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

Diplomová práce

Použití metod multiagentních systémů pro implementaci umělé inteligence v real-time strategiích

Plzeň 2016 David Fiedler

Prohlášení

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

V Plzni dne 12. května 2016

David Fiedler

Abstract

This work focuses on the use of multi-agent systems (MAS) for implementation of Artificial Intelligence (AI) for Real-time strategy games (RTS). As part of the work, the game StarCraft has been chosen as a platform for developing the multi-agent AI. The AI has been implemented as a framework, so other developers of AI for RTS can build on this work. According to the collected resources, implementation of this framework is a new step in the field of applying the multi-agent principles in developing AI for RTS.

Abstrakt

Tato práce se věnuje využití multiagentních systémů (MAS) při vývoji umělé inteligence (AI) do real-time strategií (RTS). V rámci práce byla vybrána hra StarCraft jako platforma pro vývoj AI s prvky MAS. Tato AI byla implementována ve formě frameworku tak, aby na ni mohli navazovat další vývojáři umělé inteligence pro RTS. Implementace tohoto frameworku je, alespoň dle shromážděných zdrojů, novým krokem v oblasti uplatnění multiagentních principů při vývoji AI do RTS.

Obsah

Slovník

ABCD Alpha-Beta Considering Durations – modifikace alfa-beta ořezávání uvažující časové závislosti. 18

ABL A Behavior Language – programovací jazyk pro reaktivní řízení popsaný v práci Michaela Matease a Andrewa Sterna. 15

AI Artificial Inteligence - umělá inteligence. 1, 2, 4–11, 13–43, 45, 46, 48, 52–54, 58, 59

AIIDE AI for Interactive Digital Entertainment conference – konference zabývající se umělou inteligencí, případě turnaj ve hře StarCraft pořádaný u příležitosti této konference. 14, 35

API Application programming interface – rozhraní pro přístup k aplikaci. 31, 33, 34, 36, 42

ASP Answer Set Programming – deklarativní programování určené k řešení výpočetně složitých problémů. 21

BWAPI Brood War API – rozhraní pro přístup ke hře StarCraft: Brod War. 31–34, 36, 42, 44, 52, 53

BWTA Brood War Terain Analysis – nástroj pro analýzu terénu pro hru StarCraft: Brood War. 20, 33

BWTA2 pokračování BWTA. 33, 42

CBP Case-Based Planning – viz CBR. 16

CBR Case-Based Reasoning – rozhodování na základě zkušeností z podobných případů v minulosti. 16–18

DEACCON Decomposition of Enviroments for the Creation of Convex-region Navigation-meshes – algoritmus pro dělení prostoru na oblasti. 20

EIS Enviroment Interface Standard – standard pro interakci agentů s prostředím. 36

FOW Fog of war – válečná mlha. 6

FPS First-person Shooter – „střílečka“. 5, 17, 36

FSM Finite-state machine – konečný automat. 13

GDA Goal-Driven Autonomy – rozhodovací metoda založená na samostatné volbě cílů. 15, 16, 35

HFSM Hierarchical finite-state machine – hierarchický konečný automat. 13

HMM Hidden Markov Models – skryté Markovovy modely. 17

HSMM Hidden semi-Markov Models. 17

IMTrees Influence Map Trees – mapy vlivu organizované ve stromové struktuře. 14

JNI Java Native Interface – rozhraní umožňující propojení programů běžících ve virtuálním stroji Javy a mimo něj. 32, 42

JVM Java Virtual Machine – virtuální stroj pro jazyk Java. 42

MAS Multi-agent System/s - multiagentní systém/y. 1, 2, 4, 5, 11, 12, 26, 27, 35–40, 42, 43, 52, 58

MCTS Monte Carlo tree search – prohledávání rozhodovacího stromu s využitím heuristiky na principu metody Monte Carlo. 15, 18

MIT Massachusetts Institute of Technology – prestižní technická univerzita v USA. 26

ORTS Open Real-time Strategy – open-source RTS určená pro vývoj AI. 24, 25, 27, 29

OSS open-source software – otevřený software. 23

RL Reinforcement Learning – zpětnovazebné učení. 18

RTS Real-time Strategy – Real-time strategie. 1, 2, 4–9, 11, 13, 14, 16–19, 22, 24–28, 33, 35, 40, 41, 58

SSCAIT Student StarCraft AI Tournament – turnaj pro AI do hry StarCraft. 23

UCT Upper Confidence Bound – součást/varianta MTCS. 18

1 Úvod

Tato práce se zabývá využitím multiagentních systému (MAS) pro implementaci umělé inteligence (AI) v real-time strategiích (RTS). Účelem práce je prostudovat koncepty AI v RTS a současný stav vývoje v této oblasti, dále prozkoumat dostupné technologie pro vývoj AI pro RTS, na základě získaných poznatků pak navrhnout řešení na principu MAS, které bude vhodným rámcem pro další vývoj AI a toto řešení následně implementovat. Již v úvodu této práce jsou zmíněny některé zásadní pojmy především z oblasti umělé inteligence. Význam těchto pojmů je uveden v kapitole 2.

1.1 Motivace

Real-time strategie jsou jedním z předních žánrů počítačových her. Díky rozlehlosti herního prostoru, délce partií a velkému počtu jednotek jsou tyto hry velkou výzvou pro tvorbu umělé inteligence.

Multiagentní systémy jsou počítačové systémy složené z více inteligentních agentů, kteří mezi sebou interagují. Slouží k řešení úkolů, které samotný agent nebo obecně program řešit neumí, nebo v případech, kde by návrh monolitických systémů řešících daný úkol byl obtížný.

Jedním z takových úkolů je návrh umělé inteligence do real-time strategií. Jedná se o aplikaci, kde se naplno využije vlastností multiagentních systémů jako distribuovanost nebo autonomie agentů. Navíc je zde zavedení agentů přirozeným procesem, neboť velká část agentů bude mít svou reprezentaci ve hře ve formě jednotek. Celkový přehled důvodů, které vedly k implementaci AI do RTS pomocí MAS, se nachází v kapitole 3.4.

1.2 Cíle práce

Hlavní cíle práce jsou:

1. Prozkoumat základní koncepty AI v RTS: Zejména se seznámit s rozdíly mezi RTS a běžnými hrami ve vztahu k AI (kapitola 3.1), určit specifické požadavky na AI v RTS (kapitola 3.2) a rozdělit úlohy AI v RTS do několika oblastí (kapitola 3.3).

2. Prostudovat jednotlivé oblasti vývoje AI pro RTS: Tedy prozkoumat techniky existující v jednotlivých oblastech AI v RTS a jejich současný stav vývoje (kapitola 4).

3. Seznámit se s Platformami a nástroji pro tvorbu AI pro RTS: Nejprve prozkoumat dostupné platformy pro vývoj AI pro RTS a pomocí srovnání dospět k výběru vhodné platformy pro vývoj (kapitola 5). Následně představit podrobněji vybranou platformu, její vlastnosti a na ní dostupné nástroje (kapitola 5.2).

4. Seznámit se s existujícími aplikacemi MAS v RTS (kapitola 6): Tedy rozebrat nalezené teoretické práce popisující MAS v AI pro RTS (kapitola 6.1) a dále analyzovat vybrané existující AI pro RTS ve vztahu k MAS (kapitola 6.2).

5. Pro vybranou platformu navrhnout a implementovat robustní rozšiřitelnou AI na principu MAS (kapitola 7).

6. Vytvořit demonstrátor pro implementovanou AI a ten prezentovat (kapitola 7.9).

2 Základní pojmy

V této kapitole jsou vysvětlené stěžejní pojmy týkající se především zadání práce. Velká část pojmů z oboru umělé inteligence je bohužel velmi rozmělněná. Proto zde nejsou uvedeny exaktní definice (neboť ty v pracích buď uváděny nejsou, případně se v různých pracích navzájem popírají), ale spíše neformální ukotvení významu pojmů.

2.1 Agenty

Pojem agent je jedním z pojmů, o jehož významu nepanuje mezi odborníky shoda [1, 2]. Například podle Russella a Norviga je agent entita, která vnímá prostředí skrze senzory a ovlivňuje ho pomocí efektorů [3]. Podle Wooldridge [4] mají agenty dvě zásadní vlastnosti: jsou autonomní, a navzájem interagují. Maes [5] rozšiřuje první z definic o autonomii, cílevědomost.

Sjednocením všech tří definic získáme význam pojmu, jaký je použit v této práci. Pod pojmem agent se zde rozumí entita, která má následující vlastnosti:

- **Cílevědomost:** Agent má svoje cíle, které určují jeho jednání.
- **Autonomie:** Agent jedná samostatně, není přímo ovládán jinou entitou. Jeho chování se tedy zvnějšku nedá jednoznačně předvídat.
- **Interakce:** Agent interaguje s prostředím i s dalšími přítomnými agenty.
- **Lokalita:** Tato vlastnost v žádné ze tří definic není, ale přímo z nich vyplývá: Získává-li agent informace o prostředí skrze senzory, vytváří se z těchto informací jeho vlastní – lokální pohled na prostředí. O prostředí tedy nemá kompletní informaci.

Agenty tvoří rozmanitou skupinu, od nejjednodušších reaktivních agentů, jejichž chování je pouhou reakcí na změnu prostředí až po komplexní uvažující agenty [6].

2.1.1 Pojem agent v českém jazyce

Slovo agent je v celé práci skloňováno podle vzoru hrad. Je tomu tak proto, aby se zdůraznila neživá podstata, a pojem se tak odlišil od živých agentů (lidí). Jedná se o doporučený postup Ústavu pro jazyk český v těchto případech [7], který se používá i v některých publikacích [6].

2.2 Multiagentní systémy

Multiagentní systémy (dále MAS) jsou počítačové systémy složené z více agentů (viz kapitola 2.1), kteří spolu interagují [4]. Tyto agenty interagují jak se sebou navzájem, tak s prostředím, ve kterém se pohybují. Z toho, že se systém skládá z agentů, vyplývá také to, že jsou decentralizované, neboť agenty jsou autonomní. Podstatným rozdílem oproti jiným systémům je jejich nepredikovatelnost, nelze totiž předvídat stav systému po určitém čase, neboť ten záleží na interakcích agentů v systému.

Obvykle spolu agenty v systému také spolupracují a společnými silami řeší problémy, které by byly pro jednoho agenta nezvládnutelné. Takovým MAS říkáme kooperativní MAS [8]. Pokud bude v této práci dále zmíněn pojem MAS, myslí se tím automaticky kooperativní MAS.

2.3 Real-time strategie

Strategické počítačové hry jsou takové počítačové hry, ve kterých pomocí manipulace mnoha objektů v herním prostoru (mapě) dosáhneme vítězství. Zaměřují se zejména na strategické, taktické, ekonomické nebo logistické výzvy.

Real-time strategie (dále RTS) jsou subžánrem strategických her. Jejich hlavním znakem je, že se odehrávají v reálném čase, tedy že herní čas běží bez ohledu na to, zdali hráč vykonává nějaké akce. Většinou se jedná o válečné strategické hry, kde je cílem hry nashromáždit zdroje, postavit dostatek jednotek a s pomocí nich pak zničit jednoho nebo více protivníků.

Známými zástupci žánru RTS jsou např. herní série WarCraft[1], Command and Conquer[2], StarCraft[3] nebo Age of Empires[4].

2.4 Umělá inteligence

Podle Russella a Norviga [3] je umělá inteligence (Dále AI – Artificial Intelligence) obor zabývající se studiem agentů (kapitola 2.1) pohybujících se v prostředí, vnímajících a jednajících. Podle McCarthyho [9] je to pak věda zabývající se tvorbou inteligentních strojů, především inteligentních počítačových programů.

[1] `http://us.blizzard.com/en-us/games/war3/`
[2] `http://www.ea.com/pc/strategy/cnc`
[3] `http://us.blizzard.com/en-us/games/sc/`
[4] `http://www.ageofempires.com/`

V rámci této práce je pojem AI použit nikoliv jako odkaz na vědní obor, ale právě na stroje z McCarthyho definice, tedy inteligentní počítačové programy. K tomuto účelu jsou kromě tohoto pojmu [10, 11, 12, 13] používány v rámci komunity i další: bot [10, 14, 11, 15], nebo agent [12, 16, 11]. Určit, který termín je nejpřesnější je obtížné až nemožné (již pohled na reference použití pojmů ukazuje, že jsou tyto pojmy používány ve stejném významu i v rámci jedné práce!).

Termín bot (také video game bot – bot do počítačové hry), který vznikl ze slova robot, nebyl v této práci použit z toho důvodu, že se často používá v kontextu FPS (First-person Shooter – česky „střílečka“), ve významu počítačem ovládané postavy. Počítačový program se zde ztotožňuje se svou virtuální reprezentací. V RTS ale program ovládající hráče svoji virtuální reprezentaci nemá, vykonává totiž veškeré činnosti jednotek, budov, diplomacie atd., použití pojmu bot by tak mohlo být matoucí.

Termín agent (také game agent, AI agent) je pro použití v této práci také nevhodný. Vyskytuje se zde totiž v jiném významu, a to jako součást MAS, který má být dle zadání práce použit v navrženém řešení. Agent by tak obsahoval další agenty, což by přehlednosti práce jistě neprospělo.

Jak je vidět, jsou pojmy bot a agent pro reprezentaci inteligentního stroje nevhodné. V neposlední řadě byl termín AI vybrán proto, že je použit již v zadání práce. Bude-li tedy dále používán termín AI, myslí se tím inteligentní počítačový program.

2.5 Framework

Framework je programový rámec určený jako základ pro vývoj dalších aplikací v dané oblasti. Oproti knihovně, která je taky určená pro volání z jiných programů, se framework zaměřuje na celkové řešení konkrétního problému. Nemá být tedy v aplikaci využit pouze místně, ale má tvořit základ – jádro navazující aplikace. Na rozdíl od knihovny také framework přebírá kontrolu nad aplikací a sám volá její kód. Framework tedy není totožný s knihovnou, ale může knihovny obsahovat (a obvykle i obsahuje).

Framework by měl být rozšiřitelný aniž by bylo nutno modifikovat jeho zdrojový kód. Příkladem frameworku v Javě může být Spring, z ostatních jazyků například Symfony pro PHP nebo Ruby on Rails pro Ruby.

Tento pojem se v práci nejprve opakovaně objevuje v teoretické části, kde jsou představeny různé frameworky vytvořené v oblasti AI. Dále je AI prezentovaná jako výsledek této práce vytvořena jako framework.

3 Pozadí práce

3.1 Rozdíly mezi RTS a tradičními hrami

Mezi RTS a tradičními hrami jako šachy nebo go, pro které již existuje umělá inteligence schopná porazit profesionální hráče [17, 18], je několik podstatných rozdílů. Tyto významné rozdíly, stejně jako široká komunita amatérských a profesionálních hráčů, naznačují možnost, že poražení profesionálních hráčů v RTS bude dalším milníkem v celkovém vývoji AI [19].

Přímo fakt, že se hra odehrává v reálném čase, má následující důsledky [20]:

- **Souběžnost:** Všichni hráči hrají najednou, nečekají na sebe. Z toho vyplývá nutnost okamžitých rozhodnutí: hráč, který s rozhodnutím vyčkává, se dostává do nevýhody.
- **Spojitost:**
  - **Časová:** Čas běží bez přestání[1], většina akcí zabere nějaké množství herního času, přechod mezi stavy objektů nebývá okamžitý.
  - **Prostorová:** Některé akce (pohyb, útok, výstavba) je možné provádět ve spojitém prostoru[2], vybrané prostory těchto akcí se mohou dokonce překrývat (oproti šachu, kde jsou pole na herním plánu jasně oddělené).

Většina RTS má také tyto vlastnosti [20]:

- **Částečná viditelnost:** Území mimo dohled vlastních jednotek je pro hráče neviditelné, navíc v místech, kam se ještě nedostaly jednotky hráče, chybí informace o terénu. Tomuto efektu se říká válečná mlha – Fog of war (FOW).
- **Nedeterminismus:** Velké množství akcí ve hře je nedeterministických, tedy jejich výsledek podléhá určitému pravděpodobnostnímu rozdělení.
- **Rozloha:** Počet jednotek (je možné chápat jako obdobu figur v šachu) je výrazně vyšší, vyšší bývá i počet proveditelných akcí.

[1] Ve skutečnosti je herní čas diskrétní, ale pro člověka se jeví jako spojitý, např. ve hře StarCraft se stav hry mění 24x za sekundu (při standardní rychlosti hry).

[2] prostor samozřejmě spojitý není, ale hráčům se tak jeví. Např ve hře Starcraft se na běžné mapě pohybuje počet pozic na které může jednotka vstoupit kolem 250 000.

3.1.1 Složitost

Podstatnou vlastností RTS plynoucí částečně z vlastností uvedených výše je jejich vysoká složitost. Ta je způsobena spojitým časem (narůstá počet tahů) a velkým počtem možných akcí (díky spojitému prostoru a většímu množství akcí i jednotek).

Pokusíme se nyní podle Synnaeva [21] srovnat složitost klasických her šachy a go s RTS StarCraft. Začneme tím, že si představíme celou hru jako strom, kde uzly jsou herní stavy a přechody jsou přechody mezi herními stavy. Stanovíme veličinu b jako počet potomků každého uzlu (branching factor). Dále stanovíme veličinu d jako hloubku stromu (depth). Ve vztahu ke hře zde b reprezentuje počet možných variant v každém tahu a d počet tahů.

Dále víme, že počet prvků ve stromu se rovná b^d. Toto číslo pak můžeme považovat za složitost hry. Pro šachy platí $b \approx 35$ a $d \approx 80$. Pro go pak platí $b \approx 30$–300 a $d \approx 150$–200.

Abychom mohli srovnávat, musíme určit tyto hodnoty i pro StarCraft. Zde již budeme postupovat jinak než Synnaeve, neboť ten složitost počítal podle množství akcí prováděných lidskými hráči. Zde se pokusíme odhadnout složitost pro AI, která může manipulovat všemi jednotkami zároveň, frekvence tahů je pak omezena pouze frekvencí herních obrátek. Budeme postupovat podobně jako v [20], přičemž faktor b a tedy i celý výsledek se zde liší kvůli rozdílné metodice určení počtu možných akcí.

Nejprve se zaměříme na počet variant v každém tahu b. Zde budeme uvažovat pro jednoduchost jen několik základních akcí pro každou jednotku: pohyb, útočný pohyb a stráž (patrol). Tyto akce mají zvláštní význam, neboť je můžeme cílit na každou plochu na mapě, kam může jednotka vstoupit (akci patrol dokonce cílíme na dva body, mezi kterými jednotka hlídá). Běžná mapa pro StarCraft má například 128×128 polí pro stavbu, na každém z těchto polí je pak 16 pozic, kam může jednotka vstoupit. Nebudeme-li uvažovat, že některé oblasti mohou být nepřístupné, vyjde nám počet pozic, mezi kterými vybíráme jako $p = 128 \cdot 128 \cdot 16 \approx 250000$. Když uvážíme 4 možné pozice, vyjde nám pro každou jednotku počet možných akcí v jeden okamžik $a = 10^6$. Průměrný počet jednotek lze odhadnout jako $u = 50$–200, přičemž akce jednotky výrazně nezmenší počet možností pro akce ostatních jednotek. Výsledný počet variant b tak vyjde $b = a^u = 10^{300}$–10^{1200}.

