
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**
**DEPARTMENT OF INTELLIGENT SYSTEMS**


# MULTIAGENTNÍ PODPORA PRO VYTVÁŘENÍ STRATEGICKÝCH HER

**MULTIAGENT SUPPORT FOR STRATEGIC GAMES**

**DIPLOMOVÁ PRÁCE**
**MASTER'S THESIS**


**AUTOR PRÁCE** — **Bc. LUKÁŠ VÁLEK**
**AUTHOR**

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



**BRNO 2018**

# 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, TBS, Multi-agentní systémy, UI, agent, umělá inteligence, real-time strategy, turn-based strategy

# Keywords

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



# Citace

VÁLEK, Lukáš. *MULTIAGENTNÍ PODPORA PRO VYTVÁŘENÍ STRATEGICKÝCH HER.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. František Zbořil, Ph.D.

# MULTIAGENTNÍ PODPORA PRO VYTVÁŘENÍ STRATEGICKÝCH HER

## 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
22. května 2018

## 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. [13]

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, nebo Spacewar! byly založeny na diskrétní logice a 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.

Obtížnost vytvoření centralizované UI pro strategie není jediný problém. Tento přístup vyžaduje velké množství výpočetních zdrojů, které si herní vývojáři nemohou dovolit postrádat. Výzkum umělé inteligence v RTS významně ovlivnily multiagentní systémy. Mul-

[1] Verze šachů, která vznikla v Japonsku.

tiagentní systémy (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 počítačovou hru, a který bude schopný hrát tak, aby byl pro hráče důstojným soupeřem.

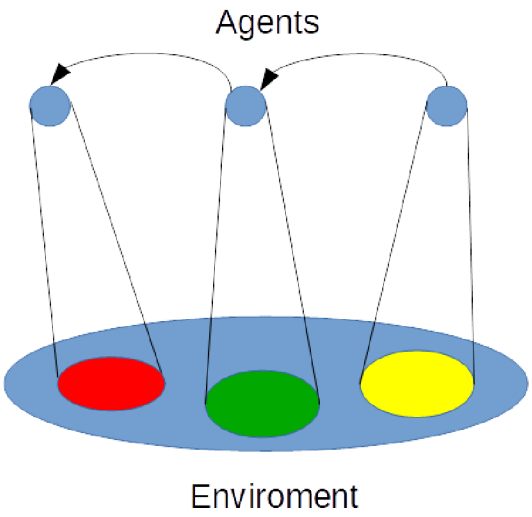
Tato práce byla vytvořena ve spolupráci s Bc. Petrem Knapkem, který měl za úkol navrhnou a vytvořit procesy rozhodování, adaptace a učení pro navržené agenty. Dále bylo jeho úkolem implementovat pomocí frameworku testovací model, na kterém byla ověřena jeho funkčnost. Mým úkolem bylo vyvinout a implementovat navržený agentní systém, společně s komunikačními protokoly reprezentujícími sociální schopnosti agentů a návrh rozhraní pro propojení frameworku s vybranou hrou.

Práce je rozdělena na několik částí. První kapitolu tvoří úvod práce. Ve druhé kapitole budou uvedeny základní informace o multiagentních systémech, které budou požívány v dalších částech práce. Třetí kapitola se věnuje popisu počítačových strategických her, jejich mechanik a prvků. Jejím účelem je seznámit čtenáře s problematikou a zasvětit ho do základních problémů, s nimiž se hráč potýká. Čtvrtá kapitola se věnuje využití systémů umělé inteligence ve strategických hrách. V této kapitole je vysvětleno s jakými problémy se setkávají vývojáři při tvorbě strategických her a ukáže potenciál agentních systémů v těchto hrách. Pátá kapitola se věnuje přístupů k agentním systémům a jejich využití ve strategických hrách. Šestá kapitola obsahuje popis návrhu, jenž bude použit jako základ frameworku, jehož implementace je popsána v sedmé kapitole. Jsou zde popsány použité technologie a implementace všech hlavních částí frameworku. V osmé kapitole je popsán testovací model, který byl použit k validaci implementovaného frameworku. Poslední dvě kapitoly se věnují shrnutí a zhodnocení výsledků práce.

# Kapitola 2

# Multiagentní systémy

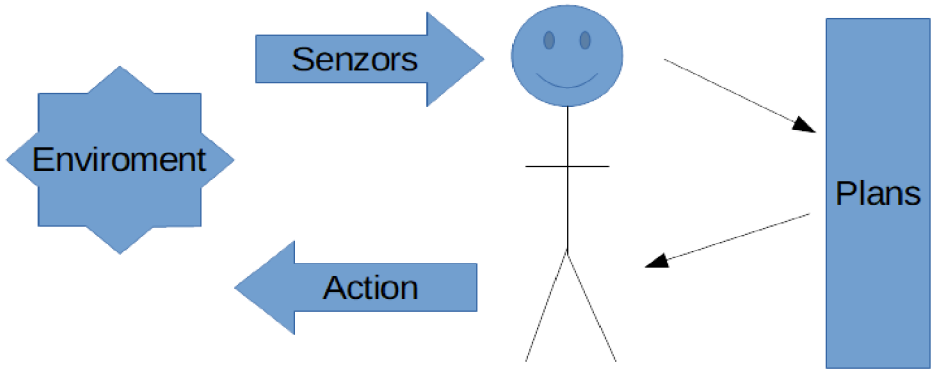
Multiagentní systém (MAS) je systém složený z několika autonomních a interagujících softwarových částí, které jsou umístěny v prostředí, jenž mohou ovlivňovat. Tyto softwarové části se nazývají agenti. Agenti v MAS se vyznačují tím, že mohou samostatně operovat jen ve své vlastní sféře vlivu. Ta omezuje 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[16]. Tyto relace jsou naznačeny v obrázku 2.1.




Obrázek 2.1: Schéma MAS. Šipky naznačují komunikaci mezi agenty a barevné elipsy, oblast se kterou agent pracuje.

## 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 2.2).[14]




Obrázek 2.2: 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. Tato mechanika umožňuje agentovi plánovat nejlepší cestu ke svému cíli dynamicky a bez potřeby dalších vstupů. Jednoduše řečeno jsou agentovy akce pod jeho vlastní kontrolou a nejsou řízeny dalšími entitami.
- **Proaktivita** — je schopnost vykazovat cílem řízené chování. Jinými slovy, agent se bude snažit dosáhnout cílů, které jsou mu delegovány. Cílem řízené chování je provádění určité posloupnosti akcí vedoucích ke splnění cíle.
- **Reaktivita** — reprezentuje schopnost agenta reagovat na změny v prostředí. Agent by měl na tyto změny reagovat efektivním vyvažováním mezi cílem řízeným chováním a reaktivním chováním. 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 Belief-Desire-Intention Agent

BDI agent je typ agenta, který je inspirován a založen na modelu lidského chování, vyvinutého na filosofických základech. Záměr je vytvořit agenta, který je stejně jako člověk, řízen svými psychickými stavy. Prakticky se jedná o počítačové programy s výpočetními analogiemi Víry (Belief), Touhy (Desire) a Záměru (Intention). Postoje agenta k těmto psychickým stavům určují jeho chování a jsou kritické k dosažení optimálního a adekvátního výsledku.[15]

- **Beliefs** – Reprezentuje znalosti, které agent má o svém prostředí. Tyto informace může buď vyčíst ze svého okolí, nebo si je může být schopný sám odvodit. Stejně jako u člověka tyto informace mohou být nepřesné, zastaralé, nebo taky úplně špatné.
- **Desires** – Reprezentuje všechny možnosti cílů, o které by mohl agent usilovat. Jedná se o jistou množinu možností, ze kterých si může agent vybírat a pracovat na jejich uskutečnění. Ne všechny touhy musejí být kompatibilní a ne nutně je agent schopný uskutečnit všechny své touhy.
- **Intentions** – Reprezentuje stav záležitostí, na kterých agent pracuje. Mohou to být cíle, které jsou agentovi delegovány, a nebo se může jednat o výsledek agentova rozhodnutí na základě možností, které jsou mu k dispozici. Záměry jsou tedy zvolené touhy na kterých se agent rozhodl pracovat. Po zvolení nějakého cíle agent je zavázán ho splnit. Cíl může být zahozen jen v případě, že je splněn, nebo nastala situace, která dělá splnění cíle nemožné. Intentions mají tedy 3 vlastnosti. Jsou perzistentní, omezují budoucí možnosti a vedou k nějaké akci.

# Kapitola 3

# 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. Zjednodušeně jde na jakoukoliv strategickou hru pohlížet jako na hru šachu. Úkolem hráče je podniknout určité kroky, které omezí možnosti jeho soupeře, a dát soupeři tzv. „šach mat“. Ovšem v počítačových strategických hrách je potřeba mnohem více kroků, abychom soupeře porazili. Hráč se musí starat o infrastrukturu své civilizace, produkovat jednotky a sbírat suroviny. Strategické hry lze rozdělit na Turn-Based Strategy Games (TBS) a Real-Time Strategy Games (RTS).

## 3.1 Real-Time Strategy Games

Real-Time strategy, neboli zkráceně RTS, jsou převážně 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.

### 3.1.1 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 svižná a vyžaduje schopnost rychlého rozhodování a hbité 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.

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


Obrázek 3.1: Ukázka uživatelského rozhraní ze hry Starcraft.

„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. Pro hráče je tedy extrémně důležité posílat zvědy a objevovat mapu, aby měl aktuální informace a mohl provádět informovaná rozhodnutí.

### 3.1.2 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. Platí zde princip "kámen–nůžky–papír", kdy určité jednotky mají výhodu proti konkrétnímu druhu nepřátelských jednotek, ale jsou zase naopak slabé proti jiným jednotkám nepřítele.

### 3.1.3 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.

## 3.2 Turn-Based Strategy Games

Turn-Based Strategy, neboli zkráceně TBS, jsou typ strategických her, kde se hráči střídají po kole. Na rozdíl od RTS zde mají hráči mnohem více času, aby mohli přemýšlet o svých akcích a plánovat všechny své kroky před tím, než je budou muset skutečně provést. Hlavním rozdílem je tedy jak v těchto hrách plyne čas. Zatímco v RTS mohou všichni hráči zároveň reagovat na události herního světa, v TBS má každý hráč svůj herní čas jen pro sebe. Hráč má na každé kolo určitý počet akcí, které může vykonat. Jakmile jsou tyto akce vyčerpány, přichází na řadu další hráč v pořadí.

### 3.2.1 Prvky a mechaniky

TBS, na rozdíl od RTS, nejsou tak pevně definovány. Mezi jednotlivými TBS lze najít mnohem více rozdílu, ať už v herních mechanikách nebo způsobu, jakým se hrají. Přesto ale lze najít několik prvků, které platí pro všechny. Tím nejdůležitějším je právě rozdělení herního času na kola. Každý hráč dostane ve svém kole příděl akcí, které může vykonat. Co taková akce může být se liší, ale většinou tyto akce říkají o kolik kroků může posunout své jednotky nebo kolik budov může postavit. Tyto kola nemusejí být nijak časově omezena, takže hráč si může vzít na přemýšlení o svých akcích kolik času potřebuje. Tento fakt významně mění hratelnost i přístup hráčů k samotné hře. Zatímco RTS je převážně taktická hra, kde se hráč rozhoduje ve zlomku sekundy, TBS je více o strategii a dlouhodobých plánech.

Na začátku hry hráč dostává kontrolu nad nějakou civilizací a stejně jako u RTS bývá většinou jeho úkolem porazit civilizace jeho soupeřů. To, jak toho dosáhne, se může v jednotlivých hrách lišit, ale většinou to je poražením nepřátelských sil a dobytím jeho základny. Pro TBS bývá typické, že mají dva módy. Jeden je určen pro přesun jednotek po mapě a kontrolu základen (Obrázek 3.2), druhý mód je určen pro kontrolu boje mezi jednotkami hráče a nepřítele (Obrázek 3.3).

