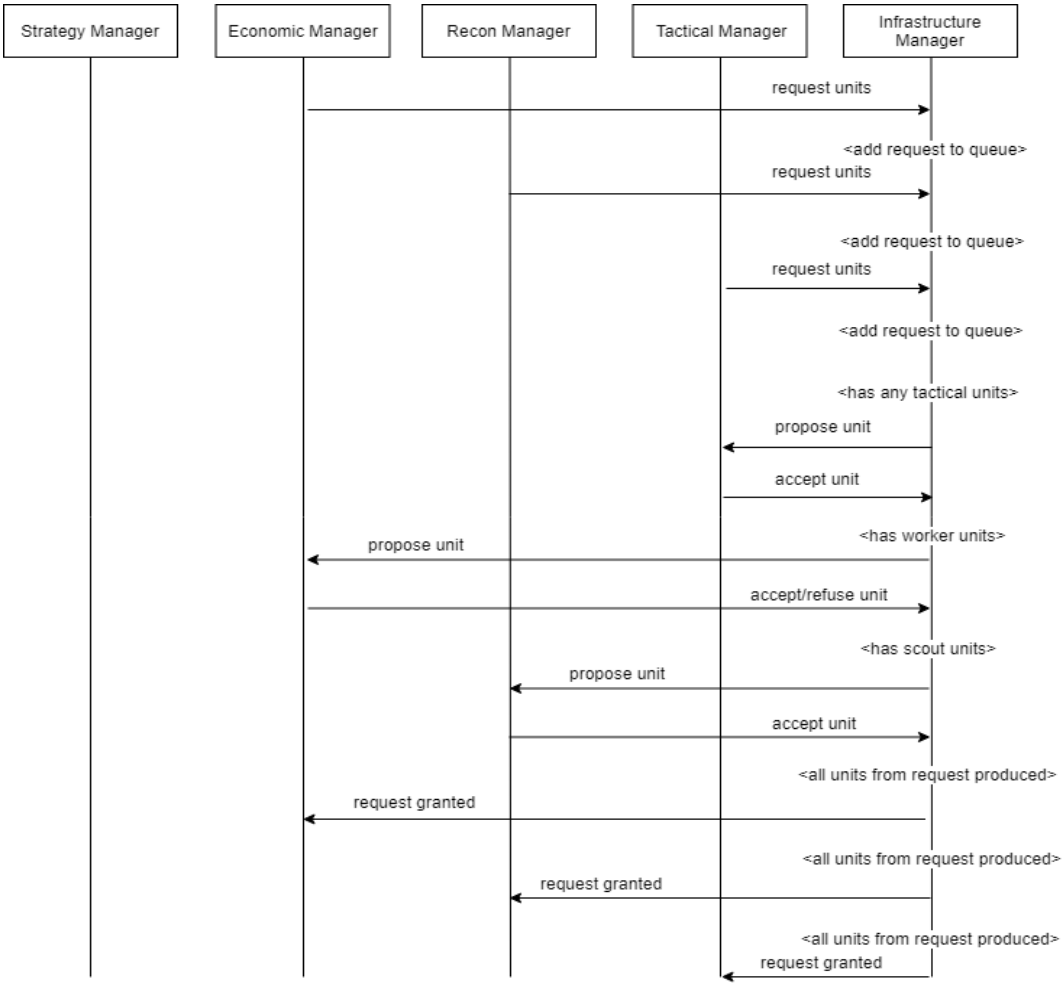
K plnění své hlavní funkce, kterou je stavění budov a produkce jednotek, využívá stavový automat. Po přijetí strategie je z definovaných kroků vytvořeno pole akcí, které je třeba vykonat. Stavový automat postupně odebírá jednotlivé položky z pole naplánovaných akcí a vykonává je. Automat začíná ve stavu IDLE. Pokud není seznam naplánovaných akcí prázdný, vybere první akci ze seznamu a přejde do stavu ASK_RESOURCE. V tomto stavu zašle Economic managerovi zprávu, že by chtěl zarezervovat zdroje pro jednotku/budovu, jenž je výsledkem právě prováděné akce a přejde do stavu ASK_FACILITY. Zde pomocí další zprávy zjistí, zda budova, která produkuje právě vytvářenou jednotku/budovu, není zrovna obsazená. Pokud není, tak ji zarezervuje a přejde do stavu ASK_WORKER. Zde zašle opět zprávu Economic managerovi, kterou ho požádá o vydání stavitele. Tím přechází do stavu DONE, kde čeká na odpovědi od Economic managera. Pokud byla odpověď na některou ze zpráv negativní, automat přechází do příslušného stavu a požádá o tuto akci znova. V případě, že odpověď na všechny žádosti byla pozitivní, automat provede zadanou akci a odstraní ji z pole naplánovaných akcí. V běhu automatu se může stát, že projde znovu stavy, ve kterých se zasílají žádosti o rezervaci. V agentovi je poznačeno, že tato žádost již byla splněna, a není pro současnou akci znova posílána. Tím je zajištěno, že vše je přiděleno pro každou akci maximálně jednou. Po provedení akce jsou uvolněny zarezervované zdroje a stavitel vrácen Economic managerovi. Obojího je docíleno zasláním potvrzovací zprávy z Infrastructure managera.

Předávání jednotek

Všechny produkční budovy, dělníci, vojáci i vozidla jsou předávány Infrastructure managerovi. Tyto objekty jsou v něm ukládány do příslušných seznamů. Tyto seznamy jsou pak v manažerovi procházeny a je rozhodováno, kterou jednotku kam poslat. Ve výchozím nastavení jsou všechny vytvořené dělnické jednotky posílány Economic managerovi a ve chvíli kdy má minimum jednotek pro svou funkci, může být nově vytvořený dělník zaslán i Recon managerovi, pokud Infrastructure manager v minulosti přijal zprávu o tom, že Recon manager potřebuje jednotky, které ještě nebylo vyhověno. Všechny bojové jednotky jsou zase zasílány Tactical managerovi.

Pokud byla od některého manažera přijata žádost o produkci jednotek, tak je taková jednotka přidána do příslušného seznamu v Infrastructure managerovi a ten při nejbližší možné příležitosti tuto žádost odbaví a zašle příslušnému manažerovi zprávu o kladném vyřízení jeho žádosti.

Tyto zprávy jsou zobrazeny v komunikačním grafu na obrázku 3.7.



Obrázek 3.7: Komunikační schéma Infrastructure managera pro předávání jednotek

3.5 Tactical manager

Tactical manager se stará o ovládání jednotek určených k boji a volbu vhodné bojové taktiky. Jeho hlavním úkolem je shlukovat všechny dostupné vojáky do skupin a přidělovat těmto skupinám cíle. Tito vojáci jsou mu přiřazováni Infrastructure managerem, který mu předává kompletní kontrolu nad všemi jednotkami určenými k boji. Cíle, které jsou jednotlivým skupinám přidělovány, jsou určeny podle aktuálně zvolené strategie.

Pokud nemá zadanou nadřazenou strategii od Strategy managera, volí si sám na jaký cíl ze své databáze znalostí má zaútočit a kdy se stáhnout k bodu na mapě, který je třeba bránit. Toho dosahuje tím, že neustále vyhodnocuje sílu své a nepřátelské armády pomocí znalostí, které získal buď on sám během svých útoků, a nebo pomocí znalostí, které objevil Recon manager.

Tactical manager byl navržen tak, aby podporoval shlukování armád a řízení útočných akcí. Tento návrh byl částečně inspirován prací Gabriela Synnaeve a Pierra Bessiere[21], kteří zde tyto problémy řeší.

3.5.1 Povinnosti Tactical managera

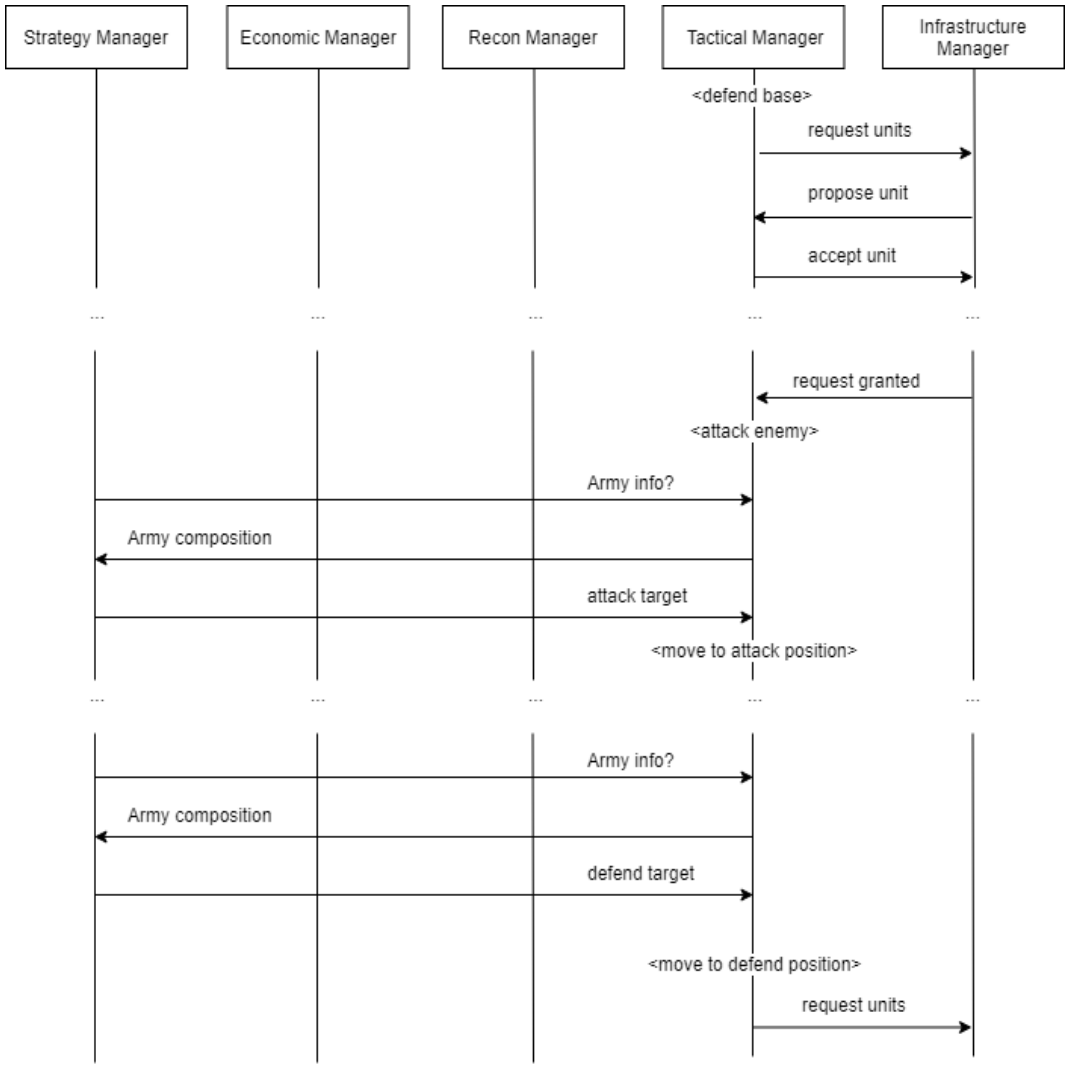
Tactical manager potřebuje ke své činnosti minimální kontakt s ostatními agenty. Komunikuje pouze se Strategy managerem a Infrastructure managerem. V obou případech se jedná o obousměrnou komunikaci a jedná se převážně o předávání informací.

- Infrastructure manager – Tactical manager je průběžně informován o každé jednotce určené k boji, kterou tento manažer vytvořil. Kontrola nad těmito jednotkami je po vzájemné domluvě předána Tactical managerovi. Ten v průběhu svého životního cyklu průběžně informuje Infrastructure managera o tom, zda aktuálně potřebuje více jednotek, nebo když zjistí, že nepřítel využívá strategii, proti které potřebuje specifickou jednotku, může ho informovat, že od něj vyžaduje konkrétní jednotku.

- Strategy manager – Tactical manager od něj může dostat nařízeno útočit, či bránit konkrétní oblasti. Tento příkaz dočasně omezuje schopnost agenta měnit si své vlastní cíle. Tactical manager v tuto chvíli používá většinu svých bojových sil k útoku nebo obraně zadané oblasti, dokud nedostane od Strategy managera zprávu o tom, že si zase může cíle libovolně měnit. Strategy manager si také může od Tactical managera vyžádat informace o aktuálním složení armády, aby mohl vyhodnotit její sílu, podle vlastních měřítek, a vhodně zvolit novou strategii.

3.5.2 Návrh komunikace

Pro plnění těchto povinností byla navržena následující komunikace:



Obrázek 3.8: Komunikační schéma Tactic managera

Vedení boje

Dokud Tactical manager nemá nepřátelskou základnu ve své databázi znalostí, tak zůstává poblíž základny, kde hlídá nejbližší "choke point". Jedná se o místo označující úzký vstup do prostoru, kde se nachází základna hráče. Ve chvíli, kdy Recon manager objeví nepřátelskou základnu a Tactical manager vyhodnotí, že jeho armáda má větší sílu než nepřítel, tak přejde do útoku. Tactical manager neustále vyhodnocuje sílu své a nepřátelské armády. Síla nepřátelských vojsk je vypočítávána podle uložených znalostí o nepřátelích, které které získal od Recon managera, nebo svých jednotek. Síla jeho armády není počítána z celkového množství jednotek. ale pouze z těch, které jsou momentálně členem útočné skupiny.

Dokud Tactical nedostane jiný příkaz, snaží se shlukovat všechny jednotky, které jsou mu přiděleny do jedné taktické skupiny, kterou posílá útočit na nepřátelské oddíly a základny,

jejichž síla je menší, než jeho. Pokud žádná taková místa nejsou, stáhne se na poslední zadanou pozici, kterou je třeba bránit, a čeká na posily. Všechny ostatní jednotky, které jsou vytvořeny a předány Tactical managerovi, se vydají směrem k nejbližší taktické jednotce a ve chvíli, kdy se dostanou dostatečně blízko středu některé této skupiny se stávají jejím členem, jejich síla se začne připočítávat k celkové síle armády. Pro toto chování je třeba implementovat metodu pro útok na danou pozici, při kterém budou jednotky útočit na vše co jim přijde do cesty, metodu pro bránění zadané pozice a funkci, která dokáže vypočítat vzdálenost jednotky od středu taktické skupiny.

Komunikace se Strategy managerem

Tactical manager může přijímat cíle k obraně a útoku od Strategy managera, které dočasně zablokují schopnost Tactical manager měnit si tyto cíle. Ve chvíli kdy dostane zprávu tohoto typu, tak má Tactical manager zakázáno tento cíl změnit, dokud nedostane novou zprávu o tom, že krize byla překonána, a Tactical manager může zase plnit svou roli bez omezení. Poslední zprávou kterou přijímá je žádost o popis aktuální armády, jejíž odpověď Strategy manager používá pro vytváření nových strategií, nebo výběru cílů k obraně, či útoku.

3.6 Recon manager

Úkolem Recon managera je aktivně prozkoumávat mapu a získávat informace pro ostatní agenty. Hledá na mapě potencionální místa k expanzi, základny nepřátel, pozice soupeřových armád a zjišťuje jejich složení. Průzkumné jednotky získává od Infrastructure managera a cíle průzkumu mu buď přiděluje Strategy manager nebo si je určuje sám. Jak Strategy manager, tak Recon manager by měli mít ve své databázi znalostí uložené znalosti o tom, co se kde na mapě může nacházet. Úkolem Recon managera je zjistit, co na těchto místech skutečně je.