Toto číslo je na první pohled „astronomicky“ vysoké. O střízlivější odhad se pokouší v práci [20], kde uvažují možnost pohybu v každý okamžik jen v osmi směrech v okolí jednotky. Na druhou stranu ve zmíněné práci uvažují větší množství možných akcí pro každou jednotku. Výsledný faktor b odhadují na $b = 10^{50}$–10^{200}. Podle tohoto střízlivého odhadu je tak faktor b pro hru StarCraft stále „nepředstavitelně“ větší než pro hru go.

Výpočet hloubky d můžeme uvažovat dle délky hry a snímkovací frekvence [20]. Typická hra trvá 25 minut, snímkovací frevence je 24 snímků za sekundu, z čehož získáme $d = 25 \cdot 60 \cdot 24 \approx 36000$.

Přehledné srovnání můžeme vidět v následující v tabulce 3.1.

hra	$\boldsymbol{b}$	$\boldsymbol{d}$	**složitost**
šachy	35	80	$\approx 10^{125}$
go	30–300	150–200	$\approx 10^{220}$–10^{500}
StarCraft	$10^{300} - 10^{1200}$	36000	$\approx 10^{10000000}$–$10^{40000000}$
StarCraft[1]	$10^{50} - 10^{200}$	36000	$\approx 10^{1800000}$–$10^{7200000}$

[1] Střízlivý odhad podle [20].

Tabulka 3.1: Srovnání složitosti klasických her a RTS

3.2 Požadavky na AI v RTS

Jak se ukázalo, rozdíly mezi RTS a klasickými hrami jsou značné. Tyto odlišnosti kladou na umělou inteligenci nové požadavky zejména pak [22]:

- **Práce se zdroji:** Hráči musí shromáždit herní zdroje, za které si poté pořizují jednotky, budovy, případně vylepšení pro poražení nepřítele. Důležitá je jak schopnost získávání zdrojů, tak správné hospodaření se získanými zdroji.
- **Schopnost rozhodnout se s neúplnými informacemi:** Hráči se musí rozhodnout, aniž by znali úplný stav hry, tedy například rozmístění nepřátelských jednotek, pozice nepřátelských základen, ale třeba i umístění zdrojů na mapě. Také je třeba neustále získávat alespoň část informací pomocí průzkumu.
- **Uvažování o prostoru a čase:** Analýza herního území a schopnost najít cestu mezi jeho částmi je nutnou součástí AI pro RTS. Rovněž schopnost znát časové souvislosti mezi jednotlivými akcemi je velmi důležitá.
- **Spolupráce:** Každý hráč obvykle kontroluje v RTS mnoho jednotek zároveň. Spolupráce těchto jednotek a schopnost koordinovat jejich akce je nezbytná pro dosažení vítězství.

- **Učení:** Schopnost učit se sice není nezbytnou součástí AI (ani AI pro RTS), nicméně výrazně zvyšuje její úspěšnost. Zvláště proti lidským hráčům (kteří mají schopnost učit se) je učení klíčovou vlastností, neboť tito jinak brzy odhalí slabé stránky umělé inteligence a tomu přizpůsobí svůj styl hry.
- **Plánování:** Na rozdíl od klasických her, v RTS není možné plánovat ve smyslu atomických akcí, neboť stavový prostor takové úlohy by byl příliš velký. Namísto toho je třeba zvolit vhodnou abstrakci, která zmenší počet možností tak, aby bylo možné provádět výpočet v reálném čase.

3.3 Dělení AI na oblasti

Ať už pro vyjasnění terminologie, nebo pro zjednodušení návrhu, rozděluje se obvykle AI na několik částí.

3.3.1 Rozdělení podle lidských hráčů

Lidští hráči obvykle rozdělují svoje rozhodnutí na dvě oblasti [10]:

- **Micro** (management): Schopnost ovládat individuální jednotky a budovy. Klíčová je zde rychlost. Hráč dobře zvládající micro obvykle ztrácí v boji méně jednotek, případně vyhrává početně vyrovnané souboje.
- **Macro** (management): Schopnost shromáždit zdroje a vybudovat silnou armádu. Dobrý makro hráč má obvykle větší armádu a více základen a zdrojů.

3.3.2 Obvyklé rozdělení při tvorbě AI

Při tvorbě AI se obvykle používá rozdělení na tyto oblasti [10]:

- **Strategie:** Nejvyšší úroveň abstrakce rozhodování. Soustřeďuje se zejména na dlouhodobé cíle. Zahrnuje celý soubor jednotek vlastněných hráčem, tedy hráč má jednu strategii. Příkladem strategie je tzv. *rush*, kdy jsou všechny jednotky a zdroje hráče soustředěny na rychlý útok co možná nejdříve, naopak dlouhodobá perspektiva je ignorována.
- **Taktika:** Zabývá se aplikací strategie. Soustřeďuje se na střednědobé cíle. Obvykle zahrnuje skupiny jednotek, každá skupina může používat jinou taktiku. Příkladem může být útok skupiny jednotek na jednu ze základen protihráče.

- **Reaktivní řízení:** Zabývá se aplikací taktiky. Soustřeďuje se na krátkodobé cíle. Na této úrovni vykonává obvykle každá jednotka akce samostatně. Typickým příkladem je útok na konkrétní jednotku, pohyb, nebo ústup.

- **Analýza terénu:** Analyzuje terén mapy, hledá důležité body, jako oblasti se zdroji, oblasti vhodné pro obranu, možné lokace nepřátelských základen atd. Tato oblast AI je velmi důležitá, neboť správné informace o mapě vedou k rychlejšímu nalezení surovin potřebných k vybudování armády na zničení soupeře, tedy k lepší strategii. Dále pak analýza terénu slouží například k vytipování vhodných míst k útoku nebo k obraně, což poskytuje taktickou výhodu.

- **Sběr informací:** Shromažďování informací o všech hráčích a okolním světě. Pomáhá odhalit strategii oponenta a správně na ni reagovat. AI, která dobře zvládá tuto část např. má informaci, kde se nachází základna nepřítele, zdali prohrává nebo vyhrává, nebo jaké bojové jednotky je nejlépe vyrábět.

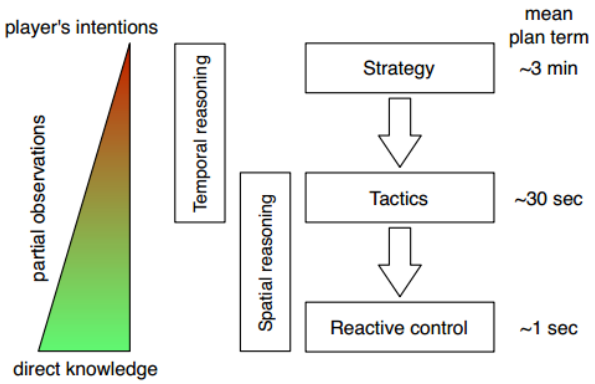
Na obrázku 3.1 můžeme vidět souvislost strategie, taktiky a reaktivního řízení s průměrným časem, na který je třeba pro danou oblast plánovat kroky. Zatímco reaktivní řízení plánujeme zhruba 1 sekundu dopředu, strategii je třeba plánovat i na několik minut napřed. Navíc je zde vidět i souvislost s množstvím známých informací. Při rozhodování reaktivního chování máme k dispozici téměř kompletní znalosti, u taktiky je třeba řadu věcí předvídat, plánování strategie pak má k dispozici jen minimum spolehlivých informací, a spoléhá se spíše na odhad záměrů hráče.

3.3.3 Rozdělení podle centralizace

Kromě výše uvedených rozdělení můžeme oblasti AI rozdělit podle míry centralizace. Řešení určité oblasti AI může být:

- **Centralizované:** V takovém případě probíhá pro jednu oblast/funkci AI centralizovaný výpočet, který vypočte výsledné akce pro všechny prvky v systému. Tedy například v případě pohybu vypočte cílové souřadnice všem jednotkám. Tento systém je výpočetně složitější, ale může dosáhnout optimálního řešení. Návrh může být velmi složitý, až nemožný, proto se i zde často vzdáváme optimality zjednodušením modelu.

- **Decentralizované:** Při decentralizovaném řešení počítá každý prvek systému svoje akce sám. Při pohybu tedy každá jednotka spočte svoje



Obrázek 3.1: Vztah strategie, taktiky a reaktivního řízení. Vlevo vidíme vztah těchto oblastí vzhledem k množství známých informací – od přímo dostupných znalostí (direct knowledge) po odhad záměrů protihráče (player's intentions). Vravo je vidět vtah vzhledem k časovému úseku, na který je třeba plánovat. Uprostřed je pak vidět přechod od rozhodování v prostoru (Spatial reasoning) k rozhodování v čase (Temporal reasoning) [20].

cílové souřadnice. Tím se vzdáváme optimálního řešení, složitost výpočtu však klesá, navíc jednodušší a robustnější bývá i návrh [23].

Obvykle je systém kombinací obou přístupů, tedy některé jeho části jsou centralizované, jiné decentralizované. MAS jsou z principu decentralizované, jakákoliv AI založená na MAS by tedy měla používat spíše decentralizované systémy.

3.4 Myšlenka implementace AI do RTS pomocí MAS

Po obeznámeni se základními problémy vývoje AI pro RTS je vhodné představit podrobně důvody, které vedly k výběru implementace AI do RTS právě pomocí MAS. Většina těchto důvodů plyne přímo z vlastností MAS:

- **Agent jako jednotka v RTS:** Reprezentace každé jednotky ve hře pomocí agenta se jeví jako přirozený a přehledný návrh.
- **Decentralizace:** Decentralizace sice způsobuje vyšší nároky na výkon (každý agent musí vykonat svůj kód), kód je ale přehlednější, neboť je rozdělen dle kompetencí jednotlivých agentů.

- **Lokalita:** Tím, že mají agenty pouze lokální informace se vzdáváme optimálního řešení, neboť na to by každý agent musel mít přístup ke všem informacím. Na druhou stranu tím přirozeně zmenšujeme prostor parametrů rozhodování, které je tak snazší. Navíc ve většině případů agent k rozhodování potřebuje jen podrobné informace ze svého okolí a malé množství abstraktních globálních informací, které mu mohou být předány prostřednictvím komunikace.
- **Podobnost s reálnou vojenskou organizací:** Reprezentace pomocí agentů umožňuje uplatnit principy podobné těm, které se používají v reálném vojsku, jako rozdělení do bojových skupin nebo plnění taktických, případně strategických cílů. Když uvážíme použití hierarchického MAS (který byl nakonec i vybrán), nabízí se možnost napodobit i léty prověřený způsob armádního velení.

4 Existující techniky pro vývoj AI do RTS

V RTS AI už nyní existuje mnoho technik pro pokrytí různých oblastí jako je strategie, taktika a další, a dále podoblasti těchto oblastí. Protože je jedním z cílů práce vytvořit univerzální framework pro RTS (tedy takový, který má potenciál pokrýt všechny tyto oblasti), je zde uveden stručný přehled používaných technik spolu s oblastmi jejich využití a se současným stavem vývoje.

Text této části se nezaobírá hodnocením jednotlivých řešení (které by bylo nad rámec této práce), ale spíše nástinem současných směrů, kterými se vývoj na poli AI pro RTS ubírá. Začlenění některých prací do určité oblasti nemusí být úplně přesné, neboť některé práce kombinují použití několika různých metod.

4.1 Techniky užívané ve více oblastech

V této úvodní části jsou uvedeny techniky, které se používají k řešení problému ve více základních oblastech (strategie, taktika, reaktivní kontrola). Dále následují kapitoly s metodami pro řešení těchto tří oblastí.

4.1.1 Finite-state machine

Asi nejčastější metodou pro implementaci jakékoliv oblasti AI je FSM – finite-state machine (konečný automat). Přestože tato metoda postrádá dynamiku, není schopná reagovat na nenadálé změny nebo nečekané chování a pravidla je snadné odhalit, je stále hodně používaná zejména pro svoji jednoduchou implementaci. Při průzkumu současných soutěžních AI dospějeme nutně k závěru, že téměř všechny obsahují v nějaké svojí části právě FSM.

V případě využití FSM k implementaci **strategie** jsou stavy automatu jednotlivé typy chování/aktivit hráče (útok, obrana), k přechodům pak dochází při změně herní situace. Např. v [24] je popsáno modelování komplexních strategií pomocí FSM. Cílem zmíněné práce bylo vytvořit takovou AI, která bude schopná se přizpůsobit stavu hry. AI implementovaná v rámci zmíněné práce byla vytvořena pro hru StarCraft, jako základ byla použit existující AI OpprimoBot.

V **reaktivním řízení** stavy automatu reprezentují vnitřní stavy jednotek (jednotka útočí, utíká z boje apod.). Například práce [25] popisuje hierarchický konečný automat (HFSM Hierarchical FSM) ve hře HALO2. Hierarchie

zde pomáhá snížit počet samostatně implementovaných stavů na únosnou mez, neboť tak mohou být jednotlivé stavy obsaženy v grafu (stromu) vícekrát.

4.1.2 Bayesovská síť

Bayesovská síť (Bayesian model) je pravděpodobnostní model reprezentovaný acyklickým orientovaným grafem, kde uzly jsou náhodné veličiny a hrany pravděpodobnostní závislosti mezi nimi.

Synnaeve a Bessière [26] použili bayesovskou síť k předpovídání akcí nepřítele (podoblast **strategie**). Model se učí z dat lidských hráčů. Výsledná implementace je volně dostupná[1].

V [27] používají Synaeve a Bessière bayesovskou síť k předpovídání útoků a k **taktickému** rozhodování. Metoda byla implementována pro hru StarCraft.

Také v oblasti **reaktivního řízení** používají Synaeve a Bessière bayesovskou síť [28]. AI s tímto reaktivním řízením byla implementována pro hru StarCraft (BroodwarBotQ[2]), kde uspěla proti dalším AI (například v roce 2011 a 2012 4. místo na turnaji AIIDE – viz turnaje 5.2.2). Využitím v reaktivním řízení se zabývali také Parra a Garrido [29]. Celkové shrnutí použití bayesovské sítě v AI pro RTS pak shrnuje v [21].

4.1.3 Evoluční algoritmy

Strategická rozhodnutí mohou být optimalizována pomocí evolučních algoritmů. Např. [30] využívá evoluční algoritmy k optimalizaci map vlivu (Influence Map). Tyto mapy jsou zde organizovaný v tzv. IMTrees (influence Map Trees), ve stromové struktuře, každý list je mapa vlivu a ostatní vrcholy jsou operátory mezi mapami (součet, násobení). Evoluční algoritmy optimalizují tyto stromy pro optimální strategická rozhodnutí ve hře.

Další využití evolučních algoritmů ve strategické oblasti našli Young a Hawes [31], používají evoluční algoritmy k určení strategických cílů. Optimalizace pomocí evolučních algoritmů zde zvýšila úspěšnost proti statickému řešení.

Hned několik prací se zbývá problémem **reaktivního řízení** pomocí evolučních algoritmů. Například práce [32] používá evoluční algoritmy k optimalizaci AI ve hře Wargus[3] (hra Warcraft II s upraveným enginem, který

[1] `https://github.com/syhw/OpeningTech`

[2] `https://github.com/syhw/BroodwarBotQ`

[3] `http://wargus.sourceforge.net/index.shtml`

umožňuje vývoj AI). Zde se tento přístup ukázal jako úspěšný při optimalizaci existujících doménových znalostí.

Další práce se zabývá zlepšením reaktivního řízení pomocí evolučních algoritmů ve hře StarCraft [33]. Zde je řešení vyvinuto jako framework, který optimalizuje konkrétní zjednodušený scénář hry. V jednom scénáři je např. třeba způsobit nepříteli v boji maximální ztráty a minimalizovat celkovou dráhu pohybu. Pomocí tohoto frameworku se podařilo namodelovat a zlepšit řešení tohoto scénáře.

V práci [34] se autoři pokusili řešit problém vysoké dimensionality evolučních algoritmů. Ta je způsobená velkým počtem optimalizovaných parametrů. Zde byl počet optimalizovaných parametrů snížen na osm (v prvním příkladu to bylo 20). Díky tomu a také menší populaci (32 vzorků proti 50 z druhého příkladu) stačilo pro všechny scénáře k natrénování méně než deset generací. AI byla implementována pro hru StarCraft.

Nakonec, práce [35] používá evoluční algoritmy k optimalizaci parametrů potenciálových polí.

4.1.4 Monte-Carlo Tree Search

Monte-Carlo Tree Search MCTS je variantou prohledávání herního stromu, kde jsou prohledávané cesty vybírány náhodně. Po každém běhu jsou průchozí uzly ohodnoceny podle toho, jestli vedla daná cesta k dobrému, nebo špatnému výsledku.

Balla a Fern [36] řeší pomocí MCTS prohledávání stavového stromu v oblasti **taktického** plánování (viz kapitola 4.3.1. Řešení je implementované pro hru Wargus.

Uriante a Ontañón řeší taktéž taktické plánování pomocí MCTS [37]. Ke snížení složitosti (velikosti herního, zde taktického stromu) použili abstrakci herních stavů, přičemž testovali více druhů této abstrakce.

Také v **reaktivním řízení** se může uplatnit MCTS. V práci [38] je použito MCTS pro správu micro managementu ve hře StarCraft.

4.1.5 Goal-Driven Autonomy

Goal-Driven Autonomy (dále GDA) je rozhodovací metoda, ve které si agenty samy vybírají svoje další cíle. Cílem GDA je, aby si mohly agenty samy rozhodnout, na co se zaměří a byly tak více autonomní.

Například Weber Mateas a Jhala [39] používají GDA pro reakci na nečekané situace. K implementaci zde byl použit jazyk ABL (A Behavior Language) [40]. GDA systém je součástí AI implementované v rámci jejich práce pro hru StarCraft, která je schopná hrát kompletní hru (tedy není zaměřena

jen na jednoduché scénáře). Tato AI se nazývá EISBot a její zdrojový kód je volně dostupný[4].

Další použití GDA ukazuje např práce [41].

4.2 Strategie

4.2.1 Case-Based Reasoning

Case-Based Reasoning (dále CBR) je způsob rozhodování na základě zkušenosti z podobných případů z minulosti. Někdy se také mluví o Case-Based Planning (CBP). Jedná se o způsob řešení problémů běžně používaný člověkem, ale nelehko přenositelný do světa AI.

