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. PETR KNAPEK**
AUTHOR

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

BRNO 2018

Abstrakt

Tato práce je věnována návrhu systému schopného tvorby obecných, autonomních ovládacích prvků strategických her založených na agentních systémech s podporou sociálních schopností, inteligentního rozhodování a učení. Budou popsány základní typy strategických her a problémy náročnosti jejich hraní, stejně jako aktuální trendy ve vývoji herních inteligencí. Poté budou prezentovány návrh a implementace výsledného systému, fungující model pro konkrétní hru a výsledky dosažené jeho testováním.

Abstract

This thesis is dedicated to creating a new system with capabilities to create new generic, autonomous strategy computer game controlling elements based on multi-agent systems with social, intelligent decision-making and learning skills. Basic types of strategy games and problems of their playing will be introduced, along with currently used methods of intelligent game AI development. This thesis also presents design and implementation of the new system, working model for a specific game and results obtained while testing it.

Klíčová slova

Multiagentní systémy, strojové učení, učení agentů, strategické hry, počítačové hry, boti, rozhodování agentů, sociální schopnosti agentů, genetické algoritmy

Keywords

Multi agent systems, MAS, machine learning, agent learning, strategic games, computer games, bots, agent decision-making, agent social capabilities, genetic algorithms

Citace

KNAPEK, Petr. *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 vypracoval samostatně pod vedením pana Doc. Ing. Františka Zbořila Ph.D., a na základě znalostí získaných ve spolupráci s kolegou Bc. Lukášem Válkem, který se se mnou podílel na návrhu systému. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Knapek
22. května 2018

Poděkování

Chtěl bych poděkovat Doc. Ing. Františku Zbořilovi Ph.D., vedoucímu mé práce, za vedení, připomínky a čas, který mi věnoval. Dále bych rád poděkoval Bc. Lukáši Válkovi za hladkou a pohodovou spolupráci s návrhem a implementací výsledného systému. Rovněž bych rád poděkoval své rodině za podporu během studia i celého mého života.

Obsah

Úvod

Ačkoliv úplně první počítačové hry byly určeny pro více hráčů, velmi brzo vznikl díky rozšíření arkádových herních automatů zájem o hry pro hráče jednoho. Aby hráče bavily a byly dostatečně obtížné, vývojáři začali s implementací prvních algoritmů reprezentujících oponenty. To však není lehkým úkolem, protože počítačový protivník musel být schopen reagovat na hráčovy akce tak, aby byla hra proti němu výzvou, a navíc vykazovat známku lidského chování kvůli věrohodnosti, že je hra fér. Z těchto důvodů se na hrách začaly uplatňovat poznatky z oboru inteligentních systémů a umělé inteligence. Po čase byly počítače schopny soutěžit s člověkem v logických hrách, mnohdy je díky promyšlenosti algoritmů překonávat.

S postupem času však vývoj umělých inteligencí pro hry upadl. To je způsobeno především pro hráče zajímavějšími možnostmi vytvoření reálnější grafiky, zábavnějšího příběhu a her se zaměřením na soupeření s ostatními hráči skrze internet. Aplikace technologií z oborů strojového učení či autonomního ovládání herních prvků je však stále perspektivním oborem.

Cílem této práce je vytvoření návrhu a implementace jednoduchého, obecného frameworku použitelného k tvorbě herních oponentů pro počítačové strategické hry fungujících na principech multiagentního systému se sociálními schopnostmi, možnostmi adaptace a způsoby učení se z předchozích odehraných her.

Na projektu jsem spolupracoval s Bc. Lukášem Válkem, který se při vývoji zaměřoval na vývoj a implementaci agentního systému a komunikačních protokolů reprezentujících sociální schopnosti agentů, dále pak na návrh rozhraní mezi frameworkem a konkrétní hrou. Mým úkolem pak bylo hlavně navrhnout a vytvořit procesy rozhodování, adaptace a učení agentů, posléze implementace testovacího modelu a ověření funkčnosti celého systému.

V prvních kapitolách, 1 a 2, jsou popsány základní pojmy relevantní pro tuto práci a druhy strategických her s rozborem klíčových vlastností a aspektů pro počítačem ovládaného protivníka.

V kapitole 3 jsou popsány základní přístupy k vytváření herních umělých inteligencí. V kapitole 4 jsou uvedeny metody softcomputingu nejčastěji používané pro tvorbu inteligentních herních protivníků.

V kapitole 5 je předveden návrh obecného systému založeného na zmíněných principech. V následující kapitole, 6 je pak podrobněji popsána implementace onoho systému.

V kapitole 7 jsou představeny obecné algoritmy pro učení a adaptaci agentů vytvořeného systému.

Testovací model vytvořený pro hru StarCraft: Brood War od Blizzard Entertainment je popsán v kapitole 8, výsledky celé práce ověřené jeho testováním pak v kapitole 9.

Kapitola 1

Multiagentní systémy, herní AI

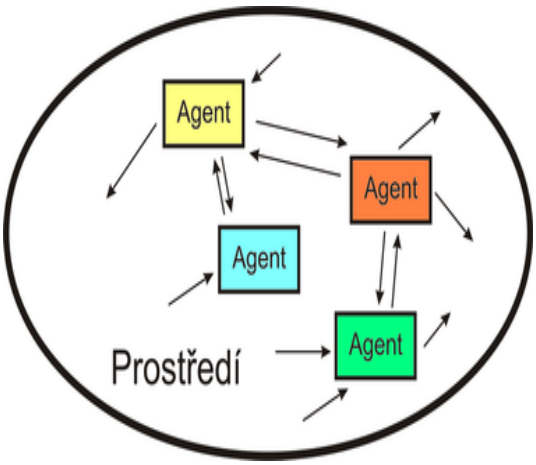
V této kapitole budou pro kontext této práce vysvětleny nejdůležitější pojmy.

1.1 Multiagentní systém (MAS)

Multiagentní [14] systém je simulované prostředí, ve kterém se vyskytují aktéři – agenti, kteří mezi sebou interagují. Ti jsou schopni vnímat prostředí nebo jeho části a svými akcemi jej měnit tak, aby dosáhli svých předem stanovených cílů. Agenti jsou:

- **Autonomní** – jednají samostatně, své jednání mají pod kontrolou.
- **Reaktivní** – reagují na změny prostředí, a to v reálném čase.
- **Proaktivní** – sami vyhledávají způsoby, jak ovlivnit prostředí k dosažení svých cílů.
- **Sociální** – komunikují s ostatními agenty a společně řeší problémy nebo vzájemné konflikty.

Pro agenta je velmi důležitá racionalita – agent si volí své cíle a následně akce s největším užitkem pro systém.



Obrázek 1.1: Systém s několika komunikujícími agenty

1.1.1 Agent podle prostředí

Agenti se dělí i podle prostředí, v jakém operují. Typy prostředí mohou být například:

- **Diskrétní/Spojité** – reálný svět je spojitý, ale počítačové simulace vždy diskrétní.
- **Přístupné/nepřístupné** – agent může vnímat celé prostředí, ve kterém se nachází, nebo jen jeho části.
- **Statické/dynamické** – podle toho, zda se prostředí mění i bez akcí agentů.
- **Deterministické/nedeterministické** – v deterministickém prostředí je jeho následující stav dán pouze současným stavem a akcí agenta.
- **Epizodní/neepizodní** – epizody jsou na sobě nezávislé části běhu systému.

V případě agentů pro strategické hry se vždy budeme bavit o diskrétním, nepřístupném a epizodním prostředí. Podle konkrétní strategické hry pak většinou o dynamickém a nedeterministickém, pakliže bereme v potaz různé vlivy prostředí herní mapy nebo jednotek soupeře.

1.1.2 BDI agenti

Zkratky pochází z anglického *Belief, Desire, Intention* [13]. Jedná se o systém s agenty řízenými záměrem.

- **Beliefs (Představy)** – soubor představ o světě, ve kterém se agent nachází. Představy nemusí být přesné a mohou se měnit.
- **Desires (Přání)** – stavy okolního světa, které by agent chtěl, nebo taky jeho úkoly. Mohou být krátkodobé i dlouhodobé. Dlouhodobé se nazývají také **Goals (Cíle)**. Agent nemusí být schopen cílů dosáhnout, některé se mohou dokonce i vzájemně vylučovat.
- **Intentions (Záměry)** – aktivity, k jejichž realizaci se může agent rozhodnout. Jedná se o přání, ke kterým má agent nějaký druh závazku.

V multiagentním systému může mít více agentů stejná přání a záměry, proto se jim obvykle vyplatí rozdělit si úlohy, a tak díky komunikaci dosáhnout lepších výsledků. Každý agent může být schopen vnímat pouze omezené okolí, a proto mezi sebou agenti sdílí informace.

1.2 Herní AI

V počítačových hrách není AI (*Artificial Intelligence, umělá inteligence*) opravdovou umělou inteligencí v pravém slova smyslu. Jedná se o širší soubor technik a algoritmů z oblastí robotiky, počítačové grafiky, prohledávacích algoritmů a skriptování řízeného spouštěči (*„Triggers“*) využívaných k simulaci lidského chování (nejen) protivníků ovládaných počítačem. Pojem umělé inteligence je spíše marketingovým tahem ze strany vývojářů (počítač hrající jako člověk), nicméně je v tomto kontextu hojně používán, a proto bude používán i v této práci.

Je třeba zmínit fakt, že herní AI nemusí ve hře ovládat pouze protivníka, ale do jisté míry může být zodpovědná i za chování jednotek ovládaných hráčem. Jedná se například o automatické získávání zdrojů, kdy jednotka sama opakuje své akce nebo se přizpůsobuje vyčerpání zdroje a nejde si jiný, automatické opravování hráčových staveb či automatická obrana základny když se nepřítel dostane na dostřel.

V dnešní době není perfektní herní AI pro herní vývojáře primárním cílem. Ti se snaží spíše o realistickou grafiku a modely, zábavnější příběh a multiplayer (hry více hráčů, obvykle online). Lidští oponenti jsou považováni za mnohem větší výzvu kvůli schopnosti intuitivně zkoušet nové taktiky a strategie, navíc s inteligentními herními AI nemá většina hráčů zkušenosti. Umělá inteligence se tedy obvykle dodělává až v posledních fázích vývoje hry a sestává se z jednoduchých skriptů vytvořených na základě pozorování a kopírování akcí z her hraných lidskými hráči.

Algoritmy strojového učení nebo jiné z oblasti inteligentních systémů se v herních AI typicky nevyskytují, avšak existuje snaha skupin fanoušků programování a studentů informačních technologií vytvářet adaptivní a vyvíjející se herní umělé inteligence pro většinou starší počítačové hry jako StarCraft, WarCraft2 apod.

1.2.1 Úplná informace

V terminologii herních umělých inteligencí „úplná informace“ znamená, že počítačem ovládaný hráč má k dispozici analýzu herní mapy. Z dat dodané analýzou je schopen vyčíst například startovní pozice hráčů, zdroje surovin, překážky, mosty a další strategicky významné věci. Úplnou informaci lze přirovnat například ke znalosti šachovnice po jejím prohlédnutí si.

Skutečnost, zda herní umělá inteligence má nebo nemá k dispozici úplnou informaci, má zásadní vliv na její proces získávání informací ve hře. Ve většině strategických her se na ni spoléhá.

Kapitola 2

Strategické hry

V této kapitole budou rozebrány základní principy strategických počítačových her a jejich nejběžnější žánry relevantní pro tuto práci – především Real-time strategie, částečně pak Turn-based strategie. Ke každému pak budou uvedeny základní vlastnosti zkoumané při tvorbě herních AI a volbě jimi používaných technik. U Real-time strategií budou krátce popsány i herní fáze určené časem stráveným v jedné hře.

2.1 Vlastnosti a historie strategických her

Strategickou hru můžeme definovat jako hru, ve které hráči ovládají více postav, které spolu obvykle spolupracují, za použití promyšlených strategií s dlouhodobým cílem, avšak schopných reagovat na okamžité změny situace. Autonomní rozhodování hráčů a jimi volené akce mají dlouhodobý dopad a vysokou měrou ovlivňují výsledek hry. Téměř všechny strategické hry po hráči vyžadují velké povědomí o situaci, stavu parametrech vlastních i protivníkových postav stejně jako sledování mnoha dalších aspektů hry. To vše musí brát při okamžitém rozhodování v potaz.

Historicky vznikly první strategické hry již ve starověku – v Egyptě, Řecku či Indii, a to ve formě deskových her, kde byly obvykle dvě shodné sady herních kamenů ovládány na hracím poli dvěma protivníky.

Nejběžnějším příkladem jsou šachy, které se staly celosvětově známé. Jejich kořeny sahají až do 7. století našeho letopočtu a vznikly v Indii. V šachu oba hráči kontrolují stejný počet herních postav s různými vlastnosti, jako třeba způsob a vzdálenost možného pohybu či způsob vyhazování protivníkových postav. V šachové partii může hráč v každém svém tahu pohnout kteroukoliv ze svých postav na kterékoliv dostupné místo na šachovnici, což je samo o sobě velké množství kombinací. Avšak pro každou z nich musí být schopen odhadnout pravděpodobné reakce protivníka několik kol dopředu, takže složitost roste exponenciálně. I proto si k šachu brzo našly cestu počítače a delší dobu se vedl souboj mezi nejlepšími lidskými šachisty světa a vyvíjejícími se algoritmy.

V dnešní době však díky velkému výpočetnímu výkonu a rychlosti, omezenému počtu efektivních strategií a kompletní znalosti celého hracího pole zvládne počítač často zvítězit jen díky možnosti uvážení všech možností. V počítačových strategických hrách však takovou možnost nemají. Obecně musí ovládat více jednotek, a každá z nich je schopna provést více akcí. Navíc běžně bývá část herního prostoru pokryta tmou, tzv. *„Fog of War“*, takže nevidí soupeřovo snažení. To musí pouze odhadovat z průběžně získávaných informací. V případě

Real-time strategických her hraje značnou roli i čas. Veškeré výpočty se musí provádět opakovaně, často během každého snímku, reakce na podněty musí být okamžité.

2.2 Real-time strategické hry

Real-time strategie, nebo taky „RTS“, jsou takové bojové strategické hry, kde se vývoj neřídí inkrementálně po střídavých tazích, ale všichni hráči řídí své postavy nebo jednotky zároveň. Typicky může každý hráč postavit více budov a jednotek různých druhů. Úkolem je pak zničení všech jednotek a budov oponentů nebo dosažení nějakých speciálních cílů, například obsazení části mapy či nashromáždění surovin. Popis v této kapitole je nejdetailnější, neboť právě těmto hrám je v práci věnováno nejvíce prostoru.

2.2.1 Aspekty a fáze hry

V RTS hraje důležitou roli tzv. ekonomika každého hráče. Všichni hráči začínají se shodnými počátečními zdroji a několika jednotkami schopnými získat další zdroje. Každý hráč pak buduje ekonomiku výcvikem dalších podobných jednotek, zvaných těžaři nebo dělníci *„Miners“* nebo *„Workers“*), kteří sice zvýší příjem surovin, ale jejich pořízení něco stojí a taky trvá, čímž může blokovat funkci nějaké výcvikové budovy nebo ubrat ze zdrojů určených k obraně. Zdroje se pak typicky mohou sestávat z několika druhů – například zlato, dřevo, jídlo apod. Ty mohou být všechny získávány jedním nebo více druhy jednotek (někdy i budov) k tomu určených. Při těžbě nashromážděné množství suroviny stoupá průběžně celou dobu těžby malými přírůstky. Občas je součástí ekonomiky mechanismus podpory života jednotek či zásob nutných k udržování populace zvaných *„Supply“* nebo populační body. Jednotky nebo budovy lze obvykle vylepšovat – útok, obrana, nové schopnosti.

Po vybudování základů ekonomiky je potřeba začít se soustředit na vybudování prvních vojenských jednotek nebo obrany. Této fázi hry se říká *„Early game“* – brzká hra. Komplikované může být rozpoznání správného okamžiku. Pokud začne hráč budovat armádu příliš brzy, může omezit svůj ekonomický růst ve prospěch svým spoluhráčů. Pokud začne příliš pozdě, může prohrát hru díky nedostatečné nebo chybějící obraně v době nepřátelského útoku. Tomuto aspektu hry se říká otevírací strategie, taky *„Opening build“* nebo *„Opening“* a pro hráče to znamená obětovat v počáteční fázi hry veškeré své zdroje a vsadit na určitý typ jednotky. V této fázi je nejdůležitější okamžitá reakce na nepřítelovu strategii, k čemuž je potřeba neustále získávat informace.

O nějaký čas později základní obrana a jednotky přestávají stačit. Hráči mohou sáhnout po lepších nebo specializovanějších jednotkách, vylepšit ty stávající nebo se zaměřit na masovou produkci. Tato fáze hry je obvykle klidnější než opening a nazývá se *„Mid game“* – středová hra. Hráči sice již mají vybudovanou ekonomiku, nicméně ta přestává stačit nebo začíná být vyčerpaná, je tedy třeba rozšířit svá území – expandovat, vytvářet expanze (sekundární základny s úkolem získávání zdrojů). Otevírají se možnosti k tvorbě více typů jednotek. V této fázi hry vznikají strategicky nejzajímavější kombinace.