Mapy TBS mají podobu, která se příliš neliší od herní desky nějaké stolní hry. Základny, doly surovin a jiné objekty jsou na mapě přítomné od začátku hry a úkolem hráče je přivlastnit si dané objekty tím, že na ně přesune své jednotky. Veškeré budovy, které hráč staví, jsou spíše vylepšení základny a fyzicky se na mapě neprojevují. Mapa má většinou podobu mřížky a hráči se mohou pohybovat po jednotlivých políčkách mapy. Políčka mohou obsahovat již zmíněné základny, generátory zdrojů, a nebo také jednotky a ne-hráčská monstra (NPC), která po své porážce hráče odmění. Rozdělení mapy do mřížky není vždy

pravidlem. Některá moderní TBS od ní již začínají upouštět a poskytují hráči větší volnost pohybu po mapě. Stejně jako v RTS, i zde se může vyskytovat Fog of War.

Obrázek 3.2: Kontrolní mód v Heroes of Might and Magic VII.


Obrázek 3.3: Bojový mód v Heroes of Might and Magic VII.

### 3.2.2 Vedení boje

Stejně jako v RTS, u většiny TBS je úkolem hráče zničit nebo obsadit základnu nepřítele. Ovšem je velký rozdíl v tom, jak k tomu jednotlivé žánry přistupují. V RTS má hráč neustále přístup ke všem svým jednotkám. Pokud se tedy silnější hráč dostane do nečekaného, nevýhodného konfliktu, může kdykoliv přivolat zbytek své armády a nepřítele porazit.

Stejná akce není možná v TBS. Zde jsou jednotky hráče rozdělené do jednotlivých armád, které se pohybují po mapě. V případě, že se střetnou dvě nepřátelské armády, hra se přepne do bojového módu a zde může každý hráč použít jen ty jednotky, které byly do konkrétní armády přiděleny. To znamená, že pokud se hráč dostane do konfliktu, na který není připraven, tak se nevyhne vážným ztrátám. Tyto ztráty mohou snadno vést k prohře, jelikož doplňování padlých jednotek v TBS mívá většinou jistá omezení.

Dalším zajímavým rozdílem mezi RTS a TBS je kolik informací má hráč o samotné hře. Jak již bylo dříve zmíněno, Fog of War není u TBS vždy přítomna. To znamená, že hráč má buď kompletní přehled o hře, nebo naopak jsou jeho informace o hře značně omezeny, jelikož u TBS z pravidla neexistují průzkumné jednotky. To vede k velmi rozdílným strategiím v jednotlivých TBS hrách.

Tyto důvody jsou příčinou toho, že přerozdělování svých jednotek mezi armádami rozmisťování na základě předpokládaných tahů nepřátel je velmi důležitý aspekt válčení v TBS. Obtížnost této činnosti velmi závisí na množství informací, které hráč o hře má. Vedení boje v tahových strategiích je velmi podobné klasické hře šachů. Je potřeba přesunovat své armády tak, aby systematicky odstraňovaly nepřátele, kteří by mohli ohrozit základnu hráče, a zároveň přesouvat své armády tak, aby mohly zaútočit a dobýt základnu nepřítele.

### 3.2.3 Ekonomika

Ekonomická stránka TBS je téměř identická s RTS. Na mapě se vyskytují zdroje surovin, které může hráč obsadit a získávat tak periodicky suroviny každé kolo. Tyto zdroje musí bránit, protože mohou být dobyty nepřítelem.

Největším rozdílem mezi TBS a RTS je fakt, že v TBS je řada omezení co se týče produkce zdrojů, jednotek a budov. V RTS může hráč snáze a mnohem rychleji upravovat míru své produkce a získávání zdrojů. V TBS je hráč mnohem více ovlivněn svými předchozími rozhodnutími. To znamená, že hráč musí více plánovat kdy a jakou budovu postavit a které generátory zdrojů obsadí. Stejně jako u všech aspektů TBS, i zde platí, že se jedná spíše o strategická rozhodnutí.

# Kapitola 4

# 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.

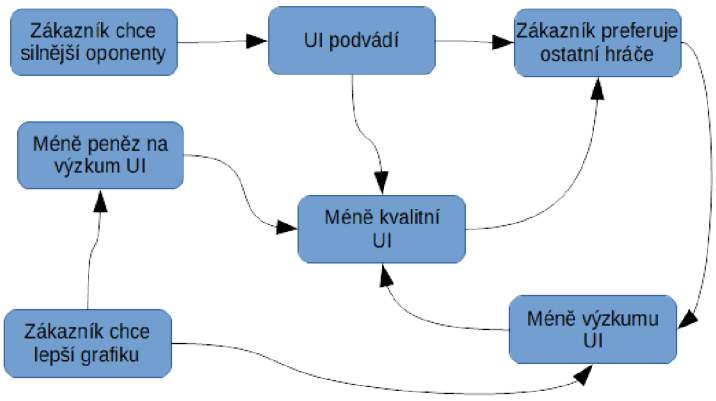
I přesto, že samotné hry se staly chytřejší a jsou schopny hráčům velmi pomáhat s ovládáním hry, tak to samé nemůže být řečeno o jejich schopnosti produkovat schopné, počítačem řízené protivníky pro hráče. Z pozorování a hraní[1] nových strategických her nelze říct, že by se UI řídící vaše protivníky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

V TBS 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 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.

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.[8] Schéma těchto závislostí lze vidět na obrázku 4.1.

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




Obrázek 4.1: Schéma relace zájmu a výzkumu UI ve strategických hrách

## 4.1 Agentní systémy jako UI ve strategiích

Jak lze z předchozích závěrů odvodit, centralizované umělé inteligence nejsou příliš vhodné pro kontrolu protivníků v RTS. Zatímco v TBS je možno s jejich pomocí dosáhnout velmi dobrých výsledků, tak v RTS existují oblasti, na které nestačí. Velkým průlomem ve výzkumu UI v RTS byl příchod agentních systémů. Agentní systémy jsou více než schopny nahradit standardní UI používanou v RTS hrách.

Je toho docíleno několika kroky:

- 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 klasickou centralizovanou UI, která se v současné době ve strategických hrách používá. V následujících kapitolách budou popsány hlavní výhody agentních systémů ve strategických hrách.

### 4.1.1 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 UI 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 bázi znalostí, ke které může libovolně přistupovat. Také musí být schopný se rozhodovat, kterou akci uskuteční na základě všech informací, které má k dispozici.

### 4.1.2 Realistické chování

Je mnoho charakteristik chování, které mohou rozlišovat umělého a skutečného protivníka. Jedná se o vlastnosti, které je třeba emulovat. UI by se tedy měla 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. Běžné UI 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.

### 4.1.3 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.

# Kapitola 5

# Agentní systémy 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.[9][11][12] V následujících kapitolách budou tyto přístupy popsány společně s výhodami a nevýhodami spojenými s jejich užitím.

## 5.1 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 obrovské množství agentů, kteří se nezávisle rozhodují na základě vnějších podnětů a zpráv od ostatních agentů, efekt motýlích křídel se zde velmi výrazně projevuje. V ideálním případě by takový systém na každou strategii hráče reagoval jiným způsobem. Ovšem tento přístup sebou přináší i několik úskalí.

Tento systém vyžaduje 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.

Tento přístup funguje velmi dobře v TBS, jelikož zde lze lépe určit, kdy je třeba aby agenti prováděli své rozhodovací cykly, a také zde je většinou potřeba menšího počtu agentů.

## 5.2 Entity kontrolované manažery

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. Tito manažeři pak kontrolují hru jen na nejvyšší úrovni a nechávají některé triviální akce na zabudované umělé inteligenci samotné hry. Tento systém poskytuje mnohem větší univerzálnost. Malé množství agentů, jejichž plány jsou z pravidla velmi obecné umožňuje znovupoužitelnost v několika různých hrách. Další výhodou je několikanásobně 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.

## 5.3 Hybridní přístup

Tento přístup spojuje obě předchozí metody. 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 potenciálně 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.

# Kapitola 6

# 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, který je řízen pomocí manažerů. Tento systém byl zvolen díky své univerzálnosti a menšímu počtu agentů nutných pro řízení systému, což by teoreticky mělo snížit náročnost napojení frameworku na libovolnou hru.

Nejprve je popsána základní funkce agenta a vysvětlen důvod jeho přítomnosti. Vzhledem k tomu, že někteří agenti spolu musejí komunikovat, je nutné navrhnout i komunikační protokoly pro tyto agenty. Jejich sociální schopnosti budou obecně popsány v této kapitole a detailněji budou vysvětleny až v kapitole o implementaci. Dále je zde naznačen základní princip ukládání znalostí v agentech, který bude také detailněji popsán v kapitole o implementaci.

## 6.1 Strategy manager

Strategy manager zastává jednu z nejdůležitějších rolí v celém agentním systému. Jeho úkolem je volit vhodnou strategii pro každou fázi hry a kontrolovat, zda je zvolená strategie stále nejlepší volbou. 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.

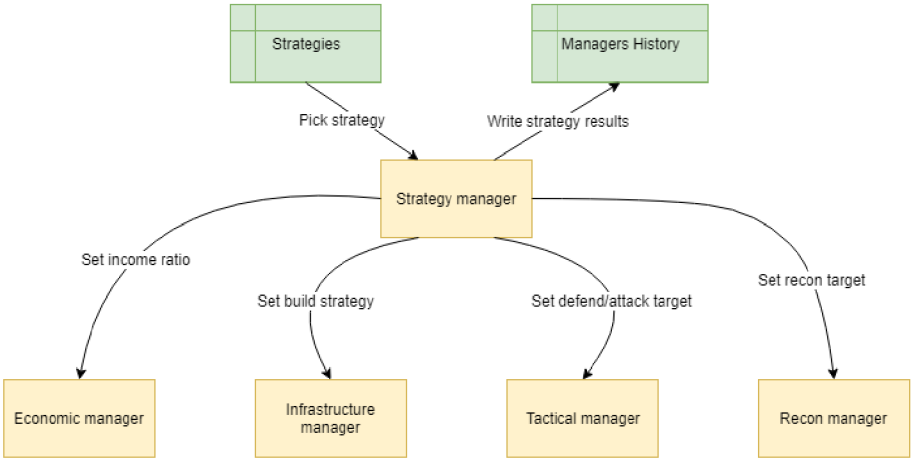
### 6.1.1 Sociální schopnosti

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.

### 6.1.2 Důvod přítomnosti manažera

Kdyby se v systému nevyskytoval Strategy manager, tak by systém mohl vykonávat akce, které nejsou třeba, a nebyl by schopen adekvátně reagovat na tahy nepřítele. Mohlo by se například stát, že se příliš mnoho zdrojů používá na expanze, když ještě není vytvořena dostatečná infrastruktura pro efektivní spotřebovávání těchto zdrojů. Také by nebylo možné měnit stavební plán v reakci na strategii nepřítele. Tyto a podobné situace by způsobili nízkou reaktivnost agentů a statické chování systému.