V [42] používá Ontañón a kolektiv CBR k plánování. Nejprve byla získána data od hráčů. Data pak expert opatřil anotacemi, aby mohla být uložena ve formě případů. Z této databáze pak AI v průběhu hry může vybrat nejvhodnější případ. AI byla implementována pro hru Wargus. Nevýhodu tohoto postupu je potřeba experta, který označí odehrané hry pomocí příznaků tak, aby z nich mohly být získány případy.

Další CBR aplikací ve hře Wargus od Ontañóna a kolektivu je práce [43]. I ta ovšem potřebuje experta, tedy neobsahuje automatickou extrakci příznaků. Ontañón a kolektiv se věnuje aplikaci CBR při tvorbě RTS do hry Wargus i v další práci [44].

Kombinaci CBR s fuzzy logikou je představena v práci [45]. Zde se pomocí CBR AI vyrovnává jak se strategickými, tak s taktickými rozhodnutími ve hře StarCraft. Využití spojení CBR se zpětnovazebným učením v oblasti GDA ukazuje práce Jaideeho a kolektivu pracující s hrou Wargus [46]. Čertický pak zkoumá možnosti CBR v oblasti správného složení armády ve hře StarCraft [47].

4.2.2 Build Order Optimization

Build Order Optimization je oblast AI, která se zabývá optimálním využitím zdrojů na výstavbu budov a produkci jednotek.

Co se týče produkce daných jednotek v existujícím souboru budov, zde je optimalita jednoduše definová, AI excelují v této oblasti již v současné době [48]. V podstatě jde o to, nenechat žádné zdroje nevyužité, a zároveň se vyhnout placení zdrojů za odložené operace (tedy platit až tehdy, kdy je možné operaci provést).

[4] `https://github.com/bgweber/eisbot`

Naopak obtížným úkolem je zvolit správné pořadí výstavby budov a správné typy jednotek k produkci. Touto problematikou se zabývali například Weber a Mateas [49], kteří optimalizovali výstavu pomocí CBR (viz. kapitola 4.2.1.).

Tuto oblast zkoumali také Churchill a Buro [50]. Zabývali se především heuristickými technikami pro zmenšení množství prohledávaných variant tak, aby se podařilo v rozumném čase sestavit kvalitní plán výstavby a produkce. Výsledky práce Churchilla a Bura jsou součástí významné aplikace, jejich plánovaní výstavby a produkce je nyní využito v AI UAlbertaBot[5].

4.2.3 Predikce chování protihráče

Tato oblast, jinak nazvaná *opponent modeling* spočívá ve schopnosti odvodit na základě informací ze hry budoucí chování protihráče. Na základě této předpovědi může být zvolena správná strategie.

Například Weber and Mateas [51] používají k predikci *data mining*. Pomocí strojového učení na datech od lidských hráčů se systém naučí předpovídat budoucí akce protihráče. Bylo testováno několik algoritmů učení, přičemž nejlepší z nich, dosáhli úspěšnosti předpovědi v průměru 69% (5 min. hry) resp. 91% (10 min. hry).

Schopnost předvídat chování oponenta se projevila i v modelu chování postaveném na HMM (Hidden Markov models – skryté Markovovy modely) [52]. Model naučený na více než 300 odehraných hrách vykazoval dále schopnost rozpoznat běžné strategie a určit pravděpodobnou posloupnost strategických rozhodnutí nebo rozpoznat neobvyklé strategie.

Dalším příkladem modelování chování oponenta je práce Schadda a kolektivu [53] pro RTS běžící na enginu Spring.

4.3 Taktika

V taktické oblasti se nacházejí zejména prostorová a časová rozhodnutí týkající se vedení bojů. Důležitou schopností v této oblasti je umět předpovědět pozici protivníka. V žánru FPS tento problém řešil Hladky a Bulitko, konkrétně ve hře Counter-Strike: Source [54]. Jejich metoda spočívá na principu strojového učení z existujících her, pomocí něhož je sestaven model (konkrétně HSMM – Hidden semi-Markov model). Ačkoliv se může zdát že se tato práce RTS netýká, není tomu tak, neboť předpovídání pohybu jednotek je v RTS stejně důležité.

[5] `https://github.com/davechurchill/ualbertabot`

Zajímavé je také vyvozování řešení taktických situací na základě podobnosti s již naučenými, testované ve hře MadRTS [55]. Tato technika se nazývá transfer learning a v zmíněné práci je implementovaná pomocí kombinace CBR a RL kterou autoři nazývají CARL.

4.3.1 Prohledávání herního stromu

Prohledávání herního stromu je technika známá z běžných her jako šachy, dáma, piškvorky, nebo go. Jde o techniku, kdy se z následujících možných stavů hry (či podmnožiny těchto stavů) sestaví strom, který se následně analyzuje za cílem nalezení cesty k nejvýhodnější listu (obvykle symbolizujícího vítězný stav).

Tyto techniky se u některých běžných her dají použít k celkovému řešení AI, u RTS je to však díky řádově vyššímu počtu stavů není možné (viz 3.1). Existují ale postupy využívající prohledávání herního stromu v taktické oblasti.

V RTS strategiích je nutné se okamžitě rozhodnout, prohledávání celého stavového stromu je tak i při omezení jen na dílčí oblast taktiky příliš náročné. Řešením je prohledávání nějakým způsobem omezit tak, aby nebylo nutné procházet všechny možné varianty. Hned několik prací navrhuje pro prohledávání herního stromu využití MCTS (viz kapitola 4.1.4).

Churchil a kolektiv [56] se s problémem vyrovnává pomocí modifikace algoritmu alfa-beta ořezávání, které nazvali ABCD (Alpha-Beta Considering Durations). Tato modifikace je přizpůsobena tomu, že přechody mezi jednotlivými stavy v herním stromu nejsou okamžité. Implementace je k dispozici jako součást knihovny SparCraft (viz kapitola 6.2.4).

Součástí nástroje SparCraft je i další metoda pro prohledávání stromu, a to sice Portfolio Greedy Search. Tato metoda je popsána v práci [57]. Tento algoritmus byl v práci srovnán s předchozím ABCD a s UCT (varianta MCTS) a ukázalo se, že ve velkých soubojích oba algoritmy překonává.

4.4 Reaktivní řízení

4.4.1 Zpětnovazební učení

Zpětnovazební učení (Reinforcement Learning – RL) je jedním z nástrojů pro řešení reaktivního řízení. Jedná se o způsob učení pomocí zpětné vazby. Agenty v systému vykonávají určité akce, následně pak dostávají zpětnou vazbu, které přizpůsobí svoje další jednání.

Například Marthi a kolektiv [58] používají zpětnovazebné učení ve hře Wargus. Pro učení používá tzv. Q-Learning.

Liu a Li [59] pomocí zpětnovazebného učení optimalizují parametry potenciálových polí (viz 4.4.3).

Dále Wender a Watson [60] používají zpětnovazebné učeni (algoritmus Q-learning) ve hře StarCraft pro trénování AI v bojových scénářích. V těchto scénářích dosahuje takto naučená AI výborných výsledků.

Obdobně je v další práci [61] využit Q-learning pro implementaci AI ve hře Wargus.

4.4.2 Navigace

Navigace je důležitou součástí AI pro RTS, pohyb tvoří značnou část aktivity na mapě. Základem a standardem je algoritmus A*, vůči kterému se většinou vymezují další nabídnutá řešení. Jeho nevýhodou je poměrně vysoká výpočetní náročnost. Tradičním řešením tohoto problému je generování zjednodušené mapy pro navigaci, která neuvažuje všechny body na mapě. Například [62] používá tzv. Triangulation-Based pathfinding. V této metodě se nejdříve prostor rozdělí na trojúhelníky tak, že se spojují okrajové body překážek v prostoru, následně se uvažují jako navigační body pouze středy těchto trojúhelníků.

Pro hodnocení kvality hledání cest je přínosná práce Sturtevanta [63], který vytvořil benchmark pro oblast navigace.

4.4.3 Potential Fields

Potential Fields (dále Potenciálová pole) jsou součástí oblasti navigace, jsou ale tak významnou oblastí výzkumu, že je jim zvlášť věnovaná celá tato sekce.

Princip potenciálových polí spočívá v tom, že se každému bodu na ploše přiřadí hodnota – potenciál. Cílem je pak navigace v poli taková, že se příslušný objekt (dále jednotka) pohybuje do oblasti s co „nejvýhodnějším“ potenciálem. Potenciál se danému bodu přiřazuje dle nejrůznějších kritérií. Nejčastějším kritériem pro výpočet potenciálu bývá cíl cesty, tvar terénu, ale i souřadnice nepřátelských jednotek.

Již Massari, Giardini a Bernelli-Zazzerapro [64] navrhují pro planetární průzkumné vozidlo navigační systém využívající nelineární potenciálové pole. Použitá Gaussova funkce zajistí jak rozšíření potenciálu v celé oblasti, tak limit potenciálu v bodě zdroje.

Aplikací v RTS pak navazuje Hagelbäck [65]. V práci představuje vzorce pro výpočet potenciálů různých zdrojů (nepřátelské jednotky, překážky,

budovy...), které se ve výsledku skládají do jednoho pole. Výsledná implementace je pro hru StarCraft.

Pro tuto hru vyvíjel AI založenou na potenciálových polích i Lebedynskyj [66].

V jiné práci [67] se podařilo ve hře StarCraft propojit oblast potenciálových polí s evolučními algoritmy (kapitola 4.1.3). pomocí evolučních algoritmů byly optimalizovány hodnoty potenciálu v jednotlivých bodech. Optimalizováno bylo sedm potenciálových polí o třech parametrech, tedy celkem 21 parametrů. Evoluční algoritmus pracoval s populací 40 jedinců a probíhal 25 generací. Výsledná optimalizace vedla k poražení silnějšího nepřítele ovládaného původní AI (tedy AI ze hry), jak můžeme vidět na demonstračním videu[6].

4.5 Analýza terénu

Specifickou oblastí v rámci AI je analýza terénu. Na rozdíl od ostatních oblastí zde není nutné okamžité vrácení výsledku, na druhou stranu se jedná o poměrně komplexní problém. Člověk v této oblasti propojuje strategické a taktické myšlení s přirozenou představou o prostoru a čase. Takové uvažování je pro AI obtížné napodobit. Přesto existují pokusy alespoň částečně analýzu terénu provést.

Například práce [68] se zabývá rozdělením terénu na oblasti. Rozdělení probíhá pomocí algoritmu DEACCON (Decomposition of Environments for the Creation of Convex-region Navigation-meshes). Jejím omezením je to, že dělí celou mapu na konvexní útvary, v případě že nějaká oblast tvoří nekonvexní útvar, je tímto systémem rozdělena na několik oblastí, přestože tvoří jednolitý celek. Na tuto práci dále navazuje [69].

Jednou z nejdůležitějších prací v této oblasti je práce Perkinse [70], na níž je postavena i dnes hojně používaná knihovna pro analýzu terénu BWTA (viz 7.4.3). V této práci je popsán analyzátor, který analyzuje mapu a vytvoří její abstrakci složenou z více-úhelníkových oblastí a uzlových bodů (Choke Points) mezi nimi.

4.5.1 Opevňování

Opevňování (walling) je jednou z oblastí, pro kterou je nutná kvalitní analýza terénu. Jedná se o stavbu budov na strategických místech s cílem zabránit nepříteli v průchodu. Rozhodnout kam budovy postavit, tak aby protihráč

[6] `https://www.youtube.com/watch?v=pBMDRdBqd2E`

nemohl projít, je pro člověka triviální, pro AI je ale tento úkol náročný, neboť je těžké převést lidské uvažování do algoritmů/pravidel pro počítač.

Tímto problémem se zabýval např. Čertický [71]. Pro hru StarCraft navrhl systém pro výpočet pozic pro hrazení na principu ASP (Answer Set Programming). Richeux a kolektivpak pro stejnou hru navrhli algoritmus pro výpočet hrazení založený na tzv. Constrain Optimization (optimalizace funkce kde její parametry jsou omezené omezujícími podmínkami) [72]. Zde se podařilo tento problém vyřešit tak, že již po 50 herních obrátkách (asi 2 sekundy) byla šance na správné řešení větší než 99%. Přitom na výpočet hrazení je ve hře daleko více času, neboť ten může běžet již od začátku hry, zatímco potřebné suroviny na stavbu budov pro hrazení bývají k dispozici až po desítkách sekund hry.

5 Platformy pro vývoj AI do RTS

RTS jsou velkým žánrem počítačových her, každý rok vyjde několik nových titulů. Počet her, pro které je možné vyvíjet vlastí AI je ale značně menší. V této kapitole se pokusíme vybrat několik platforem, kde je vývoj AI možný, srovnat jejich vlastnosti, vybrat nejvhodnější platformu pro realizaci této práce a tu pak důkladně popsat.

5.1 Vybrané platformy pro vývoj AI do RTS

V této části je popsáno šest vybraných her/platforem, pro které je možné tvořit vlastní AI. Pořadí platforem je určeno sestupně dle toho, jak vhodné se nakonec ukázaly pro vývoj. Na prvním místě je tedy platforma vybraná pro vývoj, poslední v pořadí je naopak platforma která z průzkumu vyšla jako nejméně vhodná.

Jistě by se našly i další možnosti, cílem této práce je ale najít hlavně výrazné zástupce s existující komunitou. Platforem, kde je možné vyvíjet AI jak z technického, tak z legálního hlediska, je tedy jistě mnohem více. Srovnávány budou především technické možnosti, velikost komunity vývojářů a množství vědeckých prací věnujících se projektům na dané platformě. Je třeba si uvědomit, že část tohoto hodnocení se vztahuje k aktuálnímu času, například velikost komunity kolem dané platformy se může radikálně změnit v rámci jednoho nebo dvou let.

5.1.1 Starcraft

Podpora pro vývoj AI pro hru StarCraft je značná. Protože byla tato platforma nakonec vybrána pro realizaci této práce, bude podrobně popsána až v kapitole 5.2. Zde budou stručně uvedeny především klady a zápory této platformy.

Hlavním kladem platformy StarCraft je její komunita. Existuje několik pravidelných turnajů AI do hry StarCraft (viz kapitola 5.2.2). Každoročně počet účastníků převyšuje 10, v posledním ročníku SSCAIT (tabulka 5.2) bylo dokonce skoro 50 soutěžících AI.

Dále je tato platforma pokryta velkým množstvím vědeckých prací, při rešerši pro tuto práci bylo nalezeno více než 50 prací zabývajících se vývojem AI pro tuto platformu. Kromě nich existuje řada návodů pro zprovoznění platformy a vývoj AI, na sociální síti facebook je pak skupina[1] sdružující

[1] `https://www.facebook.com/groups/bwapi/`

vývojáře AI pro StarCraft, která je velmi aktivní.

Nakonec tak byla zvolena právě tato platforma. Je zde možno vycházet z existujících prací, a tam, kde práce na dané téma nejsou, alespoň srovnat navržené řešení s existujícími technologiemi.

5.1.2 Spring

Engine Spring[2] je jednou z největších platforem pro tvorbu RTS. Přináší jak možnost vytvořit v něm vlastní hru, tak zlepšovat vlastnosti her, které jsou již v enginu vytvořené. Mezi hry vytvořené v tomto enginu patří např. Balanced Annihilation[3]. Tato hra je klonem slavné RTS Total Annihilation[4], která jako první přinesla skutečnou 3D grafiku do tohoto žánru. Snímek ze hry vidíme na obrázku 5.1. Dalšími hrami vytvořenými v tomto enginu jsou například Evolution RTS[5] nebo Spring 1944[6].

Obrázek 5.1: Snímek ze hry Balanced Annihilation vytvořené v enginu Spring. [73]

V enginu je možné měnit téměř vše, od uživatelského rozhraní přes mapy a textury jednotek až po AI. Engine je zaměřen na hraní s lidskými hráči, čemuž je přizpůsoben, například obsahuje několik verzí spouštěcích programů s vyhledáváním protihráčů. Engine je kompletně ve 3D, jeho technické

[2] `https://springrts.com/`
[3] `https://springrts.com/wiki/Balanced_Annihilation`
[4] `https://en.wikipedia.org/wiki/Total_Annihilation`
[5] `http://www.evolutionrts.info/`
[6] `http://spring1944.net/`

možnosti jsou značné, zvládne i velké mapy a až 5000 jednotek. Jak engine, tak hry na něm (až na výjimky) jsou otevřeným softwarem (open-source software – OSS).

Engine obsahuje podporu pro vývoj AI v mnoha jazycích (C++, Java, Lua, Python, C#). Již nyní existuje více než deset AI pro tento engine [74].

Komunita kolem enginu Spring existuje, dle příspěvků na fóru to vypadá, že je aktivní jak kolem několika nejhranějších her, tak v oblasti vývoje. Méně aktivní je ale komunita odborníků věnujících se této platformě, byly sice nalezeny odborné práce (např. [53, 75]), ale bylo jich velmi málo. Také Turnaje pro AI na této platformě nejsou, v minulosti fungoval žebříček AI, stránka je ale nyní nefunkční.

5.1.3 ORTS

ORTS[7] (Open Real-Time Strategy) je OpenSource prostředí pro RTS. Výhodou tohoto prostředí je fakt, že byl vytvořen přímo za cílem výzkumu a vývoje AI. ORTS má několik významných rozdílů proti běžným RTS, např:

- Komunikace mezi hráči je zajištěna na principu Server-Client (na rozdíl od obvyklého způsobu peer-to-peer). Tím je zajištěna férová hra, neboť server pošle hráčům jen ty informace o herní mapě, na které má hráč nárok (např. nepošle část mapy, kterou hráč ještě neodhalil).
- Množství dostupných druhů jednotek/budov lze upravit, díky tomu je možné snížit/zvýšit složitost hry. To může být vhodné vhodné pro jednoduché testování AI
- Uživatelské rozhraní lze libovolně upravit, nebo zcela vypnout (vhodné pro zrychlený běh hry, např. za účelem učení).
- AI může být připojena ke hře ze vzdáleného serveru. Díky tomu je možné spustit AI např. ve výpočetním centru s velmi výkonnými počítači, což může být pro vývoj AI velmi užitečné.

Tyto vlastnosti velmi zvyšují atraktivitu ORTS pro tvorbu AI. Bohužel není komunita vývojářů na této platformě příliš veliká. Turnaj pro AI se naposledy odehrál v roce 2009. Odborné práce o AI na této platformě existují, ale jde o jednotky prací ([76, 77, 78].

Snímek z ORTS můžeme vidět na obrázku 5.2.

[7] `https://skatgame.net/mburo/orts/`

Obrázek 5.2: Snímek z ORTS [79].