Když mají všichni hráči již k dispozici většinu typů jednotek, vylepšení a výzkumů, plný populační limit a několik základen, dostala se hra do *„Late game“* – pozdní hry. Hráčům již nezbývá než bojovat o zbývající zbytky zdrojů na mapě než jsou vyčerpány. Většina stran má k dispozici armády maximální velikosti sestávající se z mnoha druhů plně vylepšených jednotek, přesilu lze získat jen kompozicí armády či manévrováním a útokem ze zálohy. Do této fáze hry se v žebříčcích nejhranějších RTS her dostane spíše menší část zápasů.

2.2.2 Komplexita hry

Hlavními důvody, proč jsou strategické hry výzvou, jsou nutnost rychlého rozhodování a přizpůsobivost stále se měnící situaci, nepřetržité analýzy dostupných informací a schopnosti ovládat více jednotek či jejich skupin najednou. Rozeberme si některé podrobněji:

- **Výstavba** – hráč musí neustále řídit výstavbu budov a jednotek či výzkum vylepšení ve všech svých základnách.
- **Ekonomika** – hráč musí neustále monitorovat příjem všech surovin, tvořit nové těžaře a vyhledávat nové lokace k získávání zdrojů. Rovněž zde můžeme řadit kontrolování populačního limitu.
- **Průzkum** – nepřetržité vysílání jednotek do různých částí herní mapy za účelem získání informací o základnách, počtech a pohybech jednotek i kompozici armády nepřítele.
- **Obrana základen** – hlídání bezpečí ekonomiky a produkčních budov.
- **Armády** – pohyby s typicky několika skupinami jednotek.
- **Ovládání bitvy** – kontrola nad několika druhy jednotek, a nebo dokonce individuálních jednotek, v jedné skupině zvlášť a používání schopností jednotek. To lze úspěšně provádět pouze sledováním kompozice protivníkových jednotek a použitých schopností ve větší četnosti na menší ploše. Motivací je lepší výsledek střetu.

Tyto aktivity a jejich dílčí části lze do dvou důležitých kategorií:

1. **Micromanagement** – nejjednoznačnější zástupci z akcí vykonávaných hráči jsou například ovládání jednotek v bitvě nebo vybudování počáteční ekonomiky a prvních staveb. U těchto akcí jde o co nejpreciznější provedení, pohyb každé jednotky a lokace každé nové budovy, rozdělení úkolů každého dělníka nebo stavitele. Jeden špatný pohyb může znamenat ztrátu náskoku nad nepřítelem či zničení jednotek. Jedná se převážně o větší množství menších akcí s malým, individuálním dopadem na stav hry, které jsou však součástí většího celku. Záleží opravdu na každém detailu a drobné odchylky nebo malé časové úseky mohou znamenat rozdíl pro celou hru.
2. **Macromanagement** – převážně budování ekonomiky jako celku, průzkum, obrana základen apod. Akce jsou prováděny v mnohem větším měřítku. Nezáleží na drobných odchylkách v příjmech jednotlivých surovin nebo pozic stavby budov, avšak je třeba provádět nezbytné kroku průběžně a mít neustálý přehled o všech aspektech. Selhání může způsobit problémy, které se neprojeví okamžitě, ale později budou fatální.

Hráč, který chce vyhrát, musí mít pod kontrolou co nejvíce z micromanagementu i macromanagementu a najít rovnováhu stejně jako správný čas pro každou akci. Stejným výzvám čelí herní umělá inteligence.

2.3 Turn-based strategické hry

Tahové strategické hry, nebo taky TBS, jsou hry, kde sevývoj řídí inkrementálně po tazích. Soupeři se v tazích střídají. Tah může trvat neomezeně dlouho či mít předem daný časový limit. Během jednoho kola má každý hráč ki dispozici množství akcí, které může provádět v libovolném pořadí, které může a nemusí znamenat pro hru rozdíl. Kterákoliv informace o stavu hry dostupná na začátku tahu bývá pro hráče obvykle dostupná během celého jeho trvání, takže na ni nemusí reagovat okamžitě a má jistý čas na rozmyšlenou. Po odehrání tahu a předání řízení protihráči již nemůže nic ovlivnit a je pouze divákem soupeřů až do dalšího kola.

2.3.1 Aspekty hry

V tahových hrách se ekonomika, pokud existuje, řídí koly. Většina příjmu jednotlivých surovin se provede se začátkem kola a do konce kola jsou suroviny omezené na toto množství. Výjimkou jsou hry, kde lze po mapě sbírat malá množství surovin. Ty však ale netvoří trvalý, stálý ani konzistentní příjem. Zvýší-li si hráč příjem nějakým perzistentním způsobem, změnu pocítí až v příštích kolech.

Stavba nových budov je taktéž řízena koly. Typicky lze v jednom kole zahájit stavbu jen jedné budovy, ta může trvat jedno či více kol.

Souboje mezi hráči mohou probíhat buď na velké strategické mapě sdílené s pohybem armád a stavbou budov. V některých případech jsou na strategické mapě reprezentovány celé armády jako pouze jedna jednotka a při střetu s nepřátelskou armádou se hra přepne na tzv. taktickou mapu. Na té již může probíhat bitva se zcela jiným pohledem. Heroes of Might and Magic je typickým příkladem strategie s rozdělenou strategickou a taktickou mapou, kde se na obou hraje tahově. Hry ze série Total War mají strategickou mapu tahovou, ale bitva probíhá jako RTS.

2.3.2 Komplexita hry

Hlavní výzvou pro hráče je uvážení velkého množství možných tahů a předvídání soupeřova chování pro každý z nich. Při každé akci musí vyhodnotit, nakolik se přiblíží jím vytyčením cílům. Rovněž v jakém pořadí všechny akce, které jsou k dispozici, provést.

2.4 Budovatelské a ekonomické strategické hry

Ne všechny strategické hry jsou o zničení protivníka. V některých je hlavním cílem například stavba města, zábavního parku, zoo atd.

V takových hrách je většina hráčova snažení soustředěna do získávání surovin, vytváření rovnováhy mezi různými druhy služeb poskytovaných budovami nebo optimalizace tras používaných herními postavami.

Tato práce se takovýmito hrami nezabývá.

Kapitola 3

Agentní a jiné přístupy k tvorbě AI ve strategických hrách

V této kapitole jsou rozebrány základní přístupy používané při tvorbě umělých inteligencí strategických počítačových her, jejich výhody, nevýhody a nejčastější užití. Nejvíce je věnována právě problematice použití agentních systémů, která není častá, ale přináší mnoho výhod.

Je třeba poznamenat, že mnoho autonomního chování herních prvků je před hráčem skryto a většinou si nic neuvědomuje, ale bez jeho přítomnosti by se hra mohla stát nehratelnou nebo otravnou.

3.1 Statické skripty

Ve většině počítačových her je chování počítačem ovládaných prvků řízeno statickými skripty. Ty byly napsány za použití expertních znalostí tvůrců hry a prvních hráčů. Nejjednodušším příkladem může být seznam příkazů, které herní umělá inteligence vykonává jeden po druhém. Tento seznam se nazývá *„Řídící sekvence“*. K vykonání příkazu musí být splněny nějaké podmínky, které plnění umožní, jako například nashromáždění nějakého množství surovin pro výrobu jednotky či stavbu budovy, a nebo potřebná velikost armády k zahájení útoku.

Skripty mohou obsahovat i reaktivní nebo částečně dynamické složky, například zaútoč až budeš mít 10 jednotek typu A a 15 jednotek typu B, do té doby se věnuj výcviku. Efektivita odezvy je však nízká a čas vysoký. Prakticky všechny tyto příklady jsou však založeny jen na znalosti hry a možných akcí ve hře, téměř žádné na znalosti tahů protivníka, a proto může být herní AI zahnána do kouta v případě, že se s nějakou akcí soupeře nepočítalo – neumí na ni reagovat.

Tyto skripty neumožňují prakticky žádné sociální interakce mezi jednotkami, vše je řízeno jednou centrální entitou (*„Overlord“, „Overmind“*). Výhodou je nízká výpočetní náročnost – k vykonání dalšího příkazu stačí pouze kontrolovat, zda byla splněna podmínka.

Příklad řídící sekvence:

1. Postav budovu 1
2. Vycvič jednotku A
3. Vycvič jednotku B

3.2 Konečné stavové automaty

Komplexnější způsob řízení herní AI [7] [8]. Umožní každé jednotce volit svou následující akci podle jejího současného stavu [3]. Změna stavu však může být vyvolána nejen podnětem z okolí (v tomto případě ekvivalent abecedy přijímané konečném automatem), ale i změnou času. Tyto automaty musí být zcela deterministické.

Výhodou je vyšší autonomnost jednotlivých herních postav a možnost uplatnit základní sociální schopnosti. Nevýhodou je potřeba vyššího výpočetního výkonu a většího množství práce vývojářů investované do tvorby automatu. Pochopitelně se může stát, že automat nebude schopen reagovat na některý z podnětů.

Příkladem chování herní postavy řízené automatem můžou být tzv. Hlídači („*Patrollers*“). Ti ve svém základním režimu reprezentovaném výchozím stavem obchází kontrolní stanoviště, které dohromady tvoří delší trasu. Když hlídač projde trasu celou, vrací se na začátek (ať už průchodem celé trasy zpět nebo nejkratší cestou v prostoru) a opakuje. Pakliže spatří vetřelce, přepne se do stavu pronásledování. Když vetřelec zmizí z dohledu, hlídač ho napřed obezřetně vyhlíží a po nějakém čase se vrátí k nejbližšímu kontrolnímu bodu a pokračuje v obchůzce.

3.3 Alfa-beta ořezávání

Z anglického *Alfa-beta pruning*. Jedná se o algoritmus hledání ideálních tahů nejsilnějších pro hráče a nejslabších pro protivníka. Techniky založené na tomto algoritmu se používají pro tahové strategie, obvykle je však kvůli většímu množství akcí proveditelných v jednom tahu potřeba algoritmus přizpůsobit konkrétní hře a ve výsledku může mít svou podobou jen málo společného s původním základem.

3.4 Multiagentní systémy

Herní AI řízené agenty splňují obvykle všechny definice agentních systému. Agenti jsou autonomní, reaktivní, racionální a řízení záměrem a schopni vzdájemné komunikace umožňující jim aplikovat sociálních schopností ke společnému plnění cílů.

Tento přístup přináší mnoho výhod v reaktivnosti a kvalitě herních prvků – agentů, avšak cena výpočetního výkonu a doby vývoje bývá příliš vysoká na to, aby byl běžně využíván. Jedním z nejznámějších příkladů hry využívající multiagentní systémy je Total Commander z roku 2008, který je díky možnosti kontroly tisíců jednotek pro každého hráče po straně výkonu výzvou i pro moderní herní počítače. Nedostatečný výpočetní výkon zpomalí průběh hry natolik, že se stává nehratelnou.

Každá třída herního agenta musí mít vnitřní reprezentaci rozhodování. To lze modelovat jako statické skriptování, konečný stavový automat nebo jako několik úrovní switchů tvořených pouze z expertních znalostí hry. Je patrné, že podle zvoleného modelu lze usoudit inteligenci i náročnost agenta.

Důležitou otázkou v případě použití agentního systému je rozdělení herních entit na agenty.

3.4.1 Každá jednotka jako agent

První alternativou je použití agentů k reprezentaci všech hráčových jednotek [3]. Každá jednotka má k dispozici úplný popis možného chování a všechny možnosti k rozhodování.

Tento způsob je potenciálně nejlepší po stránce reaktivnosti a efektivity celého systému, naopak nejhorší co se týče výpočetního výkonu. Důvodem je procházení logiky agenta opakovaně a často zbytečně. Příkladem je shluk dvaceti jednotek, kdy všechny hromadně útočí. Proces nalezení optimální pozice, optimálního cíle a použitých schopností může pro většinu (nebo dokonce všechny) z nich skončit s velmi podobným či zcela totožným výsledkem. V nejhorším případě jsme tedy dosáhli až dvacetinásobného zpomalení výpočtu a to případně v každém snímku hry.

3.4.2 Manažeři

Další alternativou je použití několika málo agentů, z nichž každý ovládá větší množství jednotek či budov a je vysoce specializovaný k řešení jednoho úkolu (*„Overlords“*). Příkladem může být jeden agent pro řízení ekonomiky, jeden k výcviku jednotek a jeden k ovládání bojových jednotek. Jedná se o potenciálně mnohem rychlejší řešení, které však obětuje plnou decentralizovanost systému. Koncentrace rozhodování na méně místech přináší úlevu výpočetnímu výkonu počítače, umožňuje implementaci složitějších a detailnějších řídících algoritmů. Ztrácí se však efektivita ovládání každého agenta zvlášť, přestává záležet na každém jedinci.

3.4.3 Hybridní přístup

Tato alternativa počítá s použitím agentního přístupu pro veškeré herní entity, stejně jako aplikaci několika specializovaných manažerů. Implementačně pak bude každý agent reprezentující jednotku po rozhodovací stránce relativně slabý a bude sloužit nějakému nadřízenému. Může obsahovat například velmi jednoduchý stavový automat. Manažeři provádí nejdůležitější rozhodnutí nezávisle na svých podřízených, kterým jen dají příkaz, a ti se již starají o jeho vykonání sami. Příkladem může být příkaz k těžbě – manažer sice zadá rozkaz k těžbě suroviny, ale agent se již o odnášení do skladiště, návratu ke zdroji těžby a detekci kolize s okolními agenty stará sám. Potenciálně tento přístup nabízí nejvíce výhod tím, že kombinuje silné stránky obou předchozích, avšak je nejsložitější na implementaci a při nesprávném vyvážení úloh manažerů a jejich podřízených i příliš náročný na výkon počítače.

3.4.4 Zhodnocení agentních přístupů

Tato kapitola bude dvou herních technikách používaných v RTS hrách prezentovat, že všechny tři přístupy zmíněné jsou schopny řešit problémy relativně efektivně, což z multiagentního přístupu může činit úplné řešení ovládání herních inteligencí. Nejvhodnější způsob se však musí vybrat pro každou hru individuálně – podle běžného počtu jednotek a složitosti či množství jimi prováděných akcí. Tyto techniky jsou:

- **Dancing (Tanec)** – technika je používána pro efektivní použití jednotek schopných útoku na dálku. Spočívá ve využití časového intervalu mezi útoky k přesunu do větší vzdálenosti od cíle tak, aby byl cíl stále v dostřelu. Tato technika může potenciálně pomoci v boji proti protivníkům s menším dostřelem a s prodloužením života či záchranou jednotek. Problémem bývá volba směru ústupu a cílové pozice tak, aby nedocházelo ke kolizi více jednotek.
- **Kiting (Pouštění draka)** – obdobná technika spočívající v použití jedné nebo několika málo jednotek jakožto návnady pro nepřátelské jednotky. Tyto návnady se pak

neustále pohybují kolem spojenecké armády tak, aby na sebe zlákaly veškeré útoky a tak chránily zbytek armády, který se může soustředit na ničení. Pakliže nepřítel změní cíl, role se některých jednotek se změní, nová návnada okamžitě zahájí pohyb, předchozí se zastaví a začne útočit. Problémem bývá okamžitá detekce aktuálního cíle nepřátelských vojsk a volba cílových pozic tak, aby nedocházelo ke kolizi, nepřítel nezměnil cíl, nedohnal návnadu a spojenecké síly mohly pálit.

V případě aplikace agentního řízení na všechny jednotky velmi rychle roste efektivita využívání volného času při dancingu, detekce nepřátelského cíle a volby návnady při kitingu. Jednotky si samy hlídají čas uplynulý od posledního útoku, zda se útočí právě na ně a zda mají nepřítele v dosahu. Režie na vzájemnou domluvu mezi agenty ohledně pozic k přemístění a kolizí však rapidně roste, protože si každý agent musí sám vypočítat směr, odkud přichází nepřítel, sám, posléze se domluvit se všemi jednotkami ve stejné bojové skupině.

V případě použití manažeru pro řízení bitvy je výpočet nových pozic pro jednotky o něco jednodušší – určí se směr nepřítele pro celou skupinu tak, aby se všechny jednotky přesouvaly stejným směrem, každý jednotce se pak cíl určí. Již v tomto okamžiku je však jasné, že od určeného středu skupiny jsou některé jednotky dále a některé blíže, řešení je tedy rychlejší, ale ztrácí efektivitu pro jednotky na kraji. Detekce nepřátelského cíle pak musí proběhnout iterací přes všechny vlastní jednotky a to chvíli zabere. Reakce tedy může být efektivně i časově lepší i horší.

Kapitola 4

Techniky softcomputingu a strojového učení ve strategických hrách

V této kapitole bude velmi letmo popsáno použití technik a algoritmů k vytváření nebo optimalizaci inteligentních prvků herních AI [11] [10] [4] [6]. Některé z uvedených mohou být přímo použity jako součást řídících entit, jiné pouze jako prostředek k učení během hry nebo po ní.

4.1 Neuronové sítě

Ve strategických hrách jsou neuronové sítě používány k analýze protivníkova chování a kompozice jeho armády. Získané informace mohou být brány v potaz při krátkodobých i dlouhodobých rozhodnutích.