3.6.1 Povinnosti Recon managera

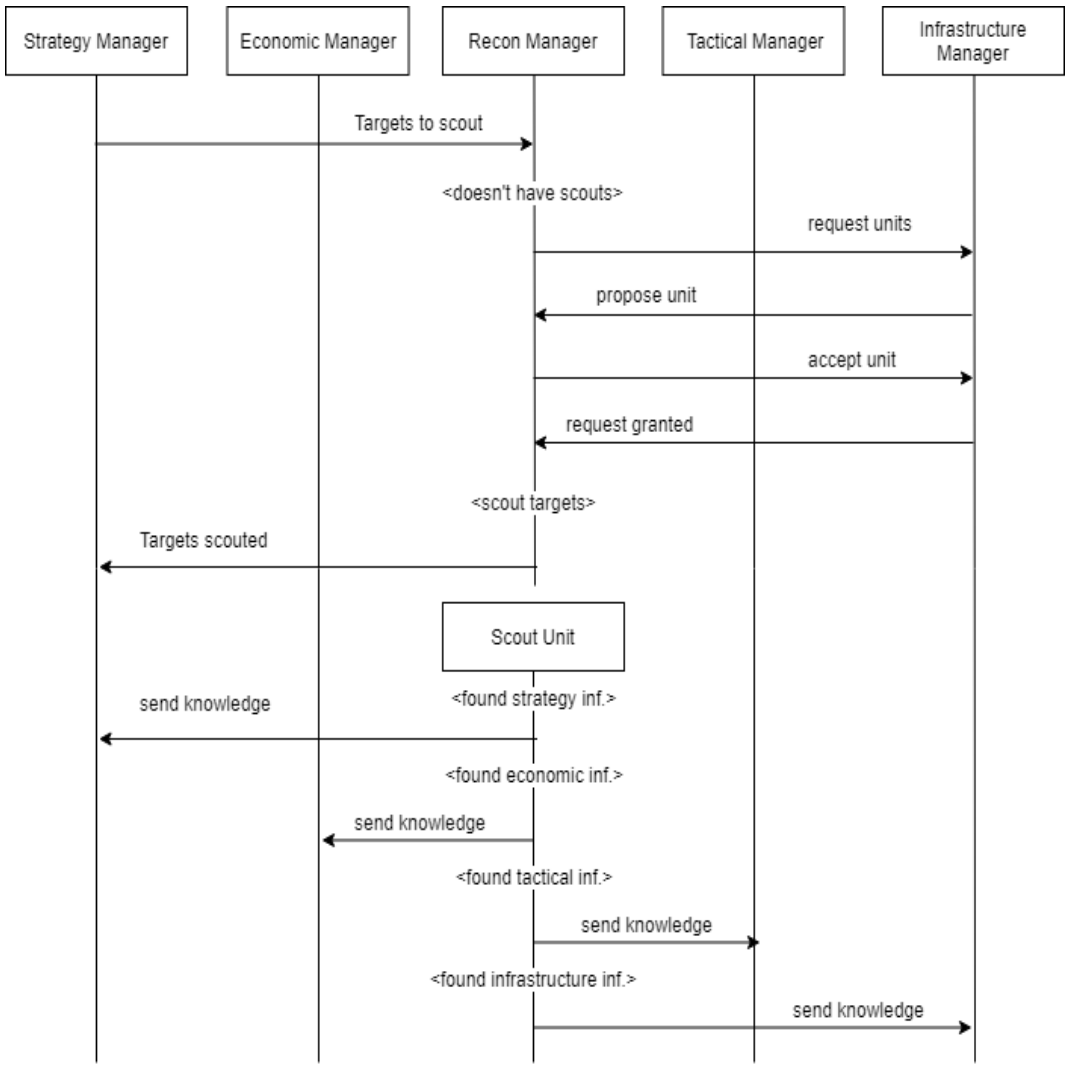
Recon manager, stejně jako Tactical manager, komunikuje obousměrně pouze s Infrastructure managerem a Strategy managerem. Recon manager může ostatním manažerům předávat informace dvěma způsoby. To, která varianta se použije, závisí na herním API a uživateli. Uživatel může předat znalosti přímo Recon managerovi, který poté pomocí svých rozhodovacích modulů zjistí, kterým manažerům má tyto informace zaslat. Druhou variantou je, že uživatel bude nové znalosti zasílat konkrétním agentům přímo pomocí jednotkových agentů. Zbytek komunikace probíhá následovně:

- Infrastructure manager – Recon manager musí žádat o zaslání průzkumných jednotek. Ve chvíli, kdy Infrastructure manager shledá, že má jednotku, jež by mohl obětovat na průzkum, tzn. že není potřeba vytvářet dělníky nebo není krizová situace, která vyžaduje prioritní vytváření bojových jednotek, vytvoří a zašle průzkumnou jednotku Recon managerovi. Recon manager si také může zažádat o speciální jednotky, které jsou například neviditelné, nebo jsou naopak schopny neviditelné jednotky detekovat.

- Strategy manager – Strategy manager vybírá ze své databáze prioritní lokace, které je třeba prozkoumat a zasílá je Recon managerovi. Ten následně volí nejlepší cestu jak tyto objekty objevit a prozkoumat. Objevené informace jsou pak ukládány do databáze znalostí Strategy managera a ostatních agentů pro které jsou objevené informace relevantní.

3.6.2 Návrh komunikace

Pro plnění role průzkumníka, byla Recon managerovi navržena následující komunikace:



Obrázek 3.9: Schéma Recon managera

Recon manager dostane od Strategy managera cíl průzkumu. Pokud nemá žádné průzkumníky, tak si o nějaké požádá Infrastructure managerovi. Jakmile získá potřebnou jednotku pošle ji na dané místo. Předání získaných znalostí je provedeno buď průzkumníkem, nebo je znalost předána ostatním manažerům přímo přes Recon managera. Pro správnou činnost tohoto manažera je nezbytné, aby uživatel jedním z těchto způsobů zasílal definovanou zprávu obsahující znalost, která je uložena do báze znalostí agenta, jež ji přijal.

Kapitola 4

Popis implementace

V této kapitole budou popsány použité technologie a konfigurační soubory sloužící k nastavení frameworku. Dále zde bude popsána báze znalostí agentů a způsob, jakým ji jednotlivý agenti využívají. Následně bude popsána implementace jednotlivých manažerů a jejich komunikačních protokolů. Framework je vytvořen v jazyce Java a implementace probíhala v IDE IntelliJ IDEA.

4.1 Použité technologie

Tato práce využívá k implementaci agentů Java Agent Development Framework (JADE), který pro komunikaci mezi agenty používá zprávy podle specifikace FIPA. Data do konfiguračních souborů jsou zapsána v JavaSript Object Notation (JSON).

4.1.1 JSON

Jedná se o způsob zápisu dat nezávislý na platformě určený pro přenos dat, která mohou být organizována v polích nebo objektovém zápisu. Vstupem může být libovolná struktura obsahující řetězec, číslo, boolean, objekt nebo pole složené ze dříve zmíněných typů. Výstupem je vždy řetězec.[7] JSON formát je v této práci použit k tvorbě konfiguračních souborů, které frameworku slouží ke snadnějšímu napojení na konkrétní hru. V tomto formátu jsou také uložena data z předchozích her, která slouží jako podpora učení.

4.2 Konfigurační a výstupní struktury

K napojení frameworku na konkrétní hru je potřeba definovat několik konfiguračních souborů. Tyto souboru jsou následně načteny do struktur, které framework používá v jednotlivých agentech k ovládání hry a správnému čtení konkrétních situací jež mohou ve hře nastat. Tento přístup je nutný k tomu, aby bylo možné framework napojit na libovolnou hru. Dále framework musí ukládat určitá data z manažerů na konci každé hry. Tyto data se načítají před každou novou hrou a využívají se v rozhodovacích algoritmech implementovaných ve vytvořeném agentním systému. Takto vytvořená databáze umožňuje agentnímu systému učit se mezi hrami. V následujících kapitolách bude blíže popsána struktura a účel jednolivých konfigurací.

4.2.1 Technology Tree

Technology Tree (zkr. TT) je konfigurační struktura obsahující popis jednotek, budov, vylepšení a všech dalších objektů, jenž se mohou ve hře vyskytovat. Do TT se vkládají pouze objekty u kterých je třeba, aby s nimi framework přímo interagoval. To znamená, že do TT není třeba dávat jednotky a budovy, které hráč neovládá. Tyto objekty jsou vkládány, až do báze znalostí, která je vytvářena dynamicky v průběhu hry. Konfigurační soubor pro TT je ve formátu JSON a jedná se o pole objektů, kde každý objekt symbolizuje jeden uzel stromu. Parametry tohoto objektu jsou popsány v tabulce 4.1 a 4.2. TT využívají ve větší či menší míře všichni manažeři, avšak jeho hlavním úkolem je sloužit jako podpora pro konfiguraci akcí a strategií, které jsou popsány v dalších kapitolách.

Tabulka 4.1: Parametry uzlu Technology Tree

Jméno	**Typ**	**Hodnota**
name	<NodeName>	Jméno TT uzlu
type	<Type>	worker|unit|building|upgrade|supply
width	<Number>	Šířka jednotky/budovy
height	<Number>	Výška jednotky/budovy
price	Array<Price>	Cena vytvoření uzlu
requires	Array<NodeName>	Seznam vyžadovaných uzlů
unlocks	Array<NodeName>	Seznam odemčených uzlů

Tabulka 4.2: Price parametry

Jméno	**Typ**	**Hodnota**
name	<ResourceName>	Jméno zdroje
value	<Number>	Počet jednotek zdroje

4.2.2 Action List

Je konfigurační struktura obsahující definici akcí, které provádí manažeři. Seznam parametrů akce je uveden v tabulce 4.3. Každá akce má kromě jména definován uzel, jenž je výsledkem této akce, a taky její kompletní cenu. Tato cena může kromě samotné ceny uzlu obsahovat také i další položky. Do ceny může být zahrnut například čas, jež dělníkovi zabere přesun na pozici stavby. V agentním systému vytvořeném pomocí frameworku musí být také určeno mapování těchto akcí, na konkrétní akci přímo v hře.

Tabulka 4.3: Parametry Action

Jméno	**Typ**	**Hodnota**
name	<ActionName>	Jméno akce
id	<number>	Unitkátní identifikátor akce
price	Array<Price>	Kompletní cena akce
node	<NodeName>	Jméno vyprodukovaného uzlu

4.2.3 Build Orders

Konfigurace obsahující popis strategií stavění, které mohou být použity. Seznam a popis jednotlivých parametrů lze najít v tabulce 4.4. Každá strategie stavění má definováno, zda se jedná o úvodní strategii, nebo o strategii určenou pro pozdější část hry. Tyto strategie definují, které akce je třeba provést a který uzel TT má být výsledkem této strategie. Seznam Build Orders spravuje Strategy manager, který pak vybranou strategii posílá Infrastructure managerovi. Infrastructure manager si pak plánuje akce uvedené v parametru *buildOrder* a doplňuje je o akce z *fill*, které jsou vhodně voleny. Tyto strategie jsou prováděny dokud není vytvořen uzel TT, který je uveden v *target* parametru.

Tabulka 4.4: Parametry BuildOrder

Jméno	Typ	Hodnota
name	<BuildOrderName>	Jméno strategie
type	<BuildOrderType>	opening|standard|lateGame
target	<NodeName>	Jméno uzlu jež je cílem této strategie
buildOrder	Array<ActionName>	Seznam akcí jež je nutno v této strategii vykonat
fill	Array<ActionName>	Seznam akcí, které jsou použity jako výplň plánu

4.2.4 Strategy List

Konfigurace obsahuje popis herních strategií, které mohou hráči ve hře použít. Tato struktura by měla obsahovat všechny známé strategie hráče i jeho protivníků. Strategy List je pole objektů (tabulka 4.5), kde každý objekt obsahuje jméno herní rasy a pole Strategy objektů. Parametry tohoto objektu jsou uvedeny v tabulce 4.6. Každá strategie má definované jméno, unikátní identifikátor, seznam jednotek, budov a vylepšení. Strategy manager může tuto strukturu používat k rozpoznání nepřátelské strategie a zjištění jakou strategii zvolit proti němu.

Tabulka 4.5: Parametry Strategy List objektu

Jméno	Typ	Hodnota
name	<RaceName>	Jméno hratelné, nebo nepřátelské rasy
strategies	Array<Strategy>	Unikátní identifikátor

Tabulka 4.6: Parametry Strategy

Jméno	Typ	Hodnota
name	<StrategyName>	Jméno strategie
id	<number>	Unikátní identifikátor
nodes	Array<{name, count}>	Seznam jednotek a budov
upgrades	Array<{name, level}>	Seznam vylepšení a jejich úrovně

4.2.5 Managers History

Framework nabízí dvě možnosti jak uchovávat data z předchozích her. První z nich je ukládání výsledků hry do objektu Managers History. Po každé hře mohou být uloženy informace, které byly získané v jejím průběhu a také výsledky akcí agentů. Například pro strategy managera jsou ukládány použité strategie a to jestli vedly k vítězství nebo prohře. V průběhu hry jsou tyto informace uchovávány v instanci třídy ManagersHistory, do které je jednotlivý agenti zapisují. Na konci hry jsou tyto informace převedny do JSONu a uloženy do souboru. Formát, v jakém jsou tyto informace uloženy, je zobrazen v tabulkách 4.7, 4.8, 4.9 a 4.10.

Tabulka 4.7: Parametry Managers History

Jméno	Typ	Hodnota
strategyManager	{<FactionObject>, ...}	Objekt obsahující hratelné rasy
reconManager	<ReconObject>	Objekt popisující výsledky průzkumu
economicManager	{<EconomicObject>,...}	Objekt s historií zdrojů

Tabulka 4.8: Parametry Faction Object

Jméno	Typ	Hodnota
<StrategyName1>	{played: <Number>, won: <Number>}	První strategie v objektu
<StrategyName2>	{played: <Number>, won: <Number>}	Druhá strategie v objektu
	...	
<StrategyNameN>	{played: <Number>, won: <Number>}	N-tá strategie v objektu

Tabulka 4.9: Parametry Recon Object

Jméno	Typ	Hodnota
<SearchMethod1>	Array<Number>	Časy nalezení nepřítele první metodou
<SearchMethod2>	Array<Number>	Časy nalezení nepřítele druhou metodou
	...	
<SearchMethodN>	Array<Number>	Časy nalezení nepřítele N-tou metodou

Tabulka 4.10: Parametry Economic Object

Jméno	Typ	Hodnota
<ResourceName1>	Array<EconomicRecord>	Záznam o příjmů z první hry
<ResourceName2>	Array<EconomicRecord>	Záznam o příjmů z první hry
	...	
<ResourceNameN>	Array<EconomicRecord>	Záznam o příjmů z N-té hry

Tabulka 4.11: Parametry Economic Record

Jméno	Typ	Hodnota
minIncome	<Number>	Minimální příjem zdrojů během jedné hry
maxIncome	<Number>	Maximální příjem zdrojů během jedné hry
avgIncome	<Number>	Průměrný příjem zdrojů během jedné hry
won	<Boolean>	Informace zda byla hra vyhrána
workers	<Number>	Počet pracovníků, kteří byli k dispozici

4.2.6 Stavy agentů

Druhým výstupem frameworku, který slouží jako podpora pro zpětnovazební učení, je ukládání kompletního seznamu stavů, kterými agent během hry prošel. Každý agent má definovaný svůj stavový objekt, který popisuje veličiny určující herní stav manažera. Vždy když agent přejde do nového stavu vytvoří instanci svého stavového objektu a uloží jej do interního seznamu. Všichni manažeři obsahují metodu, která může být zavolána na konci hry, která tento seznam převede do JSONu a uloží do zadaného souboru. Parametry stavů pro Infrastructure mangera jsou v tabulce 4.12, Economic manager je v 4.13, Recon managera v 4.14, Tactical manager v 4.15 a Strategy manager je v tabulce 4.16. Stav Strategy Managera obsahuje pouze herní čas a vybranou strategii, protože všechny ostatní informace, které jsou pro něj relevantní jsou již uloženy ve stavech ostatních manažerů.