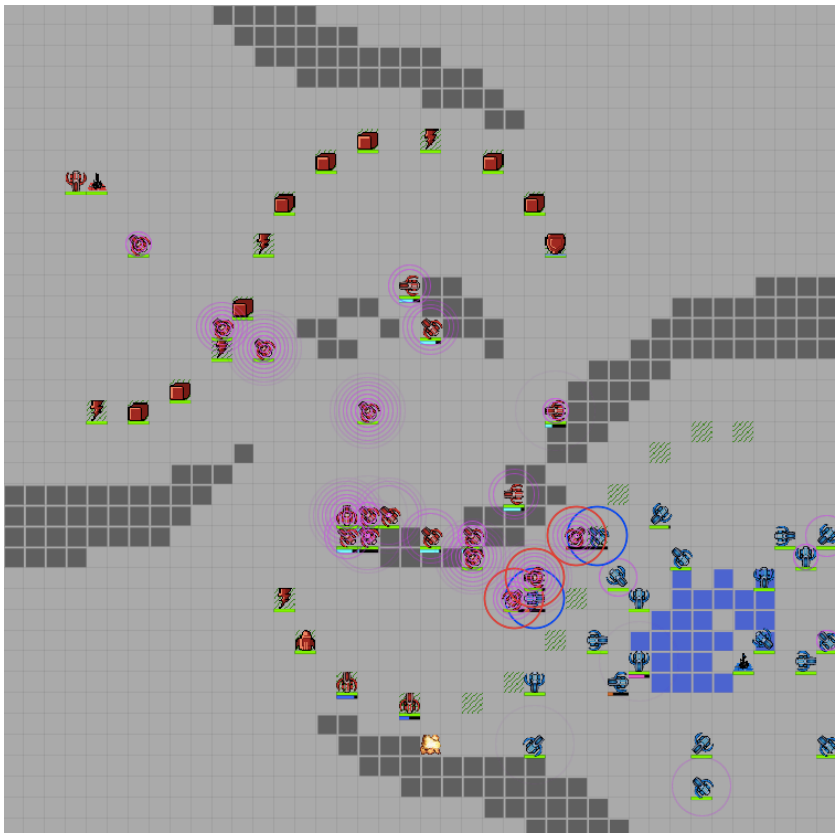
5.1.4 Battlecode

Battlecode je RTS a také název stejnojmenné soutěže na MIT (Massachusetts Institute of Technology). Tato soutěž probíhá každoročně v lednu, je naživo vysílaná a sledovaná místními studenty, vítězové si rozdělí celkem 50 000$ [80].

Ve hře je úkolem nashromáždit zdroje, vybudovat armádu robotů a s ní útočit na nepřítele a zabrat co největší území. V některých ročnících je hra ozvláštněna dalšími pravidly, například jsou přidány neutrální jednotky, které útočí na obě strany, nebo je k dosažení vítězství potřeba nejprve splnit nějaký speciální úkol.

Soutěžící AI v tomto turnaji jsou omezeny počtem volání herních funkcí nebo množstvím java bytecodu, klade se zde tedy důraz na optimalizaci. Výrazným technickým rozdílem proti ostatním platformám je to, že jednotky zde nesdílí svoje znalosti. Tím je vynuceno decentralizované a autonomní řešení. Tyto vlastnosti by mluvily pro výběr této platformy, neboť se jedná o vlastnosti MAS. Bohužel je ale platforma určena spíše pro krátkodobý vývoj, kromě soutěže pak neexistuje žádná komunita. Navíc ze všech nalezených odborných prací zabývajících se vývojem AI pro RTS není ani jedna spojena s touto platformou.

Snímek ze hry Battlecode je na obrázku 5.3.



Obrázek 5.3: Snímek ze hry Battlecode [81].

5.1.5 Warzone 2010

Warzone 2010 je RTS původně vyvíjená společností Pumpkin[8] a vydaná společností Eidos Interactive[9] v roce 1999. Od roku 2004 je hra otevřeným softwarem.

Pro hru je možné vyvíjet AI ve dvou skriptovacích jazycích. K vývoji nejen AI do hry je dostupná přehledná dokumentace[10]. Bohužel nebyla nalezena žádná práce, která by pokrývala vývoj AI pro tuto hru. Také turnaje AI pro tuto hru nejsou pořádány.

Snímek ze hry Warzone 2100 vidíme na obrázku 5.4.

[8] `http://www.ign.com/companies/pumpkin-studios`
[9] `http://www.giantbomb.com/eidos-interactive/3010-658/`
[10] `https://warzone.atlassian.net/wiki/display/jsapi`

Obrázek 5.4: Snímek ze hry Warzone 2100 [82].

5.1.6 Glest

Glest[11] je OpenSource RTS a zároveň název pro engine, na kterém hra běží. Zajímavostí enginu je například to, že veškerá data o jednotkách a technologických stromech jsou ve formátu XML (protiklad běžných her, kde jsou tato data v netransparentním a většinou proprietárním formátu).

Ve hře jsou dostupné dvě frakce, jedna se soustřeďuje na technické jednotky (Tech), druhá na jednotky s nadpřirozenými silami (Magic). Díky tomu, že je hra otevřeným softwarem, je možné měnit celý její obsah.

Návody jak vytvářet módy pro hru existují, nicméně co se týče AI, vypadá to, že žádný návod k dispozici není. Existuje pouze referenční pravidlová (rule-based) AI. Tím je tato platforma z výběru pro realizaci práce vyloučena, tam, kde není popsán ani základní postup, nemůžeme čekat fungující komunitu vývojářů.

Snímek ze hry Glest vidíme na obrázku 5.5.

5.1.7 Shrnutí

Bylo prozkoumáno celekm šest variant cílové platformy: Spring, Starcraft, ORTS, Glest, Warzone 2100 a Battlecode. Nejdříve byla vyřazena platforma

[11] `http://glest.org/en/index.php`

Obrázek 5.5: Snímek ze hry Glest [83].

Glest, neboť nebylo jasné, zdali je vůbec schůdné na této platformě AI vyvíjet. Dále byla vyřazena hra Warzone 2100, neboť je v ní možno vyvíjet AI pouze ve skriptovacím jazyce, což se zdálo pro komplexní AI na principu MAS nevhodné.

Do finálního výběru se dostaly platformy Spring a Starcraft, pro jejich výběr mluvila jejich aktivní komunita. Nakonec byla vybrána platforma Starcraft především díky velkému pokrytí odbornými pracemi. Velkým faktorem pro výběr byla také existence hned několika turnajů pro AI s poměrně velkou účastí.

V tabulce 5.1 můžeme vidět stručné shrnutí vlastností jednotlivých platforem.

5.2 Vývoj AI na platformě StarCraft

Vývoj AI pro hru StarCraft je v poměru k ostatním hrám/platformám velmi intenzivní. Pro hru StarCraft Existuje několik desítek AI a několik pomocných nástrojů pro jejich vývoj. Je pro vývojáře i hráče známá, herní mechanismy, taktiky a strategie vedoucí k vítězství jsou dobře popsány [84]. Navíc existují desítky odborných prací věnujících se vývoji AI pro tuto platformu.

Název	**Vznik**[1]	**Poslední akt.**[1]	**Soutěže AI**	**Odborné práce**	**Podporované jazyky pro vývoj**	**OS**
StarCraft	2008	2015	ano	desítky	C++, Java, Ruby, Python, C#	Windows
Spring	2005	2015	ne[5]	jednotky	C++, C, Java, Lua, Python, C#	Windows, Linux
ORTS	2003	2012[6]	ne[4]	jednotky	C++	Windows, Linux, Mac
Battlecode	2001	2016	ano	ne	Java, Scala	Java komp.
Warzone 2010	2004	2015	ne[2]	ne	Javascript, WZ script[3]	Windows, Linux, Mac
Glest	2005	2015	ne	ne	lua	Windows, Linux

[1] Týká se nástrojů pro vývoj, nikoliv hry samotné.
[2] Existují pouze občasné exhibiční turnaje.
[3] Speciální skriptovací jazyk přímo pro hru Warzone 2010.
[4] Turnaje probíhaly mezi lety 2006–2009.
[5] V minulosti existoval žebříček AI, nyní je stránka nefunkční.
[6] Poslední změna zdrojového kódu v logu.

Tabulka 5.1: Platformy pro tvorbu AI do hry StarCraft

5.2.1 StarCraft

Starcraft je RTS z prostředí vesmíru v daleké budoucnosti. Samotná hra byla vydaná v roce 1998 společností Blizzard Entertainment. Ještě v roce 1998 vyšel ke hře datadisk s názvem Brood War. Starcraft se stala velice oblíbenou hrou, časem se etablovala jako vzor toho, jak by měla RTS vypadat. Celkem se prodalo skoro 10 milionů kusů [85].

Technické vlastnosti hry jsou dnes již zastaralé. Například rozlišení hry je pouze 640x480 obrazových bodů. Tato vlastnost je však pro vývoj AI spíše dobrá, neboť to také znamená nízké nároky na hardware, a tedy možnost spustit hru ve zrychleném režimu, například za účelem strojového učení. Snímek ze hry můžeme vidět na obrázku 5.6.

Hra přinesla do žánru novinku ve formě velké odlišnosti jednotek. Ty

Obrázek 5.6: Snímek ze hry StarCraft. Na snímku boj mezi rasou Terran (modří) a Protoss (žlutí)

se zde liší mezi rasami (herní frakce) nejen vzhledem, ale i vlastnostmi a způsobem hry. Dřívějším (a částečně i současným) hrám přitom dominoval takový návrh jednotek, kdy rasy vždy měly ekvivalentní jednotky, které se lišily jen vzhledem, případně se minimálně lišily jejich vlastnosti. StarCraft toto mění, každá rasa má svůj vlastní herní styl. To samozřejmě klade zvýšené nároky na AI, ta se musí vyvíjet (tedy nejen ladit) pro všechny rasy. Ve hře je možné hrát za tři rasy:

- **Terran:** Terran je lidská rasa. Má jednotky střední ceny a síly. Budovy může stavět kdekoliv, mnoho budov se může přemisťovat. Pohyb jednotek je spíše pomalý, silná je obrana. Typické jednotky jsou pěchota, letadla, tanky.
- **Protoss:** Protoss je pokročilá mimozemská rasa. Její jednotky jsou drahé, ale silné. Všechny jednotky a budovy jsou chráněny pomocí štítů, které se automaticky regenerují. Budovy je třeba stavět v dosahu *pylonu* – speciální budovy. Má k dispozici vesmírné lodě nebo jednotky s nadpřirozenými schopnostmi.
- **Zerg:** Zerg je Insektoidní rasa podobná přerostlému hmyzu. Její jednotky jsou levné, ale ne příliš odolné, snaží se tedy zvítězit kvantitou. Nízkou výdrž jednotek dále kompenzuje jejich rychlost. Budovy je nutné stavět na povrchu, kterému se říká *creep*.

Jednotky se dále dají rozdělit na dělníky (staví budovy, opravují), bojové jednotky a podpůrné jednotky (transportní loď, medik). Jednotky jsou vyráběny buď v budovách (Terran, Protoss) nebo mutací z jiných jednotek (Zerg).

5.2.2 Turnaje AI pro StarCraft

Jedním z motorů překotného vývoje AI pro StarCraft v posledních letech jsou turnaje. V těchto turnajích proti sobě soutěží umělé inteligence vyvinuté na univerzitách a jiných vědeckých pracovištích, ale také umělé inteligence od nezávislých vývojářů. V turnajích se účastní vývojáři z více než deseti zemí [86]. Přehled nejvýznamnějších turnajů je v tabulce 5.2.

název	**od roku**	**do roku**	**účast**[1]	**pořádá**
AIIDE StarCraft AI Competition[2]	2010	2015	22	University of Alberta
CIG StarCraft RTS AI Competition[3]	2011	2015	15	IEEE Conference on Computational Intelligence and Games
SSCAIT – Student StarCraft AI Tournament[4]	2011	2015	46	ČVUT a Univerzita Komenského
StarCraft BroodWar Bots Ladder[5]	?	2015	27	Krasimir Krastev a Alberto Uriarte

[1] Účast v posledním ročníku.
[2] `https://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp/index.shtml`
[3] `http://cilab.sejong.ac.kr/sc_competition/#Important_Dates` – Odkaz na stránky ročníku 2015.
[4] `http://www.sscaitournament.com/`
[5] `http://bots-stats.krasi0.com/`

Tabulka 5.2: Turnaje AI pro hru StarCraft

5.2.3 BWAPI – API pro StarCraft

BWAPI[12] je aplikační rozhraní (API) pro programování AI do hry StarCraft. BWAPI je napsáno v jazyce C++.

Přesný význam zkratky je Brood War API, což odkazuje na rozšíření BroodWar pro hru StarCraft, pro které je toto API vytvořeno. Obecně jsou všechny AI vytvořená pro toto rozšíření, budeme-li tedy dále mluvit o AI pro StarCraft, myslí se tím AI pro hru StarCraft s nainstalovaným rozšířením BroodWar.

Prakticky všechny umělé inteligence pro StarCraft, ať už jsou napsány v jakémkoliv jazyce, používají BWAPI, buď už přímo, nebo zprostředkovaně.

5.2.4 Další nástroje pro vývoj AI pro StarCraft

Kromě BWAPI, které je pro vývoj AI pro StarCraft nezbytností, existují další pomocné nástroje pro vývoj. Tyto nástroje, usnadňující vývoj AI pro StarCraft a umožňující vývoj ve více programovacích jazycích jsou popsány v této kapitole.

Benchmarky

Pro testování AI existují referenční mapy[13], používané profesionálními hráči.

Dále byl vytvořen benchmark obsahující testovací scénáře[14] pro zajištění referenčního testování určitých vlastností AI. Tyto scénáře sice nepokrývají všechny oblasti AI, ale testují konkrétní vlastnosti v oblasti reaktivního řízení, taktiky, i strategie [11].

Jiný benchmark[15] testuje schopnost hledání cesty v terénu (pathfinding) [63].

Kromě běžného testování lze tyto benchmarky použít i jako metodu pro zhodnocení výsledků v odborných pracích [87], nebo jako doplňující měřítko úspěšnosti AI (viz zatím skromný žebříček úspěšnosti [88]). Turnaje totiž hodnotí pouze celkovou úspěšnost AI, nezohledňují však důvody úspěchu či neúspěchu, nebo míru úspěšnosti v dílčích oblastech. Stává se tak, že AI, které jsou úspěšné v turnajích, nejsou nebytně těmi technologicky nejpokročilejšími [89], ale jednají např. podle napevno nastavené strategie, která je navržená s cílem využít slabostí existujících AI [11].

[12] `http://bwapi.github.io/`

[13] `http://wiki.teamliquid.net/starcraft/Micro_Training_Maps`

[14] `https://bitbucket.org/auriarte/starcraftbenchmarkai`

[15] `http://movingai.com/benchmarks/`

API pro další programovací jazyky

Aby byla platforma úspěšná, je vhodné zpřístupnit ji pro co možná nejvíce programovacích jazyků. Základní přístup do hry, jak již bylo napsáno, funguje přes BWAPI 5.2.3, které je napsáno v jazyce C++. Dále existují rozhraní pro další programovací jazyky, všechna tato rozhraní přistupují ke hře skrze BWAPI. Existují pro jazyky:

- **Java:** Pro jazyk Java existují dva přístupy, oba ke komunikaci s BWAPI používají JNI (Java Native Interface):
  - **BWMirror:** V současnosti nejpoužívanější způsob, viz kapitola 7.4.2.
  - **JNIBWAPI**[16]**:** Alternativa k BWMirror. V současnosti není aktualizovaná (není kompatibilní s aktuální verzí BWAPI).
- **Python:** Pro připojení z jazyka python existuje API CyBW[17] Pro spojení k BWAPI používá rozhraní Cython[18]. Tato API je aktuální.
- **C#:** Pro C# existuje již neaktuální knihovna bwapi-mono-bridge[19] se svým pokračováním bwapi-mono-bridge2[20]. K připojení k BWAPI používají obě API nástroj SWIG[21]. Tento nástroj umožňuje i připojení dalších jazyků, teoreticky je tedy možné propojit pomocí bwapi-mono-bridge2 s BWAPI i další jazyky (Java, python, ruby, R, Prolog) [90].

Nástroje pro analýzu terénu

Analýza terénu je důležitou součástí každé AI pro RTS. Tato problematika je tak široká, že pro ni byly implementovány samostatné knihovny. Tyto knihovny využívají BWAPI a nabízejí funkce pro práci s mapou, například umí nalézt významné body na mapě, kde je výhodné koncentrovat obranu. V současnosti jsou k dispozici knihovny:

- **BWTA2:** Asi nejznámější knihovna pro analýzu terénu, více v kapitole 7.4.3.
- **BWTA**[22]**:** Předchůdce BWTA2, nyní již neaktualizovaný projekt.

[16] `https://github.com/JNIBWAPI/JNIBWAPI`
[17] `https://bitbucket.org/ratiotile/cybw`
[18] `http://cython.org/`
[19] `https://code.google.com/archive/p/bwapi-mono-bridge/`
[20] `https://github.com/suegy/bwapi-mono-bridge2`
[21] `http://www.swig.org/`
[22] `https://code.google.com/archive/p/bwta/`

BWSAL

BWSAL[23] je knihovna (sama sebe označuje addon library) pro BWAPI, která si klade za cíl poskytovat vývojářům AI některé často používané postupy. Přináší především abstrakci ve formě manažerů (Worker Manager, Build Manager, Defense Manager...), dále pak další užitečné objekty, například pro nalezení místa pro stavbu budovy (Building Placer). BWSAL je napsán v jazyce C++ a připojuje se přímo k BWAPI. Knihovna v poslední době není moc aktivní, poslední aktualizace pochází ze začátku roku 2015.

Atlantis

Atlantis[24] je framework napsaný v jazyce Java. Zaměřuje se na implementaci často používaných technik a funguje jako další vrstva abstrakce nad API BWMirror, kterou používá ke spojení s BWAPI. Cílem frameworku je odstínit tvůrce AI od ekonomických a dalších záležitostí tak, aby se tvůrci AI mohli soustředit na bojové oblasti AI. Atlantis nabízí například plány výroby (build orders), optimální sběr surovin, průzkum, základní implementaci reaktivního řízení, analýzu výsledku soubojů a mnoho dalšího. Framework je aktivně vyvíjen.

[23] `https://github.com/Fobbah/bwsal`
[24] `https://github.com/Ravaelles/Atlantis`

6 Existující multiagentní AI

Definice MAS AI pro RTS je zatím značně rozmělněná. To že vědecká práce mluví o multiagentním systému nebo přístupu, většinou neznamená implementaci celé AI jako MAS, ale spíše využití MAS pro řešení konkrétních problémů, zatímco jiné části AI jsou řešeny odlišnými způsoby, jako například v práci [66], kde je pomocí agentů řešeno reaktivní řízení na principu potenciálových polí.

Fungujících AI, které používají ve větší míře multiagentní systémy je málo. Například z rozboru sedmi AI soutěžících na AIIDE [20] v roce 2013 jen dva (BTHAI[1], Nova[2]) obsahují v návrhu inteligentní agenty, a žádný z nich se nedá nazvat plně agentní, neboť jsou pomocí agentů navrženy jen některé části AI.