Obrázek 6.1: Schéma Strategy managera

## 6.2 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.

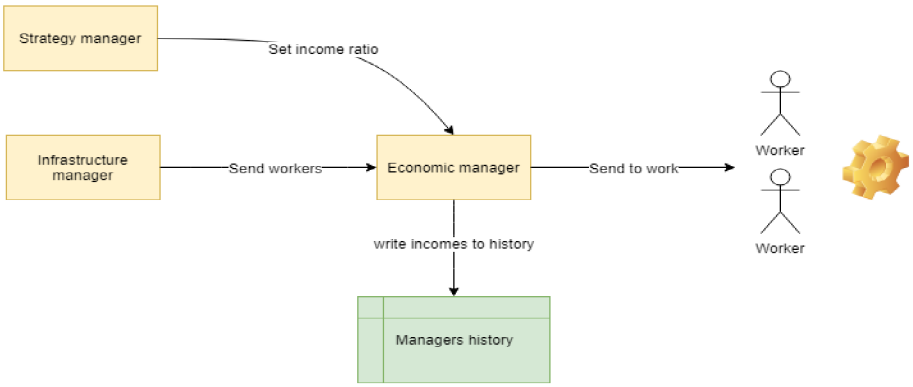
### 6.2.1 Sociální schopnosti

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.

### 6.2.2 Důvod přítomnosti manažera

Bez Economic managera by bylo velmi obtížné synchronizovat příjem surovin, stavbu nových budov a produkci jednotek. Mohly by nastat situace jako je použití dělníků ke stavbě na úkor přísunu zdrojů, a nebo konflikty v přidělení surovin, kdy by byly dvěma různým akcím přiděleny stejné zdroje.




Obrázek 6.2: Schéma Economic managera

## 6.3 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.

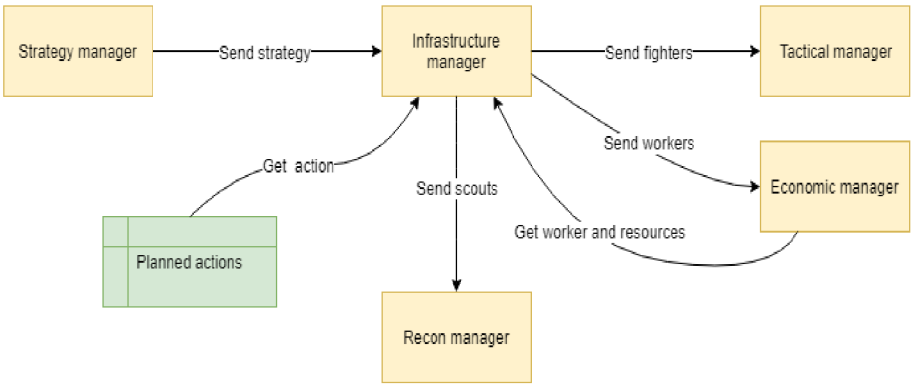
### 6.3.1 Sociální schopnosti

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 druhů komunikace. 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 - Jedná se o jediného agenta, se kterým má Infrastructure manager jednosměrnou komunikaci. Pouze od něj přijímá zprávy tak, jak bylo popsáno v kapitole 6.1

### 6.3.2 Důvod přítomnosti manažera

Bez Infrastructure managera by neexistovala žádná herní ekonomika. Nebylo by možné získat další jednotky k těžbě surovin, nestavěly by se žádné budovy a nevytvářela by se žádná armáda. Ve chvíli, kdy by tyto činnosti dělali ostatní agenti, by se velmi rozrostly komunikační protokoly mezi jednotlivými agenty a jejich práce v agentním systému by se stala zbytečně složitější. Také by bylo velmi obtížné tuto ekonomiku nějakým způsobem organizovat.




Obrázek 6.3: Schéma Infrastructure managera

## 6.4 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.

### 6.4.1 Sociální schopnosti

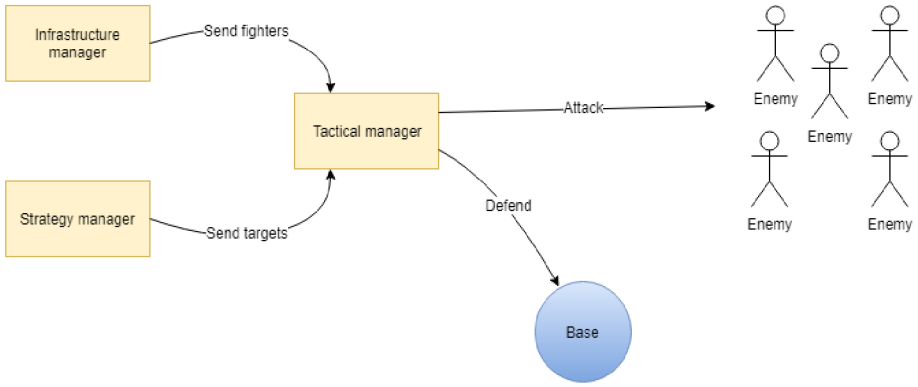
Tactical manager potřebuje ke své činnosti minimální kontakt s ostatními agenty. Komunikuje pouze se Strategy managerem a Infrastructure managerem. V obou případech se jedná o obousměrnou komunikaci a jedná se převážně o předávání informací.

- 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 akci.

- 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.

### 6.4.2 Důvod přítomnosti manažera

Bez Tactical managera by nebylo možné sofistikovaně řídit útočné jednotky. Jednotky by buď stály na místě nebo by jim byl nařízen útok a po jednom by útočily na nepřátele bez jakékoliv organizace či schopnosti se reaktivně stahovat a přeskupovat. Také by nebylo možné poručit jednotkám stáhnout se zpět k obraně základny.




Obrázek 6.4: Schéma Tactic managera

## 6.5 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.

### 6.5.1 Sociální schopnosti

Recon manager, stejně jako Tactical manager, komunikuje pouze s Infrastructure managerem a Strategy managerem. Je to tak právě proto, že stejně jako on potřebuje minimální kontakt se svým okolím. Jeho komunikační protokoly by byly mnohem složitější, kdyby musel všechny nalezené informace předávat manažerům, aby si je mohli ukládat do svých databází znalostí. Tento přístup však není příliš vhodný, protože uživatel frameworku stejně vždy musí vytvořit nějakou detekci nově vzniklých objektů a jejich mapování z herní reprezentace na typy používané v navrženém agentním systému. Takže je mnohem efektivnější vložit tyto znalosti příslušnému manažerovi do databáze znalostí ve chvíli, kdy jsou objeveny, než vytvořit rozsáhlé komunikační cesty, kterými by si všichni agenti předávali znalosti, které objevili při své činnosti. Tím pádem jediná komunikace, která musí probíhat, je:

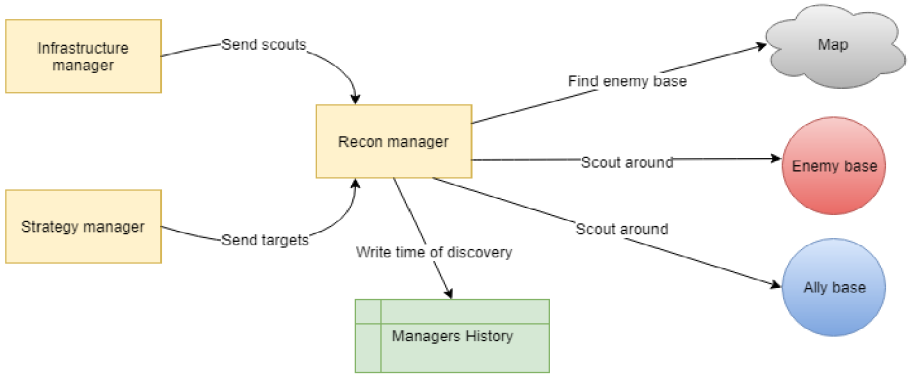
- Infrastructure manager – Recon manager musí žádat o zaslání průzkumných jednotek. Ve chvíli, kdy Infrastructure manager shledá, že má jednotku, jež by mohl obětovat na průzkum, tzn. že není potřeba vytvářet dělníky nebo není krizová situace, která vyžaduje prioritní vytváření bojových jednotek, vytvoří 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.

### 6.5.2 Důvod přítomnosti manažera

Bez Recon managera by měl celý systém jen informace, které by objevil při svých standardních činnostech. Bylo by nemožné získávat informace o nepřátelských armádách a základnách předtím, než by se s nimi Tactical manager setkal, a tím pádem by Strategy manager nemohl vybírat vhodné strategie proti konkrétním protivníkům. Mimo jiné by taky Tactical manager neznal pozice, kde má zaútočit, a musel by prozkoumávat jednotlivé lokace sám, a tím zbytečně otevírat základnu potencionálnímu útoku.




Obrázek 6.5: Schéma Recon managera

# Kapitola 7

# Popis implementace

V této kapitole budou popsány použité technologie a konfigurační soubory sloužící k nastavení frameworku. Dále zde bude popsána Knowledge Base 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.

## 7.1 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).

### 7.1.1 JADE

Je softwarový framework v jazyce Java, který usnadňuje implementaci multiagentních systémů, které komunikují s využitím FIPA specifikace zpráv[3]. Obsahuje komunikační architekturu umožňující flexibilní a efektivní zasílání zpráv, při kterém JADE vytváří a spravuje příchozí ACL zprávy. Dále obsahuje abstraktní třídy pro implementaci agentů a jejich chování. Chování agentů v JADE se definuje v Behaviour třídách. V těchto třídách jsou definovány akce, jež agent vykonává buď periodicky, nebo na základě přijmuté ACL zprávy. Díky tomu lze snadno rozdělit chování agenta pro konkrétní situace. Lze tak vytvořit např. chování pro příjmání zpráv, nebo Behaviour modul, který bude periodicky pročítat a vyhodnocovat informace z databáze. JADE mimo komunikace obstarává i paralelizaci jednotlivých agentů a jejich chování. Každý agent tak vykonává své Behaviour moduly 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.

### 7.1.2 FIPA ACL Message

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[7]. ACL Message je zpráva, která obsahuje množinu jednoho a více předem defi-

novaných parametrů. Které parametry budou použity se liší podle konkrétní situace a ne všechny musejí být validní pro konkrétní systém. Kompletní seznam parametrů ACL zprávy je zobrazen v tabulce v tabulce 7.1. Jediným povinným parametrem zprávy je položka *performative*, avšak přepokládá se, že ve většině případů budou obsaženy i položky *sender*, *receiver* a *content*. Právě tyto parametry jsou použity pro komunikaci agentů v navrženém frameworku.

| 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 7.1: FIPA ACL Message parametry[2]

### 7.1.3 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.[4] 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.

## 7.2 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.

### 7.2.1 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 Knowledge Base, 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 7.1 a 7.2. 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 7.1: Parametry uzlu Technology Tree

| **Jméno** | **Typ** | **Hodnota** |
|---|---|---|
| name | <NodeName> | Jméno TT uzlu |
| type | <Type> | worker\|unit\|building\|upgrade |
| 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 7.2: Price parametry

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

### 7.2.2 Action List

Je konfigurační struktura obsahující definici akcí, které provádí manažeři. Seznam parametrů akce je uveden v tabulce 7.3. 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 7.3: Parametry Action

| **Jméno** | **Typ** | **Hodnota** |
|---|---|---|
| name | <ActionName> | Jméno akce |
| price | Array<Price> | Kompletní cena akce |
| node | <NodeName> | Jméno vyprodukovaného uzlu |

### 7.2.3 Strategy List

Konfigurace obsahující popis strategií, které mohou být použity. Seznam a popis jednotlivých parametrů lze najít v tabulce 7.4. Každá strategie 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 strategii 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 7.4: Parametry Strategy

| **Jméno** | **Typ** | **Hodnota** |
|---|---|---|
| name | <StrategyName> | Jméno strategie |
| type | <StrategyType> | 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 |

### 7.2.4 Managers History

Z hlediska dlouhodobého učení mezi hrami je nutné uchovávat výsledky jednotlivých her. Ty se používají pro inicializaci učících algoritmů v jednotlivých agentech. Po každé hře je nutné uložit informace získané v jejím průběhu a jaké výsledky akce agentů měly. Například pro strategy managera je třeba ukládat použité strategie a jestli vedly k vítězství nebo prohře. V průběhu hry si tyto informace uchováváme ve struktuře, 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 7.5, 7.6, 7.7 a 7.8. Zisk informací a důvod jejich ukládání je popsán v kapitolách věnujících se popisu jednotlivých manažerů.