Tabulka 4.12: Infrastructure Manager State

Jméno	Typ	Hodnota
bases	<Number>	Počet hlavních budov
workerRatio	<Number>	Poměr dělnických jednotek k fázi hry
nodes	Array<{name, count}>	Seznam postavených budov
upgrades	Array<{name, level}>	Seznam vyprodukovanych vylepšení
freeSupply	<Number>	Počet volné populace
facilities	<Number>	Počet produkčních budov

Tabulka 4.13: Economic Manager State

Jméno	Typ	Hodnota
gameTime	<Number>	Aktuální čas
baseCount	<Number>	Počet hlavních budov
workersCount	<Number>	Počet dělníků
resources	Array<{name, count}>	Seznam aktuálních surovin

Tabulka 4.14: Recon Manager State

Jméno	Typ	Hodnota
gameTime	<Number>	Aktuální čas
chosenSearchMethod	<String>	Zvolená metoda prohledávání
exploredLocationCount	<Number>	Počet prozkoumaných lokací
timeOfFirstEnemyBaseDiscovery	<Number>	Čas objevení prvního nepřítele
timeSinceLastScoutingAction	<Number>	Čas od poslední průzkumné akce
unexploredLocationsCount	<Number>	Počet neprozkoumaných lokací

Tabulka 4.15: Tactical Manager State

Jméno	Typ	Hodnota
gameTime	<Number>	Aktuální čas
armyPower	<Number>	Síla armády
enemyPower	<Number>	Síla nepřítele
fightersCount	<Number>	Počet vojáků
attacking	<Boolean>	Příznak zda útočí či brání

Tabulka 4.16: Strategy Manager State

Jméno	Typ	Hodnota
gameTime	<Number>	Aktuální čas
chosenStrategy	<String>	Zvolená strategie

4.3 Databáze znalostí

Do tohoto objektu jsou každému agentovi ukládány jeho znalosti o hře. Každý agent má svou vlastní databázi znalostí, do které jsou vkládány znalosti jež jsou pro něj relevantní. Předávání a odebírání znalostí manažerům probíhá pomocí ACL zpráv. Uživatel frameworku musí zajistit, aby tyto zprávy byly zasílány vždy, když ve hře vznikne, či zanikne nějaká znalost. Uživatel pro tuto činnost může buď použít funkce jednotkového agenta, nebo může tyto znalosti předávat Recon managerovi, který je rozešle ostatním agentům. Manažeři jsou navržení tak, aby byli schopní tento konkrétní typ zprávy přijmout od jakéhokoliv agenta v systému.

V databázi znalostí (zkr. DZ) by se neměly vyskytovat duplicitní informace. To znamená, že pokud přidávaná znalost již v DZ je, jen se aktualizují údaje v ní uložené. Každý agent má množinu znalostí, které ho zajímají, a pokud se ve hře objeví znalost, která do této kategorie spadá, měla by být zaslána do DZ konkretního agenta. Agenti jsou pak schopní v této databázi hledat a rozhodovat se na základě informací, které zde mají uložené. Tyto znalosti nemusejí být nutně vždy pravdivé ani aktuální. Aby byla pro agenty práce s DZ co nejjednodušší, musí být vhodně strukturována. První formou rozdělení je kategorizování znalostí podle vlastníka dané informace. Vlastníkem je myšlena herní entita, která vlastní objekt, který uložená znalost popisuje.

Ve frameworku rozlišujeme čtyři typy vlastníků:

- PLAYER – znalosti týkající se hráče
- ALLY – znalosti týkající se spojenců
- ENEMY – znalosti o nepříteli
- NEUTRAL – neutrální objekty, které nikdo nevlastní (typicky se jedná např. o neobsazené zdroje surovin).

Znalosti rozdělené podle vlastníka jsou pak dále kategorizovány podle typu znalosti. Typy znalostí jsou následující:

- INFORMATION – Typ udávající abstrakní informaci, která se nemusí týkat reálného objektu ve hře.
- RESOURCE – Typ označující zdroj surovin.
- BUILDING – Typ označující budovy.
- UNIT – Typ označující jednotky.
- UPGRADE – Typ označující vylepšení
- WORKER – Typ označující speciální typ jednotky, která je schopna shromažďovat suroviny a stavět budovy.
- UNKNOWN – Typ rezervovaný pro znalosti neurčité povahy.

V jednotlivých kategoriích jsou pak uloženy seznamy objektů, které obsahují znalost. Tento objekt obsahuje parametry, jako jsou např. pozice, se kterou je znalost spojena, ukazatel na jednotkového agenta, kterého se znalost týká, nebo uživatelem určený parametr síly jednotky. Jednotliví agenti si do databáze znalostí ukládají následující informace:

- Strategy manager – Informace o všech základnách hráče i nepřítele, síle nepřátelských jednotek, potencionálních místech k expanzi a míst, kde se mohou nacházet nepřátelé.
- Economic manager – Informace o místech, kde se nachází suroviny, jednotkách určených k těžbě těchto surovin a budovách, které je mohou produkovat.
- Infrastructure manager – Informace o existujících budovách hráče, jednotkách určených ke stavbě budov a místech, kde tyto budovy mohou být postaveny.
- Tactial manager – Informace o všech jednotkách určených k boji jak hráče, tak nepřítele a potencionálních míst, na které může být proveden útok.
- Recon manager – Informace o jednotkách určených k průzkumu a míst, které by měly být prozkoumány.

4.4 Implementace agentů

Framework pracuje se dvěma druhy agentů. Jsou jimi manažeři a jednotkoví agenti. Jednotkoví agenti jsou implementováni jako abstraktní třída, která definuje sadu metod, které slouží k vykonání konkrétních akcí ve hře. Tato třída obsahuje i již implementované metody, které zapouzdřují určité akce, které uživatel musí provádět pro správný běh manažerů.

Manažeři jsou implementováni jako rozšíření třídy `Agent` z frameworku JADE. Všichni manažeři obsahují svou databázi znalostí, rozhodovací modul a několik chování. Chování jsou třídy rozšiřující třídu `Behaviour` z frameworku JADE a jsou v nich umístěny funkce manažerů, které je potřeba provádět nezávisle na běhu hry. Rozhodovací moduly jsou rozhraní definující metody, jejichž návratová hodnota může být výsledek učícího algoritmu, či jiné formy rozhodování.

Framework také obsahuje třídu `FrameworkController`, která slouží k tomu, aby uživateli usnadnila inicializaci agentního systému, vytvoření vlastních rozhodovacích modulů a přidávání jednotkových agentů do systému.

4.4.1 `FrameworkController`

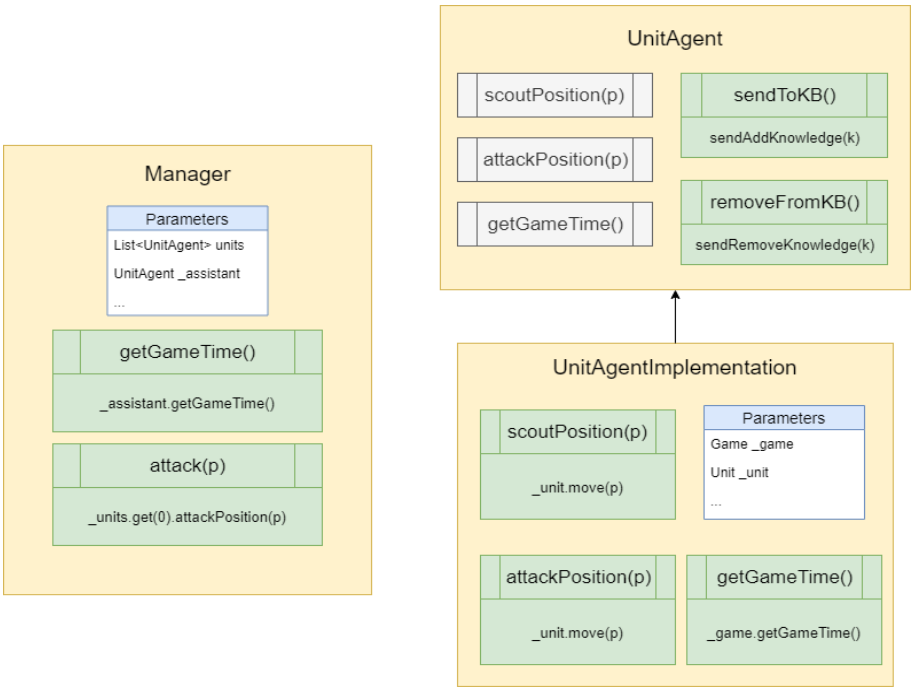
Jedná se o třídu, která slouží k inicializaci frameworku a zapouzdření inicializace JADE. Skrz instanci této třídy by měli být vytvářeny instance jednotlivých manažerů a instance tříd které čtou konfigurační vstupy. Pokud by uživatel chtěl rozšířit některé manažery, měl by to udělat právě přetížením metody této třídy pro vytvoření rozšířeného manažera a také by zde měli být frameworku předány cesty ke konfiguračním souborům. Tato třída zařizuje inicializaci JADE, vložení manažerů do příslušných JADE struktur a jejich nastartování. `FrameworkController` zařizuje i přidávání jednotkových agentů do agentního systému a jejich inicializaci.

4.4.2 `UnitAgent`

Je abstraktní třída, která obsahuje definici všech metod, které volají jednotlivý manažeři k vykonání akcí ve hře. Důvodem, proč manažeři volají metody nad UnitAgentem místo toho, aby mu poslali zprávu se žádostí o vykonání dané akce, je že většina metod, které takto manažeři volají, vrací nějakou hodnotu a na vstupu mají složité objekty. JADE sice umožňuje přidávat libovolné objekty k ACL zprávám, ale tato praktika není součástí FIPA specifikace ACL zprávy. Druhou možností by bylo převádět objekty do textového formátu (např. JSON) a z něj poté objekt rekonstruovat, ale i to má své úskalí. Dále je tu problém s asynchronitou, protože většina metod slouží manažerům k rozhodování o svých dalších akcích a čekání na odpověď pro každý takový dotaz by vytvářelo zbytečné zpomalení a složitější životní cyklus agenta. Proto bylo rozhodnuto, že přínos plně agentního přístupu k propojení se hrou není natolik významný, aby vyvážil problémy, které by tím vznikli.

Proto jediná agentní komunikace, která je v třídě `UnitAgent` implementována je zasílání informace o vzniku a zániku znalostí jednotlivým manažerům. Bylo tak rozhodnuto kvůli tomu že znalost ve frameworku jde jednoduše serializovat a u databáze znalostí je počítáno s tím, že není aktuální. Třída obsahuje metody pro vytvoření obsahu zprávy vkládající znalost, zprávu odstraňující znalost a metody pro odesílaní těchto zpráv.

U jednotkových agentů se počítá s tím, že každá jeho instance bude spojena s nějakou jednotkou, která se ve hře aktuálně vyskytuje a metody třídy `UnitAgent` budou reprezentovat stav a informace o této konkrétní jednotce. Jedinou výjimkou je instance třídy `UnitAgent`, která vstupuje do konstruktoru všech manažerů a je interně nazývána "asistent". Všichni manažeři občas potřebují zjišťovat určité informace ze hry, které nejsou vázané na konkrétní herní entitu (např. zjištění herního času), nebo potřebují zjistit nějakou herní informaci i když žádnou jednotku nevlastní (např. Strategy manager nekontroluje jednotku nikdy). Proto má každý manažer svého asistenta, který mu poskytuje trvalé spojení se hrou a který není vázaný na fyzický objekt ve hře.



Obrázek 4.1: Zjednodušený diagram vztahu UnitAgenta a manažera

4.4.3 Manager

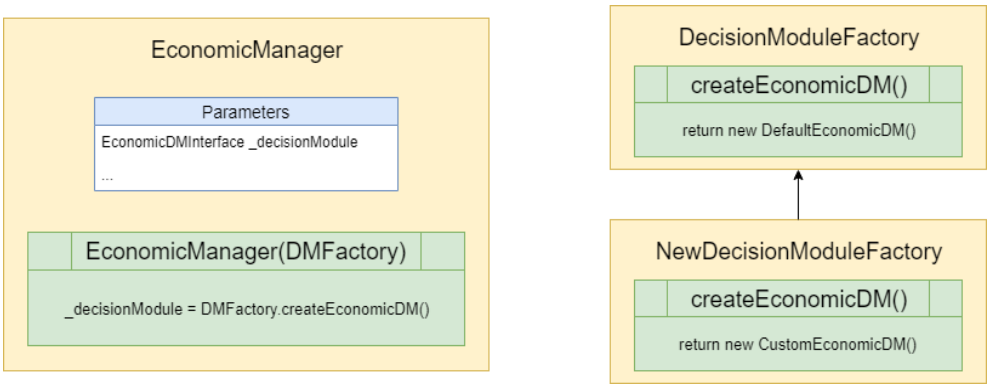
Jedná se o základní třídu, ze které vychází všichni ostatní specializovaní manažeři. Tato třída zapouzdřuje chování, které využívají všichni manažeři. Jedná se především o odesílání ACL zpráv, vytváření databáze znalostí a zpracovávání obsahu ACL zpráv od jednotkových agentů, které popisují objekty, co mohou být do databáze znalostí vloženy, nebo které je třeba z ní naopak odstranit.

Dále jsou zde definovány tři metody, které musí každý manažer implementovat. Jedná se o metody `transferUnit()`, `freeUnit()` a `onFrame()`. Metody `transferUnit()` a `freeUnit()` slouží k předávání jednotek mezi manažery a jsou volány interně v reakci na příslušné zprávy. Chování jednotlivých manažerů, včetně zasílání a přijímání zpráv je zapouzdřeno v *Behaviour* třídách JADE agentů a je prováděno asynchronně po celý život manažera. Metoda `onFrame()` je místem kde manažer předává svůj současný stav a rozhodnutí hře, kterou chceme frameworkem ovládat. Uživatel frameworku musí zajistit, aby tato metoda byla ve vhodný okamžik pravidelně volána nad všemi manažery.

Dále jsou definovány metody `updateState()` a `writeStates()`. Metoda `updateState()` vytvoří aktuální stav manažera uloží jej do interního seznamu. Metoda `writeStates()` zase slouží k tomu, aby stavy uložené v tomto interním seznamu přidala do zadaného souboru. Tyto metody jsou takto definovány, aby bylo možné poskytnout uživateli kontrolu nad tím co se do jednotlivých stavů ukládá.