6.1 Použití MAS v konkrétních oblastech AI

Použití MAS v RTS strategiích je pořád spíše v počátcích. Ačkoliv se slovo agent vyskytuje v mnoha odborných pracích na toto téma, většinou je to ve významu celé AI (jak je vysvětleno již v představení základních pojmů 2.4), nikoliv agentů jako součásti multiagentního systému. Prací, zabývajících se využitím MAS v RTS je tedy velmi málo. Ještě méně je pak aplikací tohoto principu, jak se ukáže v následující kapitole.

Většina současných aplikací MAS je v oblasti reaktivního chování. Zde se naplno uplatní výhoda decentralizace, která výrazně zmenší rozhodovací prostor a umožní jednotlivé agenty ovládat jednodušším algoritmem, než který by byl potřeba pro centální řízení reaktivního chování [34].

Tradičně se MAS principy uplatňují při implementaci potenciálovcýh polí (viz kapitola 4.4.3). Příkladem mohou být mnohé práce [91, 14, 65] Johana Hagelbäcka na toto téma, ale i další autoři v této oblasti uvažují multiagentní řešení [59, 67, 66].

Další přirozeně agentní prostředí je oblast GDA (viz kapitola 4.1.5, kde jsou uvedeny odborné práce na toto téma). V tomto modelu uvažování je agent základní a nepostradatelnou součástí.

Mimo oblast RTS se tématu MAS dlouhodobě věnuje Agent Technology Center[3] při ČVUT. Zajímavé jsou projekty na téma kooperativního plánování [92] nebo v oblasti řízení letecké dopravy [93].

[1] Již neexistuje, autor nyní vyvíjí AI OpprimoBot - viz 6.2.1

[2] `http://nova.wolfwork.com/`

[3] `http://agents.felk.cvut.cz/`

S hierarchickou multiagentní architekturou se můžeme setkat v práci Tana a Chenga [94] zabývající se strategickým a taktickým plánováním. Každý agent zde obsahuje strategickou a taktickou osobnost. Zatímco taktická osobnost produkuje chování agenta, strategická určuje rozkazy pro podřízené jednotky. Obě osobnosti vybírají chování z předem definovaného výběru. Práce byla implementována jako scénář podobný FPS v enginu Truevision3D[4].

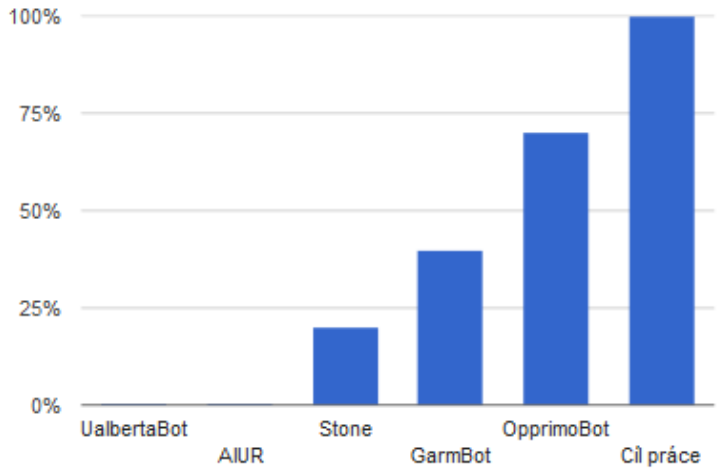
Zajímavým projektem je rozhraní EIS[5] (Environment Interface Standard), které umožňuje propojení agentů implementovaných v nějakém APL (Agent Programing Language – speciální jazyk pro vývoj agentů) k různým prostředím ve kterých by měly agenty fungovat [95]. Tento standard podporuje již několik APL, např. Jason[6], GOAL[7], 2APL[8] nebo Jadex[9]. Agenty napsané v těchto jazycích pak mohou fungovat v libovolném prostředí splňujícím standard EIS. Rozhraní EIS se navíc nedávno podařilo propojit i k BWAPI [96] (a tedy ke hře StarCraft) pomocí API pojmenovaného EISBW[10]. Toto API je k propojení nezbytné, neboť BWAPI není s rozhraním EIS kompatibilní. Tato implementace používá APL GOAL.

6.2 Srovnání existujících AI ve vztahu k MAS

AI v této kapitole jsou porovnávány zejména s ohledem na použití principů MAS. Pro porovnání bylo vybráno celkem pět AI. První tři byly vybrány z důvodu, že se v nich nacházejí ve významné míře MAS prvky. Další dvě pak byly vybrány, neboť se vyvíjí a účastní soutěží již delší dobu, navazují na ně další AI a mohou tak sloužit jako etalon AI pro Starcraft. Výčet AI není ani zdaleka kompletní, celý souhrn a porovnání vlastností AI pro hru StarCraft by vystačil na samostatnou práci.

Srovnání vybraných AI na poli „agentovosti“ (tedy podílu oblastí AI řešených na principu MAS) najdeme na obrázku 6.1. Zcela vpravo se nachází cíl této práce, kterým je implementace všech oblastí AI pomocí MAS.

[4] `http://www.truevision3d.com/`
[5] `https://github.com/eishub/eis/wiki`
[6] `http://jason.sourceforge.net/`
[7] `http://ii.tudelft.nl/trac/goal/`
[8] `http://apapl.sourceforge.net/`
[9] `https://sourceforge.net/projects/jadex/`
[10] `https://github.com/eishub/Starcraft`



Obrázek 6.1: Přibližné srovnání vybraných AI podle podílu oblastí implementovaných pomocí MAS.

6.2.1 OpprimoBot

OpprimoBot[11] je z hlediska aplikace MAS principů asi nejpříkladnější existující AI. Všechny jednotky ve hře jsou implementované pomocí agentů [97]. Armáda je tvořená trojvrstvou hierarchií, kde v čele je hlavní velitel (commander), jehož činnost je určena pomocí pravidel, pod ním jsou velitelé bojových skupin a na nejnižší úrovni přímo bojující jednotky. Bojující jednotky a velitelé bojových skupin jsou implementováni jako *reaktivní agenty*. Další agenty pak ovládají jednotky konstruující budovy, průzkumné jednotky apod.

OpprimoBot používá jednoduchou metodu učení. Po skončení hry si ukládá nastavení pořadí stavby (build order) společně s informací o vítězství. Na základě toho pak vybírá takové pořadí staveb, kde je poměr mezi vítězstvími a prohrami nejpříznivější.

Tato AI je také významná svojí pokročilou metodou navigace, kromě klasického algoritmu A* používá při kontaktu z nepřítelem tzv. *potenciálová pole* (Potential Fields) [98]. Díky této metodě zůstávají jednotky v optimální vzdálenosti od nepřítele (viz kapitola 4.4.3.

[11] `https://github.com/jhagelback/opprimobot`

6.2.2 GarmBot

GarmBot[12] používá multiagentní principy na ovládání jednotek [99]. Nejedná se o hierarchický MAS, jednotky spolu dokonce ani přímo nekomunikují, jediným nástrojem interakce je objekt který autor sám nazývá *global knowledge* (globální znalosti), a který nejvíce připomíná tzv. tabuli (blackboard) známou z tabulové architektury (blackboard architecture) [100]. Každý agent je pak implementován jako konečný automat.

Ačkoliv se na první pohled jedná o příliš jednoduché řešení, které ve velkém počtu jednotek nebude fungovat, GarmBot dokáže díky založení velkého množství základen a s tím spojenou těžbou surovin produkovat velké množství jednotek, takže doslova „zaplaví“ celou mapu [101].

Také tato AI měla původně využívat potenciálová pole, ve výsledné implementaci se ale tato mtoda kvůli technickým problémům neobjevila.

6.2.3 Stone

Stone[13] používá MAS k distribuovanému chování jednotek SCV (jednotka určená pro výstavbu budov, která je ale schopná i útoku) [102]. Ne všechny části chování SCV jsou ale distribuované, například skupinový útok je řízen běžným způsobem, tedy centrálně, bez využití MAS.

AI používá jednoduchou verzi učení na terénu mapy. Místa, ve kterých dojde k zaseknutí jednotky, jsou označena, a ostatní agenty se jim později vyhýbají. Podobný systém, který s názvem *feromonové stopy* (Pheromone Trails) se používá např. při úniku z lokálního minima potenciálových polí [103].

Dále používá tato AI algoritmy pro předpovídání výsledku soubojů nebo pro reaktivní kontrolu (provokování, pronásledování).

6.2.4 UAlbertaBot

UAlbertaBot je jednou z nejznámějších AI pro StarCraft. Její kód je umístěn ve veřejně dostupném repositáři[14], i s přehlednou dokumentací. Díky této přehledné dokumentaci je tato AI oblíbená jako startovní bod pro vývoj nových AI, existuje řada projektů které na projekt UAlbertaBot navazují [104, 105, 48, 106, 107].

UAlbertaBot používá heuristický prohledávací algoritmus pro plánování pořadí výstavby (build order) [50]. Před každou bitvou vyhodnocuje její vý-

[12] `http://wiki.teamliquid.net/starcraft/GarmBot`
[13] `http://bwem.sourceforge.net/Stone.html`
[14] `https://github.com/davechurchill/ualbertabot`

sledek pomocí bitevního simulátoru SparCraft[15], jehož výsledek rozhodne, zdali se bitvy zúčastnit, nebo se jí raději vyhnout. Tento simulátor používá pro něj vyvinutý algoritmus Alpha Beta Considering Durations [56], který je modifikací Alfa-beta ořezávání.

Z architektonického hlediska ale není UAlbertaBot pro tuto práci příliš zajímavý, neboť nepoužívá MAS v žádné ze svých částí.

6.2.5 AIUR

AIUR[16] je další z etablovaných AI pro StarCraft. Zkratka AIUR znamená Artificial Intelligence Using Randomness, umělá inteligence používající náhodu [13]. Její hlavní devizou je právě náhoda, na začátku AI zvolí náhodnou "náladu", podle té se pak řídí produkce jednotek, stavba budov, načasování útoků i další oblasti AI. AIUR se navíc učí, které nálady byly účinné proti kterému protihráči, ty pak vybírá proti konkrétnímu hráči s větší pravděpodobností.

Stejně jako UAlbertaBot neobsahuje AIUR MAS v žádné ze svých částí. V současnosti (2016) to vypadá, že se projekt dále nevyvíjí, poslední příspěvek v repositáři je dva roky starý.

[15] `https://github.com/davechurchill/ualbertabot/wiki/SparCraft-Introduction`

[16] `https://github.com/AIUR-group/AIUR/tree/v2.2`

7 AgentSCAI Framework

Na základě získaných znalostí prezentovaných v předchozích kapitolách byla navržena a implementována AI pro RTS StarCraft. Tato AI byla implementována ve formě frameworku a dostala název AgentSCAI.

7.1 AgentSCAI jako framework

Ještě před motivací a cíli je třeba vysvětlit, proč byla AgentSCAI implementovaná jako framework. Především se ukázalo, že není možné jako základ použít zdrojový kód nějaké existující AI, neboť ty nesplňují požadavky kladené na MAS (viz kapitola 6.2). Přestože se v některých AI prvky MAS vyskytovaly, ani v jednom z případů nebyla celá AI navržena jako MAS (což je cílem této práce). Bylo tedy nutné začít od základů. Některé techniky sice mohly být převzaté, ale základ AI, tedy multiagentní systém, musel být implementován celý v rámci práce.

V průběhu práce na MAS pak vyšlo najevo, že AI, která by dodržovala principy MAS, byla rozšiřitelná a škálovatelná, není možné v rámci jedné práce dopracovat do takového stavu, aby mohla konkurovat etablovaným AI. Tuto hypotézu potvrzují jak existující soutěžní AI (u kterých je většinou důraz na výkon, nikoliv návrh), tak přímo někteří autoři AI [108, 89, 97]).

Proto bylo nakonec rozhodnuto implementovat AI jako framework. Veškerý kód implementující konkrétní jednání AI byl pak oddělen do dalšího projektu. Tento druhý projekt pak slouží jako ukázka (demo) možností frameworku, čímž může značně usnadnit práci budoucím vývojářům.

7.2 Motivace

Hlavní motivací pro tuto práci bylo vytvořit skutečný MAS framework pro RTS. Na rozdíl od jiných AI, implementovaných s využitím prvků MAS (kapitola 6.2), byl zde kladen důraz na celkové řešení implementované jako MAS. V jistém smyslu framework navazuje na AI OpprimoBot (kapitola 6.2.1), která implementuje všechny jednotky jako agenty [97].

Cílem AgentSCAI bylo ale navíc implementovat pomocí agentů i další součásti frameworku, tedy různé řídící a koordinační prvky. Celý framework by tak měl být jakýmsi strojem pro MAS, ve kterém jsou jak agenty s realizací ve hře, tak agenty virtuální. Díky tomu bylo rozhodnuto nenavazovat na existující kód, ale namísto toho implementovat celý framework od základu. Celý kód je tak implementován v rámci této práce, převzatá je pouze část

pracující s mapu (`MapTools`) a systém pro umisťování budov (BuildingPlacer). Tyto části byly převzaté z AI UalbertaBot.

Kromě tohoto motivu měla tvorba frameworku cíle, které jsou popsány v následující kapitole.

7.3 Cíle

Na základě analýzy potřeb AI v RTS ve vztahu k multiagentním systémům, studia problémových oblastí AI v RTS a průzkumu existujících AI s multiagentními prvky i bez nich, byly pro framework stanoveny tyto cíle:

- **Hierarchické velení:** S ohledem na to, že se jedná o AI pro válečnou hru, bylo zvoleno hierarchické velení, na framework se můžeme dívat jako na hierarchický multiagentní systém [6].
- **Nedeterministické chování:** Jednou z hlavních slabin současných AI v souboji proti lidským hráčům je jejich předvídatelnost. Lidskému hráči stačí odhalit vzor/systém, podle kterého se AI řídí a může ho v následující hře porazit. Prvek náhody tuto nevýhodu značně omezuje.
- **Abstrakce a izolace:** Velící agenty mají jen globální (v jistém smyslu abstraktní) informace. Podřízené jednotky mají naopak jen lokální informace, zato však přesné. Tyto vlastnosti mají za následek přirozenější návrh systému, ve vztahu ke skutečnému vedení války. Navíc se zjednoduší případná budoucí implementace prohledávání stavového prostoru pro plánování, neboť agenty budou mít méně informací, na základě kterých by se měli rozhodnout.
- **Absence doménových znalostí:** Framework bude implementován bez doménových znalostí, tedy bez konkrétních strategických nebo taktických dovedností, bez technik reaktivního řízení, navigace nebo analýzy terénu. Budoucí AI používající framework jako svůj základ budou moci doménové znalosti implementovat, tyto ale nebudou součástí frameworku. Výjimku tvoří ukázka (demo) frameworku, kde jsou implementovány jednoduché dovednosti a techniky za účelem demonstrace použití frameworku.

7.4 Použité technologie

7.4.1 Jazyk Java

AgentSCAI framework je napsán v jazyce Java. Tento jazyk byl vybrán z následujících důvodů:

- **Objektovost:** Objektovost jazyka pomáhá dobrému návrhu MAS, zvláště pokud chápeme agenty jako posun objektů směrem k vyšší abstrakci (tak jako v [6]).
- **Jednoduchost:** Jednoduchá syntaxe jazyka umožňuje navrhnout framework tak, aby bylo jasné, kde navazovat.
- **Robustnost:** Silná typová kontrola spolu s genericitou umožňuje stanovit jasná pravidla, jak framework používat

7.4.2 BWMirror API

Stejně jako ostatní AI pro StarCraft používá AgentSCAI k připojení ke hře BWAPI (viz 5.2.3), napsané v jazyce C++. Mapování objektů z jazyka C++ do jazyka Java je realizováno prostřednictvím API BWMirror[1]. Právě přes toto API přistupuje framework AgentSCAI k BWAPI. Celkové schéma přístupu můžeme vidět na obrázku 7.1

BWMirror používá k mapování Java Native Interface[2] [109] (JNI) – rozhraní jazyka Java pro spolupráci s knihovnami napsanými v jiném programovacím jazyce.

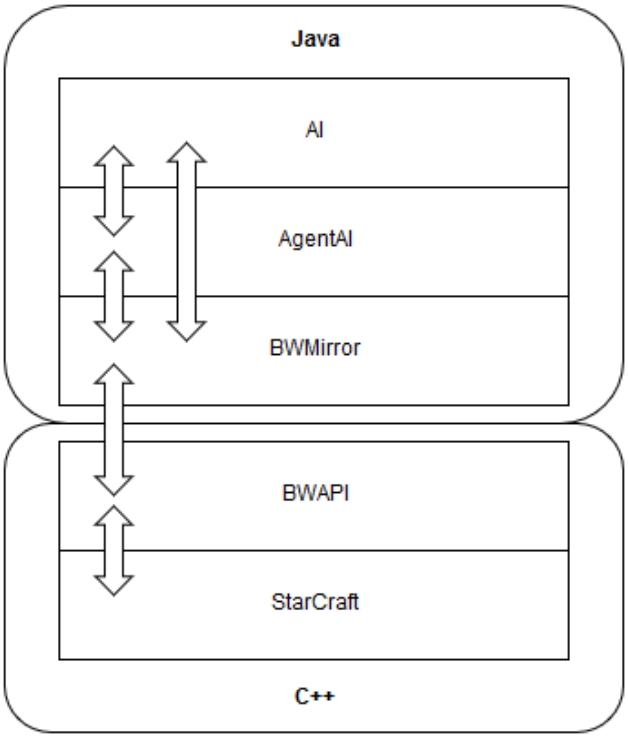
BWMirror mapuje prakticky všechny objekty z BWAPI do jazyka Java. Dále poskytuje některé pomocné metody, které zastiňují odlišnosti jazyka, a celkově posunuje API k více objektovému návrhu.

Spolupráce s knihovnou mimo JVM klade speciální požadavky na virtuální stroj – aplikaci je třeba spouštět ve 32bitové verzi Javy [109].

7.4.3 Knihovna BWTA2

Pro analýzu terénu je použitá knihovna BWTA2[3]. Tato knihovna postavená na práci [70] umožňuje analyzovat celou mapu a rozpoznat na ní důležité body, zejména:

[1] `https://github.com/vjurenka/BWMirror`
[2] `https://docs.oracle.com/javase/8/docs/technotes/guides/jni/`
[3] `https://bitbucket.org/auriarte/bwta2`



Obrázek 7.1: Přístup ke hře StarCraft pomocí frameworku AgentSCAI. AI symbolizuje umělou inteligenci vytvořenou s pomocí frameworku AgentSCAI. Šipky značí komunikaci mezi systémy

- Regiony mapy.
- Uzlové body (choke points) – tedy místa vhodná k obraně. Oddělují regiony.
- Lokace pro základny – tedy místa s možností těžby surovin.