Tabulka 7.5: 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 7.6: 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 7.7: 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 7.8: 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 7.9: 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 |

## 7.3 Knowledge Base

Do tohoto objektu jsou každému agentovi ukládány jeho znalosti o hře. Každý agent má svou vlastní Knowledge base, do které jsou vkládány znalosti jež jsou pro něj relevantní. V Knowledge Base (zkr. KB) by se neměly vyskytovat duplicitní informace. To znamená, že pokud přidávaná znalost již v KB je, jen se aktualizují údaje v ní uložené. Přidávání znalostí do KB se děje na základě akcí agentů. Každý agent má množinu znalostí, které ho zajímají, a pokud se ve hře objeví znalost, která do této kategorie spadá, je přidána do KB 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. Knowledge base je ve své podstatě datovou strukturou reprezentující položku Beliefs v BDI modelu agenta. Aby byla pro agenty práce s KB 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.
- 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 reálný objekt ve hře nebo uživatelem určený parametr síly jednotky. Jednotliví agenti si do Knowledge Base 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.

## 7.4 Implementace manažerů

Manažeři jsou hlavní částí frameworku a tvoří základ celého multiagentního systému. Byla vytvořena základní třída Base Agent, která rozšiřuje třídu Agent z frameworku JADE. Tato třída obsahuje obecné funkce, které jsou společné pro všechny agenty, aby mohli plnit roli manažera ve strategické hře. Ostatní agenti z této třídy dědí a rozšiřují ji o specifické funkce a Behaviour moduly, aby mohli plnit roli, pro kterou jsou určeni. Jednotlivé třídy manažerů neobsahují kompletní implementaci své funkčnosti, ale jedná se spíše o interface s předdefinovaným chováním. Pro každého manažera je třeba vytvořit jeho implementaci pro konkrétní hru, ve které budou naprogramovány abstraktní funkce, kter0 jsou v manažerovi definovány. Jedná se převážně o funkce určené k ovládání jednotek nebo získání specifické informace o herním objektu, které nelze vložit do TT nebo Knowledge Base. V této kapitole jsou popsány nejdůležitější aspekty jednotlivých tříd.

### 7.4.1 Base Agent

Jedná se o základní jednotku agentního systému ve frameworku. Definuje několik funkcí jež musejí být vytvořeny v definici manžera, nebo jeho implementaci a poskytuje rozhraní na vytvoření ACL zprávy, pro komunikaci s ostatními agenty.[10] Každý manažer, kromě Strategy managera, v určitou chvíli disponuje nějakými jednotkami a musí být schopen se těchto jednotek vzdát. Proto je nutné implementovat funkce pro příjem a odevzdání jednotek. Tyto funkce nemohou být univerzální, protože každý manažer s jednotkami operuje jiným způsobem. Dále také BaseAgent obsahuje rozhraní pro vytvoření pole akcí, jež může agent provádět společně se svou běžnou funkcí. Pro aktivování tohoto chování je třeba implementovat funkci pro provedení akce z tohoto pole. Poslední funkcí, kterou je třeba implementovat, je funkce onFrame, jež slouží pro synchronizaci agenta s instancí hry. Tato funkce by měla být volána při příchodu eventu o vykreslení obrazovky hry. Slouží k tomu, aby se hře předali příkazy, jenž agent vyhodnotil, že je třeba provést, aby dosáhl svého cíle. Tato funkce je již implementována v definicích manažerů.

### 7.4.2 Strategy Manager

Strategy manager se stará o výběr vhodné strategie pro různé fáze hry a zařizuje, aby ostatní manažeři zvolenou strategii realizovali. Tyto strategie jsou vybírány ze Strategy listu podle typu nepřítele, jeho strategie a současné situace ve hře. Dalším aspektem, který se na výběru strategie může projevit, je složení a aktuální síla vlastní armády. V první fázi hry je vybrána strategie typu "opening". Po dosažení cíle specifikovaného v *targer* parametru strategie je zvolena nová strategie typu "standard", a po splnění i cíle v této strategii, se již volí jen strategie typu "lateGame". Toto chování je implementováno v třídě *StrategyAnalyzeBehaviour*, která rozšiřuje třídu *SimpleBehaviour*, která je třídou frameworku JADE pro implementaci chování agenta. Tento behaviour je prováďen periodicky a v každé iteraci kontroluje, zda se v Knowledge Base agenta neobjevil záznam o existenci objektu uvedeného v parametru *target* právě prováděné strategie.

#### Komunikace s Infrastructure managerem

Po zvolení vhodné strategie je o této strategii informován Infrastructure manager pomocí INFORM zprávy obsahující jméno strategie, podle které si musí naplánovat své akce tak, aby se dostal k cíli co nejefektivněji. Mimo to je také možné v průběhu provádění stra-

tegie zjistit, že mimo akcí uvedených v parametru *buildOrder* je potřeba vykonat nějaké dodatečné akce, a i o těch je možno informovat Infrastructure managera pomocí zprávy INFORM.

### Komunikace s Economic managerem

Strategy manager může mít implementovánu schopnost vypočítat potřebný příjem surovin, aby byla nově zvolená strategie splněna co nejrychleji a pomocí INFORM zprávy informovat Economic managera o této hodnotě. Pro umožnění tohoto chování je třeba implementovat výpočet této hodnoty a následně poslat předdefinovanou INFORM zprávu. Tato hodnota je pak uložena v Economic managerovi a uživatel může její hodnotu používat při rozesílání dělníků k surovinám a rozhodování, kolik worker jednotek potřebuje pro svou činnost.

### Komunikace s Tactical a Recon managerem

Posledními dvěma komunikacemi, které jsou ve Strategy managerovi implementovány, je přiřazování cílů Recon managerovi a Tactical managerovi. Oba typy komunikace probíhají formou INFORM zpráv, kdy Strategy manager vybírá pozice ze své Knowledge base o kterých se domnívá, že by byly vhodné k útoku nebo průzkumu. Pro rozhodování o vhodných pozicích k útoku či obraně si může od Tactical managera pomocí REQUEST zprávy vyžádat informace o složení armády. Na tuto zprávu mu Tactical manager odpoví INFORM zprávou obsahující seznam jednotek, kterými disponuje. Formát všech zmíněných zpráv je uveden v tabulce 7.10.

### Historie použitých strategií

Při výběru strategií pomáhá Strategy managerovi i historie jeho předchozích rozhodnutí z předešlých her. Aby tato funkce mohla být implementována je nutné, aby bylo ukládáno jaké strategie byly použity a jejich výsledky. Pokaždé, když je zvolena nějaká strategie, si Strategy manager poznamená, že se tak stalo. Na konci hry jsou do Managers History uloženy záznamy o všech strategiích jež byly použity. Musí být poznamenáno proti jakému typu nepřítele byla strategie použita a zvýšen počet her ve kterých byla zahrána, popřípadě i počet vítězství, které díky ní bylo dosaženo.

Tabulka 7.10: Zprávy odesílány ze Strategy Managera

| **Perfomative** | **Content** | **Receiver** |
|---|---|---|
| REQUEST | armyInfo | Tactical Manager |
| INFORM | newStrategy-<StrategyName> | Infrastructure Manager |
| INFORM | plannedActions-<ActionName>;<ActionName>... | Infrastructure Manager |
| INFORM | income-<Value> | Economic Manager |
| INFORM | reconTargets-<X>-<Y> | Recon Manager |
| INFORM | target-defend-<X>-<Y> | Tactic Manager |
| INFORM | target-attack-<X>-<Y> | Tactic Manager |
| INFORM | noActionRequired | Tactic Manager |

### 7.4.3 Economic Manager

Úkolem Economic managera je zajišťovat stálý příjem surovin a přidělovat je Infrastructure managerovi. Mimo to také monitoruje průměrný příjem surovin, který je na konci uložen do Managers history. Economic manager má výhradní kontrolu nad jednotkami určenými pro těžbu surovin, což jsou ve většině případů i jednotky určené ke stavbě budov. Z toho důvodu bylo nutné vytvořit komunikaci, kterou se budou Economic manager a Strategy manager domlouvat na předávání jednotek. Také je nutné vytvořit komunikační protokoly k rezervaci a uvolnění surovin.

#### Získávání jednotek

Předávání jednotek je implementováno tak, že worker jednotky jsou nepřetržitě nabízeny Economic Managerovi, který je přijímá zasláním zprávy ACCEPT_PROPOSAL. Po přijmutí jednotky se tento dělník předá Economic managerovi a je poslán těžit prioritní surovinu. To je realizováno virtuální metodou, kterou musí uživatel frameworku implementovat. 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 workerů pomocí INFORM zprávy.

Tato komunikace je implementována ve třídě *EconomicMessageBehaviour* a jedná se o rozšíření *SimpleBehaviour* třídy frameworku JADE. Detailnější popis těchto zpráv se nachází v tabulkách 7.11 a 7.12.

#### Rezervace surovin a dělníků

Další komunikace,která musela být implementována, je příjem a odesílání odpovědí na REQUEST zprávy Infrastructure managera, které žádají o přidělení zdrojů, nebo workerů. Po přijetí REQUEST 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 CONFIRM zprávu v případě, že zdroje bylo možné zarezervovat, nebo REFUSE zprávu pokud není dostatek zdrojů na vyřízení této žádosti. Obdobná komunikace probíhá i při žádosti o workera, kde se Economic manager vzdává jednoho ze svého workerů, pokud uzná, že tím nebude výrazně narušen jeho přísun prioritních surovin.

Tato komunikace je také implementována ve třídě *EconomicMessageBehaviour*.

#### Historie průměrných příjmů

Poslední činností Economic managera je průběžné ukládání průměrného příbytku zdrojů v čase. Pro tuto funkcionalitu musí být implementována abstraktní metoda pro získání herního času, která je schopna vrátit současný čas i čas posunutý o zadanou hodnotu. Economic manager si poté v každém framu uloží aktuální hodnotu zdrojů a po uběhnutí určitého počtu framů pak z těchto hodnot vypočítá průměrný příjem, který je uložen do seznamu. Na konci hry jsou z tohoto seznamu vypočítány průměrné, minimální a maximální příjmy surovin za celou hru a společně s počtem dělníků, které měl k dispozici jsou uloženy do Managers history. Toto chování je implementováno ve třídě *EconomicIncomeAnalyzingBehaviour*, což je opět rozšíření třídy *SimpleBehaviour*.