V obou případech učených informací jsou zásadními problémy získání nebo vytvoření trénovacích množin a výpočetní výkon nutný pro analýzu každého protivníkova tahu. V případě analýzy soupeřových pohybů je pak aplikace nejnáročnější a lze ji aplikovat pouze s využitím algoritmu pro sjednocování jednotek do skupin – analýzou celé skupiny jako celku je problém redukován na analýzu pohybu několika málo bodů, což vede k menší výsledné síti.

Používány jsou pak především sítě s co nejmenším počtem skrytých vrstev.

4.2 Učení a posilované učení

Techniky učení jsou používány na nejvyšších úrovních abstrakce. Mohou sloužit k ohodnocení schopnosti rozpoznávání nepřátelských taktik a hledání vhodných kompozic armády. Aplikované jsou rovněž pro optimalizace hladin a mezních hodnot pro různé autonomní části systémů, jako například poměry těžby surovin a množství postavených budov jistého druhu.

Při použití strojového, posilovaného učení vězí největší problém ve volbě správného okamžiku k vyhodnocení zpětné vazby. Základními možnostmi jsou:

1. Za každé rozhodnutí – krátkodobě se řešení může zdát jako dobré, negativní vliv na výsledek hry se projeví až později.Tento přístup však umožňuje učení se již během hry – pokud strategie opakovaně nevychází, zvolí se jiná.

2. Za kompletní hru – ačkoliv všechny zvolené možnosti vedly k pozitivnímu výsledku, nelze jednoznačně identifikovat slabší články.

4.3 Evoluční algoritmy a dynamické skriptování

Technika se uplatňuje především u herních AI psaných formou statického skriptování nebo konečného automatu [7] [1]. Jde o změnu a optimalizaci seznamu úkolů či grafu automatu pomocí genetických algoritmů [5]. V prvním případě se jedná o úpravy možné převážně pouze mezi jednotlivými hrami přepisem celé AI. Tyto techniku mohou být použity jak k optimalizaci strategie tak k její kompletní proměně. Velkým problémem je totiž volba vhodné fitness funkce.

V agentním systému se tento přístup hodí jen k úpravám chování individuálních agentů a nikoliv k úpravě celého systému. Aplikovatelný je jak v epizodním tak neepizodním prostředí.

Prakticky lze tento postup využít jen je-li známa strategie, proti které potřebujeme vyhrát. Toho lze docílit například použitím neuronové sítě k odhalení protivníkových cílů. Pakliže má agent k dispozici tuto analýzu, je si schopen vybrat jeden z několika předem připravených vzorců chování spočítáním jednoduché, ale obvykle nepřesné fitness funkce. Pakliže se žádná taktika nejeví jako efektivní, jsou aplikovány evoluční algoritmy s typicky nízkým počtem nových generací, a to buď k nalezení nových vzorců chování nebo k optimalizaci. Po úspěšné hře je třeba, aby byla nová strategie uložena či nahradila původní.

4.4 Ant Colony Optimalization

Optimalizace mravenčí kolonií se používá k vyhledávání nejvhodnějších cest v rozhodovacích automatech agentů změnou přechodových funkcí. Rovněž může být použita k vyhledávání nejvhodnějších tras na mapě – pro přesun jednotek nebo výstavbu základen. Tento postup však vyžaduje využití algoritmu pro seskupování jednotek do skupin. Typicky není dost času pro aplikaci více než několika málo pokusů o průchod grafu najednou, je však možné zjistit alespoň nějakou cestu a v dalším snímku se pokusit o další kroky k nalezení lepšího řešení.

4.5 Particle swarm optimalization a shlukování

Tyto techniky [9] [12] se využívají k rozdělování jednotek do bojových skupin a manipulaci s nimi – především sjednocování, dělení a pohyb po mapě.

4.6 Blackboard

Metody založené na hledání řešení problému přidáváním dodatečných informací a parciálních řešení.

Kapitola 5

Návrh agentního systému řízení ve strategických hrách

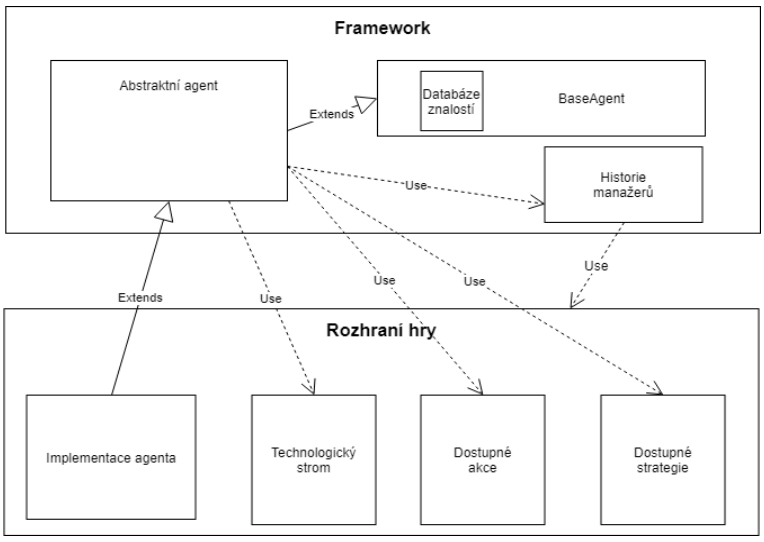
V této kapitole jsou popsány hlavní entity navrženého multiagentní systém reprezentujícího řízení jednotek jednoho hráče.

Nejprve je potřeba zmínit principy procesů sloužících k ukládání a získávání znalostí agentů o hře. Ty budou rozebrány primárně v kapitole o implementaci 6, avšak informace jsou užitečné při popisu agentů, kteří je využívají.

Poté následuje popis rolí a principu činnosti jednotlivých agentů. Rovněž je vysvětlen důvod přítomnosti agenta s danou úlohou v systému a pro ilustraci propojení popsána situace jeho odstranění ze systému. U každého dále pak vysvětlení rozhodování a volby cíle (ve kterých se uplatňuje učení a další techniky softcomputingu), kterým se tato práce věnuje nejvíce.

Jako poslední pak popis principu komunikace a sociálních schopností agentů vyplývajících z nutnosti spolupráce.

Návrh je nejvíce inspirován systémem z [8] s důrazem na možnosti optimalizací a rozšíření o inteligentní rozhodování [6].



Obrázek 5.1: Obrázek znázorňující strukturu frameworku a rozhraní

5.1 Technologický strom

Reprezentace přímého napojení na konkrétní hru. Jedná se o acyklický, orientovaný graf nezbytný pro popis jednotek, budov a ostatních konkrétních dat. Každý uzel stromu reprezentuje jednu herní entitu a nese informace o ceně její výstavby či produkce, typu a výčet uzlů, které musí být již zhotoveny pro zpřístupnění.

5.2 Databáze znalostí

V podstatě se jedná o službu poskytující způsoby ukládání a vyhledávání znalostí (představ, také *„beliefs“*, viz. 1.1.2) jednotlivých agentů o aktuálně hrané hře. Každá znalost je uchovávána jako samostatný objekt a může reprezentovat například konkrétní uzel technologického stromu nebo informace o mapě. Typicky se každý znalost může skládat dále z pozice a typu jí reprezentovaného objektu.

Duplicity v databázi znalostí jsou obvykle nežádoucí a při pokusu o opětovné přidání stejného objektu do databáze by se měly pouze aktualizovat již uložené stavy, nikoliv přidat další záznam. Příkladem vyhledávání může být nalezení všech budov určitého hráče hry, který lze použít ke zjišťování, zda jsou splněny veškeré závislosti k postavení nějaké další, pokročilejší stavby.

5.3 Strategy manager

Jak již bylo zmíněno v kapitole **??**, strategický manažer zastává nejdůležitější funkci pro průběh hry – volí herní strategii. Ta ovlivňuje rozhodování a plánování všech agentů systému. I tímto obstarává souhru všech manažerů. K volbě strategie je potřeba přehled o stavu ostatních manažerů a analýza informací o všech zaznamenaných jednotkách, budovách a pozicích všech hráčů. Relevantní mohou být poznatky o herní mapě, například její velikost, na ní ležící překážky, pozice nepřátelských základen a míst potenciálně vhodných k expanzi (díky přítomnosti surovin).

5.3.1 Rozhodování a učení

Ve výběru strategie v navrženém systému hrají roli tyto faktory:

1. Kompletní informace o hře – dostupnost všech dat je podmínkou k optimálnímu rozhodování. Stav, kdy si je agent vědom všeho, je však ve strategických hrách obvykle téměř nedosažitelný. Existují dvě možnosti, jak pohlížet na dostupné informace – jako na kompletní a jako na neúplné. V prvním případě je potřeba při každé změně přehodnotit herní plán. Ve druhém je klíčové správně uhodnout podstatu chybějících dat. Vzhledem ke značné složitosti druhé možnosti byl zvolen obvyklejší postup – pravidelné přehodnocování aktuálních poznatků.

2. Dostupné strategie – tvorba zcela nových herních strategií je komplexní úkol, a proto systém předpokládá existenci několika uživatelem definovaných. Ty však nejsou žádným způsobem striktní a slouží spíše jako nasměrování a doporučení.

3. Hodnocení dostupných strategií – k efektivnímu výběru strategie se používá dat z předešlých použití, jako například četnost použití a poměr výher (anglicky znám také jako *„win ratio“*).

Strategie jsou rozděleny do několika kategorií podle fáze hry, pro kterou jsou určeny (viz. 2.2.1). Každá strategie obsahuje cíl (například stavba konkrétní budovy), po jehož naplnění se strategie považuje za splněnou, doporučená fronta budov a jednotek ke stavbě během celé doby aktivity strategie a doporučené jednotky k výcviku kdykoliv, kdy jsou k dispozici zdroje navíc.

Taková podoba strategií umožňuje průběžnou optimalizaci a nebo okamžitou výměnu strategií posilující reaktivitu agentů systému. Rovněž lze vytvořit přímé vazby mezi kompozicí protivníkovy armády a jednotlivými strategiemi uložením statistik o jejich úspěchu proti obdobným hrozbám. Tyto statistiky musí být ukládány s koncem a načítány se začátkem každé hry. Tento proces je vhodný k učení s učitelem i bez učitele.

5.3.2 Systém bez strategického manažera

Bez strategického manažera by byl systém neschopen jakékoliv komplexnější reakce na tahy nepřítele. To zahrnuje změnu kompozice armády, důkladnější volby času útoku a obrany. Jinými slovy by zbytek manažerů postrádal souhru a společný vyšší cíl. Ačkoliv by všichni vykonávali své povinnosti, objevily by se pouze malé variace předem daného plánu a chování by bylo více statické a repetitivní.

5.4 Infrastructure manager

Produkční manažer, kontrolující výstavbu budov a výcvik jednotek. Jeho úkolem je efektivní a rychlá výstavba všeho, co je potřeba k realizaci cíle strategie zvolené strategickým manažerem. k tomu je potřeba neustálá revize a optimalizace plánu akcí, kde každá akce odpovídá výstavbě, výcviku či výzkumu herních prvků. Před vykonáním akce zjišťuje její cenu a požadavky pro realizaci, kterým se snaží vyhovět. Všechny potřebné suroviny a případnou jednotku vhodnou k realizaci výstavby budov musí napřed získat od jiných manažerů. V případě akcí reprezentujících výcvik jednotek pak musí najít vhodnou budovu schopnou výcviku, v případě výstavby budov najít optimální stavební umístění.

5.4.1 Adaptace a optimalizace

Vzhledem k tomu, že se stará o veškeré hráčovy zdroje, a to nejen ve formě využívání surovin, ale i v podobě zástavy volného místa na mapě nebo využívání již hotových budov, je kladen důraz na vysokou spolehlivost a efektivitu. Vzhledem k velkému množství parametrů se pak proces rozhodování a plánování zde stává nejsložitějším.

Fungování produkčního manažera si lze představit jako plánovač akcí ve formě fronty budov a jednotek k výstavbě, který se může měnit se strategií. Z přidělené aktivní strategie musí převzít úvodní doporučený plán akcí. Následně musí z požadavků pro realizaci cíle strategie a již realizovaných závislostí naplánovat nejkratší cestu k cíli. Tu potom vhodně začlenit do doporučeného plánu akcí tak, aby vznikla fronta reprezentující vhodný poměr mezi snahou o dosažení cíle a zdržením se od cíle za úmyslem výcviku bojových jednotek k obraně.

Po každé realizované akci přehodnotí veškeré zbylé akce ve frontě a plán se změní. Tento proces je ovlivněn především těmito dvěma aspekty:

- Validita dalších akcí ve frontě – validita některých akcí se může vzhledem k situaci ve hře měnit. Typickými příklady mohou být pozbytí nutnosti danou akci provádět nebo neuspokojené podmínky pro realizaci – chybějící zdroje, nedostatečná volná populace

jednotek. Takováto zjištění typicky vedou k úpravě fronty odstraňováním nehodících se a přidáváním nově potřebných akcí.

- Požadavky ostatních manažerů – například žádosti o nové dělníky v případě nedostatečné rychlosti získávání zdrojů od ekonomického manažera nebo žádosti o nové bojové jednotky od taktického manažera.

Tvorba perfektního plánu vyžaduje ohodnocení velkého množství potenciálních možností a proces může být až příliš výpočetně náročný a mohl by vést k pozdržení akcí. Řešení bylo naleženo v aplikaci genetických algoritmů běžících v sekundárním vlákně aplikace a schopných prakticky neustále vylepšovat aktuálně nejlepší řešení. Výsledná posloupnost proto obvykle není optimální, ale je dostatečně kvalitní a velmi adaptabilní.

Pokud produkční manažer provede akci označenou jako cíl aktivní strategie, pošle o události zprávu strategickému manažerovi a očekává přidělení dalšího úkolu, se kterým proces začíná nanovo.

5.4.2 Systém bez produkčního manažera

Bez budovatelského manažeru by se po stavbě úvodní budovy již nerozrůstala ekonomika (bez výcviku nových dělníků), nestavily další budovy a nikdy se nevytvořily žádné jednotky. Kdybychom povolili produkci jednotek a budov každému manažerovi, který by potřeboval, nároky na komunikaci by nekontrolovatelně vzrostly a nikdy by nebylo možné dosáhnout uspokojující organizovanosti.

5.5 Economic manager

Ekonomický manažer spravuje jednotky a budovy určené k získávání surovin. Jeho cílem je udržení příjmu hráče na správné úrovni a ve správném poměru mezi konkrétními komoditami. Rovněž poskytuje přehled o množství surovin okamžitě k dispozici nebo o množství zarezervovaném aktuálně prováděnými akcemi jiných manažerů. Suroviny poskytuje primárně produkčnímu manažeru. Pokud jednotka získávající suroviny zároveň plní roli stavebního dělníka, musí být ekonomický manažer požádán o její poskytnutí k výstavbě.

5.5.1 Statistiky průměrných příjmů a adaptace

Manažer si pro každou surovinu uchovává od začátku až do konce hry okamžité průměrné hodnoty příjmů v předem daných intervalech. Z nich se rovněž hledá nejnižší a nejvyšší hodnota. Tyto údaje slouží k informování ostatních manažerů a k analýze příjmů.

Informace se po každé hře uloží a mohou být využity k odhadnutí správné úrovně potřebné k optimálnímu vykonání akcí strategií. V dalších hrách jsou využity k nalezení rovnováhy v příjmu jednotlivých surovin a detekce vhodného času k ekonomické expanzi již v raných fázích hry.

Nejjednodušším je učení bez učitele, kterým se dá snadno odhadnout nedostatečnost příjmů vzhledem k realizované strategii nebo požadavkům ostatních manažerů na suroviny.

5.5.2 Systém bez ekonomického manažera

Bez ekonomického manažera by se po úvodní fázi hry řízené strategickým manažerem ekonomika hráče přestala rozrůstat a hlídat, neměl by kdo uvolnit suroviny ke stavbě a výcviku

jednotek, takže by se nic nevytvářelo. Po vytěžení surovin dostupných hned po začátku hry by nikdo nehlásil nutnost expanze k dalším zdrojům.

Tyto úkoly by mohly být vykonávány produkčním manažerem, avšak oddělení bylo shledáno logickým krokem vzhledem ke komplexitě jeho zbylých povinností.

5.6 Tactical manager

Taktický manažer ovládá vojenské jednotky. Rozhoduje o jejich taktice – o akcích, kterými se řídí celé skupiny jednotek a typicky ne individuální vojáci. Jeho úkolem je podle aktuálně zvolené strategie bránit vlastní nebo útočit na protivníkovy základny za použití všech dostupných prostředků.

Efektivně se snaží všechny jednotky dělit na bojové skupiny skupiny. Při každé bitvě určuje hlavní cíle a nepřetržitě vyhodnocuje, zda nemá protivník přílišnou převahu, kdy může dát povel k ústupu. Může řídit i každý pohyb vojáků. Kontrolu nad nově vycvičeními jednotkami získává od produkčního manažera a poskytuje jednotky nejvhodnější k průzkumu mapy průzkumnému manažeru.

Ve výchozím nastavení o svých akcích rozhoduje sám, ale může přijmout rozkaz nadřízeného strategického manažera, například pro prioritní obranu základny.