4.4.4 Rozhodovací moduly

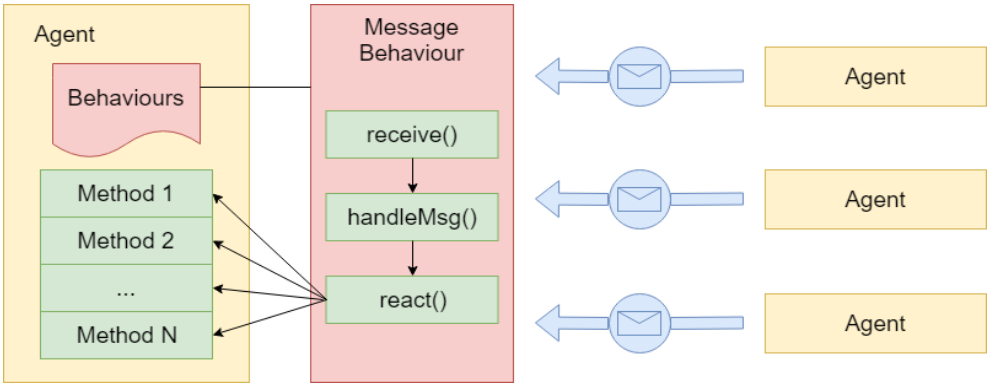
Všichni manažeři obsahují svůj vlastní rozhodovací modul. Ve frameworku je implementována třída `DecisionModuleFactory`, která vytváří instance rozhodovacích modulů pro jednotlivé manažery. Metody těchto tříd jsou používány k rozhodování netriviálních problémů, jejichž řešení může být řešeno pomocí nějaké formy strojového učení. Rozhodovací moduly jsou definovány jako rozhraní, které je třeba implementovat. Implementace těchto rozhraní není pro ovládání strategické hry povinná, protože framework obsahuje výchozí implementaci pro všechny manažery. Tyto implementace obsahují jen základní funkčnost a jejich rozhodnutí nejsou založena na žádné metodě strojového učení. Definované rozhraní rozhodovacích modulů pro jednotlivé manažery nejsou příliš rozsáhlé, ale lze je jednoduše rozšířit uživatelem, který může rozšířené rozhodovací moduly používat pomocí JADE *Behaviour*, která vybranému manažerovi může jednoduše přidat.



Obrázek 4.2: Zjednodušený diagram předání vlastního rozhodovacího modulu

4.4.5 Přijímání zpráv

Přijímání zpráv probíhá ve všech manažerech velmi podobně. Všichni manažeři mají vytvořené chování, které obstarává příjem a zpracovaní zpráv od ostatních agentů. Většinu této funkcionality obstarává framework JADE, framework musí je zajistit že v implementovaném chování je volána funkce JADE agenta pro příjem zprávy a její následné zpracování. Obecně je u každé zprávy zjištěn její odesílatel (*sender*) a typ komunikace (*performative*). Následně je podle kombinace těchto dvou parametrů ACL zprávy určeno jaké informace zpráva nese a ty jsou pak vyextrahovány z obsahu zprávy (*content*). Poté je podle získaných informací zavolána požadovaná metoda agenta. Vizualizace těchto akcí je na obrázku 4.3



Obrázek 4.3: Přijímání zpráv v manažerech

4.4.6 Strategy Manager

Strategy Manager je implementován do manažera se dvěma chováními.
`MainStrategyBehaviour` zajišťuje aby Strategy manager zasílal ostatním manžerům zprávy definované v tabulce 4.17. Tyto zprávy odpovídají komunikaci navržené v kapitole 3.2. O tom zda bude zpráva odeslána rozhodují parametry a metody ve Strategy managerovi. Ty jsou konfigurované druhým chováním agenta implementovaným ve třídě `StrategyAnalyzeBehaviour`. Toto chování analyzuje databázi znalostí a pomocí rozhodovacího modulu se snaží rozpoznat strategii nepřítele a podle toho určit priority ostatním manažerům a vytvořit tím svou vlastní strategii. Mimo to také ukládá tyto informace do struktury `ManagersHistory`.

Rozhraní rozhodovacího modulu

- `int recognizeEnemyStrategy(List<BasicKnowledge> enemyUnits)`

  Vstupem metody je seznam nepřátelských jednotek, které má manažer ve své databázi znalostí a výstupem metody by měl být index strategie nacházející se v konfigurační struktuře `StrategyList`.

- `int getResponseToEnemyStrategy(int enemyStrategyId)`

  Vstupem je index strategie nepřítele a výstupem index strategie, kterou použít proti zvolené nepřátelské strategii.

- `String selectOpeningBuildOrder(List<String> strategies)`

  Vstupem metody je seznam strategií pro prvotní fázi hry, výstupem je vybraná strategie.

- `String selectStandardBuildOrder(List<String> strategies)`

  Vstupem metody je seznam strategií pro pokročilou fázi hry, výstupem je vybraná strategie.

- `String selectLateGameBuildOrder(List<String> strategies)`

  Vstupem metody je seznam strategií pro závěrečnou fázi hry, výstupem je vybraná strategie.

- `Position getAttackPointOfInterest(KnowledgeBase knowledgeBase, PositionType positionType)`

  Do metody vstupuje databáze znalostí agenta a typ pozice co v ní mýt hledána. Výstupem je požadovaná pozice pro útok, obranu, nebo průzkum.

- `HashMap<String,Integer> getNeededResources(Integer strategyId, BuildOrder currentBuildOrder)`

  Na vstupu metody je aktuálně zvolený index strategie hry a objekt zvolené stavební strategie. Výstupem by měla být mapa potřebných surovin, která bude zaslána Economic managerovi.

Tabulka 4.17: Zprávy odesílány ze Strategy Managera

Perfomative	**Content**	**Receiver**
REQUEST	armyInfo	Tactical Manager
INFORM	newStrategy-<StrategyId>	Infrastructure Manager
INFORM	buildOrder-<BuildOrderName>	Infrastructure Manager
INFORM	income-<name>:<value>;...	Economic Manager
INFORM	reconTargets-<X>-<Y>	Recon Manager
INFORM	target-defend-<X>-<Y>	Tactical Manager
INFORM	target-attack-<X>-<Y>	Tactical Manager
INFORM	noActionRequired	Tactical Manager

4.4.7 Economic Manager

Economic manager je také implementován do manažera se dvěma chováními. První chování `EconomicDecisionBehaviour` pomocí rozhodovacího modulu zjišťuje zda má manažer k dispozici dostatek dělníků a případně pomocí INFORM zprávy z tabulky 4.18 požádá Infrastructure managera o nové. Chování `EconomicIncomeAnalyzingBehaviour` počítá průměrný příjem jednotlivých surovin, které jsou ukládány do struktury `ManagersHistory` a mohou být použity ke strojovému učení. Z tohoto chování jsou také Strategy managerovi posílány aktuální stavy surovin.

Vyjednávání s Infrastructure managerem o přidělení surovin a dělníku, popsané v kapitole 3.3, probíhá v *MessageBehaviour*, kde jsou v reakci příchozí zprávy volány metody Economic managera, které rozhodují o odpovědi na tyto žádosti.

Posílání dělníků do práce probíhá v metodě `onFrame()`, kde si manažer při každém zavolání ověří stav svých jednotkových agentů a přiřadí jim požadované úkoly. O tom co se bude těžit a v jaké míře rozhoduje podle informací, které jsou mu dány Strategy Managerem.

Rozhraní rozhodovacího modulu

- `int checkWorkerCount(int numberOfWorkers, int gameTime, HashMap<String, Double> totalResourcesMined)`

  Tato metoda má na vstupu aktuální počet dělníků, herní čas v sekundách a mapu herních surovin s aktuálními hodnotami. Výstupem by měl být počet dělníků o které si má manažer zažádat.

- `TechnologyTreeNode shouldBuildExpansionBuilding( List<BasicKnowledge> buildings, List<String> resourceGenerators, List<String> bases, int gameTime)`

  Metoda má na vstupu seznam již postavených budov, seznam jmen hlavních budov a generátorů surovin získaných ze struktury `TechnologyTree` a aktuální herní čas. Výstupem této metody je list technologického stromu reprezentující hlavní budovu, nebo zdroj surovin o který si má manažer zažádat.

Tabulka 4.18: Zprávy odesílány z Economic Managera

Perfomative	Content	Receiver
ACCEPT_PROPOSAL	worker	Infrastructure Manager
CONFIRM	worker	Infrastructure Manager
CONFIRM	resources	Infrastructure Manager
REFUSE	worker	Infrastructure Manager
REFUSE	resources	Infrastructure Manager
INFORM	need-<NodeName>-<Amount>	Infrastructure Manager
INFORM	currentIncome-<name>:<value>;...	Strategy Manager

4.4.8 Infrastructure Manager

Infrastructure manager je opět implementován do manažera se dvěma chováními. `InfrastructureMainBehaviour` obsahuje implementaci konečného automatu z obrázku 3.6, který se obstarává odebrání akce ze seznamu naplánovaných akcí a získání všech prerekvizit pro její provedení. Infrastructure manager žádá o dělníky a suroviny pomocí zpráv Economic managera a jeho odpovědi jsou zpracovávány, stejně jako všechny ostatní zprávy v *MessageBehaviour*.

Ve chvíli kdy má Infrastructure manager všechny prerekvizity pro provedení akce, tak je v metodě `onFrame()` zavolána metoda `runAction()` nad UnitAgentem, který zajistí provedení požadované akce ve hře. Je vyžadováno od uživatele, aby nad Infrastructure managerem zavolal metodu `actionStarted()`, aby tak dal manažerovi vědět, že akce je

ve fázi kdy už pro ni nemusí držet rezervované suroviny, nebo dělníky. To je nutné kvůli tomu, že ve strategických hrách se může lišit to, kdy je možné dělníky a suroviny uvolnit.

Pokaždé když `InfrastructureMainBehaviour` vybírá ze seznamu naplánovaných akcí novou položku, je vytvořen aktuální seznam možných akcí podle databáze znalostí a technologického stromu. Následně je zkontrolováno zda Infrastructure manager nemá nevyřízené požadavky na nové jednotky a případně vloží akce pro produkci těchto jednotek do povolených akcí. Poté je nad seznamem naplánovaných akcí volána korekční metoda rozhodovacího modulu. Uživatel frameworku zde může dynamicky vytvářet pořadí akcí, např. pomocí strojového učení. Pokud toto chování není v rozhodovacím modulu implementováno, tak pořadí akcí zůstává tak jak je dáno produkční strategií, kterou Infrastructure manager dostal od Strategy managera a do seznamu je pouze přidána jednotka ze zásobníku žádostí.

Všechny vyrobené jednotky si Infrastructure manager ukládá do interních seznamů, které pak prochází v `InfractructureUnitPassingBehaviour` a rozesílá jednotky na příslušná místa. Vytváření jednotek a zasílání jednotek využívá komunikaci navrženou v kapitole 3.4 a jednotlivé zprávy jsou v tabulce 4.19.

Rozhraní rozhodovacího modulu

- `List<String> correctPlannedActions(List<String> plannedActions, InfrastructureManagerState state, BuildOrder buildOrder, List<String> availableActions, HashMap<String, HashMap<String, Integer» requestedUnits)`

  Vstupem této metody je aktuální seznam naplánovaných akcí, aktuální stav manažera, zvolená strategie stavění ze struktury `BuildOrders`, seznam akcí, které mohou být aktuálně provedeny a mapu ve které jsou uloženy žádosti o jednotky z ostatních agentů. Výstupem této metody může být libovolně upravený seznam akcí, ale uživatel by měl zajistit, aby do plánu byly přidávány i jednotky o které bylo požádáno.

- `Boolean shouldAddSupplyAction(ValueLevel freeSupply, InfrastructureManagerState lastState)`

  Vstupem této metody je úroveň volné populace a poslední známý stav manažera. Výstupem funkce by měl být *True*, pokud je volné populace málo a je třeba přidat do naplánovaných akcí stavbu budovy, která slouží jako generátor populace. Pokud je volné populace dostatek, nebo je přidávání generátorů obstaráváno metodou na korekci naplánovaných akcí, metoda by měla vracet *False*.

Tabulka 4.19: Zprávy odesílány z Infrastructure Managera

Perfomative	Content	Receiver
INFORM	<NodeName>-resourceRelease	Economic Manager
INFORM	workerReturn	Economic Manager
INFORM	strategyCompleted	strategy
REQUEST	<NodeName>-resources	Economic Manager
REQUEST	worker	Economic Manager
PROPOSE	worker	Recon Manager
PROPOSE	worker	Economic Manager
PROPOSE	fighter	Tactical Manager
CONFIRM	unitRequest-granted	Tactical Manager
CONFIRM	unitRequest-granted	Economic Manager
CONFIRM	unitRequest-granted	Recon Manager

4.4.9 Tactical Manager

Tactical manager je implementován jako manažer s jedním chováním. V tomto chování je pomocí rozhodovacího modulu kontrolováno aktuální složení armády a případně odesílány požadavky na nové jednotky. Tactical manager má taky možnost zaslat Infrastructure managerovi žádost o urychlení produkce bojových jednotek v případě krize. Infrastructure manager by měl pokud přijme takovou zprávu až do odvolání dávat prioritu všem žádostem o bojové jednotky. Přijímání jednotek probíhá stejně jako u všech ostatních manažerů v *MessageBehaviour* chování agenta.

Tactical manager v každém volání `onFrame()` vyhodnocuje stav svých podřízených jednotek `UnitAgent`. Všechny jednotky, které příjme jsou uloženy do interního seznamu a v této funkci jsou posílány do konkrétní útočné skupiny. K tomu se využívá funkcí rozhodovacího modulu, které určují kolik útočných skupin vytvořit a určení síly nepřátelské armády. K výběru cíle útoku využívá informace získané z rozhodovacího modulu, nebo cíle ze zprávy Strategy Managera.

Rozhraní rozhodovacího modulu

- `BasicKnowledge choosePrimaryTarget(List<BasicKnowledge> allEnemyAssets)`

  Vstupem této metody je seznam nepřátelských jednotek a budov. Výstupem by měl být prioritní cíl pro útok.

- `TacticalGoal selectCurrentGoal(KnowledgeBase knowledgeBase, List<UnitAgent> squad)`

  Vstupem metody je databáze znalostí agenta a seznam jednotek v aktuální útočné skupině. Výstupem je vhodně zvolený taktický cíl (útok, obrana, čekání na posily).

- `int getSquadPower(List<UnitAgent> squad)`

  Vstupem metody je seznam jednotek v útočné skupině a výstupem je síla armády vyjádřená celým číslem.

- `int getEnemyPower(List<BasicKnowledge> enemyAssets)`

  Vstupem metody je seznam nepřátelských budov a jednotek získaný z databáze znalostí agenta a výstupem je síla nepřítele vyjádřená celým číslem.

- `Pair<TechnologyTreeNode, Integer> shouldRequestNewUnits( List<BasicKnowledge> allEnemyAssets, List<UnitAgent> squad, KnowledgeBase knowledgeBase)`

  Tato metoda má na vstupu seznam nepřátelských budov a jednotek, seznam jednotek v aktuální útočné skupině a databázi znalostí agenta. Výstupem by měla být dvojice s jednotkou o kterou je třeba zažádat a číslem určující množství těchto jednotek.

Tabulka 4.20: Zprávy odesílány z Tactic Managera

Perfomative	Content	Receiver
ACCEPT_PROPOSAL	fighter	Infrastructure Manager
REQUEST	fighters-need	Infrastructure Manager
INFORM	armyInfo-<NodeName>;<NodeName>;...	Strategy Manager
INFORM	fighters-stop	Infrastructure Manager
INFORM	need-<NodeName>-<Amount>	Infrastructure Manager

4.4.10 Recon Manager

Recon manager je svou implementací velmi podobný Tactical managerovi. Taky je implementován jako manažer s jedním chováním, ve kterém zjišťuje jestli má žádat o průzkumné jednotky a jaký druh jednotek potřebuje. Mimo to také přeposílá získané znalosti ostatním manažerům podle svých rozhodovacích modulů.