7.5 Architektura

Rozdělení architektury na jednotlivé části bylo provedeno až na výjimky podle principů MAS, zásadní roli v něm hraje agent. Jedná se o netradiční řešení, dekompozice AI je u většiny AI provedena podle expertních znalostí (znalostí lidských hráčů) [110, 20], nikoliv podle principů (multiagentních, nebo jiných).

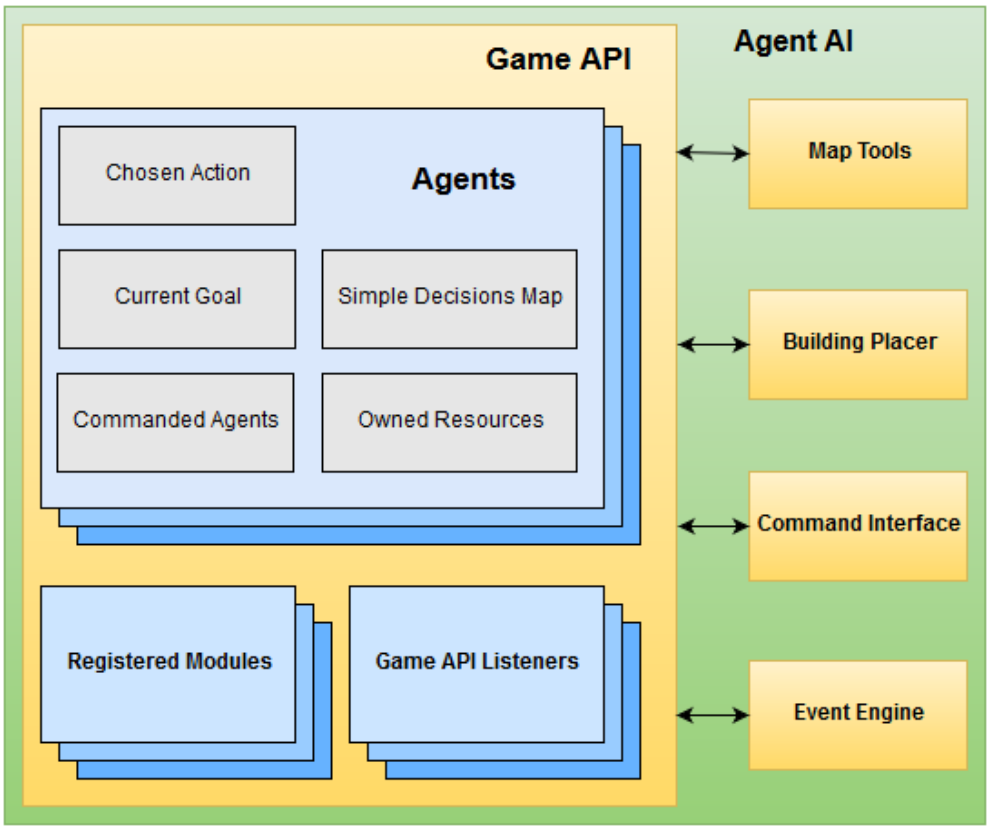
AgentSCAI má dvě základní části: **framework** – na něj by měla navazovat případná implementace AI a **demo** – což je sada příkladů, která ukazuje

možnosti frameworku a poskytuje vzor toho, jak na framework správně navázat. Pokud nebude řečeno jinak, týkají se veškeré informace v této kapitole frameworku, nikoliv dema.

7.5.1 Základní vlastnosti

Aplikace je celkově, s ohledem na to že má jít o implementaci multiagentního systému, navržena decentralizovaně. Přestože je aplikace decentralizovaná, není v ní využita paralelizace. Bylo tak rozhodnuto z toho důvodu, že BWAPI nepodporuje vícevláknový přístup [111]. Samotný kód agentů by sice mohl běžet v různých vláknech, při komunikaci s hrou by ale bylo třeba vždy vlákna synchronizovat.

Kromě agentů má aplikace i několik dalších zásadních komponent. Tyto komponenty jsou popsány v dalších sekcích, schéma celého systému je vidět na obrázku 7.2. Podrobnější schémata se pak nacházejí u popisu jednotlivých komponent.



Obrázek 7.2: Schéma komponent frameworku AgentSCAI

7.5.2 Agenty

Agenty jsou základními prvky AgentSCAI. V každé herní obrátce (*frame*, ve StarCraftu přibližně každých 42ms nová obrátka) vykoná každý agent svoji základní logiku. V AgentSCAI existují dva základní typy agentů:

- **Herní agent (Game Agent):** Agent s reprezentací ve hře (jednotkou, nebo budovou). Svoje cíle plní skrze tuto reprezentaci. Každý druh jednotky má svou vlastní agentní třídu.
- **Velící agent (Command Agent):** Agent reprezentující virtuální jednotku – velitele. Ve hře nemá žádnou reprezentaci, svoje cíle plní prostřednictvím podřízených agentů.

7.5.3 Řetězec velení

Řetězec velení je v AgentSCAI velmi striktní, ale zároveň velmi pružný. Co se týče plnění rozkazů je přísný. Jedním z hlavních cílů na framework bylo hierarchické velení, podřízené jednotky jsou tak vždy poslušné svému veliteli.

Co se týče přesné podoby hierarchie, je framework velmi pružný a dokáže se přizpůsobit různým návrhům AI.

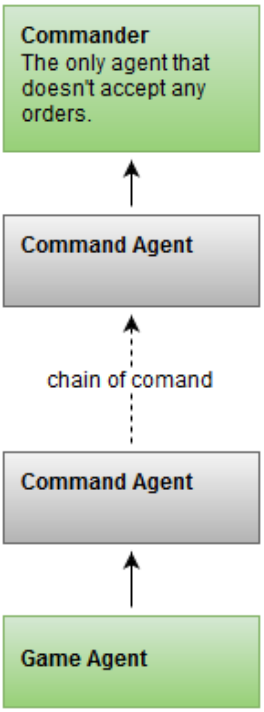
Jak řetězec velení vypadá je vidět na obrázku 7.3. Počet úrovní hierarchie není omezen. Minimální verze (na obrázku zeleně) se skládá z:

- *Commandera*, vrchního velitele, který kromě jiného spravuje všechny zdroje a jednotky.
- Jemu podřízených *herních agentů*, tedy agentů, které zapouzdřují jednotky existující ve hře.

Dále je možné přidat další velitele mezi tyto dvě úrovně a vytvořit tak tří, čtyř, ale i více úrovní velení.

7.5.4 Systém rozkazů

Systém rozkazů je jedním ze základních pilířů AgentSCAI. Bez rozkazů by všichni agenty jednaly jen podle svých defaultních cílů, což v důsledku znamená, že by nespolupracovaly. Skrze rozkazy mohou být strategické cíle přetaveny v cíle taktické, a ty potom v konkrétní akce. Výběr aktivit (aktivity dále v kapitole 7.5.5) agenta bývá nejvíce ovlivně právě příchozími rozkazy, ať už je zvolen jakýkoliv způsob rozhodování (způsoby roz. dále v kapitole 7.5.6).



Obrázek 7.3: Systém velení. Šipky směřují od podřízených agentů k jejich velitelům. Zeleně jsou značeny povinné součásti systému.

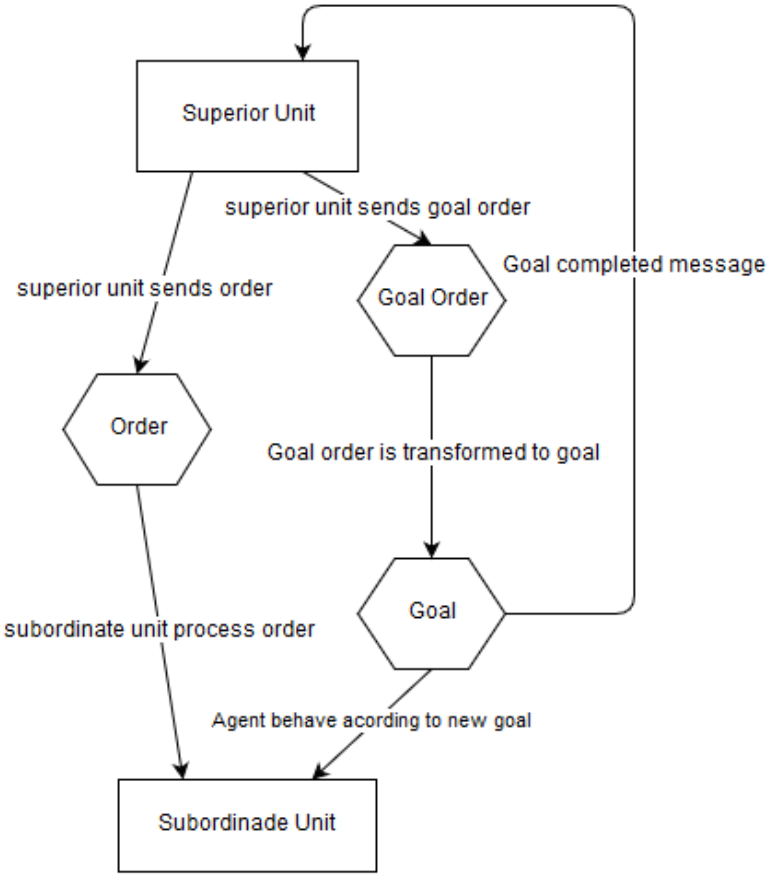
Rozkazy je samozřejmě možné posílat jen od nadřízených jednotek k podřízeným.

Jak můžeme vidět na obrázku 7.4, existují dva základní typy rozkazů:

- Order
- Goal Order

Jednoduchý rozkaz (Order) slouží k příkazu jednoduchých akcí, jako vyžádání informací, převelení jednotek apod. Tyto rozkazy jsou splněny okamžitě a jejich splnění by mělo být zaručené.

Cílový rozkaz (Goal Order) slouží ke změně cíle podřízeného agenta. Nový cíl je přímo obsahem rozkazu. Podřízený agent změní svůj cíl na cíl z rozkazu, tomuto cíli pak přizpůsobuje svoje další aktivity. Splnění cíle kontroluje samotný objekt cíle, v případě splnění cíle pak informuje agenta, z jehož rozkazu cíl vzešel.



Obrázek 7.4: Systém rozkazů

7.5.5 Aktivity

V aktivitách se odehrává téměř veškerá logika AI. Ať už jednotka plánuje strategii, taktiku, nebo někam jde či útočí, rozhodlo se o tom v těle aktivity.

Tak jako je agent základním objektem frameworku, aktivita je základní činností ve frameworku. Téměř veškerá činnost agentů je vykonávána prostřednictvím aktivit. Tyto aktivity může sdílet více agentů, není proto nutné implementovat stejnou činnost pro každý typ agenta zvlášť.

Více úrovní aktivit

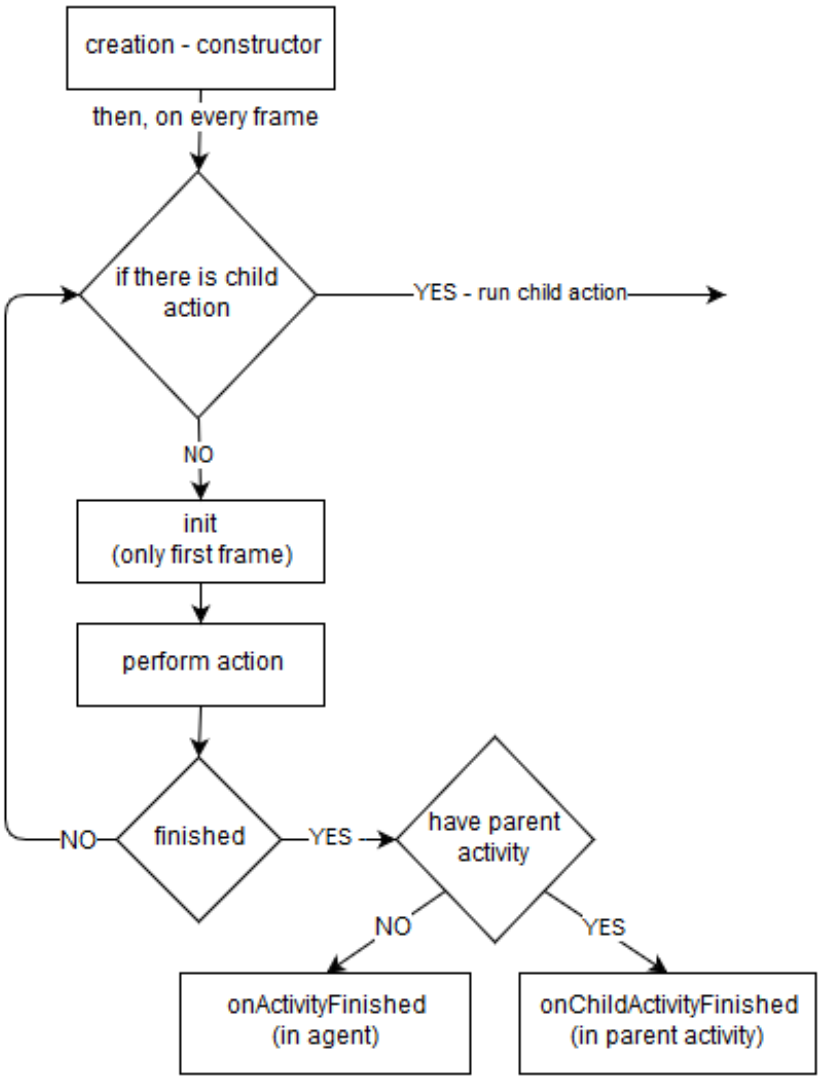
Některé aktivity jsou velmi složité a přirozeně se skládají z dílčích činností, které je možné vykonávat i samostatně. Pro sdílení těchto dílčích činností mezi aktivitami je tu víceúrovňový systém aktivit.

Aktivita volaná z jiné aktivity je její dílčí aktivitou (Child Activity).

Volající aktivita je pak rodičem (Parent Activity)[4]. Jakákoliv aktivita může být volána jako potomek jiné aktivity. Počet úrovní není nijak omezen, tedy dílčí aktivita může mít další dílčí aktivity.

Životní cyklus aktivit

Životní cyklus aktivity vidíme na obrázku 7.5. Vidíme zde volání dílčí aktivity i signál rodiči o dokončení aktivity. Tento signál je vyslán jak v případě, kdy je rodičem jiná aktivita, tak v případě, že je rodičem přímo agent.



Obrázek 7.5: Životní cyklus aktivity

[4]Toto názvosloví nemá nic společného s dědičností objektů

7.5.6 Rozhodování

Rozhodování se v AgentSCAI děje uvnitř agenta, tedy každý agent se rozhoduje, co bude v danou chvíli dělat. Toto rozhodnutí se děje na úrovni aktivit 7.5.5.

Nejjednodušší možností je vybrat aktivitu na základě aktuálního cíle agenta. Toto rozhodování je ve frameworku označeno jako *simple decision making*.

Dále je zde běžné rozhodování u kterého je možné zvolit i další kritéria, jako rasu hráče, současný stav hry apod. Všechny tyto parametry lze využít najednou a rozhodovat se na základě jejich kombinace, tato sada parametrů se pak v AgentSCAI nazývá *decision tables map key* (obrázek 7.6 – vlevo). Tento systém je navržen s důrazem na obecnost, do rozhodování je tak možné vložit téměř libovolný nový parametr (parametr má generický typ), aniž by bylo třeba systém přepracovat. Na základě *decision tables map key* se následně vybere z kolekce rozhodovacích tabulek (*decision tables map*), která je pro každý typ agenta individuální, příslušná rozhodovací tabulka (*decision table*). V případě deterministického rozhodování je v této tabulce právě jeden záznam s vybranou akcí agenta. Tento systém rozhodování je součástí modulu (více o modulech v sekci 7.5.7) s názvem *Decision Module*.

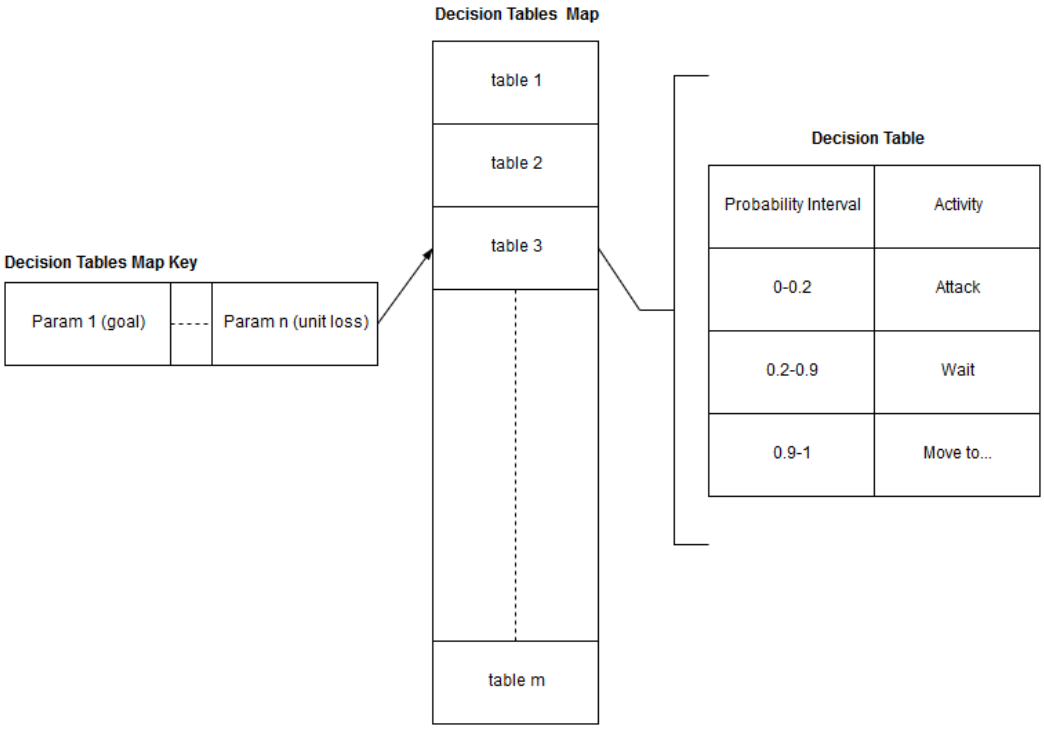
Nedeterministické rozhodování

Kromě jednoznačných rozhodnutí je možné také přidat do rozhodování prvek náhody. To může výrazně zvýšit úspěšnost AI, neboť chování agentů je hůře předvídatelné.

Při rozhodování je možné rozdělit interval mezi 0 a 1 (tedy 0% a 100%) na neomezené množství rozhodnutí. Například pokud je cílem hráče rush a rasa nepřítele je Protoss, můžeme nastavit šance na rozhodnutí tak, že s pravděpodobností 0.7 (70%) podnikne okamžitý rush a s pravděpodobností 0.3 (30%) nejprve expanduje na nejbližší základnu. Rozdělení pravděpodobnosti mezi více akcí docílíme přidáním více záznamů do rozhodovací tabulky, každý záznam obshuje aktivitu a k ní příslušnou pravděpodobnost. Jak takové rozhodování vypadá můžeme vidět na obrázku 7.6 (rozhodovací tabulka vpravo).