Tabulka 7.11: 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 |

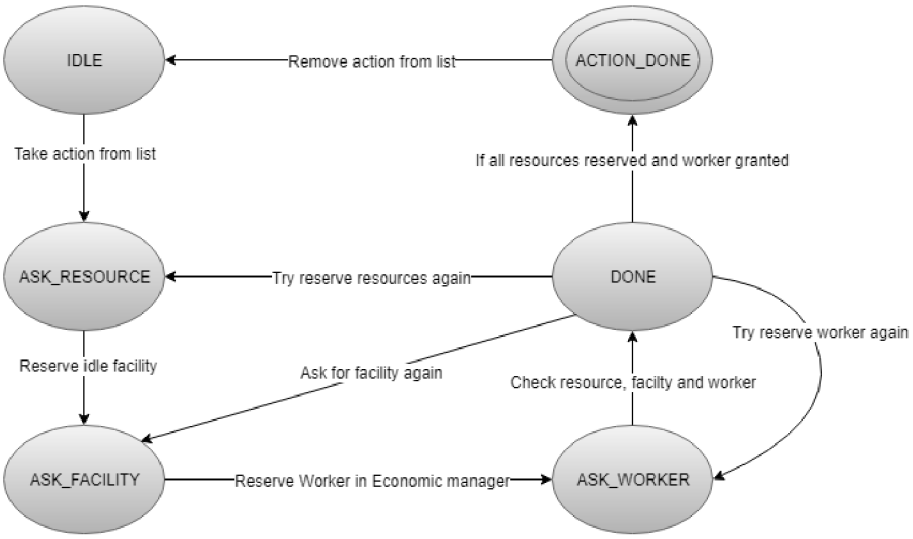
### 7.4.4 Infrastructure Manager

Tento manažer se stará o výstavbu budov, produkci jednotek a vytváření vylepšení. Tuto činnost koná podle strategie, kterou mu nastavuje Strategy Manager, a využívá k jejímu plnění zdroje vytěžené Economic managerem. Dále je zodpovědný za předávání vytvořených jednotek příslušným manažerům a příjem mimo plánových požadavku od těchto manažerů. Musí tedy i být schopen své naplánované akce neustále optimalizovat, měnit jejich pořadí a přidávat do nich akce nové, a to podle měnících se požadavků a současného stavu systému.

#### Provádění plánovaných akcí

K plnění své hlavní funkce, kterou je stavění budov a produkce jednotek, využívá stavový automat. Po přijetí strategie je z parametru *buildOrder* vytvořeno pole akcí, které je třeba vykonat. Stavový automat implementovaný v třídě *InfrastructureMainBehaviour*, rozšiřující standardní *SimpleBehaviour* frameworku JADE, postupně odebírá jednotlivé akce z pole a vykonává je. Tento stavový automat je zobrazen na obrázku 7.2. 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 REQUEST zprávu, že by chtěl zarezervovat zdroje pro uzel TT, jenž je výsledkem právě prováděné akce a přejde do stavu ASK_FACILITY. Zde zavolá uživatelem implementovanou funkci, jež zjistí, zda budova, která produkuje právě vytvářený uzel TT, není zrovna obsazená. Pokud není, tak tuto budovu zarezervuje a přejde do stavu ASK_WORKER. Zde zašle další REQUEST zprávu Economic managerovi, kterou ho požádá o vydání stavitele. Tím přechází do stavu DONE, kde čeká na odpovědi od Economic managera. Pokud byl odpověď na některou z REQUEST zpráv REFUSE, automat přechází do příslušného stavu a požádá o tuto akci znova. V případě, že odpověď na všechny žádosti byla CONFIRM, 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 stavvitel vrácen Economic managerovi. Obojího je docíleno INFORM zprávou z Infrastructure managera.

Aby stavový automat plně fungoval, je nutné implementovat funkci pro provedení akce, která je definována v třídě *BaseAgent*, kterou Infrastructure manager rozšiřuje. Tato funkce vytváří namapování akce ve formátu, se kterým pracuje framework, na skutečnou akci ve hře. Také je potřeba implementovat funkci pro zarezervování budovy, pro výrobu jednotky, která je definována v samotném Infrastucture managerovi. Tato funkce by měla být schopná podle aktuálně produkovaného TT uzlu zjistit, kterou budovu je třeba použít a zarezervuje




Obrázek 7.2: Stavový automat zajišťující provádění akcí v Infrastructure managerovi

ji. Také by měla kontrolovat, zda daná budova již něco nevyrábí, aby neprobíhala rezervace zdrojů, které nejsou aktuálně využívány.

### Předávání jednotek

Další funkcí Infrastructure managera, která zde bude popsána je jeho schopnost předávat vyrobené jednotky příslušným managerům. 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 behaviour modulu *InfractructureUnitPassingBehaviour*, který rozšiřuje *SimpleBehaviour*, procházeny, a jsou činěna rozhodnutí o tom, 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.

Pokud byla od Tacical managera přijata zpráva, že potřebuje vojáky, jsou všechny vytvořené bojové jednotky zasílané jemu. Pokud byla přijata zpráva, že Tactical manager nepotřebuje další jednotky, jsou tito vojáci ponecháni v kontrole Infrastructure managera, který je nechává v základně, případně i omezí jejich výrobu.

Formát jednotlivých zpráv zmíněných v této kapitole se nachází v tabulce 7.12.

### Optimalizace naplánovaných akcí

Poslední funkcí, kterou Infrastructure manager obstarává, je upravování listu naplánovaných akcí. To je řešeno genetickými algoritmy, kde je nad polem naplánovaných akcí prováděna evoluce, která počítá fitness funkci podle toho jak efektivně současný plán vede k cíli. Po provedení každé akce vyhodnocuje momentální plán a přidává do něj mimo plánové akce získané z ostatních manažerů, nebo akce uvedené v paramteru *fill*, aktuálně prováděné strategie.

Tabulka 7.12: Zprávy odesílány z Infrastructure Managera

| Perfomative | Content | Receiver |
|---|---|---|
| INFORM | <NodeName>-resourceRelease | Economic Manager |
| INFORM | workerReturn | Economic Manager |
| REQUEST | <NodeName>-resources | Economic Manager |
| REQUEST | worker | Economic Manager |
| PROPOSE | worker | Recon Manager |
| PROPOSE | worker | Economic Manager |
| PROPOSE | fighter | Tactic Manager |

### 7.4.5 Tactical Manager

Úkolem tohoto manažera je starat se organizování jednotek určených k boji a jejich přesouvání k vybraným cílům. Jeho úkolem je volit jednotky pro tyto činnosti a rozhodovat o tom, zda je třeba bránit či útočit. Tactical manager je schopný tyto rozhodnutí činit samostatně, ale tento rozhodovací proces může být omezen Strategy managerem. Jednotky jsou mu pravidelně zasílány Infrastructure managerem a nepřátelské základny pro něj objevuje Recon manager.

#### 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 na základě *power* parametru u uložených znalostí v kategoriích ENEMY->UNITS a ENEMY->BUILDINGS, které má uložené ve své databázi znalostí. 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 INFORM zprávu tohoto typu, tak má Tactical manager zakázáno tento cíl změnit, dokud nedostane novou INFORM zprávu o tom, že krize byla překonána, a Tactical manager může zase plnit

svou roli bez omezení. Kromě těchto zpráv přijímá taka PROPOSE zprávy on Infrastructure managera, pomocí které mu nabízí předání kontroly na jednotkami. Poslední zprávou kterou přijímá je REQUEST zpráva od Strategy managera po které zašle zpět INFORM zprávu obsahující jména všech jednotek, jimiž Tactical manager disponuje. Formát všech zpráv, které tento manažer odesílá, jsou v tabulce 7.13.

Tabulka 7.13: 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 |

### 7.4.6 Recon Manager

Recon manager má na starosti průzkum mapy a získávání aktuálních informací o mapě. K tomu používá průzkumné jednotky, jež jsou mu přidělovány Infrastructure managerem. Jeho hlavním úkolem je co nejdříve nalézt nepřátelskou základnu a předat tuto znalost Strategy managerovi. Po nalezení nepřátelské základny, se jeho priorita mění na průzkum klíčových oblastí mezi základnou hráče a nepřítele. Většinou bude prozkoumávat nejvhodnější místa k expanzi, nebo zdroje prémiových surovin. Posledním úkolem Recon managera je měřit čas za který byla nepřátelská základna objevena a uložit ho do Managers History.

Aby mohl Recon manager začít plnit svou roli musí si zažádat o průzkumné jednotky. To je provedeno zasláním REQUEST zprávy Infrastructure managerovi. Ten jakmile dostane jednotku vhodnou k prozkoumávání mapy, zašle PROPOSE zprávu, kterou nabídne Recon managerovi průzkumnou jednotku. Recon manager zašle ACCEPT_PROPOSAL zprávu a po přijetí zprávy je mu navrhovaná jednotka přidělena. Stejně jako ostaní manažeři může i Recon manager zaslat INFORM zprávu, kterou zažádá Infrastructure managera o specifickou jednotku. Toto chování je implementováno ve Behavior třídách *ReconMainBehaviour* a *ReconMainBehaviour*. Detailní popis zpráv je v tabulkách 7.14 a 7.12.

Ke správné funkci Recon managera je nutné, aby uživatel frameworku implementoval funkce pro nalezení nepřátelské základny a sekundární průzkum po nalezení nepřítele. Agent také podporuje implementaci více metod prohledávání mapy. Recon manager pak ukládá pro každou z těchto metod časy, kdy byla nalezena nepřátelská základna do Managers History, a používá tyto časy k výběru metody s nejlepšími výsledky.

Tabulka 7.14: Zprávy odesílány z Recon Managera

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

# Kapitola 8

# Testovací model

V této kapitole je popsán testovací model jež byl vytvořen pomocí implementovaného frameworku. Vytvořený model je jednoduchý agentní systém ovládající hru StarCraft (1998) vydanou společností Blizzard Entertainment. Tato hra byla vybrána kvůli tomu, že je dostatečně komplexní na to, aby se zde plně projevily vlastnosti vytvořeného frameworku, a kvalitnímu API, které lze snadno využít k napojení vytvořeného agentního systému na samotnou hru. V této kapitole jsou uvedeny některé důležité informace o hře, o API, které slouží k jejímu ovládání, a na konci kapitoly je uveden stručný popis samotné implementace včetně popisu konfiguračních souborů.

## 8.1 Starcraft

StarCraft je science fiction RTS,z roku 1998 vydaná společností Blizzard Entertainment. V této hře se hráč ujme kontroly nad jednou ze tří hratelných ras. Hratelné rasy jsou Terrani, lidé vyhoštění ze Země, Zergové, insectoidní rasa prahnoucí po genetické dokonalosti a Protossové, humanoidní, vysoce technologicky pokročilá rasa s psionickými schopnostmi. Každá z těchto ras má specifické výhody a nevýhody, které se projevují na tom, jakým směrem se mohou ubírat jejich strategie. V této hře všechny rasy pracují se dvěma zdroji. Prvním z nich jsou minerály, které jsou těženy dělníky z velkých modrých krystalů rozmístěnými po mapě. Z tohoto zdroje jsou stavěny všechny základní budovy, dělníci a první bojové jednotky. Druhým zdrojem, který se ve hře vyskytuje, je vespene plyn. Pro těžení této suroviny je třeba postavit speciální budovu na vespene gejzíru a pridělit dělníky, kteří z ní budou plyn těžit. Tento zdroj je používaný k vytváření vylepšení pro jednotky, stavění pokročilejších budov a produkci lepších jednotek.

- Zerg – soustředí se na rychlou produkci jednotek, způsobujících velké poškození, ale mají slabou obranu. Vyžadují velkou kontrolu mapy a multitasking.
- Protoss – Soustředí se na přístup tzv. "hrubé síly". Mají velmi silné jednotky, které excelují v obraně i útoku. Nevýhodou je, že jejich jednotky jsou drahé a produkce trvá dlouho.
- Terran – Jsou dobří ve všem, ale v ničem nejsou nejlepší. Spoléhají na svou dobrou obranu v prvotní části hry díky bunkrům a relativně silným počátečním jednotkám.