Během každého zavolání metody `onFrame()` je zjištěno zda jsou nějaké cíle, které je třeba prozkoumat a jestli jsou k dispozici nějaké průzkumné jednotky. Pokud nejsou zadány cíle od Strategy Managera, Recon Manager hledá nepřátelské základny a pokud již všechny našel, tak tyto objekty pravidelně prozkoumává.

Rozhraní rozhodovacího modulu

- `ReconSearchMethod getSearchMethod()`

  Metoda zvolí metodu prohledávání mapy pro tuto hru. Navrácená hodnota je uložena v Recon managerovi.

- `boolean shouldSendKnowledgeToManager(BasicKnowledge knowledge, ManagerType manager)`

  Metoda má na vstupu znalost a typ agenta. Pokud je žádoucí aby agent tuto informaci dostal metoda vrátí *True*, jinak *False*.

- `boolean shouldRemoveKnowledgeFromManager(OwnerType owner, KnowledgeType type, UnitAgent object, ManagerType manager)`

  Na vstupu metody je vlastník znalosti, její typ a instance třídy `UnitAgent`. Metoda rozhodne zda znalost spojená s těmito parametry by měla být odstraněna z agenta označeného vstupním parametrem `ManagerType`.

- `boolean shouldRemoveKnowledgeFromManager(OwnerType owner, KnowledgeType type, Position p, ManagerType manager)`

  Stejná funkcionalita jako předchozí metoda, ale místo objektu rozhoduje odstranění z databáze agenta podle pozice znalosti.

- `boolean shouldRemoveKnowledgeFromManager(OwnerType owner, KnowledgeType type, String name, ManagerType manager)`

  Další variace metody pro odstranění znalosti z databáze agenta, ale tentokrát podle jména spojeného se znalostí.

Tabulka 4.21: Zprávy odesílány z Recon Managera

Perfomative	**Content**	**Receiver**
ACCEPT_PROPOSAL	worker	Infrastructure Manager
REQUEST	scouts-need	Infrastructure Manager
INFORM	need-<NodeName>-<Amount>	Infrastructure Manager
INFORM	targetsScouted	Strategy Manager

Kapitola 5

Testovací model

Tato kapitola je rozdělena na tři části. První část se věnuje volbě vhodného prostředí pro otestování validity frameworku. Druhá část je věnována popisu tvorby testovacího modelu pro zvolenou hru. V této části jsou popsány jednotlivé kroky, které bylo nutné učinit pro propojení agentního systému a konkrétní hry. Třetí část je zaměřena na rozbor poznatků získaných při tvorbě testovacího modelu. Je zde zhodnoceno jak obtížné bylo testovací model vytvořit, jak efektivně pracoval ve své minimální implementaci a jaký byl jeho přínos pro tvorbu systému schopného ovládat strategickou hru.

5.1 Volba testovacího prostředí

Jako testovací prostředí byla zvolena populární strategická hra Starcraft: Brood War a k jejímu ovládání byl použit framework BWAPI, konkrétně jeho nástavba JNIBWAPI a BWMirror, která slouží pro použití tohoto frameworku v jazyce Java.

5.1.1 Starcaft

Starcraft je strategická hra odehrávající se v reálném čase a žánrově se řadí mezí sci-fi válečné hry. Hra byla vyvinuta společností Blizzard Entertainment a vydána 31. března 1998. Hra obsahuje všechny prvky typické pro RTS, které byly popsány v kapitole 2.1 a to včetně jednotek se speciálními schopnostmi. Starcraft má také rozsáhlou historii se strojovým učením a umělou inteligencí. Mnoho studentů a programátorů testuje své dovednosti tvorbou tzv. botů naprogramovaných ke hraní právě Starcraftu. Mimo to jsou také pořádány turnaje, kde tito lidé staví své boty proti sobě. Takovým turnajem je třeba Student StarCraft AI Tournament and Ladder (SSCAI)[11], pořádaný studenty a výzkumníky z ČVUT. Z těchto důvodů byla hra Starcraft vybrána jako vhodný kandidát k otestování užitečnosti a použitelnosti vyvíjeného frameworku.

5.1.2 BWAPI

The Brood War Application Programming Interface (zkr. BWAPI) je framework pro jazyk C++ umožňující interakci s rozhraním hry Starcraft: Brood War, což je verze hry nejčastěji používaná v turnajích zmíněných v předchozí kapitole. Nad BWAPI existují ještě frameworky JNIBWAPI a BWMirror, které slouží jako Java wrapper, aby mohli boti pro Starcraft být vyvíjeni i v jazyce Java. Společně s BWAPI je třeba využít i program ChaosLauncher, který slouží jako most mezi běžícím agentem a spuštěnou hrou.

5.2 Tvorba modelu

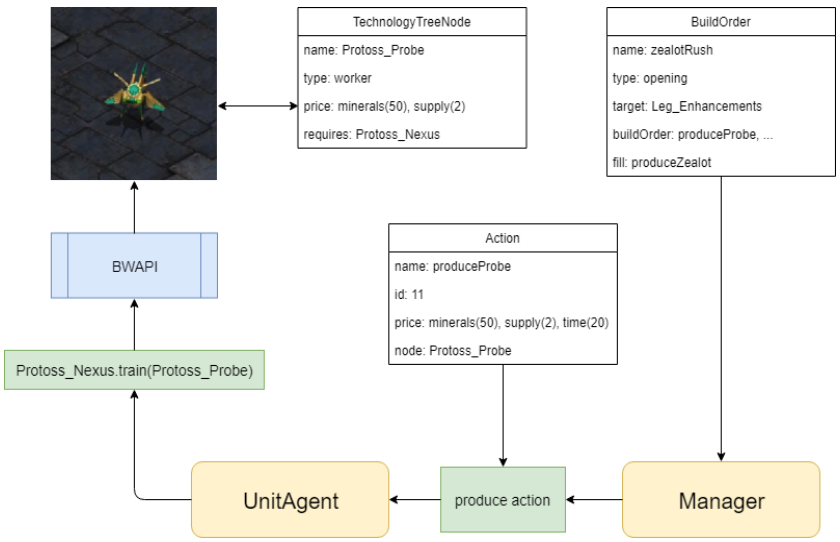
Vytvoření testovacího modelu se skládalo ze čtyř částí. V první řadě bylo třeba vytvořit konfigurační struktury pro framework, díky kterým je schopný agentní systém interpretovat informace ze hry v podobě, se kterou je schopný pracovat. Druhou částí bylo implementovat specializovanou třídu `UnitAgent` z vytvořeného frameworku. Tento UnitAgent pak slouží agentnímu systému jako jeho proxy pro komunikaci s BWAPI a předávání příkazů hře. Dále bylo třeba implementovat obecné výchozí rozhodovací moduly pro testování kompletní funkčnosti frameworku. Posledním krokem bylo samotné spuštění agentního systému a jeho propojení se spuštěnou hrou.

5.2.1 Konfigurační struktury

Prvním krokem k vytvoření agentního systému ovládájícího zvolenou hru bylo vytvořit konfigurační struktury, které slouží jako reprezentace herních elementů a akcí které s nimi jdou provádět v podobě, které framework rozumí a je s ní schopný pracovat. Aby mohl agentní systém hrát úplnou hru je třeba vytvořit technologický strom (Technology Tree) obsahující všechny jednotky, budovy a vylepšení, které může hráč vyprodukovat. K tomuto stromu je potřeba vytvořit list akcí (Action List), který popisuje akce co můžeme ve hře udělat, abychom vyprodukovali uzly stromu. Dále je třeba definovat herní strategie (Build Orders), které určují výchozí pořadí stavby budov pro Infrastructure managera.

Data pro technologický strom byla získána ze Starcraft wiki[10], což je fanoušky vytvořená databáze obsahující informace o všech elementech hry. Při tvorbě technologického stromu bylo třeba identifikovat dvě speciální entity. Těmi jsou dělník a generátor populace. Testovací model byl vytvářen, tak aby hrál za rasu Protoss, takže jako dělník byly označena jednotka Probe a jako generátor populace budova Pylon.

Data pro herní strategie byla získána z Liquipedie[8]. Opět se jedná o fanoušky tvořenou databázi, ale více zaměřenou na kompetitivní stránku hry. Obsahuje data z oficiálních turnajů hry a strategie profesionálních hráčů. Bylo vybráno několik klasických počátečních a pokročilých strategií pro hraní rasy Protossů a ty byly přepsány do podoby pro konfigurační strukturu `BuildOrders`.

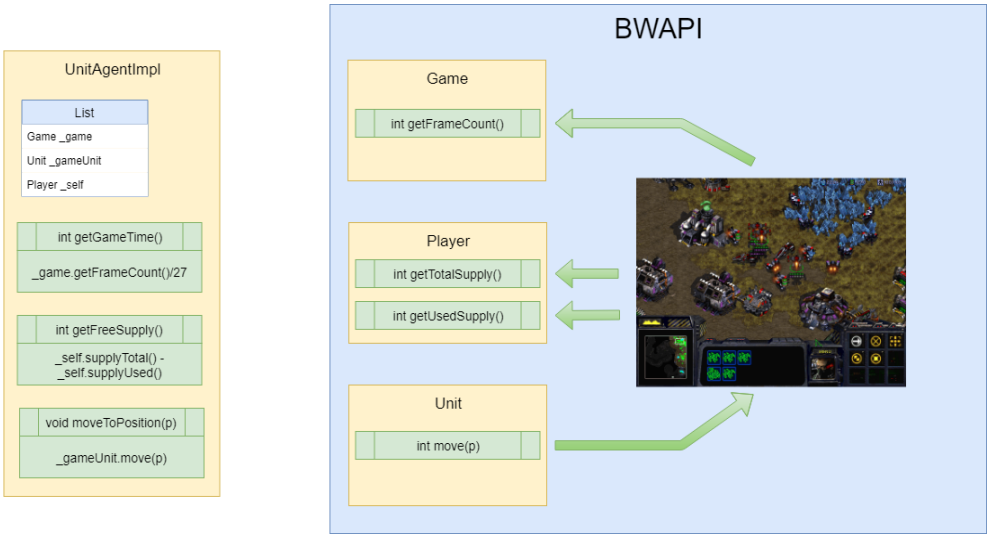


Obrázek 5.1: Schéma propojení pomocí konfiguračních struktur

5.2.2 Implementace jednotkového agenta

Další prerekvizita k propojení agentního systému se hrou je implementace jednotkového agenta. Třída `UnitAgent` definuje seznam abstraktních metod, které agenti frameworku volají pro vykonání určitých akcí ve hře. Byla tedy vytvořena třída `UnitAgentImpl`, která rozšiřuje třídu `UnitAgent`. Této nově vzniklé třídě jsou předány herní objekty z frameworku BWAPI. Jedná se o instance tříd `Game`, `Player` a v případě, že se jedná o instanci `UnitAgentImpl`, která představuje budovu, či jednotku, je jí předána i instance třídy `Unit`, která slouží k ovládání konkrétní jednotky v aktuální hře.

Objekty `Player` a `Game` umožňují získat ze spuštěné hry všechny informace, které agenti potřebují. Poskytují metody pro získání například herního času, množství dostupných surovin a zjištění pozice nebo vzdálenosti herních objektů. Objekt `Unit` zase umožňuje pohybovat s jednotkou, stavět budovy, či produkovat nové jednotky, protože BWAPI používá objekt `Unit` i k reprezentaci budov.



Obrázek 5.2: Schéma propojení třídy UnitAgent a BWAPI

Během implementace třídy `UnitAgent` byl z části použit i framework Atlantis[2], což je Java framework sloužící jako obal nad BWAPI, který usnadňuje vytváření nových botů pro ovládání hry Starcraft. Tento framework je použit k implementaci metod UnitAgenta pro útok na nepřátelskou pozici a obstárává mikromanagment jednotky.

5.2.3 Implementace rozhodovacích modulů

Poslední věcí kterou je třeba implementovat, aby byl agentní systém schopný hrát zvolenou hru efektivně jsou rozhodovací moduly jednotlivých manažerů. Implementace rozhodovacích modulů není nutná, protože framework obsahuje sadu obecných výchozích rozhodovacích modulů, i přesto je však doporučeno, aby uživatel vytvořil vlastní implementaci. Tyto moduly nejsou řízené strojovým učením a jejich účel je zajistit, aby framework všechny činnosti plnil alespoň s minimální efektivitou a jsou určeny především pro testování frameworku.

Chování jednotlivých výchozích modulů je následující:

- `DefaultStrategyDecisionModule`

  Modul vybírá náhodně strategie hry a strategie stavění ze struktur StrategyList a BuildOrders. Je tomu tak protože bez znalosti konkrétní hry, či strojového učení je velmi obtížné navrhnout obstojný systém pro výběr vhodných strategií.

- `DefaultEconomicDecisionModule`

  Modul má stanovený počet dělníků, které požaduje a pravidelně o ně žádá dokud je nezíská. Expanze pak zase povoluje, až po uplynutí definovaného času a splnění daných prerekvizit. Těmi jsou nestavění hlavní budovy pokud není postavený zdroj surovin a naopak nevytváření nového zdroje surovin, pokud není postavená nová hlavní budova.

- `DefaultInfrastructureManager`

  Ve výchozím rozhodovacím modulu není plán akcí nijak upravován, takže Infrastructure manager staví budovy v pořadí které má definované v přiřazené strategii stavění. Aby se ve strategii nemuseli definovat generátory populace, je nastaveno, aby rozhodovací modul vrátil *True* při dotazu, zda postavit generátor populace, pokud je volná populace na úrovni `LOW`, nebo nižší.

- `DefaultTacticalManager`

  Výpočet nepřátelské síly probíhá sečtením atributu *power*, který je uložen u nepřátelských jednotek v databázi znalostí. Síla útočné jednotky se vypočítá sečtení hodnoty, kterou vrací metoda `getPower()` třídy `UnitAgent`. O nové jednotky je požádáno vždy, když je síla nepřítele výrazně vyšší než síla hráče. Typ jednotky o kterou je požádáno a primární cíl útoku jsou voleny náhodně, protože stejně jako u strategií, bez znalosti konkrétní hry je obtížné toto rozhodnutí vhodně učinit.

- `DefaultReconManager`

  Správně by mělo být v implementaci rozhodovacího modulu zařízeno, aby manažerům chodili znalosti jen o relevantních informacích. Co však je relevantní informace pro kterého manažera nelze jednoznačně určit bez znalosti konkrétní hry. Nelze například rozhodnout o kterých budovách by měl mít Tactical manager znalosti. Tento agent potřebuje jen informace o takticky významných budovách, ale jaká budova do této skupiny patří se může lišit nejen mezi rozdílnými hrami, ale i podle aktuálního protivníka. Z tohoto důvodu a pro zjednodušení testování, výchozí rozhodovací modul Recon managera přeposílá všechny informace do všech agentů.

5.2.4 Spuštění agentů a propojení se hrou

BWAPI definuje třídu `DefaultBWListener`. Instance této třídy umožňuje odchytávat herní události jako je start hry, vytvoření jednotky a vykreslení snímku. Rozšířením této třídy získáme prostředí ve kterém mužeme vytvořit, inicializovat a řídit agentní systém. Vytvoření a ovládání agentního systému vytvořeného frameworkem je rozprostřeno ve funkcích, které jsou volány BWAPI, když je zachycena příslušná událost. Těmito funkcemi jsou:

- `onStart()`

  V reakci na tuto událost je vytvořena instance tříd `FrameworkController`, `DecisionModuleFactory` a `UnitAgentImpl` (asistent). Poté jsou pomocí FrameworkControlleru vytvořeny instance konfiguračních struktur a manažerů. Následně je nad FrameworkControllerem zavolána funkce `StartMAS()`, které zajistí spuštění JADE a vložení manažerů do inicializovaného agentního systému.