7.5.7 Moduly

Moduly jsou klíčem k tomu, jak rozšiřovat framework AgentSCAI nejen tam, kde to je zamýšleno, tedy prostřednictvím nových agentů, akcí, cílů, rozkazů apod., ale i zcela nepředvídaným způsobem. Stačí modul zaregistrovat ve



Obrázek 7.6: Systém Rozhodování. Nejprve je na základě parametrů získaných ze současného stavu hry (decision tables map key) vybrána z kolekce (decision tables map) správná rozhodovací tabulka (decision table). Následně je z této tabulky náhodně (s ohledem na pravděpodobnostní rozdělení záznamů) vybrán jeden záznam, agent pak začne provádět aktivitu z tohoto záznamu.

frameworku a ten je následně volán z různých částí frameworku, těchto volání pak může využít k výkonu své funkce. Zatím jsou implementovány jen tři moduly související s rozhodováním:

- **Decision Module:** Modul pro rozhodování. Bez Tohoto modulu je k dispozici pouze rozhodování ve zjednodušené formě.
- **Decision Storage Module:** Tento modul umožňuje načítání rozhodnutí pro rozhodovací modul ze souborů ve formátu XML.
- **Learning Module:** Modul pro učení. Umožňuje spustit scénář, který v průběhu mnoha her mění rozhodovací parametry agentů, následně pak vybere nejlepší parametry, které uloží do XML.

Moduly mohou mít mezi sebou závislosti (dependencies). Například Modul pro učení závisí na modulu pro práci s rozhodnutími ve formátu XML a ten zase na rozhodovacím modulu.

7.6 Použité návrhové vzory

Protože byla práce implementována jako framework, bylo třeba postupovat pokud možno tak, aby bylo co nejvíce vlastností frameworku vynuceno jazykem Java. To se v zásadě povedlo, v AgentSCAI je jen několik oblastí, kde nejsou konvence vynuceny (tedy lze je obejít). Tohoto stavu bylo dosaženo za použití několika návrhových vzorů.

7.6.1 Copy constructor

Tento známý vzor [112] byl použit pro aktivity, herní agenty a také pro commandera. Jedná se o alternativu k použití klíčového slova `clone`, tedy o kopírování objektu.

U **aktivit** bylo tohoto vzoru použito kvůli rozhodovacímu modulu. Pro jeho použití je třeba nejprve inicializovat tabulky s výběrem aktivity pro určité parametry hry (tedy pro určité stavy hry) ke každému typu agenta. Vlastní rozhodování pak funguje tak, že agent dle parametrů hry vyhledá tabulku aktivit, z této tabulky vybere aktivitu dle svého současného cíle, z aktivity potom vytvoří (pomocí copy konstruktoru) novou instanci.

Nejedná se zde o přesné použití vzoru copy constructor, protože nekopírujeme všechny vlastnosti aktivity, ale jen některé její parametry, zatímco stavové proměnné, ale třeba i některé parametry nás nezajímají. Příkladem může být aktivita skupinového útoku. Zatímco velikost útočné skupiny se kopíruje z původní aktivity, cíl útoku nebo proměnná určující dosažení cíle se vytváří až při výběru aktivity.

V případě **herních agentů** se vzor copy constructor používá při registraci typů herních agentů pro jednotky ve hře. Když se pak ve hře objeví nová jednotka, v mapě registrovaných herních agentů se vyhledá příslušný herní agent, ze kterého se vytvoří nová instance inicializovaná objektem jednotky.

U **commandera** copy constructor slouží k opětovnému vytvoření commandera v případě opakovaných her. V nové hře se kopírují jen parametry commandera (počáteční cíl), stavové proměnné commandera se inicializují v každé nové hře na defaultní hodnoty.

Kód 7.1: Bezpečná generická metoda pro výběr z kolekce

```
protected final <T> T getCommandedAgent(Class<T>
    agentClass) {
     for (Agent subordinateAgent : commandedAgents) {
          if (agentClass.isInstance(subordinateAgent)) {
               return (T) subordinateAgent;
          }
     }
     return null;
}
```

7.6.2 Omezování generických parametrů u odvozených tříd

Tato metoda je zhusta použita například u aktivit. Každá aktivita obsahuje referenci na agenta, který aktivitu provozuje. Agent je v aktivitě určen generickým parametrem. Hlavním účelem tohoto parametru ale není, tak jako je to u generických typů obvyklé, umožnit využití jedné třídy s více generickými parametry. Tento generický parametr je totiž omezen (většinou na konkrétní typ) u odvozených tříd (odvozených aktivit). Takový způsob umožňuje značnou a bezpečnou úsporu kódu, neboť odkaz na agenta je používán v rámci hierarchie aktivit na mnoha úrovních, aniž by bylo nutné ho přetypovat. Jedná se vlastně o bezpečné a efektivní prostoupení polymorfismu do oblasti generických vlastností. V rámci dědičnosti takto můžeme měnit nejen třídy, ale i jejich vlastnosti.

7.6.3 Bezpečná generická metoda pro výběr z kolekce

Touto metodou byl implementován výběr z kolekce podřízených agentů. Bylo totiž velmi příhodné získat metodu, která vrátí z kolekce konkrétní typ agenta tak, aby bylo zaručeno že nedojde k chybě při přetypování. Výsledek je vidět v kódu 7.1.

7.6.4 Command pattern

Pro udílení rozkazů, posílání požadavků a komunikaci byl použit tzv. *command pattern* [113]. Tento vzor umožňuje již při vytvoření rozkazu zafixovat jeho příjemce, čím lze předejít řadě potenciálních chyb.

7.6.5 Observer pattern

Tento známý návrhový vzor je v AgentSCAI využit ve dvou případech. Prvním případem je GameAPI - hlavní objekt frameworku. K tomuto objektu je možno se registrovat jako pozorovatel. Tato registrace slouží pouze k volání metody `onStart` u pozorovatele. Toto volání je nutné pro registraci herních agentů k typům jednotek ve hře, neboť tato registrace musí (kvůli BWAPI) proběhnout až po startu hry.

Druhým případem užití tohoto vzoru je *event engine*, který má jako registrovaného pozorovatele GameAPI, která tak zachytí potřebné události.

7.7 Použité MAS techniky

Jak již bylo napsáno, celý framework je implementován jako multiagentní systém. Prostředí pro běh agentů tvoří základ celého frameworku. Všechny akce jednotek a budov ve hře jsou následkem jednání agentů. Implementace samotných agentů již byla probrána v kapitole 7.5. Zde jsou uvedeny některé další multiagentní techniky, které dotvářejí obraz frameworku AgentSCAI jako plnohodnotného multiagentního systému. Jedná se o:

- **Koordinaci** agentů prostřednictvím rozkazů, žádostí a zasílání informací. Tato oblast je popsaná v kapitole 7.7.1.
- **Lokalita** agentů – v AgentSCAI není žádná centrální databáze informací. Agenti nemají informace o herním světě, s výjimkou informací z jejich okolí a informací obdržených pomocí rozkazů a zpráv od ostatních agentů
- **Vnímaní světa skrze senzory**: Framework pro agenty simuluje některé senzorické informace (objevené jednotky). Tato vlastnost frameworku je popsaná v kapitole 7.8.2
- **Autonomní rozhodování** agentů, probrané v kapitole 7.5.6. Ať už se agenti rozhodují pomocí systému jednoduchých rozhodnutí nebo používají rozhodovací modul, jejich rozhodnutí jsou vždy samostatná. Jedná se tedy o decentralizované rozhodování.

7.7.1 Koordinace

Co se týče koordinace agentů, není AgentSCAI příliš striktní. Teoreticky je možné ovládat jednotky jen pomocí rozkazů, bez jakékoliv zpětné vazby. Framework však kromě rozkazů obsahuje ještě informace (informační zprávy) a

požadavky (obdoba informace, ale od adresáta se očekává reakce). Pomocí těchto nástrojů spolu mohou agenty libovolně komunikovat. Hustota komunikace pak záleží na návrhu konkrétního chování.

Tato volnost je nejlépe ukázána v demu v ukázce formací (viz obrázek 7.7: (`FormationDemoStarter`). Zde Postupují jednotky vpřed tak, že obklopují v půlkruhu VIP jednotku ve svém středu. Formace přitom může být sestavena buď centrálně velitelem skupiny, nebo dohodou agentů mezi sebou. V obou případech je výsledek stejný, agenty se nakonec seřadí správně. Tím se ukazuje, že v případě vzájemné komunikace není v AgentSCAI jen jedno správné řešení, ale naopak že je možné svobodně zvolit míru vzájemné komunikace.

Obrázek 7.7: Ukázka formace

7.8 Vlastnosti frameworku

Kromě dříve zmíněných má framework AgentSCAI další důležité vlastnosti. Ty jsou postupně probrány v této kapitole.

7.8.1 Učení

AgentSCAI obsahuje modul učení. Tento modul umožňuje vytvoření vlastích scénářů učení. Během scénáře se sledují agenty registrované v modulu učení

a parametry jejich rozhodnutí. Tyto parametry se pak na konci hry spolu s výsledkem hry ukládají do XML formátu. Scénář umožňuje pro každou hru zvolit jiné parametry aktivit, díky tomu je možné získat porovnatelné výsledky hry jak pro různé aktivity, tak pro různé parametry aktivit.

Výsledky pak umí modul učení i vyhodnotit. Ve frameworku je k dispozici zatím jen základní algoritmus učení, který vybírá nastavení s nejlepším průměrným výsledkem. Modul je ale navržen tak, že se dá tento systém rozšířit bez zásahu do frameworku (tedy i bez zásahu do rozhodovacího modulu).

Naučené parametry nakonec modul uloží opět do XML. Optimální konfigurace pak může být načtena pomocí modulu pro načítání rozhodnutí z XML a použita ve hře.

7.8.2 Systém událostí

Framework plně využívá systém událostí dostupný z BWAPI prostřednictvím BWMirror. Navíc tento systém transformuje tak, aby emuloval multiagentní prostředí. Například v případě, že konkrétní jednotka vidí ve hře nepřátelskou jednotku, BWAPI vytvoří příslušnou událost, tato událost ovšem nemá žádný vztah k tomu, která jednotka nepřítele odhalila. AgentSCAI se pak postará, že se událost dostane k příslušné jednotce, čímž se emulují senzorické schopnosti agenta typické pro MAS.

Obdobně i další herní události jsou propagovány příslušným agentům, nikoliv do nějaké centrální databáze informací. Tím se sice snižuje informovanost agentů, ale na druhou stranu se tak vytváří přirozené multiagentní prostředí.

7.8.3 Rozšiřitelnost

Framework je možné rozšířit při tvorbě AI téměř ve všech směrech. Je možné:

- Nastavit vlastní **rozhodovací mapy**, která přiřazuje parametry stavu hry k aktivitám pro danou jednotku.
- Vytvořit vlastní **agenty** ať už úplně od základu nebo odvozením od agentů existujících. Tyto agenty je možné ve hře tvořit dle potřeby jako virtuální, nebo je dokonce zaregistrovat jako herní agenty pro příslušnou jednotku. Zatím je pomocí herních agentů implementováno 14 typů jednotek, díky možnosti registrace ale není problém implementovat nové herní agenty a ty pro potřebnou jednotku zaregistrovat.

- Implementovat vlastní **aktivity**, opět odvozením nebo od základu tak jako agenty. Tyto aktivity lze pak přidat agentům pomocí rozhodovacího modulu nebo jen pro jednoduché rozhodování.
- Vytvořit nové **rozkazy**, **cíle**, **informace** nebo **žádosti**.
- Implementovat vlastní rozšiřující **moduly**.
- Implementovat vlastní **parametry pro rozhodování**. Charakter ani počet těchto parametrů není nijak omezen.

7.8.4 Další důležité vlastnosti

- **Jsou dostupné všechny rasy:** Framework nabízí nástroje pro vývoj AI pro všechny rasy. Vývojář má tak svobodu volby, pro jakou rasu chce AI vyvíjet. Navíc je možné tvořit AI pro více ras (i všechny tři) což může být konkurenční výhodou, neboť i soupeři pak musí připravit AI schopnou reagovat na všechny tři rasy. AI které pracují se všemi rasami zatím mnoho není, v turnajích ji zatím použil pouze UAlbertaBot [86].
- **Automatické načtení herních agentů:** Agenty reprezentující jednotky ve hře mají ve frameworku nejen svojí třídu, ale tato třída je navíc automaticky instancována, objeví-li se ve hře daná jednotka. Tvorbu herních agentů je tak možné nechat na frameworku, není třeba se o nic starat.

7.9 Prezentace projektu

Zdrojové kódy celého projektu jsou verzované pomocí systému git a nachází se na serveru Github[5]. Tam se také nachází:

- **Návod na instalaci[6]:** V návodu na instalaci je popsáno jak založit AI projekt na frameworku AgentSCAI. Zde jsou také popsány požadavky na instalaci dalších nástrojů (BWAPI, samotná hra StarCraft...).
- **Návod na implementaci vlastní AI[7]:** V návodu je popsáno jak na framework navázat a používat jeho jednotlivé části. V této části je také návod na zprovoznění ukázkového projektu – dema.
- **Popis architektury frameworu[8]:** Aktualizovaný popis architektury

[5] `https://github.com/F-I-D-O/AgentSCAI`
[6] `https://github.com/F-I-D-O/AgentSCAI/wiki/Installation`
[7] `https://github.com/F-I-D-O/AgentSCAI/wiki/Build-your-own-bot`
[8] `https://github.com/F-I-D-O/AgentSCAI/wiki/Architecture`

frameworku ve formě wiki.

- **Demo[9]:** Zdrojové kódy dema. V těchto ukázkových kódech je předvedena na příkladech prakticky veškerá funkčnost frameworku.
- **Instruktážní videa:** Na stránkách projektu se také nacházejí krátká instruktážní videa, která pomohou s prvními kroky ve frameworku a jsou tak doplňkem k ostatním informacím na webu projektu.

7.9.1 Demo

Demo je ukázkový projekt k frameworku AgentSCAI. Představuje možnosti frameworku na několika příkladech. Celkem se jedná o pět ukázek:

- `BBSStarter`: Ukázková AI hrající za rasu Terran. Uplatňuje rush strategii BBS (barracks barracks supply).
- `OutbreakStarter`: Ukázková AI pro rasu Zerg. Kromě těžby surovin zvládá expanzi na novou základnu.
- `ProtossStarter`: Ukázková AI za rasu Protoss – pouze sběr surovin.
- `Starter`: Ukázka používání modulu pro načítání rozhodnutí ze souboru a modulu učení.
- `FormationDemoStarter`: Ukázka centralizovaného a decentralizovaného používání formací.

Všechny ukázky byly testované na mapě *Destination*, s výjimkou ukázky formací, která jde spustit pouze na speciální mapě *formationTest*, která je součástí repositáře a je nutné ji nainstalovat[10].

[9] `https://github.com/F-I-D-O/AgentSCAI/tree/master/demo`
[10] Návod na instalaci map je součástí návodu na instalaci.

8 Závěr

V této práci bylo navrženo multiagentní řešení AI v RTS. V kapitole 3 bylo nastíněno pozadí práce. Dále byly v této kapitole uvedeny základní rozdíly mezi RTS a klasickými hrami, ukázalo se, že tyto rozdíly jsou významné, zvláště pak odhadnutá složitost hry se ukázala jako dramaticky vyšší než u klasických her. Dále byly určeny specifické požadavky na AI pro RTS. Nakonec byly úlohy AI pro RTS rozděleny do několika oblastí.

Jednotlivé oblasti AI pak byly prozkoumány v kapitole 4. Bylo nalezeno velké množství zdrojů věnujících se tématu tvorby AI do RTS, především pak na platformě StarCraft. Průzkum dále ukázal, že prací zaměřujících se na MAS principy v AI pro RTS je málo.

V kapitole 5 byly analyzovány různé platvormy pro tvorbu AI do RTS. Bylo prozkoumáno celkem šest platforem. Následně byla vybrána pro vývoj v rámci této práce platforma StarCraft, která byla dále podrobně rozebrána.

Dále byl v kapitole 6 shrnut současný stav vývoje v oblasti MAS pro AI do RTS. Především byly zmíněny odborné práce na toto téma a srovnány existující AI s prvky MAS.

Nakonec byl v kapitole 7 navržen framework AgentSCAI implementovaný dle principů MAS. V úvodu této kapitoly byl vymezen posun od existujících AI k AgentSCAI, jeho přínosy a cíle. Následně byl framework implementován dle návrhu. Framework je navržen tak, aby byl schopen řešit všechny představitelné problémy AI v RTS. Problémy, na které není framework připraven v současném stavu, je možné implementovat pomocí rozšiřujících modulů.

Implementace je prezentována prostřednictvím veřejného repositáře spojeného s webovými stránkami, prezentace je popsána v kapitole 7.9.

Celkově se tedy podařilo prozkoumat vlastnosti vývoje AI pro RTS, analyzovat současný stav vývoje, vybrat vývojovou platformu, navrhnout framework pro vývoj AI a ten nakonec implementovat. Především byl pak splněn hlavní cíl, totiž realizovat práci s využitím principů MAS.

8.1 Budoucí vývoj

Práce byla koncipována jako MAS framework. K frameworku je přiloženo demo, které ale slouží pouze k demonstračním účelům. V budoucnu by bylo vhodné navázat vývojem kompetitivní AI založené na tomto frameworku. Kromě zdokonalení frameworku by to vedlo především k propagaci, která je nutná, s ohledem na to, aby byl framework do budoucna využívaný komunitou vývojářů. Práci by bylo také vhodné propagovat dalšími kanály,

například založením kanálu na sociálních sítích.

Další možnost vývoje se nabízí v oblasti rozšiřování frameworku. V budoucnu by bylo možné rozšířit framework pomocí převzetí osvědčených metod z existujících AI, tak jako byla převzata část kódu z AI UAlbertaBot. Potenciál je například pro implementaci některých aktivit s využitím algoritmů reaktivního řízení z existujících AI nebo rozšíření množství implementovaných jednotek, budov a jejich speciálních akcí. Další možností je například implementovat učení se proti konkrétním AI, což se ukázalo jako efektivní strategie na turnajích AI [20].

Bibliografie

[1] Stan Franklin a Art Graesser. „Is It an Agent, or Just a Program?: A Taxonomy for Autonomous Agents“. In: *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*. ECAI '96. London, UK, UK: Springer-Verlag, 1997, s. 21–35. ISBN: 3-540-62507-0. URL: http://dl.acm.org/citation.cfm?id=648203.749270.