5.6.1 Shlukování a výpočet síly skupin

Jednotky je potřeba rozdělit do skupin shlukováním. Skupina má typicky nějaký střed, ke kterému jsou nově přidělené jednotky poslány a při přiblížení se propojí se skupinou. Pohyby jednotlivých skupin je třeba taktéž řídit.

Zd databáze znalostí musí být vypočtena síla nepřátelských armád. To lze realizovat přesně, součtem atributů jednotek, jako například jejích souhrnná síla útoku, a nebo klasifikací. Pro skupiny účastnících se potyček si musí být manažer jist jejich šancí na vítězství. V případě poklesu síly skupiny nebo objevení nových nepřátelských jednotek se může poměr sil změnit v cela nepříznivý a zavelí se ústup. Ve výpočtech nepřátelské síly hraje roli i vzdálenost jednotek – například není třeba počítat s obrannými budovami v oponentově základně při šarvátce ve středu mapy, na neutrálním území. Toto omezení lze realizovat například aplikací shlukování i na protivníkovy jednotky nebo sčítáním sil pouze jednotek nacházejících se pouze v kruhu o určitém poloměru se středem ve centru bojové skupiny. Pro pohyb bojových skupin se hodí optimalizace hejnem částic.

5.6.2 Systém bez taktického manažera

Bez taktického manažeru by vojenské jednotky po vytvoření pouze nehybně stály na místě a podle typu hry se maximálně bránily přímému napadení. Nešlo by koordinovaně útočit na nepřítele ani efektivně bránit žádnou základnu.

5.7 Recon manager

Průzkumný manažer aktivně prohledává mapu za účelem nalezení potenciálních míst k ekonomické expanzi, odhalení vojsk (počty jednotek a kompozice armády) a základen soupeřů a získání co nejvíce informací o herní mapě. Jednotky k průzkumu si žádá od produkčního nebo taktického manažera a hlavní cíle průzkumu si buď určuje sám nebo jsou mu

dány strategickým manažerem. Nejvíce ze všech manažerů přispívá do databáze znalostí ostatních.

Nalezení nepřítele a cesty k němu má obvykle nejvyšší prioritu.

5.7.1 Optimalizace tras

Činnost průzkumného manažera se velmi liší podle toho, zda má k dispozici úplné informace o mapě (1.2.1).

Pakliže je má, obvykle jsou mu známy potenciální startovní lokace nepřátel a umístění surovin vhodných k vybudování expanzí. Ty musí projít v nejvýhodnějším pořadí a pak vhodně zvolit, ke kterým se bude vracet a které ignorovat. Vzdálenost mezi takovýmito místy pak není jediným faktorem pro rozhodování – prioritou je prohledat počáteční lokace pro nalezení nepřítele a posléze monitorovat jemu nejbližší zdroje surovin kvůli potenciální expanzi. Vzhledem k upřednostňování uzlů před jinými a nenavštěvováním jiných nelze hovořit o podobnosti k problému obchodního cestujícího.

Bez úplných informací o mapě je úkol podobný, avšak volba trasy je téměř náhodná. Efektivní je pouze směřování pozornosti k místům s nejvíce neprozkoumaným okolím.

5.7.2 Systém bez průzkumného manažera

Bez tohoto manažeru by strategický manažer velkou část hry dostatečné množství informací k rozhodování a taktický manažer by nemohl útočit. Rovněž by nešlo expandovat za účelem zisku dalších surovin.

5.8 Komunikace a sociální schopnosti manažerů

Sociální schopnosti mezi agenty jsou nutné pro:

- Rozhodování o strategiích a volbě cílů.
- Předávání jednotek.
- Rezervování a předávání zdrojů, kontrola příjmů.
- Sdílení informací o nepříteli, vlastních jednotkách a mapě.

Kapitola 6

Implementace

Tato kapitola pojednává o konkrétních implementačních detailech jednotlivých částí navrhovaného systému.

V první části se nachází zmínky o implementačním jazyce, použitých knihovnách a technologiích.

Následuje popis napojení funkcionality frameworku na konkrétní hru. Toto rozhraní velkou mírou ovlivnilo podobu ostatních komponent a vyžaduje ze strany programátora doplnění mnoha nezbytných dat a postupů.

Třetí část kapitoly se zabývá společnými prvky implementací chování agentů a následně jejich specifiky. U každého jsou zmíněny i jeho sociální schopnosti – s kým komunikuje a o čem.

6.1 Implementační detaily a použité technologie

Framework je implementován v programovacím jazyce Java. V prvotních fázích byl vývoj realizován v C++, od kterého se však upustilo kvůli problémům s některými knihovnami pro tvorbu agentních systémů.

Java přináší implementaci velké výhody, jako například možnost využití frameworku pro tvorbu agentů JADE nebo podporu pro snadnou implementaci vícevláknových aplikací.

Je zde však i velmi silné negativum – omezení možnosti připojení frameworku ke hrám vyvíjených v nejčastěji používaných grafických enginech, například UnrealEngine nebo Unity, kde se standardně používají jazyky C++ a C# kvůli zlepšení výkonu. Propojení je stále realizovatelné, avšak s větší režií.

6.1.1 JADE

Java Agent Development Framework (zkratka JADE) je framework implementovaný v Javě používaný pro multiagentních systémů komunikujících zprávami podle specifikace FIPA. Kromě abstraktních tříd pro tvorbu agentů a specifikací jejich chování (*„Behaviour“*) obsahuje i nástroje pro řízení komunikace ACL zprávami.

Každá specifikace chování agenta je souhrn akcí, které jsou vykonávány uď periodicky nebo v návaznosti na přijetí ACL zprávy. Jeden agent může mít takovýchto *Behaviour* mnoho, každé reagující například na jeden druh zpráv a všechny se schopností paralelního vykonávání. Díky nim může celý agentní systém fungovat paralelně a mimo hlavní výpočetní vlákno aplikace.

6.1.2 Paralelismus

V grafických aplikacích se typicky většina výpočtů realizuje na úrovni snímků. Jeden snímek vykreslení, anglicky „frame“, je pak základní jednotkou grafických operací a odpovídá vykreslovací frekvenci obrazovky nebo aplikace. Ve vykonávaném programu se pak obvykle snímek vyvolá metodou `update` nebo `onFrame`, typicky s jedním parametrem `double deltaTime` značící počet uplynulých milisekund od počátku posledního snímku. Ve hrách se pak řídící operace vykonávají v hlavním vlákně v těchto metodách.

Ve vytvořeném frameworku však mnoho výpočtů probíhá ve zvláštních vláknech – ty mohou být vytvořeny i explicitně nebo mohou náležet jednotlivým `Behaviour` definovaných pomocí JADE. Ty umožňují provádění více věcí najednou i pokud jsou natolik výpočetně náročné, že v případě volání během každého snímku by hru zpomalovaly.

Kombinace obou přístupů pak volá po nutnosti synchronizace. Ta je v našem případě realizována připravením výsledků paralelních výpočtů nebo komunikace ve vedlejších vláknech a následným nastavením flagů či přepínačů. Ty si agenti vyhodnocují v každém snímku a příslušným způsobem pak ovlivňují hru.

6.1.3 JSON

Java Script Object Notation je způsob serializace objektů do textové podoby pomocí polí a objektového zápisu atributů. Standardně JSON rozumí základním, atomickým datovým typům. Formát JSON se nejčastěji používá pro zápis konfiguračních objektů ve formě řetězce.

6.2 Rozhraní hry

Tato kapitola pojednává o způsobech použitých pro vytvoření obecným rozhraním mezi konkrétní hrou a vytvořeným frameworkem. Je patrné, že použití Javy jakožto implementačního jazyka, stejně jako další fakta uvedená v kapitole 6.1, možnosti vytváření jakékoliv mezivrstvy a kompatibilitu s běžnějšími grafickými aplikacemi omezují.

Prvky realizující propojení se dají podle principu fungování rozdělit do dvou skupin:

- **Abstraktní třídy** – obvykle třídy reprezentující řízení nebo agenty, které pro svou činnost vyžadují vyhodnocení informací nebo vykonání příkazů souvisejících s konkrétní implementací herních jednotek. Typickým příkladem je porovnání dvou objektů a zjištění, zda se jedná o jeden a ten samý. K tomu je třeba dopředu znát jejich strukturu.
- **JSON struktury** – slouží primárně k popisu herních akcí a uzlů technologického stromu. Používá se tam, kde je třeba výčtu většího množství entit se stejnými atributy (jméno, cena atd.).

6.2.1 Pozice

Různé druhy her používají lišící se způsoby reprezentace souřadnic potřebných pro stavbu budov nebo přesuny jednotek. Aby bylo pro agenty systému možné pracovat jednotně, bylo potřeba vytvořit jednoduchou třídu reprezentující X a Y souřadnice na mapě. Význam těchto souřadnic je pak již plně v režii uživatele frameworku – určuje, zda se jedná o pixely, indexy dlaždic či jiný způsob reprezentace souřadnic v konkrétní hře.

6.2.2 Technologický strom

Technologický strom (viz 5.1) je souhrn popisů jednotek, budov a vylepšení uložený jako pole elementů v JSONu. Ty reprezentují výčet všech herních entit, s nimiž je systém schopen pracovat, protože je díky definici uživatele pozná. Neexistuje možnost, jak pracovat s nedefinovanými herními prvky.

Tabulka 6.1: Podoba jediného uzlu technologického stromu

Jméno	Typ	Hodnota
`name`	<String>	Jméno uzlu
`type`	<TypeEnum>	worker|unit|building|upgrade
`width`	<Integer>	Šířka jednotky nebo budovy
`height`	<Integer>	Výška jednotky nebo budovy
`price`	List<Price>	Cena uzlu, seznam dvojic název suroviny a hodnota
`requires`	List<String>	Uzly potřebné k produkci
`unlocks`	List<String>	Vytvářené jednotky (v daném uzlu)

Atribut `name` musí být unikátním identifikátorem každého uzlu. Na začátku každé hry se všechny uzly načtou z jednoho souboru. Až po načtení a vytvoření všech se existující propojí tak, aby objekty ukazovaly například v poli `requires` přímo na další uzly a neobsahovaly pouze jméno.

Ve frameworku je definována třída `TechnologyTree` uchovávající veškeré záznamy. Její metody slouží k vyhledávání uzlů podle různých parametrů.

Je žádoucí, aby byl typ uzlu definován interním názvem používané hry. To umožní lepší propojení s databází znalostí. Kromě výčtu entit k dispozici slouží technologický strom i pro podporu strategického rozhodování produkce – výčtem cen jednotek.

6.2.3 Akce

Seznam akcí je souhrn popisů operací, které jsou jednotliví manažeři schopni provádět . Jejich skládáním do určitého pořadí vzniká jejich fronta akcí, ke které lze přistupovat i jako k prioritní frontě, pakliže je třeba.

Agenti se pak snaží sekvenčně provádět jednu akci za druhou. Tato sekvence reprezentuje cestu k jimi zvoleným cílům. Ne všichni agenti musí tyto akce využívat.

Tabulka 6.2: Podoba jedné akce

Jméno	Typ	Hodnota
`name`	<String>	Jméno akce
`price`	List<Price>	Cena akce, seznam dvojic název suroviny a hodnota
`node`	<String>	Uzel technologického stromu vytvořený akcí

Jméno akce musí být unikátním identifikátorem. Výsledkem akce může být nějaký uzel technologického stromu. Pak je tento v akci identifikován jedinečným jménem. Cena akce může a nemusí být totožná s cenou produkovaného uzlu.

Třída `ActionList` na začátku hry přečte všechny akce, vytvoří objekty pro jejich reprezentaci a propojí je s uzly technologického stromu. Dále poskytuje metody pro vyhledávání akcí, například právě podle produkovaného uzlu.

Prováděné akce musí být přímo programátorsky definované v agentovi, který je vykonává. V obecné části frameworku lze manipulovat pouze s jejich reprezentací a hodnocením, není však možné dopředu znát veškeré napojení na herní objekty. Je možné sdílet stejné akce mezi agenty, dokonce i s jejich implementací.

6.2.4 Strategie

Seznam strategií je souhrn uživatelem definovaných postupů a cílů hry.

Tabulka 6.3: Podoba jedné strategie

Jméno	Typ	Hodnota
`name`	<String>	Jméno strategie
`type`	<StrategyType>	Typ strategie – „opening“, „standard“ a „lateGame“
`target`	<String>	Jméno uzlu, jehož produkce je cílem strategie
`buildOrder`	List<String>	Seznam doporučených akcí
`fill`	List<String>	Seznam akcí k vyplnění a spotřebu přebytečných surovin

Třída `StrategyList` na začátku hry přečte ze souboru všechny strategie. Každá strategie má jako unikátní identifikátor své jméno `name`. Atribut `type` explicitně určuje, zda se jedná o strategii určenou k použití na začátku hry nebo až v návaznosti.

Nejdůležitější atributy strategie jsou `target` a `buildOrder`. Průběh strategie je primárně definován budovami a jednotkami, které se produkují jejím vlivem, avšak tato produkce může být proložena i jinými než stavebními akcemi. Seznam akcí `buildOrder` explicitně uvádí doporučený postup, jak ke strategii přistupovat. Tento postup není skálopevný, avšak ke změnám dochází až v procesech popsaných v kapitole o učení (7). Zatím uvažujme, že je to seznam akcí nařízených k výstavbě. Tyto akce rozhodují o vývoji celé hry. Uzel `target` je pak cílovým uzlem technologického stromu, o jehož produkci je ve strategii usilováno. Dokončením tohoto uzlu strategie úspěšně končí a je zvolena další. Seznam akcí `fill` Je pak jen seznam akcí například produkujících jednotky, které se mohou kdkyoliv přidávat do `buildOder` akcí kvůli efektivnějšímu využívání získaných zdrojů nebo k podpoře prohrávající armády.

6.2.5 Historie manažerů

Za účelem ukládání dat pro učení manažerů byl vytvořen soubor `ManagersHistory.json`. Tento soubor je na začátku každé hry načten a manažeři mají možnost získat z něj svá data z posledních her. Rovněž mají po skončení hry možnost tato data aktualizovat nebo rozšířit. Tento proces slouží jako prvek pro učení bez učitele, v souboru se ukládají data převážně statistického charakteru – průměrné příjmy během her, použité strategie a jejich úspěšnost apod.

Uživatel frameworku může upravovat jaká data se ukládají, avšak ve výchozím nastavení se jedná hlavně o průměrné příjmy surovin a úspěšnost jednotlivých strategií. Každý ze základních manažerů má v souboru vlastní objekt reprezentující všechna jeho data. V něm je formát dat již pro každého volitelný.

6.2.6 Knowledge base

Databáze znalostí slouží k ukládání představ agentů o hře. Jsou případy, kdy mají manažeri k dispozici přesné informace, ale není to pravidlem.

Tabulka 6.4: Podoba jedné znalosti/představy agentů

Jméno	Typ	Hodnota
`type`	<KnowledgeType>	Typ znalosti
`position`	<Position>	Pozice objektu
`owner`	<OwnerType>	Vlastník objektu
`power`	<Integer>	Síla objektu reprezentovaného znalostí
`unitType`	<String>	Typ objektu
`description`	<String>	Popis znalosti – pro uživatele frameworku
`object`	<Object>	Objekt reprezentovaný znalostí

Typem znalosti může být například informace, budova, místo na mapě, jednotka apod., vlastníkem objektu může být hráč, spojenec, nepřítel či neutrální frakce. Při ztrátě viditelnosti objektu reprezentovaného znalostí uchovává pole `position` jeho poslední známou polohu. Typ objektu je definován pomocí jména uzlu technologického stromu, pokud takový existuje.

Třída `KnowledgeBase` reprezentuje databázi znalostí agenta a poskytuje metody pro ukládání, aktualizace a mazání znalostí. Poskytuje rovněž metody pro vyhledávání znalostí podle různých kritérií, například nalezení všech vlastní jednotek nebo nalezení všech nepřátelských budov.

6.3 Implementace agentů

Hlavními řídícími prvky systému jsou manažeři (popsáni v kapitole 5) – autonomní, proaktivní, navzájem komunikující agenti.

Hlavní manažeři vyžadují od uživatele tvorbu konkrétní implementace. Ta je nutná kvůli implementaci akcí, které jsou ve frameworku pojaty obecně, ale k fungování s konkrétní hrou jsou naprosto nezbytné. Příkladem je například ověření, zda je jednotka živá a schopna útoku či k nalezení místa na mapě. Tyto metody jsou rozebrány v manuálu, který je součástí přílohy.

6.3.1 Třída BaseAgent

Abstraktní třída `BaseAgent` je základem pro všechny manažery, kteří z této třídy dědí.

`BaseAgent` pak sama dědí z třídy `Agent` z frameworku JADE. Díky tomu je možné všem agentům přidat tzv. `Behaviour` – konkrétní implementace rozhraní vzájemného sociálního chování a komunikace agentů. Ti pak mohou komunikovat pomocí ACL zpráv.

Dále pak tato abstraktní třída poskytuje základní prvky pro operace s akcemi (viz. 6.2.3):

- Množina dostupných akcí – seznam akcí dostupných agentovi pro zobrazení všech dostupných nebo omezení výběru.
- Fronta akcí – Implementována jako seznam, všichni agenti a manažeři ve frameworku mají potenciálně podporu pro sekvenční provádění naplánovaných akcí a přístup k této

frontě kvůli dynamickým úpravám (změna pořadí provádění nebo dokonce akcí samotných).