- `onUnitDiscover()`

  Přijmutí této události signalizuje, že v oblasti kterou hráč vidí se objevila nová jednotka, či budova. Je tedy vytvořena nová znalost představující objevenou entitu a je odeslána zpráva o této nové znalosti příslušnému manažerovi. Ten zprávu přijme, zpracuje a uloží znalost do své databáze.

- `onUnitDestroy()`

  Tato událost signalizuje zničení herní entity. Může se jednat o jednotku, nebo budovu libovolného hráče, který je zrovna v oblasti na kterou hráč vidí. Stejně jako při objevení znalosti je odeslána zpráva o zničení entity všem manažerům, kteří by tuto znalost mohli mít. Ti zprávu opět zpracují a případně smažou informace o zaniklé entitě ve své databázi znalostí.

- `onUnitCreate()`

  Událost oznamuje vznik nové jednotky ve hře. Je zasílána v okamžiku kdy započala produkce herní entity a byly použity zdroje, které k její výrobě byly potřeba. V reakci na tuto událost je nad Infrastructure managerem volána metoda `actionStarted()`, která uvolní zarezervované zdroje, vrátí zapůjčené dělníky a zahájí produkci nové akce.

- `onFrame()`

  Metoda volaná po každém vykreslení herního snímku. V této metodě můžou být volány metody nad objekty BWAPI, kterou jsou pak provedeny před vykreslením nového herního snímku. Při každém zavolání této metody jsou zavolána `onFrame()` metody jednotlivých manažerů, kteří tak vykonají své akce ve hře prostřednictvím přiřazených jednotkových agentů.

- `onEnd()`

  Jedná se o událost oznamující konec hry. V reakci na tuto událost jsou na manažery volány metody `writeStates()`, které zapisují stavy, jež manažeři v průběhu hry nabyly do příslušného JSON souboru. Také je zde uložena do svého souboru struktura `ManagersHistory`.

5.3 Zhodnocení testovacího modelu

Testovací model je hodnocen na základě tří kritérií. Za prvé je rozebráno jak obtížné byly vykonat jednotlivé kroky implementace testovacího modelu. Dále je popsáno jak efektivní byl agentní systém ve hraní hry obecně a jakým způsobem jednotlivý agenti plnili své povinnosti. Na závěr je vyhodnoceno co framework tvorbě systémů pro ovládání strategických her přináší a co v něm naopak chybí.

5.3.1 Základní propojení se hrou

Základní propojení vytvořeného agentního systému s konkrétní hrou není nikterak složité, ale může být relativně pracné v závislosti na zvolené hře a jejím API. Je potřeba zajistit, aby agentní systém věděl co může ve hře dělat a byl schopný zpracovat informace co mu ze hry budou chodit. K tomu slouží technologický strom, který mapuje herní entity do podoby, kterou framework dokáže zpracovat. K tomuto stromu je třeba vytvořit pole akcí, které uzly technologického stromu produkují, nebo s nimi pracují. Vytvoření těchto struktur by nemělo být komplikované, jedná se převážně o vytváření objektů v JSON formátu.

Poté je třeba zajistit, aby agentnímu systému chodili nové informace. To znamená, že je potřeba pomoci herního API získávat informace o probíhající hře a ty předávat Recon managerovi, který je rozesílá ostatním agentům v systému. Tato část může být složitá, protože uživatel musí zajistit, aby agentnímu systému nepředával špatné informace. Musí nalezené znalosti správně kategorizovat a vytvořit objekt reprezentující znalost. Je na uživateli, aby parametry objektu znalosti, nebo způsobem jakým bude znalost zasílat, zajistil že se do databáze znalostí nebudou přidávat duplicitní informace.

Posledním krokem pro propojení se hrou je vytvoření implementace třídy `UnitAgent`, přes kterou bude agentní systém vykonávat své akce ve hře. Obtížnost vytvoření této třídy je závislá pouze na zkušenostech uživatele s herním API. Agentní systém svou činností zajišťuje, že všechny prerekvizity pro vykonání herní akce jsou splněny, takže zavolaná metoda třídy `UnitAgent` by měla jen zajistit, že bude akce opravdu vykonána. Výjimkou jsou jen akce pro útok a průzkum nepřátelských oblastí. Tyto akce budou ve většině her vyžadovat nějaký mikromanagment, který framework není schopný obecně vykonávat.

5.3.2 Efektivita vytvořeného systému

Agentní systém vytvořený frameworkem je po propojení se hrou, schopný plně hrát proti zabudovaným počítačem řízeným protivníkům s použitím výchozích rozhodovacích modulů. Agentní systém dokázal tyto oponenty porazit i bez implementace strojového učení, jen s použitím definovaných strategií stavění a minimální implementací třídy `UnitAgent`. Testovací model nedokázal porazit zkušenější hráče, nebo pokročilé agenty zaměřené na hraní hry Starcraft. Tento výsledek byl však očekávaný, protože bez komplexnější implementace rozhodovacích modulů a jednotkového agenta, které by optimalizovali strategie stavby budov, těžení surovin a ovládání bojových jednotek je agentní systém příliš statický a předvídatelný.

Zhodnocení jednotlivých manažerů

- Strategy manager

  Většina funkcí tohoto manažera spočívá ve správném volání metod rozhodovacího modulu a následné zaslání zprávy příslušnému manažerovi. Vzhledem k tomu, že většina tohoto rozhodování je ve výchozích modulech implementováno jen jako zástupná funkce, tak nelze příliš hodnotit efektivitu tohoto agenta. Jediné co může být potvrzeno je, že Strategy manager správně volá metody rozhodovacího modulu pro rozpoznání nepřátelské strategie a korektně zpracovává příchozí zprávy, na které zasílá správnou odpověď.

- Economic manager

  Zajišťuje konstantní přísun surovin a samostatně si skrze Infrastructure managera obstarává dělníky. Mimo to také korektně hlídá aktuální úrovně surovin a nedovoluje, aby Infrastructure manager blokoval zdroje, které nejsou k dispozici. Economic manager je také schopný zařídit, aby byly postaveny nové hlavní budovy a zdroje surovin, jen když to aktuální finanční situace vyžaduje.

- Infrastructure manager

  Korektně obstárává makromanagment produkce budov a jednotek. Automaticky si domluvou s Economic managerem zajistí dělníka a suroviny na provedení akce. Při provádění akcí také správně zajišťuje, aby plně využíval dostupné prostředky. Příkladem toho je například přiřazovaní akcí na produkci jednotek jen továrnám, které jsou aktuálně volné. Správně také zpracovává žádosti o nové jednotky a ve chvíli kdy jsou vytvořené je předává příslušným agentům. Agent neblokuje zbytečně suroviny, ani jednotky a svůj plán plní s maximální možnou rychlostí.

- Tactical manager

  Agent vyhodnocuje sílu herních armád a podle poměru sil pravidelně přepíná mezi útočným a obranným módem. Tactical manager je schopný své vojáky stahovat z boje když se dostane do nevýhody a také si dokáže zažádat o nové vojáky. Všechny své jednotky shlukuje do skupin a zajišťuje tak, aby na nepřítele vždy útočil s maximální možnou silou.

- Recon manager

  Manažer si žádá o průzkumné jednotky a samostatně s nimi prochází mapu. Tím získává znalosti o herním stavu a tyto informace přeposílá ostatním agentům v systému. Agent je schopný samostatně vyhodnotit kterému agentovi by měl poslat kterou informaci.

5.3.3 Přínos a omezení frameworku

Hlavním přínosem frameworku je zapouzdření makromanagmentu strategické hry do obecného agentního systému. Pomocí frameworku lze vytvořit agentní systém ovládající libovolnou strategickou hru hranou v reálném čase, pokud k této hře existuje API přes které je možné hru řídit a získávat z ní informace. Framework byl sice optimalizován pro válečné strategické hry typu Starcraft, ale je použitelný i pro ostatní druhy strategických her ve kterých je třeba řešit těžbu surovin, stavbu budov, produkci jednotek, nebo útok na nepřítele.

Framework byl navíc navržen tak, aby podporoval strojové učení. Toho je docíleno systémem rozhodovacích modulů a průběžným ukládáním stavů hry. Agentní systém tak vytváří data, které mohou být použita pro strojové učení a zároveň poskytuje mechanismus jak pomocí výsledků tohoto učení ovlivňovat rozhodnutí agentního systému při řízení hry. Rozhodovací moduly byly navrženy tak, aby byly jednoduché na implementaci a použití. Podpora frameworku pro strojové učení byla ověřena Bc. Petrem Knapkem, který tento frameworku použil ve své diplomové práci zabývající se učením agentů ve strategických hrách.

Vytvořený agentní systém navíc používá pro implementaci agentů framework JADE, kde agenti mezi sebou komunikují pomocí FIPA ACL zpráv. Díky tomu je možné celý agentní systém jednoduše rozšířit. Jednotlivým agentům jde přidat nové chování, lze vytvořit další komunikace mezi agenty, nebo do systému mohou být přidáni úplně nový agenti. Tímto způsobem mohou být například rozšířeny rozhodovací moduly, nebo provedeny úpravy agentního systému kvůli ovládání hry pro kterou není optimalizován.

Hlavním omezením vyvinutého frameworku je absence vestavěného mikromanagmentu. První iterace návrhu obsahovali mikromanagment pro několik agentů, ale během prvních testů byl tento přístup z několika důvodů zavržen. Hlavním problémem bylo vyvážit obecnost frameworku a složitost propojení s konkrétní hrou. V případě obecného řešení vznikalo velké množství funkcí, které musel uživatel implementovat a úměrně s tím se zvedala složitost implementace a použití jednotlivých agentů. V případě zjednodušeného propojení se hrou zase nešlo vytvořit v agentech mikromanagment, bez toho aniž by věděli jakou hru hrají a měli přístup k jejímu API. Aby byla zachována obecnost frameworku a požadavek na jednoduchost použití, tak bylo rozhodnuto delegovat mikromanagment do funkcí jednotkového agenta, kterého implementuje uživatel pro konkrétní hru.

Bez důkladnějších testů není možné určit zda pomocí frameworku lze vytvořit agentní systém, který by bylo možné použít pro ovládání i žánrově jiných strategických her. Příkladem jsou třeba tahové, nebo budovatelské strategie. Oba žánry aspoň z části řeší podobné problémy jako válečné strategie hrané v reálném čase (RTS), ale jak je k těmto problémům přistupováno se od RTS velmi liší. Je pravděpodobné, že by šlo pomocí frameworku vytvořit agentní systém který by tyto hry ovládal, ale téměř jistě by to vyžadovalo od uživatele rozšíření a úpravy jednotlivých manažerů.

Kapitola 6

Závěr

V této práci byl navržen a implementován framework pro tvorbu agentních systémů, schopných komunikovat a ovládat libovolnou válečnou strategickou hru. Cílem práce bylo identifikovat základní prvky a mechaniky strategických her a navrhnout systém který bude schopen číst informace z běžné strategické hry a provádět nutné úkony pro její hraní. K této činnosti byl vytvořen framework, který se skládá z několika agentů, jež mohou být nakonfigurováni pro ovládání zvolené strategické hry. Jednotlivý agenti jsou také navrženi tak, aby podporovali strojové učení a byly snadno rozšiřitelní o nové funkce. Framework byl otestován vytvořením jednoduchého agentního systému, který byl schopný číst a ovládat strategickou hru Starcraft: Brood War. Podpora strojového učení byla ověřena v diplomové práci Bc. Petra Knapka, který vytvořený framework použil k implementaci systému, využívajícímu metody strojového učení pro rozhodování manažerů ve hře Starcraft: Brood War.

Framework obsahuje agenty zapouzdřující obecný makromanagment ve strategických hrách. Jednotlivý agenti jsou schopní zajišťovat těžbu surovin, stavění budov, shromažďování vojáků a stanovování herních cílů. Použitím frameworku tak uživatel musí implementačně řešit jen ovládání hry na nejnižší úrovni. Navíc díky mechanismu rozhodovacích modulů, které agenti používají k řízení oblasti za kterou jsou zodpovědní, může uživatel i ovlivňovat makromanagment hry, aniž by musel činit změny v kódu jednotlivých agentů. Další výhodou je použití frameworku JADE k vytvoření agentního systému. Agenti zde komunikují pomocí FIPA ACL zpráv, takže systém může spolupracovat i s jinými agenty využívajícími tuto standardizovanou formu komunikace.

Bylo ověřeno že agentní systém vytvořený frameworkem lze použít k ovládání válečné strategické hry. V rámci budoucí práce by bylo zajímavé ověřit zda lze vytvořit agentní systém, který bude schopen ovládat i budovatelské, či tahové strategické hry. Následně podle těchto poznatků upravit framework, aby toho buď byl schopen, nebo řídil tyto hry efektivněji. Dále by určitě bylo možné rozšířit rozhodovací moduly jednotlivých agentů a dát tak uživateli větší kontrolu nad činností agentního systému jen pomocí implementace těchto tříd.