## 8.2 BWAPI

Brood War Application Programming Interface (BWAPI) je open-source framework určený k ovládání hry StarCraft a jejího rozšíření Broodwar. S jeho použitím je možné vytvořit AI programy, které jsou schopné tuto hru samostatně hrát. BWAPI ve výchozím nastavení odhaluje pouze informace, které jsou viditelné hráči. Takže například pozice nebo zdraví jednotek, které zmizeli ve Fog of War, přestávají v tu chvíli být dostupné. Framework také může blokovat jiné uživatelské příkazy, než ty vydané vytvořeným programem. Toto výchozí chování může být změněno, ale pro potřeby tohoto projektu tak nebylo učiněno. BWAPI poskytuje většinu funkcí, které vyvíjený framework vyžaduje. Jsou jimi například událostí informující o vytvoření objektu nebo jeho objevení na mapě, které pomáhají ukládat agentům znalosti do Knowledge base nebo informují Infrastructure managera o tom, že byla stavební akce započata. Takové informace poskytují požadovanou zpětnou vazbu, které vyvinutý framework od systému hry požaduje.

## 8.3 Implementace testovacího modelu

Model byl vytvořen v jazyce Java s použitím BWAPI a frameworku popsaného v této práci. Testovací model představuje agentní systém hrající hru StarCraft: Brood War za rasu Protoss. Byly vytvořeny konfigurační soubory potřebné pro framework a implementace jednotlivých manažerů. Následuje popis těchto konfiguračních souborů a činnosti agentů v této konkrétní hře.

### 8.3.1 Konfigurační soubory

Vytvořené konfigurační soubory jsou napsány ve formátu JSON a jsou to pole objektů popsaných v kapitole 7.2. Zde je popsáno jaké hodnoty byly zvoleny pro jednotlivé parametry těchto objektů a důvody této volby.

Uzly TT jsou rozděleny do čtyř typů. Těmito typy jsou "worker", "unit", "building", "upgrade". Jako jména jednotlivých uzlů jsou uvedeny řetězcové reprezentace typů jednotek, které používá BWAPI. Všechny jednotky jsou reprezentovány prefixem "Protoss"odděleného podtržítkem od jména jednotky. Typ "worker"má pouze jednotka "Protoss_Probe". V parametru *price* je kromě ceny v minerálech a vespene plynu také uveden čas v sekundách, který zabere objekt vytvořit, a v případě jednotek i parametr "supply", který udává jaké množství populace jednotka zabírá.[6] Parametry šířky a výšky objektu jsou uvedeny v pixelech, protože tato reprezentace velikosti byla nejsnáze dostupná.[5]

Action list a Strategies list mají standardní podobu, tak jak byla uvedena v kapitole 7.2, s jediným rozdílem – že do parametru *price* nejsou započítávány žádné dodatečné položky.

### 8.3.2 Implementace propojení se hrou

Jak bylo řečeno v kapitole 7.4, každému manažerovi je třeba implementovat jisté funkce a určité metody manažerů musejí být ve vhodnou chvíli volány, aby agent mohl plnit svou činnost. Zde je popsáno proč se tyto funkce manažerů volají, popřípadě jak je zařízeno vhodné volání potřebných metod vytvořeného agentního systému. Popis metod, které je třeba jednotlivým manažerům implementovat, je uveden v manuálu, který se nachází v příloze.

### Inicializace a napojení frameworku

BWAPI obsahuje třídu *DefaultBWListener*, která slouží k zachycení nejrůznějších událostí z probíhající hry. Jedná se například o události *onStart*, která se volá na začátku hry, nebo událost *onFrame*, která se volá při vykreslování každého framu hry. Rozšíření této třídy slouží jako vstupní a výstupní bod, přes který agentní systém komunikuje s hrou. Při zaslání události *onStart* jsou vytvořeny instance všech tříd potřebných pro start agentního systému. Jsou zde vytvořeny objekty obsahující konfigurace, jednotlivé implementace manažerů a je zde vytvořen *AgentContainer* z frameworku JADE, do kterého jsou agenti vloženi a následně spuštěni. Po této počáteční inicializaci je při každé příchozí události *onFrame* zavolána u každého agenta jeho vlastní *onFrame()* metoda, co předá hře příkazy, které agent vyhodnotil, že je třeba vykonat. Jedná se o jediný vstupní bod, kterým může framework hře předávat instrukce.

### Práce s Knowledge Base

K práci s Knowledge Base slouží události *onUnitDestroy*, která přichází ve chvíli, kdy byla jednotka nebo budova zničena, a *onUnitDiscover*, která přichází kdykoliv nějaká jednotka či budova přestane být kryta Fog of War. Je to právě událost *onUnitDiscover*, která zajišťuje, aby byly agentům vkládány znalosti do Knowledge base. Při každé události tohoto typu je vytvořen objekt reprezentující znalost příchozího objektu a je vložen do Knowledge base agenta. Při příchodu události *onUnitDestroy* jsou pak ve všech agentech nalezeny znalosti popisující zničený objekt a ten je pak ze všech Knowledge base odstraněn. Vzhledem k tomu, že vytvořené objekty ve skutečnosti nezanikají, je zde nutné u agentů, kteří mohou disponovat jednotkami, zavolat metodu *unitDestroyed(Object destroyedUnit)*, která tento objekt odstraní z jeho interních seznamů dostupných jednotek.

### Práce s akcemi

K provádění akcí je nutné disponovat informacemi o tom, že byla dokončena akce Infrastructure managera. Ve chvíli, kdy je akce započata a je možno uvolnit všechny zdroje, je třeba zavolat metodu *actionStarted(String actionName)*. Tato metoda oznámí Infrastructure managerovi, že prováděna akce byl započata, dělník může být uvolněn a zarezervované zdroje byly využity. To kdy se daná metoda má zavolat se bude ve hře StarCraft měnit podle rasy, za kterou je hráno, protože každá rasa ve hře má trochu jiný systém stavění. Pro Protossy je třeba tuto metodu volat při příchodu události *onUnitCreate*.

# Kapitola 9

# Zhodnocení práce

V této kapitole je zhodnocena implementace frameworku a výsledky dosažené testovacím modelem. Mezi hodnocené prvky patří například schopnost hrát úplnou hru, vyhrávat hry, přínosy agentního přístupu a efektivnost komunikačních protokolů.

Dále vzhledem k tomu, že framework se snaží o implementaci obecného chování nutného k ovládání strategické hry, je nutné i vyhodnotit náročnost napojení na hru a množství akcí, které musí uživatel frameworku vykonat, aby tak učinil.

## 9.1 Hraní hry

Framework by měl poskytovat nástroje k tomu, aby s jeho pomocí šlo vytvořit program schopný hrát úplnou hru a také vyhrát. To znamená, že by mělo být možné provést všechny akce dostupné hráči a s jejich pomocí dosáhnout vítězství.

### 9.1.1 Schopnost hrát úplnou hru

Pro hraní úplné hry je třeba plně nakonfigurovat Technology Tree, který je popsán v kapitole 7.2.1. Dále je pro každý uzel nutné vytvořit akci, která tento uzel vyprodukuje (kapitola 7.2.2). Následně je třeba vytvořit mapování těchto akcí na konkrétní funkce, které se musí zavolat, aby daná činnost byla vykonána ve hře. Posledním krokem je vytvoření strategií, které budou tyto akce obsahovat. Pokud jsou tyto kroky uskutečněny, tak je framework schopný vyprodukovat a postavit všechny objekty ve hře.

Framework neposkytuje nástroje pro ovládání speciálních jednotek, jež mají i jiné schopnosti, kromě běžného útoku. Uživatel frameworku si však může používání těchto schopností doplnit do implementace Tactical managera, a to bez zásahu do vnitřní struktury frameworku.

Pokud by se jednalo o hru, která jednotky tohoto typu obsahuje, tak framework momentálně nepodporuje hraní úplné hry. Pokud takové jednotky hra neobsahuje, tak pokud jsou doplněny všechny náležitosti, tak je framework schopný hrát úplnou hru.

### 9.1.2 Schopnost vyhrát hru

V aktuální podobě je systém, vytvořený pomocí implementovaného frameworku, schopný vyhrát hru proti počítači či nezkušenému hráči. Jeho schopnost zvítězit ve hře však velmi závisí na konkrétní implementaci jednotlivých manažerů pro danou hru a kvalitě vytvořených strategií v konfiguračním souboru popsanému v kapitole 7.2.3.

Framework je schopný poměrně dobře zvládat produkci jednotek a budov, těžení surovin a průzkum mapy. Jeho největším úskalím je ovládání jednotek určených k boji. Není problém předávat Tactical managerovi jednotky a cíle k útoku nebo obraně, ale je velmi obtížné vytvořit obecný systém pro ovládání jednotek. Výchozí nastavení Tactical managera je schopno útočit a stahovat jednotky, ale je zde velký prostor ke zlepšení. Problém je, že tyto vylepšení by se mohly ukázat jako výhodná pouze pro implementovaný testovací model, určený pro hraní hry StarCraft. V rámci této práce však nebylo možné tuto teorii ověřit, vzhledem k náročnosti vytvoření více takových testovacích modelů.

Závěrem tedy je, že framework dokáže vyhrát hru, ale závisí na konkrétní implementaci a na hře, na jakou bude framework napojen a na kvalitě strategií, které budou frameworku poskytnuty.

## 9.2 Přínosy agentního přístupu

Použití agentního přístupu přineslo větší míru abstrakce a možnost asynchronního vyhodnocování informací nebo plánování dalších akcí. Abstrakce nemá přímo vliv na to jak řízení hry probíhá, ale usnadňuje práci vývojáře oddělením logických celků a možností dělat úpravy lokalizované jen pro určitou část řízení hry. Paralelismus zase, pokud máme k dispozici vícevláknový procesor, zvyšuje rychlost výpočtů pro řízení hry a snižuje tak počet operací, které je třeba udělat v hlavním vlákně během vykreslení jednoho framu.

### 9.2.1 Abstrakce a komunikace

Použitím agentního systému jsme dosáhli velké míry abstrakce. Každý manažer odpovídá konkrétnímu logickému celku hry, a tak je odděleno například řízení boje od stavění budov. To umožňuje provádět interní změny v jednotlivých agentech, které neovlivní zbytek systému, pokud nebude narušena definovaná komunikace. To znamená, že můžeme kompletně změnit způsob jakým např. Tactical manager řídí jemu přidělené jednotky, pokud zůstane zachováno přijímání zpráv a korektní odpovědi na zprávy z Infrastructure managera a Strategy managera.

Díky agentnímu přístupu jsme taky schopní rozšiřovat komunikační schopnosti jednotlivých agentů. Můžeme vytvořit nové zprávy a definovat chování pro přijímaní této zprávy v ostatních manažerech. To umožňuje delegování problémů na jiné manažery, místo aby se museli řešit interně v agentovi, který může na takový problém narazit, ale neměl by ho řešit. Komunikace také dále rozšiřuje možnosti paralelismu, které jsou posány v další kapitole.

### 9.2.2 Paralelismus

Další výhodou jež agentní přístup přináší je paralelismus činností agentů. Tuto funkci zajišťuje framework JADE. Je možno vytvořit několik agentních chování, kde se každé bude starat o určitou část problematiky, kterou daný manažer řeší. Můžeme tak například vytvořit chování pro příjem zpráv, analýzu Knowledge Base, nebo řízení jednotek. Tyto chování se vykonávají buď v reakci na zprávu, nebo periodicky. Tento přístup také přispívá k rozšiřitelnosti frameworku, protože tyto chování jdou do agenta jednoduše přidat.

## 9.3 Obtížnost napojení frameworku

V kapitole 7 a manuálu v příloze této práce jsou popsány kroky jež je třeba učinit k napojení frameworku na konkrétní hru. Jak je z těchto dokumentů vidět, je třeba mnoho vstupů od programátora ke zprovoznění navrženého frameworku.

Množství režie k tomu, aby framework fungoval správně a uživatel ho byl schopný používat je mnohem větší, než bylo původně předpokládáno. I přesto, že ve frameworku chybí implementace některých vlastností, které se mohou ve strategické hře vyskytovat, jako je například ovládání jednotek se speciálními schopnostmi, tak množství věcí, které musí uživatel implementovat je příliš velké.