- Limit počtu akcí – číslo určující maximální možné množství akcí ve frontě.

Navíc třída `BaseAgent` funguje i jako rozhraní. Manažeři se zavazují k implementaci především dvou čistě abstraktních metod:

- `onFrame()` – většina manažerů (typicky všichni kromě strategického manažera) vykonává činnosti přímo ovlivňující hru. Jak bylo zmíněno v kapitole 6.1.2, ty se musí provést v hlavním vlákně hry. Tato abstraktní metoda tedy umožní, aby se všichni agenti mohli dostat ke slovu během každého snímku.
- `transferUnit()` – většina manažerů (typicky všichni kromě strategického manažera, podle konkrétní hry i produkčního manažera) si uchovává přehled o jednotkách a budovách, které má pro své akce k dispozici. Tato metoda pak slouží k jejich předávání – dávající strana zavolá metodu toho, komu chce objekt předat. Ten se věnuje parametrem. Bohužel nelze bez znalosti konkrétní hry přesně určit typ předávaného objektu, proto se jedná o instanci generické Java třídy `Object`.

6.3.2 Strategy manager

Třída `StrategyManager` implementuje obecné chování popsané v kapitole 5.3. Její hlavní zodpovědností je výběr strategií ze seznamu dostupných strategií popsaných v kapitole 6.2.4 podle rozhodnutí založených na historii účinku strategií z kapitoly 6.2.5. Strategický manažer by měl být schopen poznat, zda má reagovat pouze na naučené postupy nebo má možnost dát šanci jiným – zda je aktivován režim učení.

Na začátku hry se vybírá pouze ze strategií označených jako *„opening“*. Po dokončení zvolené pak z množiny *„standard“* a obdobně nakonec z množiny *„lateGame“*.

Zvolenou strategii zasílá strategický manažer produkčnímu manažerovi. Standardně čeká, než je informován o splnění jejího cíle (obvykle vybudování uzlu označeného jako *„target“*) a pak volí další. Tento proces lze však kdykoliv změnit a strategii okamžitě vyměnit.

Strategický manažer si uchovává informace o strategiích použitých za celou hru. Na konci se pak tato historie zapíše a přidá ke stávajícím statistikám – zapamatování prohry nebo výhry u všech použitých.

Strategický manažer má rovněž možnost nařídit cíl útoku nebo obrany taktickému manažerovi. Ten si cíle volí sám, nicméně může vyvstat potřeba k takovémuto příkazu ze strategických důvodů, kterým taktický manažer nerozumí.

6.3.3 Infrastructure manager

Produkční manažer, implementován abstraktní třídou `InfrastructureManager` je nejkomplikovanější třídou i nejsložitějším manažerem celého frameworku. Hry druhu zvoleného pro zaměření frameworku počítají s řízením hry stavbou konkrétních budov a jednotek ve velmi specifických pořadích.

V produkčním manažeru se prolínají žádosti ostatních manažerů o tvorbu konkrétních jednotek se snahou přizpůsobit se co nejefektivněji strategii zvolené strategickým manažerem. Sám produkční manažer pak vykonává akce jen po získání surovin a stavebních pozic pro výstavbu. Vytvořené jednotky či budovy si pak buď nechává kvůli návazným akcím nebo předává ostatním manažerům.

6.3.4 Economic manager

Ekonomický manažer, popsaný v kapitole 5.5, jehož obecná funkcionalita se nachází ve třídě `EconomicManager` si uchovává seznam ekonomických jednotek a budov a počítá přírůstky surovin.

Metodou `transferUnit` dostává od produkčního manažera jednotky. Podle množství jednotek těžících konkrétní suroviny se jednotka přiřadí ke zvolenému úkolu. Úlohy všech jsou vykonávány do doby, než produkční manažer zažádá o dělníka ke stavbě. Po vzájemné dohodě se vybere nejméně zaměstnaný dělník nejméně vzácné suroviny – obvykle takový, který právě nic nenese. Ten je pak opět metodou `transferUnit` předán produkčnímu manažerovi.

Počítání surovinového přírůstku je realizováno v každém snímku hry. Pro každou komoditu existuje proměnná reprezentující stav z předchozího snímku. Pakliže se nová hodnota liší od staré liší, do mapy reprezentující historii se uloží záznam o změně identifikovaný číslem snímku, kdy ke změně došlo, a nesoucí hodnotu změny. Hodnota reprezentující minulý snímek se aktualizuje.

Standardně se počítá pouze přírůstek kladný, tedy změny se zápornou hodnotou se do mapy nepřidávají a pouze se aktualizuje proměnná reprezentující historii, avšak není pro uživatele obtížné proces změnit. Během hry se pak počítají průměr přírůstků – v předem nastaveném intervalu se spočítají přírůstky za poslední minutu a vydělí počtem uběhnutých snímků za jednu minutu hry. Tato proměnná vstupuje do algoritmu jako konstanta.

Pro celou hru se pak do historie manažerů ukládá nejvyšší a nejnižší minutový průměr a pak celkový průměr za celou hru. Je pochopitelně možné rozdělit průměry na intervaly podle fáze hry – je například zřejmé, že během první minuty hry bude průměr jiný než v pozdější fázi hry, kdy je aktivních mnohem více jednotek realizujících těžbu.

Tyto hodnoty pak mohou sloužit uživateli ke tvorbě strategií a vytváření potenciálních pravidel pro žádání produkčního manažera o nové dělníky nebo surovinové expanze při poklesu těžby.

6.3.5 Tactical manager

Reprezentován třídou `TacticalManager`, tento agent je pověřen kontrolou všechny bojových jednotek a řešením všech konfliktů s nepřítelem.

Bojové jednotky si ukládá v seznamu obecných objektů třídy `Object` a pro veškeré operace s nimi vyžaduje implementaci v konkrétní hře. V té musí uživatel frameworku specifikovat například operace pro zjištění, zda je jednotka naživu a schopna útočit, ověření pozice jednotky, ale i analýzu jejích atributů, jako například jejího zdraví.

Taktický manažer si sám volí cíl, který se skládá ze dvou nezbytných položek:

- Typ cíle – cílem je útok na nebo obrana místa na mapě či konkrétní jednotky.
- Pozice cíle – v případě útoku nebo obrany je potřeba znát lokaci.

Tento cíl mu však může strategický manažer kdykoliv změnit a dokud ho z jeho závazku nepropustí, sám si jej nemůže měnit.

Na začátku hry abstraktní taktický manažer požaduje od své implementace analýzu mapy kvůli získání tzv. „choke pointu“, zúženého průchodu do základy, kde je snadnější shromažďovat jednotky a bránit se prvotním útokům nepřítele. Ten je označen jako obranný cíl do doby, než průzkumný manažer nalezne nepřítele. Od prvního útoku je uchováván v paměti jako z možných shromaždišť po ústupu.

Po nalezení nepřítele se taktický manažer snaží útočit na jeho základny. Nalezení základny je mu indikováno přidáním zvláštního záznamu označeného „*BASE*“ do do databáze znalostí. Výběr základny k útoku může být realizován podle vzdálenosti nebo podle času přidání – tím se dostává vyšší priority novějším expanzím a nebo naopak prvotní základně.

Jednotky přiřazené taktickému manažerovi jsou rozděleny do bojových skupin. Skupina má vlastní cíl útoku. Taktický manažer nepřetržitě porovnává sílu svých skupin a nepřátelských jednotek podle metody implementované uživatelem. Při zjištění přesily je zavelen ústup k obraně shromaždiště a produkční manažer je požádán o další bojové jednotky. Nově vytvořená jednotka přidaná do seznamu bojových jednotek je přiřazena bojové skupině a vyslána na aktuální pozici této skupiny. Shlukování jednotek skupiny pak probíhá výpočtem vzdálenosti mezi nimi. Pakliže se nová jednotka dostatečně přiblíží některé jiné v téže skupině, stane se aktivním členem a rozkaz se jí změní s přiblížení ke skupině na aktivní příkaz celé skupiny, tedy například na útok na základnu.

6.3.6 Recon manager

Reprezentován abstraktní třídou `ReconManager`, průzkumný manažer po implementační stránce vyžaduje pro svou funkčnost tři věci:

- Průzkumné jednotky
- Znalost startovních pozic hráčů
- Znalost všech pozic potenciálních surovinových expanzí

Průzkumné jednotky mohou být jak vojenského typu, tak dělníci. Po začátku hry je vyslána žádost produkčnímu manažeru o poskytnutí alespoň jednoho špeha k okamžitému začátku průzkumu mapy. Tento požadavek není uspokojen dříve než s vytvořením první jednotky schopné pohybu. Po převzetí tohoto špeha jej průzkumný manažer okamžitě vyšle hledat nepřátelskou základnu.

Naše implementace předpokládá, že má celý agentní systém k dispozici úplnou informaci o mapě (viz. 1.2.1), což můžeme přirovnat k člověku, který by se vyznal po městě nahlédnutím do jeho mapy. Ačkoliv jsou tyto pozice požadovány po implementaci agenta pro konkrétní hru, jsou nezbytné pro tvorbu trasy k průzkumu.

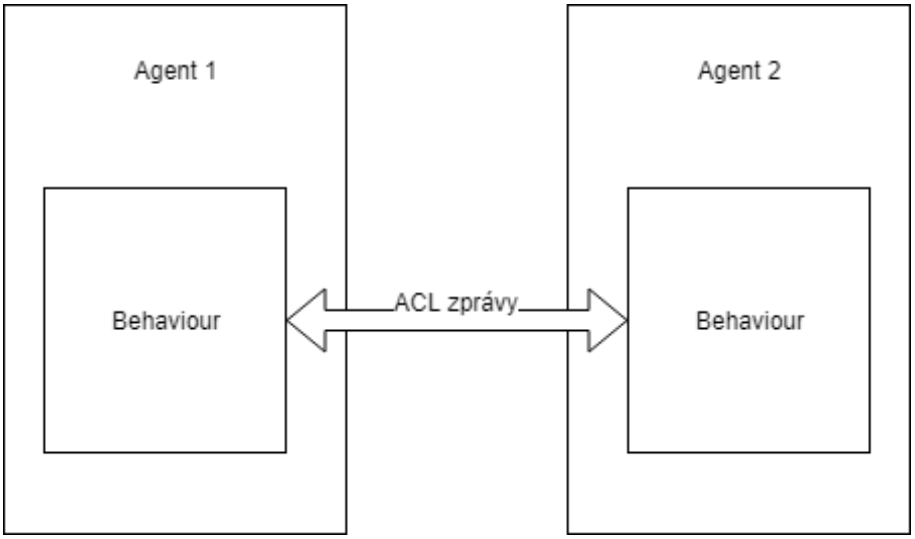
Po výpočtu první trasy proběhne průzkum všech startovních lokací vyjma vlastní. Pakliže je na některé z nich nalezen nepřítel, ostatní se již dále neprohledávají. Průzkumný manažer se pokusí svým špehem odhalit co nejvíc informací o počátečních snaženích protivníka, tedy informace potenciálně užitečné pro odhalení jeho strategie. Poté periodicky monitoruje tři nejbližší potenciální expanze.

Pakliže je jeho jednotka určena k průzkumu, požádá o další. V této implementaci se používá pouze základních dělníků pro roli zvěda.

6.4 Komunikace

Komunikaci mezi agenty je realizována pomocí zpráv podle FIPA ACL specifikace. Zasílání a přijímání těchto zpráv je zařízeno pomocí frameworku JADE, který tento typ zpráv podporuje. Každý agent má implementováno rozšíření `SimpleBehaviour` třídy, které slouží jen k obsluze příchozích zpráv. Komunikace mezi agenty je neblokující, takže mohou při čekání na zprávu dále provádět svou činnost.. ACL zprávu lze kdekoliv v implementaci agenta zaslat pomocí metody `sendMessage` definované v `BaseAgent` třídě frameworku. Navržené

komunikační protokoly může uživatel rozšířit zasláním své vlastní zprávy a implementací vlastní instance třídy `Behaviour`, která bude tuto zprávu přijímat, a přidá ho do vlastní implementace agenta.



Obrázek 6.1: Základní schéma komunikace agentů

Kapitola 7

Učení, adaptace a rozhodování agentů

V této kapitole jsou popsány schopnosti učení a adaptace agentů, které slouží hlavně jako podpora jejich rozhodování o volbě cíle nebo hledání cest jak cíle dosáhnout.

První použitou technikou je učení bez učitele. Tu používá několik manažerů ke zjištění nejefektivnějších postupů.

Za druhé jsou to genetické algoritmy použité v produkčním manažerovi pro optimalizace pořadí prováděných akcí.

7.1 Učení bez učitele

Učení bez učitele znamená, že systém hodnotí akce sám bez zásahů učitele – bez zpětné vazby od programátora nebo uživatele.

Ve vytvořeném systému je učení realizováno pomocí souboru s historií pro jednotlivé agenty popsaného v kapitole 6.2.5. Do něj si mohou po každé odehrané hře manažeři ukládat libovolná data, které mají na začátku další hry opět k dispozici. Informace uložené v těchto datech mohou typicky nabývat podoby statistik, avšak je možné uložit prakticky cokoliv. Programátor může při používání frameworku sám doplnit svá data pro uložení.

Ve frameworku se učení používá na třech místech (pro tři různé manažery), a to k rozhodování o nejlepší strategii proti konkrétnímu protivníkovi strategickým manažerem, optimalizaci vyhledávacích tras průzkumného manažera a optimalizaci získávání surovin ekonomickým manažerem.

7.1.1 Učení strategického manažera

Ve standardní podobě historie dodávané frameworkem si strategický manažer po hře aktualizuje záznam o všech strategiích, které v této hře použil. Historie obsahuje jeden statistický objekt pro všechny doposud použité kombinace frakce protivníka a strategie. Pakliže takováto kombinace doposud použita nebyla, vytvoří se nový záznam.

Tyto záznamy obsahují počet her odehraných a vítězných za použití této strategie proti konkrétní nepřátelské frakci. Tyto dva údaje dohromady dávají statistiku, která se označuje jako *„win ratio“*.

Při začátku nové hry si strategický manažer tyto statistická data načte ze své historie a podle nich se v průběhu hry rozhoduje. Algoritmus výběru strategie je možné uživatelem změnit nebo upravit, ale v podobě dodávané ve frameworku se strategický manažer řídí

i jistou mírou nahodilosti, čili si může vybrat i strategii doposud hůře hodnocenou a dát jí šanci, dokonce si může zvolit i strategii doposud nepoužitou – například po přidání nové strategie do souboru se seznamem strategií. Strategie s lepším ohodnocením mají však mnohem vyšší šance pro opětovné použití. Strategický manažer se tedy během existence konkrétního systému stále učí, pokud není nastaven jinak.

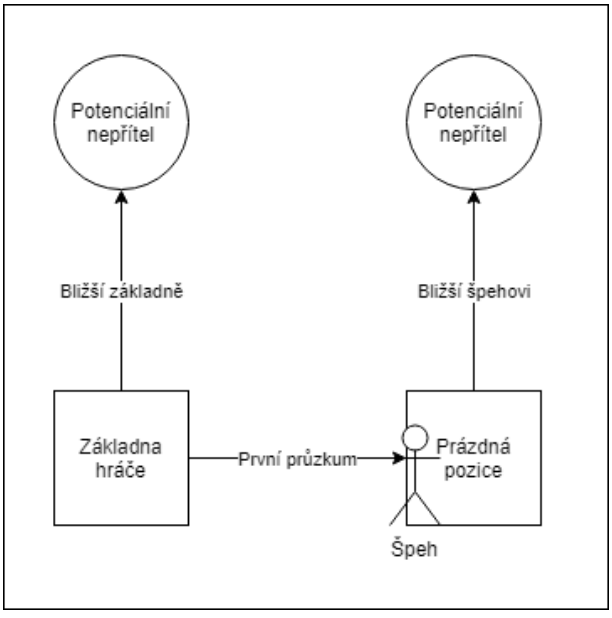
7.1.2 Učení průzkumného manažera

Podle popisu činnosti průzkumného manažera z kapitoly 6.3.6 vyplývá, že se začátkem průzkumu plánuje trasu na prohledání všech známých potenciálních startovních pozic hráčů kromě své vlastní. K průzkumu lze použít několik možných algoritmů. Ve frameworku jsou za pomoci učení zkoumány dva:

1. Nejbližší základně
2. Nejbližší zvědovi

Při začátku průzkumu je první cíl zkoumání jasný – pro oba algoritmy totožný, jelikož nový zvěd byl vyroben a předán ke kontrole průzkumnému manažerovi v hráčově základně. Pakliže není protivník nalezen, na mapách pro více než dva hráče pak zbývá ještě několik možností, kde může být. V této chvíli se může manažer rozhodnout, zda poslat zvěda na neprozkoumané místo nejbližší zvědovi nebo prvotní základně hráče. Z obrázku 7.1 je patrné, že špeh nyní přesunem k příští pozici a případným úspěchem či neúspěchem může získat nebo ztratit hodně času.

Umisťování hráčů na mapu může být v každé konkrétní hře velmi specifické a výsledek získaný učením vypovídá na takto zjednodušené mapě v podstatě jen o algoritmu náhodného výběru pozic ve hře, nikoliv o inteligentních akcích protivníka. Hodnotu získává na větších mapách.