Literatura

[1] *Game AI - Funtelligence - Extra Credits.* [Online; navštíveno 30.06.2016].
URL https://www.youtube.com/watch?v=1FBGR6vmNeU

[2] *Atlantis.*
URL https://github.com/Ravaelles/Atlantis

[3] FIPA ACL Message Structure Specification. [Online; navštíveno 15.03.2018].
URL http://www.fipa.org/specs/fipa00061/SC00061G.pdf

[4] Harvester Agent for pysc2.
URL https://github.com/0ad4ai/SC2_HarvesterAgent

[5] JADE Programming Tutorial for Begginers.
URL http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf

[6] JAVA Agent Development Framework. [Online; navštíveno 15.03.2018].
URL http://jade.tilab.com/

[7] JSON. [Online; navštíveno 15.03.2018].
URL https://www.json.org/

[8] Liquipedia. [Online; navštíveno 15.03.2018].
URL https://liquipedia.net/starcraft

[9] ResourceGather.
URL https://github.com/0ad4ai/ResourceGather

[10] Starcraft wiki. [Online; navštíveno 15.03.2018].
URL http://starcraft.wikia.com/wiki/StarCraft_Wiki

[11] Student StarCraft AI Tournament and Ladder.
URL https://sscaitournament.com/

[12] The Foundation for Intelligent Physical Agents. [Online; navštíveno 15.03.2018].
URL http://www.fipa.org/repository/aclspecs.html

[13] Cabanag, M. A. V.; Hitchens, M.; Richards, D.: A novel agent based control scheme for RTS games. [Online; navštíveno 10.01.2018].
URL https://www.researchgate.net/publication/241770342_A_novel_agent_based_control_scheme_for_RTS_gamess

[14] *Dallatana, A.: BDI agents for Real Time Strategy games. [Online; navštíveno 10.01.2016].*
URL http://amslaurea.unibo.it/4217/1/dallatana_andrea_tesi.pdf

[15] *Daylamani-Zad, D.: Swarm intelligence for autonomous cooperative agents in battles for real-time strategy games. [Online; navštíveno 10.01.2018].*
URL https://www.researchgate.net/publication/320254420_Swarm_intelligence_for_autonomous_cooperative_agents_in_battles_for_re
time_strategy_games

[16] *Maissiat, J.; Meneguzzi, F.: Adaptive High-Level Strategy Learning in StarCraft. [Online; navštíveno 6.03.2018].*
URL
http://www.sbgames.org/sbgames2013/proceedings/comp/03-full-paper.pdf

[17] *McCorduck, P.: Machines Who Think. Natick, MA: A K Peters, Ltd., 2004, ISBN 01-56881-205-1.*

[18] *MCCoy, J.; Mateas, M.: An Integrated Agent for Playing Real-Time Strategy Games. [Online; navštíveno 10.01.2018].*
URL https://www.aaai.org/Papers/AAAI/2008/AAAI08-208.pdf

[19] *Pingen, G. L. J.: An implementation of a data-mining approach to strategy selection in Starcraft: Brood War.*
URL https://theses.ubn.ru.nl/bitstream/handle/123456789/144/Pingen%2C_G.L.J._1.pdf?sequence=1

[20] *Salamon, T.: Design of Agent-Based Models. Bruckner Publishing, 2011, ISBN 978-80-904661-1-1.*

[21] *Synnaeve, G.; Bessiere, P.: A Dataset for StarCraft AI and an Example of Armies Clustering. [Online; navštíveno 6.03.2018].*
URL https://arxiv.org/pdf/1211.4552.pdf

[22] *Zbořil, F.: Podklady k předmětu AGS - VI. Multiagentní systémy. [Online; navštíveno 10.01.2018].*
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FAGS-IT%2Flectures%2FAGS16_07.pdf&cid=10764

Příloha A

Instalace a spuštění

K přeložení a spuštění testovacího modulu ve složce "testModel" přiloženém na CD je potřeba:

- Načíst přiložený projekt v Eclipse nebo IntelliJ Idea pro vývoj Javy
- Mít vývojovou platformu JDK alespoň verze 8
- Mít nainstalované BWAPI
- Mít kopii StarCraft: Brood War verze 1.16.1 s podporou Chaoslauncheru

  Návod k instalaci BWAPI a instrukce ke spuštění bota jsou nejlépe popsány na: https://sscaitournament.com/

Příloha B

Dokumentace AFSG

Agent-Framework-for-Strategy-Games (AFSG) je framework pro vytvoření agentního systému sloužícího k ovládání primárně válečných strategických her. Systém je složený z pěti agentů:

- Strategy manager - Rozpoznání strategie nepřítele, výběr vlastních strategií hry a stavby.
- Economic manager - Těžba surovin a jejich rozdělování.
- Infrastructure manager - Stavba budov, vylepšení a produkce jednotek.
- Tactical manager - Kontrola entit určených k boji a útok na nepřítele.
- Recon manager - Průzkum mapy a předávání znalostí ostatním manažerům.

Pro vytvoření a spuštění agentního systému pro kontrolu hry, je potřeba učinit čtyři kroky, které budou podrobně vysvětleny v následujících kapitolách. Tato dokumentace obsahuje popis jen rozhraní a tříd, které musí uživatel nutně implementovat. Kompletní dokumentace API je přiložena ke zdrojovému kódu frameworku a JAR knihovny, které může uživatel použít pro importování frameworku do svého projektu. Čtyři kroky, co musí uživatel učinit a které zde budou popsány jsou následující:

- Vytvoření konfiguračních struktur `TechnologyTree`, `ActionList`, `BuildOrders` a `StrategyList`.
- Implementace abstraktní třídy `UnitAgent`.
- Implementace rozhraní *DecisionModule* pro jednotlivé manažery. (Tento krok je volitelný, protože framework obsahuje základní rozhodovací moduly pro všechny manažery, ale je doporučeno implementovat si vlastní.)
- Vytvoření instancí manažerů, spuštění agentního systému a zajištění předávání informací ze hry agentnímu systému.

B.1 Vytvoření konfiguračních struktur

Konfigurační struktury slouží agentnímu systému k tomu, aby bylo možné číst informace ze hry v podobě, kterou je schopný framework interpretovat a pracovat s ní.

TechnologyTree

Do tohoto souboru reprezentujícího technologický strom se vkládá popis herních entit, které bude systém schopen vytvářet. Je zde třeba vložit popis všech objektů, které uživatel zamýšlí používat v akcích a *BuildOrderech*. Jednotlivé uzly Technologického stromu jsou načteny do třídy `TechnologyTree`, přes kterou je možné k nim přistupovat. V tomto souboru se musí nacházet JSON pole, které bude obsahovat objekty typu `TechnologyTreeNode`, který je uveden v následující tabulce.

Tabulka B.1: Parametry `TechnologyTreeNode`

Jméno	**Typ**	**Hodnota**
name	<NodeName>	Jméno TT uzlu
type	<Type>	worker|unit|building|upgrade|supply
width	<Number>	Šířka jednotky/budovy
height	<Number>	Výška jednotky/budovy
price	Array<Price>	Cena vytvoření uzlu
requires	Array<NodeName>	Seznam vyžadovaných uzlů
unlocks	Array<NodeName>	Seznam odemčených uzlů

Tabulka B.2: Parametry Price

Jméno	**Typ**	**Hodnota**
name	<ResourceName>	Jméno zdroje
value	<Number>	Počet jednotek zdroje

Následuje příklad JSON zápisu pro Technology Tree obsahující jednotku Probe rasy Protossů ze hry StarCraft: Brood War.

```
[
  {
    "name": "Protoss_Probe",
    "type": "worker",
    "width": "23",
    "height": "23",
    "price": [
      {
        "name": "minerals",
        "value": "50"
      },
      {
        "name": "supply",
        "value": "1"
      }
    ],
    "requires": [
        "Protoss_Nexus"
    ],
    "unlocks": []
  }
]
```

B.1.1 ActionList

Do tohoto souboru je třeba vložit všechny akce, kterých budou manažeři schopni. Následně je třeba v implementaci UnitAgenta provést mapování těchto funkcí na příkazy konkrétní hry. Ve výchozím nastavení frameworku je nutné v tomto souboru uvést minimálně akce pro stavění budov, produkci jednotek a výzkum vylepšení. S těmito akcemi pak pracuje Infrastructure manager nastane bez nich nekorektní chování. K seznamu akcí lze přistupovat přes třídu `ActionList`, ve které jsou tyto akce uloženy. Opět se jedná o JSON pole, které obsahuje objekty typu `Action`, jejichž definice je v následující tabulce.

Tabulka B.3: Parametry `Action`

Jméno	**Typ**	**Hodnota**
name	<ActionName>	Jméno akce
id	<number>	Unitkátní identifikátor akce
price	Array<Price>	Kompletní cena akce
node	<NodeName>	Jméno vyprodukovaného uzlu

Následuje příklad JSON zápisu pro Action List, s akcí na výrobu Proby, z předchozího příkladu.

```
[
  {
    "name": "produceProbe",
    "id": 11,
    "price": [
      {
        "name": "minerals",
        "value": "50"
      },
      {
        "name": "time",
        "value": "20"
      },
      {
        "name": "supply",
        "value": "1"
      }
    ],
    "node": "Protoss_Probe"
  }
]
```

B.1.2 BuildOrders

Do tohoto konfiguračního souboru je třeba vložit všechny *BuildOrdery* které může framework použít. Ty jsou používány Infrastructure managerem k prvotnímu naplánování svých akcí. Strategy manager se může učit tyto nadefinované *BuildOrdery* používat a vybírá ty s největší úspěšností proti současnému protivníkovi. Je doporučeno tedy nadefinovat co nejvíce možných *BuildOrderů*, pro různé části hry. *BuildOrdery* jsou uloženy v souboru ve formátu JSON, jako pole objektů typu `BuildOrder`, jehož popis je v následující tabulce.

Tabulka B.4: Parametry Strategy

Jméno	**Typ**	**Hodnota**
name	<BuildOrderName>	Jméno strategie stavění
type	<BuildOrderType>	opening|standard|lateGame
target	<NodeName>	Jméno uzlu jež je cílem této strategie
buildOrder	Array<ActionName>	Seznam akcí jež je nutno v této strategii vykonat
fill	Array<ActionName>	Seznam akcí, které jsou použity jako výplň plánu

Následuje příklad JSON zápisu pro BuildOrders, obsahující počáteční *BuildOrder* pro strategii "Zealot Rush".

```
[
  {
    "name": "zealotRush",
    "type": "opening",
    "target": "Protoss_Assimilator",
    "buildOrder": [
      "buildPylon",
      "buildGateway",
      "buildGateway",
      "buildAssimilator"
    ],
    "fill": [
      "produceZealot"
    ]
  }
]
```

B.1.3 StrategyList

Konfigurace obsahuje popis herních strategií, které mohou být ve hře použity ať už hráčem, či nepřítelem. Strategy List je pole objektů, kde každý objekt obsahuje jméno herní rasy a pole objektů *Strategy*. Parametry tohoto objektu jsou uvedeny v tabulce 4.6. Strategy manager může tuto strukturu používat k rozpoznání nepřátelské strategie a zjištění jakou strategii zvolit proti němu. Toto chování je však třeba implementovat, protože výchozí rozhodovací moduly toto chování nepodporují. Opět se jedná o JSON strukturu, která je načtena do třídy `StrategyList`, kde mohou být jednolivé `Strategy` objekty zpřístupněné identifikátorem rasy a číselným identifikátorem strategie.

Tabulka B.5: Parametry Strategy List objektu

Jméno	**Typ**	**Hodnota**
name	<RaceName>	Jméno hratelné, nebo nepřátelské rasy
strategies	Array<Strategy>	Unikátní identifikátor

Tabulka B.6: Parametry Strategy

Jméno	**Typ**	**Hodnota**
name	<StrategyName>	Jméno strategie
id	<number>	Unikátní identifikátor
nodes	Array<{name, count}>	Seznam jednotek a budov
upgrades	Array<{name, level}>	Seznam vylepšení a jejich úrovně

Následuje příklad zápisu StrategyListu obsahující strategii "3 Gateway"pro rasu Protoss.

```
[
   {
      "name": "Protoss",
      "strategies": [
         {
            "name": "3_gateway",
            "id": 0,
            "nodes": [
               {
                  "name": "Protoss_Nexus",
                  "count": 1
               },
               {
                  "name": "Protoss_Pylon",
                  "count": 2
               },
               {
                  "name": "Protoss_Gateway",
                  "count": 3
               },
               {
                  "name": "Protoss_Cybernetics_Core",
                  "count": 1
               },
               {
                  "name": "Protoss_Citadel_of_Adun",
                  "count": 1
               },
               {
                  "name": "Protoss_Assimilator",
                  "count": 1
               },
               {
                  "name": "Protoss_Zealot",
                  "count": 11
               }
            ],
            "upgrades": [
               {
                  "name": "Leg_Enhancements",
                  "level": "1"
               }
            ]
         }
      ]
   }
]
```

B.2 Implementace třídy UnitAgent

Instance třídy unitAgent ve frameworku zastupují všechny ovladatelné prvky ve hře. Slouží jako obal nad všemi jednotkami a budovami, které může hráč ovládat. Tato třída obsahuje sadu metod pomocí nichž agenti činí akce ve hře, nebo z ní získávají informace, které nelze efektivně získávat z databáze znalostí. Následuje seznam těchto metod, s popisem vstupních parametrů a předpokládaným výstupním parametrem. Ve frameworku se předpokládá existence speciálního UnitAgenta, který se předává do konstruktoru jednotlivým manažerům a skrz tento objekt manažeři získávají informace, které nejsou vázané na konkrétní herní entitu. Nad tímto objektem jsou volány pouze metody označeny jako "assistant method".

- (assistant method) `boolean isUnitBaseStructure(String unitType)`

  Do metody vstupuje buď hodnota z parametru *name* objektu `TechnologyTreeNode`, nebo hodnota parametru *unitType* z objektu `BasicKnowledge`. Návratová hodnota je *True*, pokud se jedná o budovu, která plní roli hlavní herní základny.

- (assistant method) `boolean isUnitResourceGenerator(String unitType)`

  Do metody vstupuje buď hodnota z parametru *name* objektu `TechnologyTreeNode`, nebo hodnota parametru *unitType* z objektu `BasicKnowledge`. Návratová hodnota je *True*, pokud se jedná o budovu, která genereruje nové suroviny.

- (assistant method) `boolean isUnitSupplyGenerator(String unitType)`

  Do metody vstupuje buď hodnota z parametru *name* objektu `TechnologyTreeNode`, nebo hodnota parametru *unitType* z objektu `BasicKnowledge`. Návratová hodnota je *True*, pokud se jedná o budovu, která vytváří budovy umožňující mít více jednotek, tzn. zvětšují populaci.

- (assistant method) `boolean isUnitFacility(String unitType)`

  Do metody vstupuje buď hodnota z parametru *name* objektu `TechnologyTreeNode`, nebo hodnota parametru *unitType* z objektu `BasicKnowledge`. Návratová hodnota je *True*, pokud se jedná o budovu, která produkuje nové bojové jednotky.

- (assistant method) `ValueLevel getSupplyLevel()`

  Metoda vrací aktuální hodnotu `ValueLevel` reprezentující současnout úroveň velikosti populace.

- (assistant method) `void runAction(Action action, UnitAgent currentWorker, UnitAgent currentFacility)`

  Metoda provede zvolenou herní akci. Na vstupu má parametr *name* z objektu `Action`, objekt UnitAgent zastupující dělníka, jež má akci vykonat a objekt UnitAgent představující budovu nad kterou má být akce provedena. Pokud není potřeba k akci dělník či budova, tak je místo příslušného objektu na vstupu hodnota *null*.

- (assistant method) `UnitAgent reserveFacility(TechnologyTreeNode node, List<UnitAgent> facilities)`

  Metoda má na vstupu uzel technologického stromu, který má být vyprodukován a seznam všech postavených budov. Metoda vrací unitAgenta ze seznamu, který má být použit pro produkci zadaného `TechnologyTreeNode`.

- (assistant method) `int checkResourceAvailability(String name)`

  Vstupním parametrem metody je jméno jednoho ze zdrojů a výstupem je celočíselná hodnota udávající množství tohoto zdroje.

- (assistant method) `int getGameTime(int i)`

  Metoda vrátí aktuální herní čas v sekundách. Vstupní parametr udává časový offset.

- (assistant method) `Position findChokepoint()`

  Metoda vrátí pozici nejbližšího "chokepointu", nebo jiného vhodného obranného místa poblíž základny hráče. Pozice je reprezentována `Position` objektem frameworku.

- (assistant method) `Position getSquadPosition(List<UnitAgent> squad)`

  Metoda vrátí pozici zadané útočné skupiny. Pozice je reprezentována `Position` objektem frameworku.

- `boolean isFarFromPosition(Position position)`

  Metoda vrátí *True* pokud jednotka není na zadané pozici. Pozice je reprezentována `Position` objektem frameworku.

- `void sendToPosition(Position position)`

  Metoda pošle jednotku na zadanou pozici. Pozice je reprezentována `Position` objektem frameworku.

- `void attackPosition(Position position)`

  Metoda pošle jednotku za zaútočit na zadanou pozici. Jednotka by měla také útočit na všechny nepřátelské jednotky, na které po cestě narazí. Pozice je reprezentována `Position` objektem frameworku.

- `int getPower()`

  Vrátí celočíselnou hodnotu reprezentující sílu jednotky. Toto číslo by měla mít porovnatelnou hodnotu jako je parametr *power* zadávaný znalostem v databázi znalostí.

- `boolean isCloseToSquad(List<UnitAgent> squad)`

  Vrátí *True* pokud je jednotka v dost blízko útočné skupině a může do ní být zařazena.