Snaha vytvořit obecný framework pro všechny strategie vytváří spoustu neznámých faktorů a zvyšuje množství objektů a parametrů, které musí uživatel konfigurovat. Tato skutečnost se nejvíce projevuje v Tactical managerovi, který již teď obsahuje spoustu metod, které je třeba implementovat pro konkrétní hru a to i přesto, že není připraven na všechny nároky, které by na něj mohli být kladeny v některých strategických hrách.

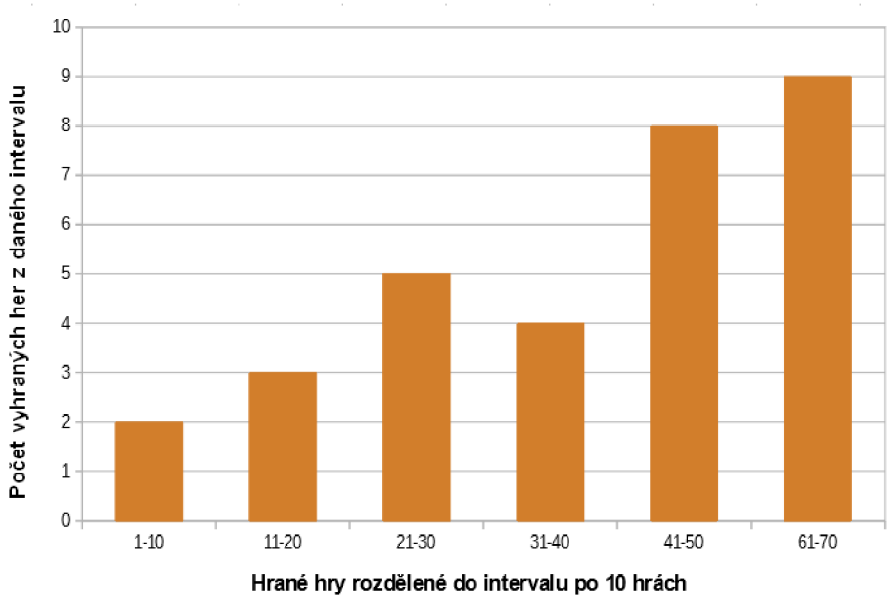
## 9.4 Zhodnocení testovacího modelu

Vytvořený testovací model byl opakovaně pouštěn proti základní umělé inteligenci ve hře Starcraft a několikrát byl otestován i proti lidským hráčům. Během her vykazoval znaky reaktivnosti tím, že například stahoval jednotky, ve chvíli kdy čelil přesile. Systém taky projevoval cílem řízené chování, ve chvíli, kdy například Infrastructure manager měl strategií zadaný cíl a za pomocí informací o hře, které měl k dispozici upravoval své pole akcí, aby se k tomu to cíli dostal efektivněji.

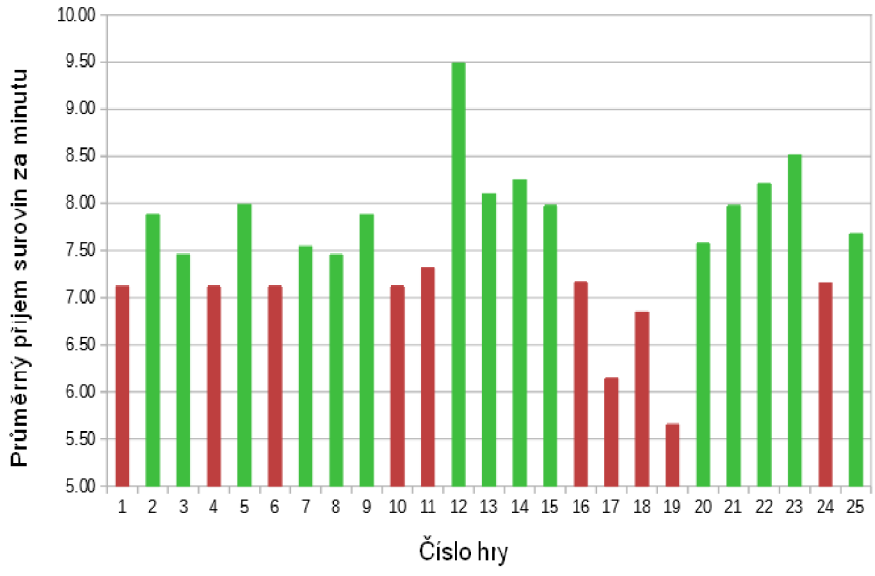
Systém byl schopný porážet ostatní umělé protivníky a méně zkušené hráče Starcraftu. Systém vykazoval velmi dobré výsledky proti počítači, protože počet strategií, které má základní AI ve Stracraftu nadefinované, je značně omezený. Testovací model měl nadefinováno několik strategií, které jsou proti těmto strategiím velmi účinné, takže se relativně rychle naučil vybírat si vhodnou strategii, podle typu soupeře. Výsledky tohoto testování jsou zobrazeny v obrázku 9.1.

Proti lidským hráčům byla jeho úspěšnost menší, protože ani testovací model neměl definováno příliš mnoho strategií a tak jeho možnosti se učit neměli natolik velký rozsah, aby se mohli vyrovnat hráči. Také bylo z časových důvodů náročné uskutečnit dostatek her proti lidským oponentům, aby se mohlo učení nějak výrazněji projevit.

Dalším zajímavým hodnocením úspěšnosti testovacího modelu bylo sledovat závislost průměrného příjmu surovin a šancí na výhru. Na obrázku 9.2 je zobrazen průměrný příjem za minutu herního času pro 25 her. Zelenou barvou jsou v grafu označeny vítězné hry a červenou prohry. Graf je vytvořen podle příjmu minerálů, které jsou hlavní produkční surovina ve Starcraftu. Staví se z nich první budovy a základní jednotky. Všechny testovací hry byly prováděny vůči stejné strategii nepřítele, aby bylo docíleno co nejmenší odchylky. Můžeme vidět, že v tomto konkrétním případu jistá závislost mezi příjmem surovin a vítězstvím opravdu existuje. Lze zde vidět, že pokud systém dosáhl průměrného příjmu přesahujícího 7,25 minerálů za minutu, tak hra končila vítězstvím.




Obrázek 9.1: Zobrazení zlepšení pravděpodobnosti vítězství, pomocí úpravy výběru strategie na základu předchozí úspěšnosti.




Obrázek 9.2: Zobrazení průměrných příjmu za minutu pro 25 her. Zelená barva označuje vyhrané hry a červená prohrané.

# Kapitola 10

# Závěr

Tato práce popisuje návrh a implementaci obecného frameworku, který umožní vytvářet počítačem řízené protivníky do strategických her. Tito protivníci jsou vytvořeni jako agentní systémy, které jsou schopny se přizpůsobovat situacím ve hře a učit se z her předchozích.

Navržený systém obsahuje definici pětice agentů, kteří jsou schopni ovládat jednotlivé aspekty strategické hry. Konkrétně se jedná o výběr strategie, stavění budov, těžení zdrojů, ovládání bojových jednotek a průzkum mapy. Tito agenti jsou ve frameworku nazýváni manažeři a jsou schopni spolupráce díky navrženým komunikačním protokolům, které umožňují předávat si jednotky, strategie, pozice nepřátel a další informace potřebné k řízení hry.

Pro správnou funkci systému je nutné implementovat jisté metody těchto agentů. Jedná se především o metody, které poskytují informace o herních objektech, nebo zadávají příkazy specifické pro hru. Dále je nutné, aby byl poskytnut popis objektů ve hře, akcí, které s nimi lze provádět a strategií určených pro danou hru.

Pomocí implementovaného frameworku byl vytvořen testovací model, pro hru Starcraft od společnosti Blizzard Entertainment. Tento testovací model je schopen hrát za jednu z hratelných ras, konkrétně se jedná o tzv. Protossy a slouží k ověření funkčnosti navrženého frameworku. Model je schopný ovládat všechny objekty, které jsou definovány v jeho konfiguraci a je schopný vyhrávat nad základní umělou inteligencí, která ovládá protivníky ve hře.

Všechny cíle práce byly splněny. Při tvorbě frameworku byl kladen velký důraz na jeho univerzálnost a rozšiřitelnost. V původním návrhu také mělo být velmi jednoduché ho napojit na jakoukoliv strategickou hru, ale tato vlastnost se ukázala být implementačně velmi obtížná. Framework je stále relativně jednoduchý na použití, ale uživatel musí implementovat více částí agentů, než se původně předpokládalo. Na druhou stranu je framework mnohem univerzálnější a rozšiřitelnější, než bylo při návrhu zamýšleno. Uživatel může jednoduše rozšířit funkce jednotlivých manažerů, nebo jim i přidat úplně nové chování s tím, že to příliš neovlivní jejich základní implementaci a běžné činnosti, které provádí. Tyto rozšíření by však byly užitečné většinou jen pro konkrétní hru, takže nedává příliš smysl zahrnovat je v základním frameworku.

Možným pokračováním práce by mohla být úprava frameworku pro konkrétní typ strategických her a tím docílit možného odstranění jedné z největších vad systému, kterou je vyšší náročnost napojení frameworku na hru.

# Literatura


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

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

*[3] JAVA Agent DEvelopment Framework. [Online; navštíveno 15.03.2018].*
*URL http://jade.tilab.com/*

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

*[5] liquipedia. [Online; navštíveno 15.03.2018].*
*URL*
*https://liquipedia.net/starcraft/List_of_Unit_and_Building_Sizes*

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

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

*[8] 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*

*[9] 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*

*[10] Jean Vaucher, A. N.: JADE Tutorial and Primer. [Online; navštíveno 15.03.2018].*
*URL*
*https://www.iro.umontreal.ca/~vaucher/Agents/Jade/JadePrimer.html*

*[11] Josh MCCoy, M. 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*

*[12] Matt Arvin Villas Cabanag, D. R., Michael Hitchens: 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

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

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

*[15] Zbořil, F.: Podklady k předmětu AGS - III. Formální logiky pro agentní a multiagentní systémy. [Online; navštíveno 10.01.2018].*
*URL* https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FAGS-IT%2Flectures%2FAGS15_03.pdf&cid=10764

*[16] 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 modlu 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

# Manuál

Pro použití frameworku je třeba vytvořit implementace jednotlivých manažerů, které budou převádět pokyny z agentního systému na příkazy konkrétní strategické hry. Dále je třeba vytvořit nebo upravit konfigurační soubory, které těmto manažerům slouží k orientaci ve hře a kategorizují jim jednotlivé její části do podoby, které rozumí. Následuje popis těchto konfiguračních souborů. Aa nimi je uveden seznam manažerů a u nich popis abstraktních metod, které je třeba implementovat.

Konfigurační soubory:

- TT.json - Technology Tree

  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 strategiích. 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 |
| 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": "time",
        "value": "20"
      },
      {
        "name": "supply",
        "value": "1"
      }
    ],
    "requires": [
        "Protoss_Nexus"
    ],
    "unlocks": []
  }
]
```

- Actions.json - Action List

  Do tohoto souboru je třeba vložit všechny akce, kterých budou manažeři schopni. Následně je třeba v implementaci manažerů, kteří by měli tyto akce používat, 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 řpes 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 |
| 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",
    "price": [
      {
        "name": "minerals",
        "value": "50"
      },
      {
        "name": "time",
        "value": "20"
      },
      {
        "name": "supply",
        "value": "1"
      }
    ],
    "node": "Protoss_Probe"
  }
]
```

- Strategie.json - Strategies List

  Do tohoto konfiguračního souboru je třeba vložit všechny strategie se kterými může framework pracovat. Tyto strategie jsou používány Strategy managerem k naplánování prvních akcí Infrastructure managerovi. Strategy manager se učí tyto nadefinované strategie používat a vybírá ty s největší úspěšností proti současnému protivníkovi. Je doporučeno tedy nadefinovat co nejvíce možných strategií, pro různé části hry. I strategie jsou uloženy v souboru ve formátu JSON, jako pole objektů typu Strategy, jehož popis je v následující tabulce.