Obrázek 7.1: Obrázek naznačující volby příští lokace k průzkumu při hledání nepřítele

7.1.3 Učení ekonomického manažera

Jak bylo již popsáno v kapitole 6.3.4, ekonomický manažer si za celou hru počítá průměrné přírůstky jednotlivých surovin. Z výpočtu si pak ve výchozím nastavení ponechává nejnižší, nejvyšší a průměrnou hodnotu. Na konci hry si tyto informace uloží společně s příznakem, zda tento příjem stačil pro výhru či nikoliv. Na základě těchto informací může během hry například žádat produkčního manažera o další dělníky.

7.2 Genetické algoritmy

Ve vytvořeném frameworku slouží genetické algoritmy k optimalizaci naplánovaných akcí [2]. Konkrétně je to v produkčním manažeru, jehož proces výběru a realizace akcí má největší dopad na výsledek hry.

Pro připomenutí z kapitoly 6.3.3, produkční manažer si uchovává seznam budov a jednotek ke stavbě ve frontě. Tato fronta je vytvořena ihned po přidělení strategie strategickým manažerem. Cílem produkčního manažera je co nejefektivněji postavit všechny entity ve frontě a uzel, který je cílem strategie. Avšak akce ve frontě mohou být v neoptimálním, dokonce nevalidním pořadí.

Fronta není validní, pokud takovou obsahuje posloupnost akcí, že při jejich sekvenčním provádění se manažer dostane do stavu, kdy jeho následující akce nemůže být nikdy provedena. Tato situace může nastat například z těchto důvodů:

- Neuspokojení podmíněnosti uzlu – pakliže následující akce produkuje uzel tehcnologického stromu (jednotku či budovu), který nemůže být podle interní logiky hry postaven kvůli skutečnosti, že uživatel doposud nepostavil všechny uzly, na kterých tento závisí, produkční manažer nemůže pokračovat a zasekne se.
- Nedostatečné populační body – uzel technologického stromu produkovaný následující akcí je jednotkou, avšak hráč nemá v současné době dostatek volných populačních bodů k její vytvoření. V tomto případě se produkční manažer zacyklí při snaze zarezervovat všechny zdroje.
- Fronta neobsahuje cílový uzel strategie – tento nedostatek nijak nepřeruší činnost produkčního manažera, avšak může hráče stát hru díky uváznutí produkce na málo pokročilých jednotkách.

Tyto problémy lze obvykle řešit deterministicky , avšak nalezení optimálního řešení může vyžadovat neadekvátní množství výpočetního času a při kombinaci příliš mnoha hodnotících faktorům nemusí být nalezeno vůbec. Genetické algoritmy nemusí být schopny nalézt nejlepší řešení, ale vždy naleznou alespoň nějaké. To mají posléze možnost ještě vylepšovat nebo upravovat při změně podmínek ve hře, čímž dodávají celému procesu na dynamičnosti.

Dalším problémem je optimálnost pořadí prováděných akcí. Fronta akcí není optimální například pokud:

- Akce je závislá na předcházející – například při stavbě produkční budovy, kdy následující akcí je stavba jednotky v ní. Akce výcviku jednotky tedy dlouhou dobu čeká na dokončení této budovy, zatímco by nahromaděné suroviny mohl produkční manažer použít na jinou, splnitelnou akci.
- Populační body – v době začátku výcviku jednotky musí mít hráč dostatek populačních bodů. Pakliže se však budova poskytující tyto body teprve staví, opět musí produkční manažer čekat. Je proto lepší stavět takovéto budovy s předstihem.

- Cíl bude splněn příliš pozdě – pakliže má hráč dostatek zdrojů a jsou splněny všechny prerekvizity cílového uzlu, je zbytečné otálet.
- Hromadí se suroviny – pakliže má hráč dostatek surovin na provedení dalších akcí a momentálně necvičí tolik bojových jednotek, kolik by mohl, lze do fronty akcí doplnit akze z pole `fill` strategie a získat tak snáze převahu nad nepřítelem.
- Neodpovídá požadavkům jiných agentů – ostatní manažeři mohou během hry dynamicky žádat o další jednotky. Přesně vypočítaný plán s těmito požadavky nemusí počítat a je náročné ho vytvořit celý znovu.

Dohromady tyto faktory tvoří až příliš mnoho požadavků a vstupních proměnných pro exaktní algoritmus, avšak je potřeba na ně dynamicky reagovat.

7.2.1 Obecný postup

Při převzetí nové strategie od strategického manažeru spustí produkční manažer první kolo evolučních algoritmů. V něm proběhnou následující akce:

1. Převezme se `buildOrder` seznam akcí strategie a z něj vytvoří fronta akcí.
2. Na její konec se doplní nejkratší cesta akcí k cílovému uzlu vypočítaná technologickým stromem, pak akce produkující samotný cílový uzel.
3. Vypočtou se místa ve frontě, kde dojde k nedostatku populačních bodů. Pakliže jsou taková místa nalezeny, vloží se před ně akce vedoucí k navýšení.

V této fázi je garantována, že je fronta akcí validní vzhledem k přítomnosti akce produkující cílový uzel a dostatku populačních bodů, ne však vzhledem k podmíněné splnitelnosti všech akcí.

Z této prvotní fronty akcí se mutací vytvoří počáteční generace. Každý jedinec je ohodnocen fitness funkcí. Poté proběhne několik kol křížení. Všichni noví jedinci jsou před ohodnocením mutováni a prověřeni na validitu populačních bodů, která může být opětovně doplněna novými akcemi pro její dosažení. Křížení může v krajních případech odstranit akci produkující cílový uzel, tne může být znovu doplněn.

Po předem daném počtu iterací křížení se vrátí nejlépe ohodnocený jedinec, který následně slouží jako fronta akcí produkčního manažera. Tato fronta nemusí být validní a vzhledem k velkému množství proměnných zasahujících do evoluce a typicky malému počtu generací nebude s nejvyšší pravděpodobností ani optimální. To však není problém, protože ihned po provedení první akce ve frontě jsou egenetické algoritmy opětovně spuštěny. V těch figuruje současná fronta jako počáteční jedinec.

7.2.2 Mutace

Mutace je v těchto genetických algoritmech variačním operátorem sloužící primárně pro optimalizaci. Jejím cílem je nalezení efektivnější pořadí akcí změnou jejich pořadí a doplnění akcí do míst, kde nejsou efektivně využívány suroviny.

Mutace může mít nepříznivý efekt na validitu pořadí akcí, například přemístěním uzlu závislém na jiném před něj či přidáním jednotek do míst, kde bude jejich tvorba nevalidní vzhledem k populačním bodům.

Mutace se skládá z těchto hlavních operací:

- Přidání dodatečných akcí produkujících bojové jednotky z položky `fill` strategie na náhodná místa fronty akcí.
- Prohození pořadí sousedních akcí fronty na náhodných pozicích.
- Posunutí akce produkující cílový uzel strategie blíže počátku fronty.

7.2.3 Výběr jedinců a křížení

Jedinci jsou vybírání náhodně. Vždy se vyberou dvě dvojice náhodných jedinců, z každé dvojice je lépe ohodnocený jedinec zvolen jako rodič. Z genů rodičů (jednotlivé akce v jejich frontě) jsou náhodně vybírány geny výsledného jedince. Ten je po konci křížení zmutován, ohodnocen fitness funkcí a v populaci nahradí nejhůře ohodnoceného jedince.

7.2.4 Fitness funkce

Vzhledem k předpokladu, že žádná fronta akcí vytvořená křížením a mutací není optimální, hodnota fitness byla zvolena jako celé číslo typu `integer` reprezentující množství nedokonalostí jedince reprezentovaného seznamem akcí. U nového jedince začíná jeho počáteční fitness na nule. S každou nedokonalostí je odečten jistý počet bodů reprezentující prioritu daného aspektu hodnocení a míru neuspokojení. Ve výchozím nastavení frameworku jsou aspekty porušující validitu hodnoceny mnohem přísněji, než ty způsobující pouze horší efektivitu.

Mezi nejdůležitější aspekty patří například:

- Neřítomnost cílového uzlu ve frontě – hodnocena velmi tvrdě.
- Rozdíl vzdálenosti cílového uzlu od počátku fronty a délky nejkratší možné cesty k uzlu – pakliže je před vybudováním cílového uzlu nutné postavit další dva, optimální pozice pro cílový uzel je třetí místo od počátku fronty. Pakliže se cílový uzel vyskytuje ve formě blíž, není splnitelný. Pakliže dál, postaví se později, než může. Tato vzdálenost od nejlepší pozice je pak záporným hodnocením tohoto aspektu.
- Počet míst nevalidních po stránce populačních bodů – hodnoceno relativně mírně vzhledem k faktu, že tento problém je možnost opětovně řešit v příštích generacích.

V plánu bylo hodnotit frontu například i podle míry paralelismu plnění akcí nebo podle doby čekání před počátkem plnění další akce (kromě případu, kdy se čeká pouze z důvodu chybějících surovin). S narůstajícím počtem parametrů fitness funkce však složitost rapidně stoupala a vznikaly další požadavky pro implementaci některých hodnocení uživatelem, což vedlo k nižší obecnosti procesu.

Při frontě obsahující zhruba dvacet akcí se hodnocení jedinců běžně pohybuje v intervalu (-120, 30).

Kapitola 8

Model pro StarCraft:Brood War

Pro ověření použitelnosti vyvíjeného frameworku bylo nutností vytvořit testovací model. K tomuto účelu byla vybrána strategická hra StarCraft, konkrétně její rozšíření Brood War. K důvodům, proč právě takto starší hra, patří jednoduchost ovládání, dostupné API a velké strategické možnosti – vítězství velmi často neznamená jen silnější armádu. Jedna z nejsilnějších motivací pak byla existence stále aktivních soutěží pro studenty, odkud bylo možné se dozvědět o běžně používaných technikách a nebo poměřit síly s déle existujícími boty.

V této kapitole jsou popsány základní vlastnosti a aspekty StarCraftu, jako umístění děje hry, hratelné frakce a princip fungování ekonomiky.

Dále pak následuje popis konkrétní implementace rozhraní mezi hrou a vyvíjeným frameworkem navazující na popis z kapitoly 6.2.

8.1 StarCraft: Brood War

V roce 1998 vydala společnost Blizzard Entertainment science fiction RTS StarCraft a ještě téhož roku i rozšíření Brood War. Hra se po dlouhou dobu těšila přízně velkého množství fanoušků, především z USA a Jižní Korei. Přestože po vydání druhého dílu StarCraft II: Wings of Liberty v roce 2010 první díl hraje méně lidí, stále se ve StarCraft: Brood War pořádají turnaje a neumírající zájem dokonce přesvědčil společnost Blizzard Entertainment vydat v roce 2017 StarCraft: Remastered – verzi téměř totožnou s původní, avšak s lepší podporou hry online, s lepší grafikou a s možností přepnutí do vyššího rozlišení. Hlavním úkolem hráče je zničení všech nepřátel.

8.1.1 Popis děje a světa

Děj StarCraftu se odehrává přibližně kolem roku 2490 v sektoru Koprulu – vzdáleného několik desítek let letu vesmírem od Země. Zde lidé (taktéž pojmenovaní jako Terrani – pozemšťané)kolonizovali několik obydlených planet. Po asi dvou stech letech od kolonizace a od odloučení od Země se pak lidské frakce připletou do války mezi Protossy, pokročilou a pradávnou rasou válečníků se psionickou silou, kteří se spoléhají na technologii a boj se ctí, a Zergy, nestvůrami s čistě organickými technologiemi, které se spoléhají především na sílu ve velkém počtu a primitivní boj z blízka pomocí drápů.

8.1.2 Grafika a vizuální stránka hry

Grafika StarCraftu je dvourozměrná (2D), avšak snaží se perspektivou o semi-3D dojem. V roce 1998 byly vykreslovací technologie mnohem horší než dnes, hra proto standardně funguje při rozlišení 640x480 a mezi 6 a 23,5 snímků za sekundu (FPS). Rychlost hry je svázána s FPS, což se v dnešní době považuje za nežádoucí. Na svou dobu má hra poměrně pokročilý antikolizní systém jednotek.

8.1.3 Herní suroviny a ekonomika

Během hry jsou důležité tři suroviny:

- **Minerály (Minerals)** – základní surovina v podobě modrých krystalů. Lze ji těžit již od úplného začátku hry a prakticky všechny jednotky a budovy ve hře potřebují minerály ke své produkci. Rozvinutí dostatečné těžby minerálů je od začátku hry hlavním ekonomickým cílem hráče a je potřeba použít minimálně desítky těžebních jednotek.
- **Vespene (Vespene gas)** – pokročilá surovina potřebná až pro lepší druhy jednotek a jejich vylepšení. Těžbou vespenu může hráč získat technologický náskok nad protivníky. Na počátku hry je potřeba nalézt gejzír vespenu, postavit na něm speciální budovu realizující těžbu a v ní následně „zaměstnat“ dělníky. Těžba vespenu je mnohem pomalejší vzhledem k nízkému počtu dostupných gejzírů a je potřeba s touto surovinou více šetřit.
- **Populace (Supply)** – všechny jednotky ve hře obsadí svým vyrobením část hráčovy populace (často jednotky obsadí více populačních bodů než 1). Maximální možná velikost je 200 obsazených populačních bodů. Od počátku hry však hráč může mít jen mnohem méně jednotek a k tomuto číslu se může dostat pouze stavbou speciálních budov nebo jednotek, které mu další populaci poskytnou. Všechny tři herní frakce mají vlastní verzi populačních bodů.

Na začátku má hráč k dispozici čtyři dělníky a hlavní budovu schopnou jejich produkce. Ty musí okamžitě poslat těžit minerály, ty pak použít k výstavbě dalších dělníků. Populace k dispozici je v tomto okamžiku kolem desíti a je rychle využita, je tedy třeba první budovy přidávající další. Posléze se typicky začíná se stavbou prvních vojenských staveb schopných produkce vojenských jednotek.

Když začnou docházet minerály v prvotní základně, a nebo hráč cítí bezpečí, přichází na řadu fáze expanze – stavba nové hlavní budovy na místě s dalšími minerály a gejzírem vespenu.

8.1.4 Herní frakce Terranů

Reprezentace pokročilejší verze naší současné technologie, Terrani jsou fyzicky nejslabší frakcí zcela se spoléhající na útok z dálky – konvenční střelné zbraně a bojová vozidla a tanky či vesmírné lodě. Jejich budovy při poškození samovolně hoří, takže mohou být při nepozornosti hráče úplně zničeny jen pomocí pár útoků nepřítele. Terranský hráč má však možnost opravovat budovy a mechanické jednotky, což v kombinaci se silnou palebnou silou činí tuto frakci velmi silně přirozeně orientovanou na obranný způsob hry. Dělník této frakce musí být přítomen u stavby budovy až do jejího konce. Vzhledem k pomalejšímu postupu k nepříteli je tato frakce méně náročná na multitasking.

8.1.5 Herní frakce Zergů

Technologicky velmi primitivní, avšak evolučně velmi přizpůsobiví a houževnatí Zergové se spoléhají na útoky velkým množstvím slabších jednotek, které kombinují možnost ničit nepřítele z blízka i z dálky. Jejich jednotky jsou velmi rychlé a schopny způsobit velké škody během krátké doby. Rovněž se veškeré jejich jednotky i budovy neustále regenerují. Vzhledem k organické podstatě budov mohou stavět pouze v blízkosti již existujících tam, kde ty stávající rozšířily „*creep*“ – fialovou látku schopnou budovy vyživovat. a při výstavbě budovy dělník zanikne, v budovy se vlastně promění. Schopnost rychlého pohybu a doplňování ztracených jednotek je činí frakcí vhodnou k agresivnějšímu způsobu hry a ovládnutí celé mapy, k čemuž je však potřeba nejvíc multitaskingu hráče.

8.1.6 Herní frakce Protossů

Velmi vyspělá rasa s velmi odolnými, ale drahými jednotkami, které se trénují delší dobu než u ostatních ras. Spoléhají se na silné energetické zbraně a štíty, které se samy regenerují. Kombinují velmi silné jednotky pro boj z blízka a rychlé jednotky pro boj na dálku. Většina jejich budov musí být postavena pouze v okolí tzv. „Pylonů“, speciálních budov přidávajících populační body a generujících energii pro ostatní stavby. Dělník nemusí být přítomen u budovy a může se vrátit ki těžbě – stavba budovy je vlastně jen teleport již hotové konstrukce na zvolené místo. U Protossů existuje velmi dobrá rovnováha mezi útokem a obranou, avšak kvůli menšímu počtu jednotek se jim může snadno stát, že je nepřítel obklíčí, hlavně v prvotních fázích hry.

Obrázek 8.1: Zergské jednotky ničí základnu a dělníky Protossů

8.2 BWAPI

K vytvoření rozhraní mezi StarCraftem a vyvíjeným frameworkem bylo použito BWAPI – Brood War Application Programming Interface [6]. Jedná se o open-source framework ve formě dynamické knihovny pro C++, ovšem existuje i verze pro Javu realizovaná pomocí JNI – Java Native Interface.