- `boolean scoutEnemyBase(Position basePosition)`

  Metoda pošle jednotku prozkoumat nepřátelskou základnu zadané pozici. Pozice je reprezentována `Position` objektem frameworku. Metoda vrací *True* pokud byl průzkum základny úspěšný a *False* pokud se průzkum z nějakého důvodu nezdařil.

- `boolean scout()`

  Metoda pošle jednotku prohledávat mapu za účelem nalezení nepřítele. Metoda vrací *True* pokud byly všichni nepřátelé objeveni.

- `boolean isScouting()`

  Metoda vrací *True*, pokud jednotka aktuálně provádí průzkum.

- `boolean isWorker()`

  Metoda vrací *True*, pokud se jedná o dělnickou jednotku.

- `boolean isSupply()`

  Metoda vrací *True*, pokud se jedná o jednotku, či budovu zvětšující hodnotu populace.

- `boolean isFighter()`

  Metoda vrací *True*, pokud se jedná o útočnou jednotku.

- `boolean isBuilding()`

  Metoda vrací *True*, pokud se jedná o budovu.

- `boolean isResource()`

  Metoda vrací *True*, pokud se jedná o budovu, či jednotku produkující nové zdroje.

- `boolean isIdle()`

  Metoda vrací *True*, pokud jednotka zrovna nevykonává žádnou akci.

- `boolean isAlive()`

  Metoda vrací *True*, pokud je jednotka živá a je možné ji ovládat.

- `boolean isCompleted()`

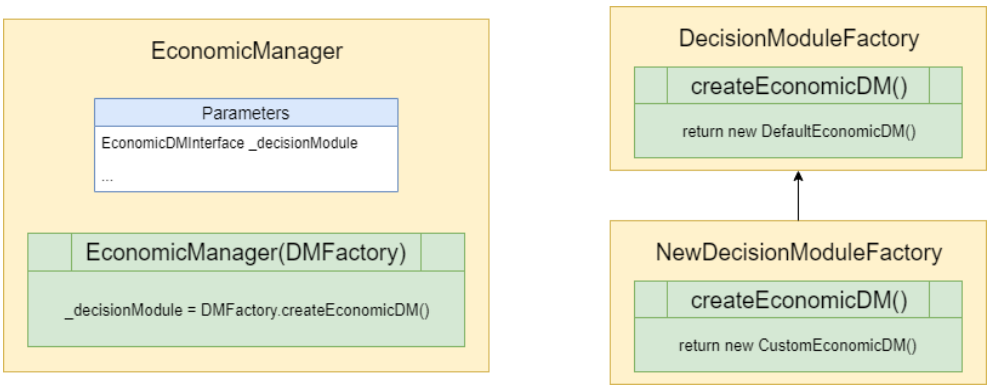
  Metoda vrací *True*, pokud je produkce jednotky, či stavba budovy dokončena.

- `boolean startMineResources()`

  Metoda pošle jednotku těžit aktuálně netěžené zdroje.

B.3 Implementace tříd DecisionModule

Rozhodovací moduly jsou definovány jako rozhraní. Uživatel musí toto rozhraní implementovat a vytvořit vlastní implementaci třídy `DecisionModuleFactory`, která bude vytvářet instance ním vytvořených rozhodovacích modulů. Vlastní instanci `DecisionModuleFactory` pak předá *FrameworkControlleru* při vytváření instancí manažerů.



Obrázek B.1: Zjednodušený diagram předání vlastního rozhodovacího modulu

Následuje popis rozhraní jednotlivých rozhodovacích modulů.

B.3.1 StrategyDecisionModule

- `int recognizeEnemyStrategy(List<BasicKnowledge> enemyUnits)`

  Vstupem metody je seznam nepřátelských jednotek, které má manažer ve své databázi znalostí a výstupem metody by měl být index strategie nacházející se v konfigurační struktuře StrategyList.

- `int getResponseToEnemyStrategy(int enemyStrategyId)`

  Vstupem je index strategie nepřítele a výstupem index strategie, kterou použít proti zvolené nepřátelské strategii.

- `String selectOpeningBuildOrder(List<String> strategies)`

  Vstupem metody je seznam strategií pro prvotní fázi hry, výstupem je vybraná strategie.

- `String selectStandardBuildOrder(List<String> strategies)`

  Vstupem metody je seznam strategií pro pokročilou fázi hry, výstupem je vybraná strategie.

- `String selectLateGameBuildOrder(List<String> strategies)`

  Vstupem metody je seznam strategií pro závěrečnou fázi hry, výstupem je vybraná strategie.

- `Position getAttackPointOfInterest(KnowledgeBase knowledgeBase, PositionType positionType)`

  Do metody vstupuje databáze znalostí agenta a typ pozice co v ní mýt hledána. Výstupem je požadovaná pozice pro útok, obranu, nebo průzkum.

- `HashMap<String,Integer> getNeededResources(Integer strategyId, BuildOrder currentBuildOrder)`

  Na vstupu metody je aktuálně zvolený index strategie hry a objekt zvolené stavební strategie. Výstupem by měla být mapa potřebných surovin, která bude zaslána Economic managerovi.

B.3.2 EconomicDecisionModule

- `int checkWorkerCount(int numberOfWorkers, int gameTime, HashMap<String, Double> totalResourcesMined)`

  Tato metoda má na vstupu aktuální počet dělníků, herní čas v sekundách a mapu herních surovin s aktuálními hodnotami. Výstupem by měl být počet dělníků o které si má manažer zažádat.

- `TechnologyTreeNode shouldBuildExpansionBuilding( List<BasicKnowledge> buildings, List<String> resourceGenerators, List<String> bases, int gameTime)`

  Metoda má na vstupu seznam již postavených budov, seznam jmen hlavních budov a generátorů surovin získaných ze struktury TechnologyTree a aktuální herní čas. Výstupem této metody je list technologického stromu reprezentující hlavní budovu, nebo zdroj surovin o který si má manažer zažádat.

B.3.3 InfrastructureDecisionModule

- `List<String> correctPlannedActions(List<String> plannedActions, InfrastructureManagerState state, BuildOrder buildOrder, List<String> availableActions, HashMap<String, HashMap<String, Integer» requestedUnits)`

  Vstupem této metody je aktuální seznam naplánovaných akcí, aktuální stav manažera, zvolená strategie stavění ze struktury BuildOrders, seznam akcí, které mohou být aktuálně provedeny a mapu ve které jsou uloženy žádosti o jednotky z ostatních agentů. Výstupem této metody může být libovolně upravený seznam akcí, ale uživatel by měl zajistit, aby do plánu byly přidávány i jednotky o které bylo požádáno.

- `Boolean shouldAddSupplyAction(ValueLevel freeSupply, InfrastructureManagerState lastState)`

  Vstupem této metody je úroveň volné populace a poslední známý stav manažera. Výstupem funkce by měl být True, pokud je volné populace málo a je třeba přidat do naplánovaných akcí stavbu budovy, která slouží jako generátor populace. Pokud je volné populace dostatek, nebo je přidávání generátorů obstaráváno metodou na korekci naplánovaných akcí, metoda by měla vracet False.

B.3.4 TacticalDecisionModule

- `BasicKnowledge choosePrimaryTarget(List<BasicKnowledge> allEnemyAssets)`

  Vstupem této metody je seznam nepřátelských jednotek a budov. Výstupem by měl být prioritní cíl pro útok.

- `TacticalGoal selectCurrentGoal(KnowledgeBase knowledgeBase, List<UnitAgent> squad)`

  Vstupem metody je databáze znalostí agenta a seznam jednotek v aktuální útočné skupině. Výstupem je vhodně zvolený taktický cíl (útok, obrana, čekání na posily).

- `int getSquadPower(List<UnitAgent> squad)`

  Vstupem metody je seznam jednotek v útočné skupině a výstupem je síla armády vyjádřená celým číslem.

- `int getEnemyPower(List<BasicKnowledge> enemyAssets)`

  Vstupem metody je seznam nepřátelských budov a jednotek získaný z databáze znalostí agenta a výstupem je síla nepřítele vyjádřená celým číslem.

- `Pair<TechnologyTreeNode, Integer> shouldRequestNewUnits( List<BasicKnowledge> allEnemyAssets, List<UnitAgent> squad, KnowledgeBase knowledgeBase)`

  Tato metoda má na vstupu seznam nepřátelských budov a jednotek, seznam jednotek v aktuální útočné skupině a databázi znalostí agenta. Výstupem by měla být dvojice s jednotkou o kterou je třeba zažádat a číslem určující množství těchto jednotek.

B.3.5 ReconDecisionModule

- `ReconSearchMethod getSearchMethod()`

  Metoda zvolí metodu prohledávání mapy pro tuto hru. Navrácená hodnota je uložena v Recon managerovi.

- `boolean shouldSendKnowledgeToManager(BasicKnowledge knowledge, ManagerType manager)`

  Metoda má na vstupu znalost a typ agenta. Pokud je žádoucí aby agent tuto informaci dostal metoda vrátí True, jinak false.

- `boolean shouldRemoveKnowledgeFromManager(OwnerType owner, KnowledgeType type, UnitAgent object, ManagerType manager)`

  Na vstupu metody je vlastník znalosti, její typ a instance třídy UnitAgent. Metoda rozhodne zda znalost spojená s těmito parametry by měla být odstraněna z agenta označeného vstupním parametrem ManagerType.

- `boolean shouldRemoveKnowledgeFromManager(OwnerType owner, KnowledgeType type, Position p, ManagerType manager)`

  Stejná funkcionalita jako předchozí metoda, ale místo objektu rozhoduje odstranění z databáze agenta podle pozice znalosti.

- `boolean shouldRemoveKnowledgeFromManager(OwnerType owner, KnowledgeType type, String name, ManagerType manager)`

  Další variace metody pro odstranění znalosti z databáze agenta, ale tentokrát podle jména spojeného se znalostí.

Příloha C

Manuál pro použití systému

Tento manuál předpokládá, že uživatel již implementoval třídy UnitAgent, vybrané rozhodovací moduly a vytvořil konfigurační struktury popsané v příloze B. Tento manuál ukazuje jak tyto třídy a struktury použít ve frameworku, jak celý systém inicializovat a také způsob předávání informací ze hry vytvořenému agentnímu systému.

C.1 Vytvoření a inicializace agentního systému

Prvním krokem je vytvoření instance třídy `FrameworkController` a načtení konfiguračních struktur. To je učiněno zavoláním metody `createConfigStructure(`
`String TechnologyTreeFilePath, String ActionListFilePath,`
`String BuildOrdersFilePath, StrategyListFilePath, String PlayerRace)`

```
this.frameworkController = new FrameworkController();
frameworkController.createConfigStructures(
     "data/config/TT.json",
     "data/config/Actions.json",
     "data/config/BuildOrders.json",
     "data/config/Strategies.json",
     "Protoss"
);
```

Následuje vytvoření instancí DecisionModuleFactory, UnitAgent asistenta a následné vytvoření instancí jednotlivých manažerů.

```
DecisionModuleFactory dmFactory = new CustomDMFactory();
UnitAgentImpl assistant = new UnitAgentImpl(null);

frameworkController.createInfrastructureManager(dmFactory, 64, 64, assistant);
frameworkController.createEconomicManager(dmFactory, resources, assistant);
frameworkController.createReconManager(dmFactory, assistant);
frameworkController.createTacticalManager(dmFactory, assistant);
frameworkController.createStrategyManager(dmFactory, assistant);
```

Poté už jen stačí spustit agentní systém a vložit instanci UnitAgent do běžícího systému.

```
frameworkController.startMAS();
assistant.createUnitAgentInstance(frameworkController);
```

Tímto krokem je inicializace spuštění agentního systému kompletní.

C.2 Vytváření nových znalostí a jednotek

Aby agentní systém mohl správně rozhodovat o akcích, které ve hře vykonává je třeba mu předávat informace o vzniku a zániku herních entit. Framework poskytuje několik možností jak předávat informace agentnímu systému. Nejjednodušší variantou je předávat všechny znalosti Recon managerovi, který se postará o to, aby byly informace předány agentům, kteří je potřebují. To může být učiněno třeba následujícím způsobem:

```
BasicKnowledge knowledge = new BasicKnowledge();
knowledge.setParameters(type,
     position,
     faction,
     ownerType,
     power,
     unitType,
     desc,
     agentName);

((ReconManager)Manager.getManagerByName(ManagerType.RECON.toString()))
     .addKnowledge(knowledge);
```

Rozhodování o tom co bude Recon manager posílat kterému agentovi lze ovlivnit pomocí rozhodovacích modulů. Pokud by uživatel chtěl může použít i definované metody UnitAgenta pro vytvoření a odeslání zprávy. Jedná se o metody:

- `String createKnowledgeMessageContent(KnowledgeType type, Position position, String faction,OwnerType owner, int power, String unitType, String description, String unitAgentName)`
- `void sendKnowledgeToManager(ManagerType manager, String messageContent)`

Společně se znalostmi je třeba i hlídat vytvoření nových jednotek a budov hráče. Pokaždé když je taková entita vytvořena je třeba pro ni vytvořit instanci UnitAgenta a tu předat Infrastructure managerovi, tak jak je ukázáno v příkladu:

```
newAgent = new UnitAgentImpl(unit);
newAgent.createUnitAgentInstance(this.frameworkController);
Manager.getManagerByName(ManagerType.INFRASTRUCTURE.toString())
     .transferUnit(newAgent);
```

Metoda `void createUnitAgentInstance(FrameworkController frameworkController)` zajistí přidání vytvořeného UnitAgent do běžícího agentního systému, aby mohl přijímat a odesílat ACL zprávy ostatním agentům.

C.3 Předávání akcí hře

Je třeba identifikovat místo, kde bude agentní systém předávat povely hře. Většina herních API bude mít event onFrame. Je doporučeno použít reakci na tento event k zavolání `onFrame()` metody nad všemi manažery, tak jak je ukázáno v tomto příkladu:

```
for (Manager manager : Manager.getAllManagers()) {
     manager.onFrame();
}
```

Tímto je zajištěno, že agentní systém bude schopný skrze přidělené UnitAgenty provádět akce ve hře.

C.4 Informování systému o začátku akce

Pokaždé když je započata akce zavoláním metody `void runAction(Action action, UnitAgent currentWorker, UnitAgent currentFacility)` třídy UnitAgent, je třeba pohlídat zda byla akce opravdu provedena a v tu chvíli zavolat metodu `void actionStarted( String actionName)` nad Infrastructure managerem. Když je tato metoda zavolána nemusí nutně být požadovaná akce dokončena, ale musí být v té chvíli již být použity požadované zdroje a musí být možné vrátit dělníka Economic managerovi. Příkladem je třeba volání metody v `onUnitCreate(Unit unit)` metodě třídy `DefaultBWListener`.

```
@Override
public void onUnitCreate(Unit unit) {
     InfrastructureManager manager = (InfrastructureManager)Manager
          .getManagerByName(ManagerType.INFRASTRUCTURE.toString());

     Action actionForNode = ActionList.getInstance()
          .findActionForNode(unit.getType().toString());
     manager.actionStarted(actionForNode.get_name());
}
```

C.5 Uložení výsledků hry

Tento krok je volitelný a není nutné ho provádět. Pokud má uživatel v plánu učit agentní systém mezi hrami může na konci hry uložit stavy, kterými manažeři během hry prošli a uložit aktualizované data ve struktuře *ManagersHistory*. To je provedeno pomocí metody `void writeStates(AgentStatesHistoryWriter writer, String filePath, Boolean alreadyExists)` na jednotlivými manažery a metodou `writeToFile( String FilePath)` nad objektem `ManagersHistory`.