Tabulka B.4: Parametry Strategy

| **Jméno** | **Typ** | **Hodnota** |
|---|---|---|
| name | <StrategyName> | Jméno strategie |
| type | <StrategyType> | 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 Strategies List, obsahující strategií "zealot rush".

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

Implementace prvků frameworku:

Každému manažerovi je třeba implementovat několik abstraktních metod, které potřebuje pro korektní provádění své činnosti. Vzhledem k tomu, že všichni agenti jsou založeni na frameworku JADE, je možné jim přidávat nové `Behaviour` moduly, a tím pádem i novou komunikaci a akce, které tito agenti mohou provádět. Níže jsou popsány metody, které je nutné jednotlivým manažerům implementovat a co tyto metody musejí vykonávat, aby bylo zajištěno předpokládané chování agentního systému v tomto frameworku.

- Strategy manager

  Strategy manager nepotřebuje implementovat žádné metody, ale uživatel se může rozhodnout přepsat jeho metody pro výběr strategie pro různé části hry. Konkrétně se jedná o:

  - `String selectOpeningStrategy(List<String> strategies)` – Metoda pro výběr první strategie. Na vstupu má seznam jmen dostupných strategií. Výstupem je jméno vybrané strategie.
  - `String selectStandardStrategy(List<String> strategies)` – Metoda pro výběr strategie, která je použita po splnění cíle počáteční strategie. Na vstupu má seznam jmen dostupných strategií. Výstupem je jméno vybrané strategie.
  - `String selectLateGameStrategy(List<String> strategies)` – Metoda pro výběr strategie, která je použita v závěrečných fázích hry. Ve chvíli, kdy je zvolena jedna z těchto strategií, může už znova být vybrána jen strategie určená pro pozdější fáze hry. Na vstupu má seznam jmen dostupných strategií. Výstupem je jméno vybrané strategie.

- Economic manager

  - `int checkResourceAvailability(String resourceName)` – Této metodě je zadáno jméno suroviny, která se ve hře vyskytuje, a musí být vrácena její aktuální hodnota. Jména těchto surovin jsou čerpána z parametru *name* objektu `Price` konfiguračních souborů.
  - `int getGameTime(int time)` – Metoda vrátí aktuální herní čas. Musí být schopna vrátit i čas posunutý o hodnotu `time`. Hodnota `time` je zadávána v sekundách. To znamená, že pokud je metoda zavolána například s hodnotou -60, vrátí čas před 60 sekundami.
  - `void transferUnit(Object transferredUnit)` – Metoda přijme jednotku, kategorizuje ji a uloží do příslušného seznamu agenta.
  - `Object freeUnit()` - Metoda si odstraní z některého ze agentových seznamů jednotku a vrátí ji jako návratovou hodnotu.
  - `boolean isWorkerIdle(Object workerUnit)` – Vrátí `TRUE`, když zadaná jednotka zrovna nevykonává žádnou činnost, jinak vrací `FALSE`.
  - `void sendWorkersToWork(List<Object> idleWorkerUnits)` – Metoda rozešle zadané dělníky těžit zdroje. Mělo by zde být rozhodnuto, který dělník půjde těžit který zdroj.

- Infrastructure manager

- `void transferUnit(Object transferredUnit)` – Tato metoda přijímá nově vytvořené jednotky a budovy. Měla by kategorizovat všechny bojové jednotky do seznamu `_fighters`, dělníky do seznamu `_workers` a budovy do seznamu `_productionBuildings`.
- `void onUnitCreate(Object createdUnit)` – Metoda musí zavolat metodu `void actionStarted(String actionName)` Infrastructure managera. Musí této metodě předat jméno akce produkující typ objektu ze vstupu.
- `void runAction()` – Metoda zajistí provedení akce uložené v parametru `_currentAction`.
- `void reserveFacility()` – Metoda zjistí, ve které budově se vyrábí `TechnologyTreeNode` uložený v parametru `_currentNode` a tuto budovu zaregistruje uložením do parametru `_currentFacility`.
- `boolean isIdle(Object facility)` – Zjistí, zda zadaná budova právě produkuje jednotku. Vrací `TRUE`, pokud v této budově není produkována jednotka, jinak `FALSE`.

- Tactical manager
  - `void transferUnit(Object transferredUnit)` – Metoda přijme jednotku a přidá ji do seznamu `_fighters`.
  - `String getUnitName(Object unit)` - Vrací jméno zadané jednotky, které by mělo být stejné jako v Technology Tree
  - `void findChokepoint()` – Najde nejbližší místo vhodné k obraně a uloží ho do parametru `_goalPosition`.
  - `boolean isFarFromChokepoint(Object fighter, Position chokepoint)` – Vrátí, zda je `fighter` příliš vzdálen od `chokepoint`.
  - `void sendToChokepoint(Object fighter, Position chokepoint)` – Zašle `fighter`, na `chokepoint`.
  - `boolean canAttack(Object fighter)` – Zjistí, zda je `fighter` v tuto chvíli schopný zaútočit.
  - `void attackPosition(Object fighter, Position position)` – Zaútočí s `fighter` na `position`.
  - `boolean isUnitCloseToSquad(List<Object> squad, Object unit2, int radius)` - Zjistí, zda `unit2` není od `squad` vzdálena víc jak `radius`.
  - `Position getSquadPosition(List<Object> squad)` – Vrátí současnou pozici jednotek ve `squad`.
  - `boolean sentToAttackPosition(Object fighter, Position goalPosition)` – Zašle `fighter` na útočnou pozici `goalPosition`.
  - `void unitDestroyed(Object destroyedUnit)` – Odstraní `destroyedUnit` ze všech seznamů v Tactical managerovi.
  - `boolean choosePrimaryTarget(List<BasicKnowledge> potentialTargets)` – Zvolí nejvhodnější cíl k útoku z listu `potentialTargets`

- Recon manager

  V Recon mangerovi může být definováno více metod prohledávání. Manažer si pak ukládá do historie časy, kdy byla nalezena nepřátelská základna, kterou metodou a

pak podle těchto údajů vybírá nejvhodnější vyhledávací metodu a poskytuje její jméno v parametru `_searchMethod`.

  - `void transferUnit(Object transferredUnit)` – Metoda přijme jednotku a přidá ji do seznamu `_scouts`.
  - `boolean isScoutActive(Object scoutUnit)` – Metoda zjistí, zda jednotka provádí nějakou akci.
  - `boolean sendScout(Object scoutUnit)` – MEtoda zašle jednotku prozkoumávat startovní lokace na mapě. Vrací `FALSE` pokud již neexistují pozice, které by mohly být prozkoumány.
  - `boolean scoutEnemyBase(Object scoutUnit, Position basePos)` – Metoda pošle jednotku prozkoumávat nejbližší okolí zadané pozice. Vrací `FALSE`, pokud již neexistují pozice, které by mohly být prozkoumány.
  - `double getGameTime()` – Vrátí herní čas.
  - `List<String> getSearchMethods()` – Vrátí jména dostupných metod prohledávání.
  - `void unitDestroyed(Object destroyedUnit)` – Odstraní jednotku ze všech seznamů v Recon managerovi.
  - `void returnScout(Object scoutUnit)` – Vymaže jednotku ze svých seznamů a vrátí ji zasilateli.

- Technology Tree
  - `boolean isNodeComplete(Object node)` – Zjistí, zda je objekt dokončen.
  - `String getSupplyAction()` – Vrátí jméno akce, která zvětšuje populaci.
  - `int getSupplyActionValue()` – Vrátí hodnotu, o kterou akce zvětšuje populaci.

- KnowledgeBase
  - `areUnitsEqual(Object unit1, Object unit2)` - Porovná dva zadané herní objekty a zjistí, zda jsou totožné.

Inicializace frameworku:

Před vytvořením jednotlivých agentů je třeba vytvořit instance následujících singleton tříd:

- TechnologyTree()
- ActionList()
- StrategiesList()
- ManagersHistory()

Poté je třeba vytvořit instance jednotlivých manažerů a vložit je do *AgentContainer* třídy frameworku JADE, pomocí metody *acceptNewAgent(String name, Agent agent)*. Výstupem této metody je *AgentController*, který je třeba spustit zavoláním metody *start()*.

Ukázka incializace:

```
TechnologyTreeImpl tt = new TechnologyTreeImpl();
ActionList al = new ActionList(tt);
StrategiesList sl = new StrategiesList(tt, al);
ManagersHistory mh = new ManagersHistory();

_infrastructureManager = new InfrastructureManagerImpl(30, "infMgr", self, game);
_economicManager = new EconomicManagerImpl(30, "ecoMgr", resources, self, game);
_reconManager = new ReconManagerImpl(30, "recMgr", self, game);
_tacticalManager = new TacticalManagerImpl(30, "tacMgr", self, game);
_strategyManager = new StrategyManagerImpl(30, "strMgr", self, game);

Properties pp = new Properties();
pp.setProperty(Profile.GUI, Boolean.FALSE.toString());
Profile p = new ProfileImpl(pp);
jade.wrapper.AgentContainer ac = Runtime.instance().createMainContainer(p);

AgentController economicController
= ac.acceptNewAgent("economic", _economicManager);
AgentController infrastructureController
= ac.acceptNewAgent("infrastructure", _infrastructureManager);
AgentController reconController
= ac.acceptNewAgent("recon", _reconManager);
AgentController tacticalController
= ac.acceptNewAgent("tactical", _tacticalManager);
AgentController strategyController
= ac.acceptNewAgent("strategy", _strategyManager);

economicController.start();
infrastructureController.start();
reconController.start();
tacticalController.start();
strategyController.start();
```

Napojení frameworku:

Pro plné propojení frameworku se hrou je třeba zavolat určité metody jednotlivých manažerů při konkrétních událostech. Tyto metody jsou, spolu s událostí, při které musí být zavolány, popsány zde.

- `void onFrame()` – Tato funkce musí být zavolána nad každým manažerem pokaždé, když je vykreslován nový snímek hry. Slouží k předání úkolů jednotlivých manažerů hře.
- Následující funkce musí být zavolány nad Managers History objektem na konci každé hry.
  - `void addStrategyRecord(String strategyName, String faction, boolean won)`
  - `void addReconRecord(String reconMethod, double time)`
  - `void addEconomicRecord(HashMap<String,Double> avgIncome, HashMap<String,Double> minIncome, HashMap<String,Double> maxIncome, int workers, boolean won)`
  - `writeToFile()`
- `void unitDestroyed(Object destroyedUnit)` – Nad každým managerem, který má k dispozici jednotku nebo budovu ve chvíli, kdy byl tento objekt zničen.
- `void transferUnit(Object transferredUnit)` – Nad Infrastructure managerem pokaždé, když je vytvořena nová jednotka.
- `void pushKnowledge(BasicKnowledge knowledge)` – Ve chvíli, kdy je objevena nějaká znalost, musí tato metoda být zavolána nad Knowledge Base každého agenta, pro něhož je informace relevantní.
- `void onUnitCreate(Object unit)` – Musí být zavolána nad Infrastructure managerem ve chvíli, kdy byla započata stavba budovy nebo produkce jednotky, a je možné uvolnit všechny zarezervované zdroje, a to včetně dělníka, který na dané akci pracoval.