BWAPI obsahuje kompletní soubor všech tříd potřebných pro ovládání StarCraftu. A to nejen jednotek ovládaných hráčem, ale i hry samotné – například změny systémového nastavení. Primární pro účel právě je však rozhraní pro ovládání jednotek.

Implementací vzniká tzv. **bot** – samostatně běžící aplikace komunikující přes BWAPI s běžící instancí hry. Hlavní třídu bota je nutné vytvořit implementací `DefaultBWListener` třídy BWAPI, jíž dostaneme hlavní vstupní bod rozhraní.

Dědící třída je pak schopna naslouchat různým herním událostem, které na které BWAPI reaguje voláním příslušné metody:

- `onStart()` – metoda bez parametrů volaná se startem hry. Perfektní místo pro inicializaci agentů, zjištění dalších informací o hře a analýzu mapy.
- `onFrame()` – metoda volaná v při každém novém snímku (viz. 6.1.2). V této metodě je žádoucí dovolit všem agentům systému provádět akce ve hře.
- `onUnitDiscover(Unit)` – vyvolána při každé příležitosti, kdy přijde hráč do kontaktu s libovolnou herní jednotkou, jak svou, tak protivníkovou. Metoda je ideální k detekci a ukládání informací ze hry do databáze znalostí. Objevená jednotka je parametrem metody.
- `onUnitCreated(Unit)` – vyvolána pokaždé, když hráč začne s výstavbou budovy nebo výcvikem jednotky. Předává pouze vlastní jednotky, jako parametr.
- `onUnitComplete(Unit)` – vyvolána při dokončení stavby nebo výcviku hráčovy budovy jednotky. Předává pouze vlastní jednotky, jako parametr.
- `onUnitDestroy(Unit)` – vyvolána při zničení jednotky či budovy. A to i protivníkových. Zničená jednotka je předána parametrem. Ideální místo pro mazání záznamů jednotek z databáze znalostí.
- `onEnd(boolean)` – metoda indikující konec hry. Jejím parametrem je jediný `boolean` značící, zda bot ve hře vyhrál nebo prohrál. Metoda dává botovi prostor uložit si po hře jakékoliv záznamy. Právě zde se ukládají jakákoliv data o použitých strategiích, průměrných příjmech a další, která jsou relevantní pro jeho učení.

Instance třídy `Unit` reprezentující jednotky mohou mít svou pozici. Pozic jsou v BWAPI celkem dva druhy – pozice v pixelech (třída `Position`) a pozice v herních dlaždicích (třída `TilePosition`). V případě herních dlaždic je pozice vlastně indexem této dlaždice na osách X i Y a její počátek vyjádřený v pixelech je tento index vynásobený číslem třicet dva. Objekty svou pozici mohou měnit s každým snímkem hry. Pakliže nepřátelská jednotka zmizí hráčovým jednotkám z dohledu, její pozice přestává být platnou. Z tohoto důvodu je velmi užitečné mít v databázi znalostí uloženy záznamy o všech jednotkách s jejich poslední známou pozicí. Při znovuobjevení jednotky se pozice pouze aktualizuje.

Při zničení jednotek nepřestanou existovat reprezentující objekty, pouze metoda `exists()` vrací `false` místo `true`. Z tohoto důvodu je mnohem výhodnější využití herní události `onUnitDestroy()` k detekci smrti jednotky a odebrání záznamu z databáze znalostí.

8.3 Implementace bota

Testovací bot byl implementován tak, aby byl schopen hrát pouze za Protossy. Jak bylo popsáno v předchozí kapitole, každá frakce má svá specifika, kterým je třeba při implementaci čelit. Jako Protoss konkrétně se bot potýká s těmito faktory poskytujícími ze strategického hlediska výzvy:

- Detekce začátku stavby – když je Protosský dělník, tzv. *„Probe“* vyslána postavit budovy, po začátku výstavby se opět může věnovat jiným činnostem. Jde tedy především o oddělení herní události počátku a konce stavby způsobem, který neplatí pro ostatní frakce.
- Síť *„Pylonů“* – Protosská stavba poskytující populační body i energii okolním budovám. Tyto budovy je vhodné je vhodné pozicovat tak, ať dohromady tvoří energetickou síť dostatečně volnou pro průchod jednotek a s dostatkem místa pro další stavby, avšak i dost těsnou pro překrývání sfér vlivu jednotlivých Pylonů pro případ zničení, kdy by se budovy kolem něj deaktivovaly z důvodu absence energie.
- Opatrné zacházení s jednotkami – Protosské jednotky jsou drahé a je jích méně. Velmi snadno mohou být obklíčeny a zničeny. Je proto lepší útočit v mnohem sevřenějších skupinách o více členech.
- Náročnost na populaci – jednotky Protossů vyžadují pro svou konstrukci mnoho populačních bodů. To může být problémem hlavně v prvotních fázích hry, kdy hráč nemá dostatek surovin na stavbu většího množství Pylonů, přesto si musí udržovat armádu s dostatečnou silou na obranu základny.

8.3.1 Implementované mapování technologického stromu

Pro použití v příslušných strategiích bylo vytvořeno rozhraní pro tvorbu a ovládání části jednotek Protossů. Bot neumí stavět a kontrolovat všechny typy uzlů, avšak pro testování strategií jsou dostatečné. Bot umí kontrolovat a vytvářet tyto typy budov:

- **Nexus** – centrální budova schopná výcviku dělníků a fungující jako skladiště surovin. Nexus je hráči poskytnut při začátku hry.
- **Assimilator** – budova, kterou lze postavit pouze na Vespeneovém gejzíru a umožňující jeho těžbu.
- **Pylon** – jak blyo již zmíněno, budova poskytující energii pro fungování produkčních budov (kromě Nexusu a Assimilatoru) a populační body.
- **Gateway** – lze označit jako kasárna, hlavní produkční budova pro Protosské pozemní jednotky.
- **Forge** – budova schopná vylepšování Protosských pozemních jednotek a energetických štítů. Rovněž umožňuje stavbu obranných staveb.
- **Cybernetics Core** – budova schopná vylepšeování Protosských vzdušných jednotek. Mnohem dřív však slouží k odemčení Dragoonů, nejdříve dosažitelných Protosských jednotek schopných útoku z dálky, stejně jako dalších pokročilějších Protosských staveb.

- **Citadel of Adun** – budova sloužící k vylepšování základních Protosských jednotek, Zealotů.
- **Robotics Facility** – produkční stavba pokročilých Protosských jednotek.
- **Observatory** – budova umožňující stavbu pokročilých špionážních jednotek v Robotics Facility a jejich vylepšování.

Jednotky je bot schopen vycvičit a ovládat tyto:

- **Probe** – Protosský dělník. Těží minerály i Vespene a začíná konstrukci budov.
- **Zealot** – základní jednotka, k dispozici hned po postavení první konstrukční stavby (Gateway). Zpočátku pomalá, což je její hlavní slabost vzhledem ke schopnosti útoku pouze z blízka. Velmi odolná. Později lze vylepšit její rychlost v Citadel of Adun.
- **Dragoon** – V pořadí druhá Protosská bojová jednotka, schopna silného útoku na dálku. První jednotka, jejíž cena zahrnuje i Vespene. i přes výrazně vyšší cenu není odolnější než Zealot, avšak je mnohem rychlejší a její už tak dobrý dosah zbraní lze ještě vylepšit.
- **Observer** – průzkumná, vzdušná jednotka bez schopnosti útočit. Jejími hlavními přednostmi jsou maskování (bez detekce na ni nepřítel nemůže zaútočit) a detekce nepřátelských maskovaných jednotek.

8.3.2 Strategie

Na začátku hry je první starostí překonat první útok z strany výchozích herních inteligencí ve StraCraftu. Ten je vytvořen tak, aby byl unikátní pro každou frakci a znamenal pro hráče dostatečnou hrozbu. Po odražení tohoto útoku začíná fáze rychlejší expanze a technologického postupu. Tyto prvotní hrozby jsou:

- Výchozí útok Terranů – útok s necelou dvacítkou pěchotních vojáků a občas nějakým lékařem schopným uzdravovat pěchotu. Útok není sám o sobě velkou hrozbou pokud hráč zaměří lékařské jednotky, jinak může jeho odražení trvat delší dobu a hráče ekonomicky vyčerpat.
- Výchozí útok Protossů – útok s více než deseti Zealoty. Díky velké výdrži a silnému útoku Protosských jednotek, které jsou ještě navíc koncentrované na jednom místě, vyžaduje odražení útoku koncentraci sil.
- Brzký útok Zergů – také známý jako *„Zergling Rush“*, je to útok v tak brzké fázi hry, že hráč ještě nemusel stihnout postavit první produkční budovy a bojové jednotky. Útok se skládá ze šesti nebo osmi Zerglingů – levných a slabých, ale rychlých jednotek se silným útokem, kteří jsou schopni zničit hráčovy dělníky a kompletně mu ochromit ekonomiku.
- Pozdější útok Zergů – přijde později než všechny dříve zmíněné, ale zato je mnohem silnější. Armáda nepřítele se může skládat ze třiceti až čtyřiceti bojových jednotek různých druhů, s útokem na blízku i na dálku. Největší výzvou je však i účast *„Lurkerů“* na útoku – jednotky schopné maskovat se zakopáním pod zem a útoku na slušnou vzdálenost, a to i na několik jednotek hráče najednou.

Útoky Protossů a Terranů jsou si velmi podobné co se náročnosti i časování týče, ovšem útoky Zergů jsou po strategické stránce obtížnější, protože si hráč nikdy nemůže být jist, co na něj přijde a kdy. Bot je připraven na hru proti náhodnému protivníku, je však potřeba co nejdříve zjistit, kterému čelí.

K dispozici má několik implementovaných strategií.

Zealot Rush

Nejjednodušší *opening* strategie. Spoléhá na co nejrychlejší produkcí co největšího množství Zealotů, ideálně z několika produkčních budov najednou. Úkolem této strategie je co nejrychlejší vybudování základní armády a útok na nepřítele. Strategie končí výstavbou Assimilatoru pro přechod na komplexnější navazující strategii.

Dragoon Rush

Riskantnější *opening* strategie. Spoléhá se rychlý technologický pokrok k pokročilějším jednotkám, Dragoonům, pro získání převahy nad nepřítelem. Bot musí riskovat pozdější začátek produkce jednotek a vystavuje se hrozbě brzkých útoků nebo přílišné velké ztrátě času vedoucí k malé velikosti armády v době útoku nepřítele. Strategie končí vylepšením dosahu útoku Dragoonů.

Citadel

Jednodušší *standard* (navazující) strategie zaměřená na masovou produkci Zealotů a jejich vylepšení. Strategie končí vylepšením rychlosti Zealotů, kteří jsou díky němu mnohem efektivnější.

Observer

Standard strategie zaměřená na co nejrychlejší výrobu Observeru pro efektivní boj s maskovanými jednotkami. Kromě zahájení výroby Observerů se zaměřuje na výrobu Zealotů a Dragoonů během celého průběhu.

Late

Strategie určená pro dohrání hry masovou produkcí Zealotů a Dragoonů.

Kapitola 9

Zhodnocení

Tato kapitola prezentuje a hodnotí implementaci frameworku a výsledky dosažené testovacím modelem. Hodnocení se týká míry splnění cíle vytvořit obecný framework schopný mapování na konkrétní hru řízeného sociálně fungujícími agenty s možnostmi učení.

Hodnoceny jsou například schopnost agentního systému z frameworku hrát úplnou hru, tedy provádět všechny akce dostupné lidskému hráči a za jakých podmínek toho lze či nelze dosáhnout. Dále pak schopnost vyhrávat hry a jak velké zlepšení je dosaženo díky podpoře učení a adaptace agentů. Posléze je zhodnocena přínos komunikace díky JADE a vliv paralelismu na výkon a rychlost.

Jelikož se jedná o framework snažící se o implementaci obecného chování, nutné je také hodnocení náročnosti rozhraní potřebného pro spojení s konkrétní hrou a s tím související režie pro programátora.

V poslední části je pak ze všech hodnocení vytvořen závěr a jsou prezentovány myšlenky, jejichž realizace by mohla vést k dalšímu zlepšení.

9.1 Hra úplné hry

Principy a struktury popsané v kapitolách 6.2 a 6.3 je nutné pro korektní funkčnost frameworku vytvořit mapování technologického stromu a akcí konkrétní hry, stejně jako implementovat tyto akce v herních interních metodách. Zároveň je nutné vytvořit strategie obsahující použití těchto akcí a uzlů technologického stromu, aby si je framework byl schopen zasadit do kontextu a vytvořit s nimi produkční plán. Po vytvoření úplného mapování jednotek pro rozhraní jsou agenti schopni stavět všechny budovy, cvičit všechny jednotky a vyzkoumat všechna dostupná vylepšení.

Některé hry, dokonce i námi používaný StarCraft, umožňují určitým jednotkám používat speciální schopnosti, typicky s nějakým intervalem po použití než je možno ji použít znovu a obvykle i s nějakou další cenou ve formě energie či „*many*“. Podpora pro používání těchto schopností ve vyvíjeném frameworku chybí. Uživatel si ji však má možnost doplnit v implementaci taktického manažera bez zásahu do frameworku.

Pakliže tedy hra umožňuje jednotkám používat speciální schopnosti, hra úplné hry není momentálně podporována. V opačném případě je při dostatečné implementaci rozhraní možná a závisí jen na úplnosti popisu hry.

9.2 Micromanagement jednotek

Podrobněji vysvětlen v kapitole 2, pojmem *„micromanagement“* se v kontextu strategických počítačových her rozumí precizní ovládání jednotek během boje. Tím je možno dosáhnout lepších výsledků, než když jsou jednotku pouze vyslány kupředu.

Snahy o implementaci obecných procesů a algoritmů snažících se o micromanagement jednotek ve střetu s protivníkem bez návazností a závislostí na konkrétní hře se setkaly s neúspěchem. Důvodem je složitost zobecnění a obrovské nároky na množství informací, které musí dodat uživatel frameworku.

Taktický manažer, který ovládá jednotky, totiž neví nic o jejich povaze. Zda se jedná o jednotky pro boj z blízka nebo z dálky, jak jsou odolné nebo rychlé. Některé tyto informace lze vypsat, i když se jedná o velké množství dat. Mnohem větší je však režie na přidělení rolí jednotkám a kombinace armády složené z více druhů jednotek.

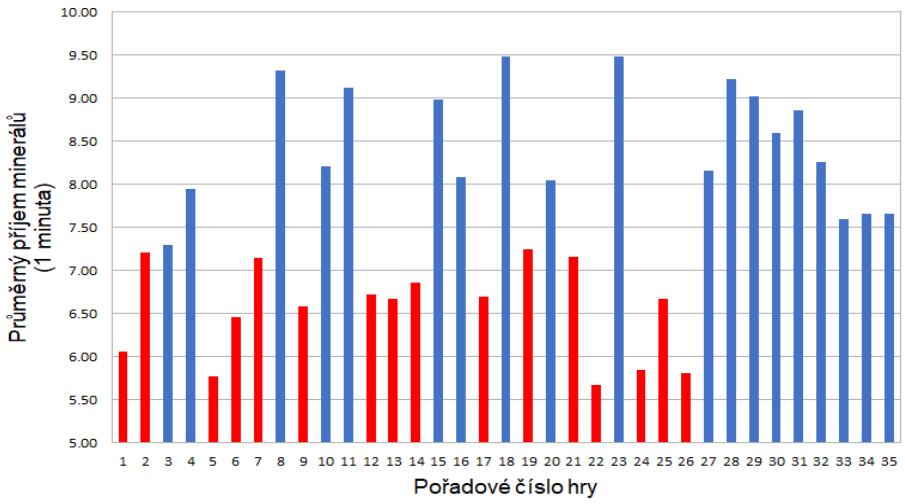
9.3 Učení a Adaptace

Pro zlepšení funkčnosti a úspěšnosti frameworkem řízených botů jsou použity dvě hlavní metody - učení bez učitele a genetické algoritmy.

9.3.1 Učení bez učitele

Učení bez učitele přineslo menší úspěch ve formě optimalizace tras používaných průzkumným manažerem. Ten byl díky výběru statisticky výhodnějších postupů pro volby příštího prohledávaného místa schopen nalézt nepřítele o 12% rychleji Tento údaj byl ověřen na dvacítce odehraných her.

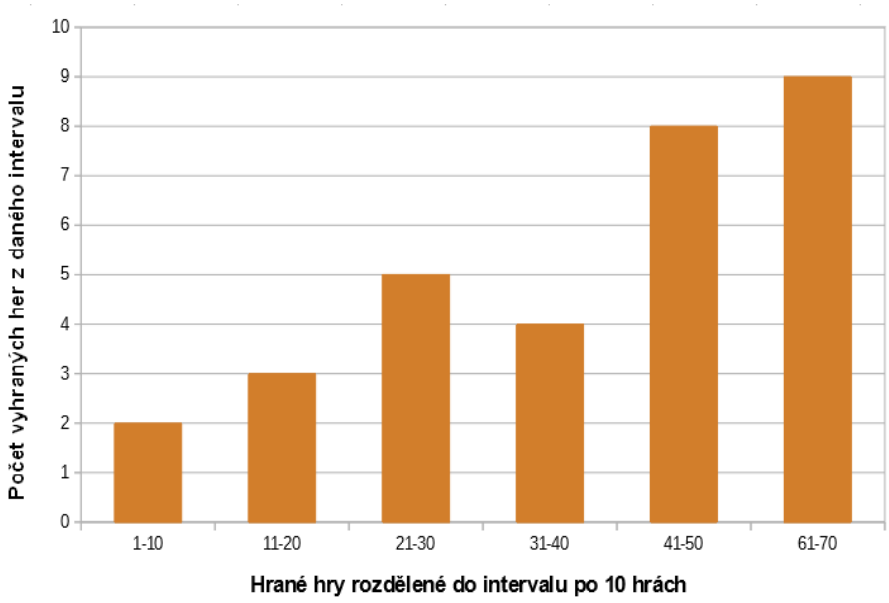
Větším úspěchem byla analýza průměrného příjmu surovin během odehraných her. Zkoumány byly hry, kdy nedošlo ke zničení nepřítele či prohře příliš brzy, tedy dříve, než se plně rozvinula hráčova ekonomika. Vynechány byly i případy, kdy naopak suroviny po delším čase došly.



Obrázek 9.1: Graf průměrných příjmů během 30 her, modře výhry a červeně prohry

Testování probíhalo přidáváním a ubíráním počtu dělníků, které bot za hru postavil, a pamatováním si výsledku hry a průměrného zisku minerálů za celou hru. Testování rozvněž probíhalo za použití pouze jedné strategie, tzv. *„Zealot rushe“*. Na grafu 9.1 jsou na ose X zaneseny jednotlivé hry započítané do statistik a na ose Y hodnoty průměrných příjmů minerálů za danou hru. Modře jsou vyznačeny případy, kdy bot vyhrál, červeně pak prohry. Je patrné a sledováním odehraných her snadno ověřitelné, že omezení ekonomiky mělo přímý vliv na úspěch bota. Bot se je tímto postupem schopen naučit hranice, které musí překonat, aby měl větší šance na výhru, a to pro každou konkrétní strategii a surovinu.

Největším úspěchem učení je schopnost výběru strategií, které se proti danému typu protivníka v předešlých hrách nejvíc oplatily. Příkladem je učení StarCraft bota proti Zergům. Započteny jsou pouze hry, kdy Zergové uplatnili strategii zahrnující Lurkery (jednotky schopné maskování a útoku ze zálohy). Bot měl k dispozici tři navazující strategie, z nichž pouze jedna obsahovala Observera (jednotku shcopnou Lurkery odhalit). Obecně lze říci, že pokud si bot zvolil jinou strategii, hru prohrál. S Observerem hry vyhrával. Graf 9.2 poměr vyhraných her, kdy na ose X jsou zaznačeny odehrané hry po desítkách a na ode Y počet vyhraných her z dané desítky pokusů. I při silné preferenci jedné strategie bot dává stále občas šanci ostatním, proto není nikdy úspěšnost stoprocentní. Je potřeba zdůraznit, že bot se tímto postupem naučil poze která strategie je výhodnější proti celé frakci. To však nic nevypovídá o situacích, kdy by Zergové zvolili jiné kompozice armád či strategie.



Obrázek 9.2: Graf rostoucího podílu vítězství s delším učením

Za graf děkuji Bc. Lukáši Válkovi, který provedl měření.

9.3.2 Genetické algoritmy

Použití genetických algoritmů mělo rovněž vliv na úspěšnost bota. Genetické algoritmy si poradily s chybami jako chybějícím cílovým uzlem ve frontě akcí nebo s místy nedostatku populačních bodů. Tyto dva problémy by však šly vyřešit i exaktním přístupem. Větším přínosem byly optimalizace produkční fronty.

Jako test bylo spuštěno dvacet her bez optimalizace a stejné množství s genetickými algoritmy. V všech testech měl bot k dispozici stejnou strategii s frontou padesáti akcí a oponenty stejných frakcí. V testech s optimalizací:

- Cílový uzel byl postaven průměrně o 2.48 akce dříve
- V době vítězství hry měl průměrně o 4,2 bojové jednotky více
- Vítězství trvala průměrně o 8% kratší dobu
- Prohry trvaly průměrně o 5,5% delší dobu
- Vyhrál o 2 hry více

Tyto výsledky ukazují, že optimalizace přinesly zlepšení. Předpokladem však je, že zvolená strategie umožní botovi vyhrávat hry i bez optimalizace. Při testech se výrazně horší strategií nedosáhl bot žádného zlepšení stojícího za zmínku.

9.4 Schopnost vyhrát

Vyvíjený framework a na něm založený model byl primárně porovnáván s výchozí herní umělou inteligencí obsaženou ve StarCraftu. Důvodem byla snaha především o vytvoření obecných procesů použitelných jako náhradu příliš jednoduchých inteligencí ve hrách.

Vytvořený bot nemá žádné problémy s výhrou nad libovolnou strategií, kterou proti němu počítačoví oponenti ve StarCraftu použijí. Během vyvíjení se dalo dosáhnout stoprocentní úspěšnosti a naopak bylo třeba při testování různých aspektů chování bota sabotovat tak, aby se výchozí inteligenci spíše vyrovnat než ji zcela překonal. Problémem však je, že jím používané strategie musí vytvořit uživatel frameworku. Tedy člověk, který přemýšlí tam, kde bot nemůže. Jeho vítězství je tedy silně závislé na tvůrcích. S nekvalitní strategií je zcela ztracen.

Při srovnání s boty vyvíjenými v soutěži SSCAIT , kteří jsou vyvíjeni delší dobu za použití expertních znalostí jejich vývojářů a za účelem hrát pouze jednu hru nemá bot řízený tímto frameworkem velkou šanci. Přesto se mu podařilo „v laboratorních podmínkách“ proti průměrným botům vyhrát zhruba třetinu až polovinu her. Proti botům vedoucím žebříčky prakticky skoro žádnou – především díky absenci micromanagementu nebo nepřátelským strategiím, na které nebyl nikdy testován.

9.5 Agenti a komunikace

Použití agentního systému přineslo možnosti abstrakce. Každý implementovaný manažer (viz. 5) odpovídá jednomu aspektu hry. Tímto je zcela odděleno například řízení boje a ekonomiky. To nemá vliv na hru jak ji vnímá hráč hrající proti herní umělé inteligenci řízené frameworkem, je to však užitečné pro vývojáře.

Komunikace mezi agenty je pak opět druhem abstrakce. Umožňuje vývojáři žádat v rámci jednoho manažera ostatní komponenty o informace a jen si pravidelně kontrolovat, zda jsou k dispozici. Sociální schopnosti agentů rovněž umožňují vývojáři delegovat úkoly na manažery, kteří by se měli problémem zabývat a nemusí je pouze žádat o informace, které pak musí vyhodnotit.

Komunikace agentů ještě navíc přináší systému rozšířené možnosti paralelizace úkolů, to bude ovšem důkladněji rozebráno v další podkapitole 9.6.

9.6 Rychlost a paralelismus

Výhodou frameworku je možnost provádění výpočtů pro hru ve vedlejších vláknech. Paralelismus je zaručen hlavně díky použití agentního frameworku JADE (viz. 6.1.1). Jednotlivá *Behaviours* agentů jsou pak každé vlastním vláknem. V těchto vláknech se realizuje například

- Komunikace agentů – dokud se jednotliví manažeři nedohodnou na řešení problému, nemusí se na výsledek této komunikace vůbec čekat a snímky hry se vypočítávají jak mají, bez zdržení.
- Hledání tras – výpočet tras průzkumu nebo bojových jednotek. Prohledávání stavového prostoru je ve hrách obvykle náročný proces, zvláště u větších skupin jednotek. Framework dokáže výpočty provádět ve vedlejších vláknech a jednotky dostanou příkaz s konkrétním cílem až je vypočten.
- Optimalizace – příkladem jsou třeba genetické algoritmy popsané v kapitole 7.2. Ty mohou optimalizovat frontu akcí produkčního manažera prakticky nepřetržitě aniž by jej zdržovaly.

V systému se stále vyskytuje mnoho procesů, které paralelně provádět nelze a nebo to přináší spíše obtíže než užitek. Příkladem je výpočet síly protivníka v taktickém manažeru, který musí probíhat na základě viditelnosti jednotek, kterou lze zkoumat pouze v hlavním vlákně hry.

Potenciální zrychlení výpočtů je omezeno jen počtem implementovaných chování agentů a výkonem stroje, na kterém hra běží. Při testování byly běžně aktivní tři až čtyři vedlejší vlákna, obvykle komunikační.

Pakliže hra poběží na systému s procesorem s jedním jádrem, komunikace a optimalizace budou průběh hry zpomalovat. Konkrétně genetické algoritmy mohou způsobit, podle svých nastavení, zásadní zdržení a dobu potřebnou pro výpočet snímku zvětšit až na zhruba dvojnásobek. V dnešní době ale s takovým scénářem nelze počítat.

9.7 Režie pro vytvoření rozhraní, obecnost

K vytvoření funkčního rozhraní a mapování akcí konkrétní hry je potřeba dodat mnoho vstupů od programátora (popsáno v kapitole 6 a v manuálu v příloze). Je potřeba nejen kompletního výčtu všech jednotek a budov, ale i implementovat metody dávající jim příkazy nebo zjišťující jejich vlastnosti.

Množství nutné režie k tomu, aby byl framework vůbec schopen ovládat, je mnohem vyšší, než se v počátcích projektu předpokládalo. S přihlédnutím k faktu, že chybí implementace dalších věcí jako ovládání speciálních schopností jednotek nebo informovanějšímu rozpoznávání nepřátelských strategií, které by nutných popisů specifických věcí ještě přidaly, může být cena za obecný framework mnohdy příliš vysoká.

Při vývoji frameworku se začalo zcela obecným návrhem. Při tvorbě testovacího modelu i na poměrně starší hru bylo od začátku potřeba do systému přidávat nové metody a zvětšovat množství metod závislých na konkrétní implementaci. Ve frameworku prakticky zbyl jen obecný způsob řízení a procesy fungující bez závislosti na konkrétní hře. To navíc platí jen pro určitý druh her.

Ačkoliv se neobešlo bez menších ohnutí pravidel pro StarCraft bota v době, kdy nebyl framework zcela dokončen, je teď schopen fungovat zcela obecně. Nebude však nikdy v žádné hře nejlepší.

9.8 Náměty na zlepšení

Tato podkapitola obsahuje některé myšlenky, které kdyby byly realizovány, mohly by zvýšit efektivitu a úspěšnost systémů řízených frameworkem.

- Učení s učitelem a klasifikace – rozpoznávání kompozic nepřátelských armád nebo určování jeho strategie podle postavených budov by mohlo zlepšit volbu strategií. Problémem je však nutnost specifikace nepřátelských strategií uživatelem jakožto tříd pro klasifikaci. Už tak je požadavků na vstupy od uživatele frameworku mnoho.
- Rozšíření genetických algoritmů – aplikace při optimalizace dokázala přinést zlepšení, ale ne dostatečné. Důraz při případném zlepšování by měl být kladen na hodnocení paralelizace akcí a validity pořadí akcí ve frontě.
- Systém micromanagementu jednotek – bot při srovnání s ostatními ze soutěže SSCAIT obecně přicházel v šarvátkách o mnohem více jednotek a byl nucen ustupovat.
- Specializace frameworku – přestat se snažit o maximálně obecné chování a zaměřit ovládání na užší profil podobných strategických her.

Kapitola 10

Závěr

Cílem této práce bylo vytvořit návrh obecného frameworku, který by umožnil vytvářet počítačem řízené protivníky pro strategické hry v podobě multiagentních systémů schopných sociálních schopností, adaptace na situaci ve hře a poučením se z předchozích her.

Výsledkem je návrh a implementace systému, který používá obecných postupů rozdělených podle zaměření a implementovaných v řídících agentech – tzv. manažerech. Každý z manažerů je odpovědný za jeden aspekt hry, například ekonomiku nebo ovládání bojových jednotek. Manažeři jsou schopni úzké spolupráce díky sociálním schopnostem umožňujícím konzultaci strategií, žádání o zdroje či jednotky nebo sdílení informací. Někteří z těchto speciálních agentů jsou schopni učení za účelem volby nejlepších strategií, jiní optimalizují pořadí jimi prováděných akcí použitím genetických algoritmů.

Systém potřebuje od programátora specifikaci akcí, herních entit a strategií, které jsou mu v dané hře k dispozici. Rovněž pak vyžaduje implementaci specifických abstraktních metod, které poskytují informace nejen o jednotkách a budovách z konkrétní hry obecným algoritmům.

Pomocí námi vytvořeného frameworku se nám povedlo, jako model pro ověření funkčnosti, vytvořit herní umělou inteligenci pro hru StarCraft: Brood War od společnosti Blizzard Entertainment. Tento *„bot“* je schopen efektivně zacházet se všemi entitami hry definovanými v jeho rozhraní a vyhrávat nad výchozími počítačem ovládanými protivníky ve hře.

Tímto byly cíle práce splněny. Prezentované výsledky testovacího modelu osobně považuji za uspokojivé, a to vzhledem k tomu, že systém je schopen díky rozhraní fungovat obecně bez návaznosti na konkrétní hru, poučit se ze svých akcí a vyhrávat. Oproti původním plánům jsme však museli upustit od několika rozšíření, například od analýzy protivníkových tahů a strategií neuronovými sítěmi, kvůli složitosti implementace rozhraní konkrétních her.

Složitost a rozsah rozhraní potřebného pro ovládání konkrétní hry obecnými algoritmy a postupy však hodnotím jako nevyhovující a s každou další funkcionalitou by se jen zvětšovaly. Vytvoření zcela obecného systému je možné, ale daň za tuto obecnost je příliš vysoká než aby si našla širší uplatnění. Výsledný systém je použitelný a konkurenceschopný, ale pouze pro méně komplexní hry.

Literatura

[1] Antonio Gusmao, T. R.: *Reinforcement Learning In Real-Time Strategy Games. [Online; navštíveno 14.02.2018].*
URL https://pdfs.semanticscholar.org/3853/d7ced9603d93e977faeef496ff91a2c18052.pdf

[2] Churchill David, M. B.: Build Order Optimization in StarCraft. [Online; navštíveno 14.02.2018].
URL https://pdfs.semanticscholar.org/dfd9/1e739bd979c08485a75fd11c501a6ec05118.pdf

[3] Daylamani-Zad, D.: Swarm intelligence for autonomous cooperative agents in battles for real-time strategy games. [Online; navštíveno 17.11.2017].
URL https://www.researchgate.net/publication/320254420_Swarm_intelligence_for_autonomous_cooperative_agents_in_battles

[4] Harshit Sethy, V. P., Amit Patel: Real Time Strategy Games: A Reinforcement Learning Approach. [Online; navštíveno 4.12.2017].
URL
https://www.sciencedirect.com/science/article/pii/S187705091501354X

[5] Ian Watson, Y. C. W. P. G. C., Damir Azhar: Optimization in Strategy Games:Using Genetic Algorithms to Optimize City Development in FreeCiv. [Online; navštíveno 14.02.2018].
URL https://www.cs.auckland.ac.nz/research/gameai/projects/GA%20in%20FreeCiv.pdf

[6] Čertický M., C. D.: The Current State of StarCraft AI Competitions and Bots. [Online; navštíveno 15.12.2017].
URL https://sscaitournament.com/

[7] Marc Ponsen, P. S., Héctor Muñoz-Avila; Aha, D. W.: Automatically Generating Game Tactics through Evolutionary Learning. [Online; navštíveno 2.12.2017].
URL https://www.aaai.org/ojs/index.php/aimagazine/article/viewFile/1894/1792

[8] MCCoy Josh, M. M.: An Integrated Agent for Playing Real-Time Strategy Games. [Online; navštíveno 26.10.2017].
URL https://www.aaai.org/Papers/AAAI/2008/AAAI08-208.pdf

[9] Messerschmidt, L.: Using Particle Swarm Optimization to Evolve Two-Player Game Agents. [Online; navštíveno 14.02.2018].
URL http://arimaa.com/arimaa/papers/LeonMesserschmidt/thesis.pdf

[10] *Olivier Bartheye, E. J.: A PDDL-Based Planning Architecture to Support Arcade Game Playing. [Online; navštíveno 29.11.2017].*
URL https://web.stanford.edu/class/cs227/Readings/PDDLPlanningForArcadeGamePlaying.pdf

[11] *Saket Joshi, R. K.: Relational Markov Decision Processes: Promise and Prospects. [Online; navštíveno 18.11.2017].*
URL https://www.aaai.org/ocs/index.php/WS/AAAIW13/paper/viewFile/7168/6633

[12] *Wei-Bing Liu, X.-J. W.: An evolutionary game based particle swarm optimization algorithm. [Online; navštíveno 14.02.2018].*
URL
https://www.sciencedirect.com/science/article/pii/S0377042707000799

[13] *Zbořil, F.: Podklady k předmětu AGS - III. Formální logiky pro agentní a multiagentní systémy. [Online; navštíveno 6.1.2018].*
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FAGS-IT%2Flectures%2FAGS15_03.pdf&cid=10764

[14] *Zbořil, F.: Podklady k předmětu AGS - VI. Multiagentní systémy. [Online; navštíveno 6.1.2018].*
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.