[2] Nicholas R. Jennings, Katia Sycara a Michael Wooldridge. „A Roadmap of Agent Research and Development“. In: *Autonomous Agents and Multi-Agent Systems* 1.1 (led. 1998), s. 7–38. ISSN: 1387-2532. DOI: 10.1023/A:1010090405266. URL: http://dx.doi.org/10.1023/A:1010090405266.

[3] Stuart J. Russell a Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2. vyd. Pearson Education, 2003. ISBN: 0137903952.

[4] Michael Wooldridge. *An Introduction to MultiAgent Systems*. 2nd. Wiley Publishing, 2009.

[5] Pattie Maes. „Artificial Life Meets Entertainment: Lifelike Autonomous Agents“. In: *Commun. ACM* 38.11 (lis. 1995), s. 108–114. ISSN: 0001-0782. DOI: 10.1145/219717.219808. URL: http://doi.acm.org/10.1145/219717.219808.

[6] Aleš Kubík. *Inteligentní agenty. Tvorba aplikačního software na bázi multiagentních systémů*. 2004. ISBN: 80-251-0323-4.

[7] Jazyková poradna ÚJČ AV ČR. *Životnost podstatných jmen*. URL: http://pirucka.ujc.cas.cz/?id=201 (cit. 24. 04. 2016).

[8] Liviu Panait a Sean Luke. „Cooperative Multi-Agent Learning: The State of the Art“. In: *Autonomous Agents and Multi-Agent Systems* 11.3 (lis. 2005), s. 387–434. ISSN: 1387-2532. DOI: 10.1007/s10458-005-2631-2. URL: http://dx.doi.org/10.1007/s10458-005-2631-2.

[9] John McCarthy. *What is artificial intelligence?* basic questions about science discipline. Stanford University, 2012.

[10] Santiago Ontañón et al. „Encyclopedia of Computer Graphics and Games“. In: (2015), s. 1–12.

[11] Alberto Uriarte a Santiago Ontañón. „A Benchmark for StarCraft Intelligent Agents“. In: *AIIDE*. 2015.

[12] B. G. Weber et al. „Reactive planning idioms for multi-scale game AI“. In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. Srp. 2010, s. 115–122. DOI: 10.1109/ITW.2010.5593363.

[13] Florian Richoux. *Artificial Intelligence Using Randomness*. AIUR-group. URL: http://aiur-group.github.io/AIUR/ (cit. 19. 04. 2016).

[14] Johan Hagelbäck a Stefan J. Johansson. „A Multiagent Potential Field-Based Bot for Real-Time Strategy Games“. In: *International Journal of Computer Games Technology* (2009).

[15] David Churchill. *UAlbertaBot github page*. URL: https://github.com/davechurchill/ualbertabot (cit. 20. 04. 2016).

[16] Marlos C. Machado, Gisele L. Pappa a Luiz Chaimowicz. „Characterizing and Modeling Agents in Digital Games“. In: *2012 Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*. 2012.

[17] Murray Campbell, A. Joseph Hoane Jr. a Feng-hsiung Hsu. „Deep Blue“. In: *Artificial Intelligence 134* (2002), s. 57–83.

[18] *Artificial intelligence: Google's AlphaGo beats Go master Lee Se-dol*. BBC. 12. břez. 2016. URL: http://www.bbc.com/news/technology-35785875 (cit. 15. 04. 2016).

[19] Jonathan Cheng. *Computers That Crush Humans at Games Might Have Met Their Match: 'StarCraft'*. The Wall Street Journal. 22. dub. 2016. URL: http://www.wsj.com/articles/computers-that-crush-humans-at-games-might-have-met-their-match-starcraft-1461344309 (cit. 05. 05. 2016).

[20] Santiago Ontañón et al. „A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft“. In: *IEEE Transactions on Computational Intelligence and AI in Games* (2013), s. 293–311.

[21] Gabriel Synnaeve. „Bayesian Programming and Learning for Multi-Player Video Games. Application to RTS AI“. Dis. Institut National Polytechnique de Grenoble, 2012.

[22] Michael Buro. „Real-Time Strategy Games: A New AI Research Challenge“. In: *International Joint Conferences on Artificial Intelligence*. 2003, s. 1534–1535.

[23] Eugenio Bargiacchi et al. „Decentralized Solutions and Tactics for RTS“. In: *25th Benelux conference on Artificial Intelligence*. 2013, s. 372–373.

[24] Marcus Svensson. „Dynamic Strategy in Real-Time Strategy Games“. Degree project. Linnaeus University, 2015.

[25] Damian Isla. „Handling Complexity in the Halo 2 AI“. In: *GDC 2005*. 2005.

[26] Gabriel Synnaeve a Pierre Bessière. „A Bayesian Model for Plan Recognitionin RTS Games Applied to StarCraft“. In: *2011 IEEE Symposium on Computational Intelligence and Games (CIG)*. 2011, s. 79–84.

[27] G. Synnaeve a P. Bessière. „Special tactics: A Bayesian approach to tactical decision-making“. In: *2012 IEEE Conference on Computational Intelligence and Games (CIG)*. Zář. 2012, s. 409–416. DOI: 10.1109/CIG.2012.6374184.

[28] G. Synnaeve a P. Bessière. „A Bayesian model for RTS units control applied to StarCraft“. In: *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*. Srp. 2011, s. 190–196. DOI: 10.1109/CIG.2011.6032006.

[29] Ricardo Parra a Leonardo Garrido. „Advances in Computational Intelligence: 11th Mexican International Conference on Artificial Intelligence, MICAI 2012, San Luis Potosí, Mexico, October 27 – November 4, 2012. Revised Selected Papers, Part II“. In: ed. Ildar Batyrshin a Miguel González Mendoza. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. Kap. Bayesian Networks for Micromanagement Decision Imitation in the RTS Game Starcraft, s. 433–443. ISBN: 978-3-642-37798-3. DOI: 10.1007/978-3-642-37798-3_38. URL: http://dx.doi.org/10.1007/978-3-642-37798-3_38.

[30] C. Miles et al. „Co-Evolving Influence Map Tree Based Strategy Game Players“. In: *2007 IEEE Symposium on Computational Intelligence and Games*. Dub. 2007, s. 88–95. DOI: 10.1109/CIG.2007.368083.

[31] Jay Young a Nick Hawes. „Evolutionary Learning of Goal Priorities in a Real-Time Strategy Game“. In: *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-12)*. Říj. 2012.

[32] Marc Ponsen. „Improving adaptive game AI with evolutionary learning“. Dipl. Delft University of Technology, 2004.

[33] Nasri Othman et al. „Simulation-based Optimization of StarCraft Tactical AI through Evolutionary Computation“. In: *CIG*. 2012.

[34] Martin Čertický a Michal Čertický. „Evolving Reactive Micromanagement Controller for Real-Time Strategy Games“. In: *Scientific Conference of Young Researchers*. 2015, s. 225–228.

[35] Jørgen Bøe Svendsen a Espen Auran Rathe. „Micromanagement in StarCraft using Potential Fields tuned with a Multi-Objective Genetic Algorithm“. Dipl. Norwegian University of Science a Technology, 2012.

[36] R. K. Balla a A. Fern. „UCT for tactical assault planning in real-time strategy games“. In: *21st International Joint Conference on Artificial Intelligence*. 2009, s. 40–45.

[37] Alberto Uriarte a Santiago Ontañón. *High-Level Representations for Game-Tree Search in RTS Games*. 2014. URL: `http://www.aaai.org/ocs/index.php/AIIDE/AIIDE14/paper/view/9015`.

[38] Wang Zhe et al. „Using Monte-Carlo Planning for Micro-Management in StarCraft“. In: *4th Annual Asian GAME-ON Conference on Simulation and AI in Computer Games (GAMEON ASIA)*. 2012, s. 33–35.

[39] Ben G. Weber, Michael Mateas a Arnav Jhala. „Applying Goal-Driven Autonomy to StarCraft“. In: *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. 2010.

[40] Michael Mateas a Andrew Stern. „A Behavior Language for Story-Based Believable Agents“. In: *IEEE Intelligent Systems* 17.4 (čvc 2002), s. 39–47. ISSN: 1541-1672. DOI: `10.1109/MIS.2002.1024751`. URL: `http://dx.doi.org/10.1109/MIS.2002.1024751`.

[41] Anders Dahlbom a Lars Niklasson. „Goal-Directed Hierarchical Dynamic Scripting for RTS Games“. In: *AIIDE*. 2006, s. 21–28. URL: `http://www.aaai.org/Papers/AIIDE/2006/AIIDE06-008.pdf`.

[42] Santiago Ontañón et al. „Case-Based Planning and Execution for Real-Time Strategy Games“. In: *International Conference on Case-based Reasoning*. 2007.

[43] Santiago Ontañón et al. „Soft Computing Applications in Industry“. In: ed. Bhanu Prasad. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. Kap. Learning from Demonstration and Case-Based Planning for Real-Time Strategy Games, s. 293–310. ISBN: 978-3-540-77465-5. DOI: 10.1007/978-3-540-77465-5_15. URL: http://dx.doi.org/10.1007/978-3-540-77465-5_15.

[44] Kinshuk Mishra, Santiago Ontañón a Ashwin Ram. „Situation Assessment for Plan Retrieval in Real-Time Strategy Games“. In: *9th European Conference on Case-Based Reasoning (ECCBR 2009)*. 2008.

[45] Pedro Cadena a Leonardo Garrido. „Advances in Artificial Intelligence: 10th Mexican International Conference on Artificial Intelligence, MICAI 2011, Puebla, Mexico, November 26 - December 4, 2011, Proceedings, Part I“. In: ed. Ildar Batyrshin a Grigori Sidorov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. Kap. Fuzzy Case-Based Reasoning for Managing Strategic and Tactical Reasoning in StarCraft, s. 113–124. ISBN: 978-3-642-25324-9. DOI: 10.1007/978-3-642-25324-9_10. URL: http://dx.doi.org/10.1007/978-3-642-25324-9_10.

[46] Ulit Jaidee, Héctor Muñoz-avila a David W. Aha. „Case-based learning in goal-driven autonomy agents for real-time strategy combat tasks“. In: *In M.W. Floyd & A.A. Sánchez-Ruiz (Eds.) Case-Based Reasoning in Computer Games: Papers from the ICCBR Workshop. U.* 2011.

[47] Martin Čertický a Michal Čertický. „Case-Based Reasoning for Army Compositions Real-Time Strategy Games“. In: *Scienti c Conference of Young Researchers*. 2013, s. 70–73.

[48] Martin Rooijackers a Philippa Warr. *StarCraft: Building A Brilliant Brood War Bot*. Rock, Paper, Shotgun. 28. led. 2016. URL: https://www.rockpapershotgun.com/2016/01/28/starcraft-ai-bots/#more-343753 (cit. 18. 04. 2016).

[49] Ben G. Weber a Michael Mateas. „Case-Based Reasoning for Build Order in Real-Time Strategy Games“. In: *Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2009)*. Říj. 2009.

[50] David Churchill a Michael Buro. „Build Order Optimization in StarCraft“. In: *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*. 2011, s. 14–19.

[51] Ben G. Weber a Michael Mateas. „A data mining approach to strategy prediction“. In: *CIG'09: Proceedings of the 5th international conference on Computational Intelligence and Games*. IEEE Press. Piscataway, NJ, USA: IEEE Press, 2009, s. 140–147. ISBN: 978-1-4244-4814-2.

[52] Ethan Dereszynski et al. „Learning probabilistic behavior models in real-time strategy games“. In: *Artifcial Intelligence and Interactive Digital Entertainment (AIIDE)*. 2011.

[53] Frederik Schadd, Er Bakkes a Pieter Spronck. „Opponent modeling in real-time strategy games“. In: *Proceedings of the GAME-ON 2007*. 2007, s. 61–68.

[54] S. Hladky a V. Bulitko. „An evaluation of models for predicting opponent positions in first-person shooter video games“. In: *2008 IEEE Symposium On Computational Intelligence and Games*. Pros. 2008, s. 39–46. DOI: 10.1109/CIG.2008.5035619.

[55] Manu Sharma et al. „Transfer Learning in Real-time Strategy Games Using Hybrid CBR/RL“. In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. IJCAI'07. Hyderabad, India: Morgan Kaufmann Publishers Inc., 2007, s. 1041–1046. URL: http://dl.acm.org/citation.cfm?id=1625275.1625444.

[56] David Churchill, Abdallah Saffidine a Michael Buro. „Fast Heuristic Search for RTS Game Combat Scenarios“. In: *AIIDE*. 2012, s. 112–117.

[57] D. Churchill a M. Buro. „Portfolio greedy search and simulation for large-scale combat in starcraft“. In: *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. Srp. 2013, s. 1–8. DOI: 10.1109/CIG.2013.6633643.

[58] Bhaskara Marthi et al. „Concurrent Hierarchical Reinforcement Learning“. In: *IN PROCEEDINGS IJCAI-2005*. 2005, s. 779–785.

[59] L. Liu a L. Li. „Regional Cooperative Multi-agent Q-learning Based on Potential Field“. In: *2008 Fourth International Conference on Natural Computation*. Sv. 6. Říj. 2008, s. 535–539. DOI: 10.1109/ICNC.2008.173.

[60] S. Wender a I. Watson. „Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft:Broodwar“. In: *2012 IEEE Conference on Computational Intelligence and Games (CIG)*. Zář. 2012, s. 402–408. DOI: 10.1109/CIG.2012.6374183.

[61] Ulit Jaidee a Héctor Muñoz-Avila. „CLASS Q-L: A Q-Learning Algorithm for Adversarial Real-Time Strategy Games“. In: *AIIDE*. 2012.

[62] Douglas Demyen a Michael Buro. „Efficient Triangulation-based Pathfinding“. In: *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*. AAAI'06. Boston, Massachusetts: AAAI Press, 2006, s. 942–947. ISBN: 978-1-57735-281-5. URL: http://dl.acm.org/citation.cfm?id=1597538.1597687.

[63] Nathan R. Sturtevant. „Benchmarks for Grid-Based Pathfinding“. In: *IEEE Transactions on Computational Intelligence and AI in Games* (2012), s. 144–148.

[64] Mauro Massari, Giovanni Giardini a Franco Bernelli-Zazzera. „Autonomous navigation system for planetary exploration rover based on artificial potential fields“. In: *Dynamics and Control of Systems and Structures in Space (DCSSS) 6th Conference*. 2004.

[65] Johan Hagelbäck. „Multi-Agent Potential Field Based Architectures for Real-Time Strategy Game Bots“. Dis. Blekinge Institute of Technology, 2012. Kap. 8. ISBN: 978-91-7295-223-2.

[66] Serhij Lebedynskyj. „Methods of multi-agent movement control and coordination of groups of mobile units in a real-time strategy games“. Dipl. Czech Technical University in Prague, 2015.

[67] Thomas Willer Sandberg. „Evolutionary Multi-Agent Potential Field based AI approach for SSC scenarios in RTS games“. Dipl. IT University of Copenhagen, 2011.

[68] D. Hunter Hale, G. Michael Youngblood a Priyesh N. Dixit. „Automatically-generated Convex Region Decomposition for Real-time Spatial Agent Navigation in Virtual World“. In: *Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2008, s. 173–178.

[69] Frederick W. P. Heckel, G. Michael Youngblood a D. Hunter Hale. „Influence Points for Tactical Information in Navigation Meshes“. In: *Proceedings of the 4th International Conference on Foundations of Digital Games*. FDG '09. Orlando, Florida: ACM, 2009, s. 79–85. ISBN: 978-1-60558-437-9. DOI: 10.1145/1536513.1536535. URL: http://doi.acm.org/10.1145/1536513.1536535.

[70] Luke Perkins. „Terrain Analysis in Real-Time Strategy Games: An Integrated Approach to Choke Point Detection and Region Decomposition“. In: *Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. 2010, s. 168–173.

[71] Michal Certický. „Implementing a Wall-In Building Placement in StarCraft with Declarative Programming“. In: *CoRR* abs/1306.4460 (2013). URL: http://arxiv.org/abs/1306.4460.

[72] Florian Richoux, Alberto Uriarte a Santiago Ontañón. „Walling in Strategy Games via Constraint Optimization“. In: *AIIDE*. 2014.

[73] *BA 7.81 Released!* Spring Engine Forum. 18. srp. 2013. URL: https://springrts.com/phpbb/viewtopic.php?t=30827 (cit. 05. 05. 2016).

[74] *Spring Engine - AI Skirmish List*. URL: https://springrts.com/wiki/AI:Skirmish:List (cit. 25. 04. 2016).

[75] Sindre Berg Stene. „Artificial Intelligence Techniques in Real-Time Strategy Games - Architecture and Combat Behavior“. Dipl. Institutt for datateknikk og informasjonsvitenskap, 2006.

[76] Michael Buro. „Computers and Games: Third International Conference, CG 2002, Edmonton, Canada, July 25-27, 2002. Revised Papers“. In: ed. Jonathan Schaeffer, Martin Müller a Yngvi Björnsson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. Kap. ORTS: A Hack-Free RTS Game Environment, s. 280–291. ISBN: 978-3-540-40031-8. DOI: 10.1007/978-3-540-40031-8_19. URL: http://dx.doi.org/10.1007/978-3-540-40031-8_19.

[77] Samuel Wintermute, Joseph Xu a John E. Laird. *SORTS: A Human-Level Approach to Real-Time Strategy AI*. 2007.

[78] Laurent Navarro a Vincent Corruble. „Entertainment Computing – ICEC 2009: 8th International Conference, Paris, France, September 3-5, 2009. Proceedings“. In: ed. Stéphane Natkin a Jérôme Dupire. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Kap. Extending the Strada Framework to Design an AI for ORTS, s. 270–275. ISBN: 978-3-642-04052-8. DOI: 10.1007/978-3-642-04052-8_32. URL: http://dx.doi.org/10.1007/978-3-642-04052-8_32.

[79] *ORTS - A Free Software RTS Game Engine*. 25. květ. 2005. URL: https://skatgame.net/mburo/orts/ (cit. 05. 05. 2016).

[80] *What is Battlecode?* URL: https://www.battlecode.org/contestants/about/ (cit. 25. 04. 2016).

[81] *What is Battlecode?* URL: https://www.battlecode.org/contestants/about/ (cit. 05. 05. 2016).

[82] *Warzone 2100 2.3.1a*. URL: http://download.chip.eu/cz/Warzone-2100_132532.html (cit. 05. 05. 2016).

[83] *Glest - GALLERY*. URL: http://glest.org/en/gallery-screenshots.php (cit. 05. 05. 2016).

[84] *Liquipedia - The StarCraft Encyclopedia*. URL: http://wiki.teamliquid.net/starcraft/Main_Page (cit. 16. 04. 2016).

[85] Kristin Kalning. „Can Blizzard top itself with 'StarCraft II?'“ In: *NBC News* (31. květ. 2007).

[86] David Churchill. *2015 AIIDE StarCraft AI Competition Report*. tournament. University of Alberta, 2015. URL: http://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp/report2015.shtml#results (cit. 16. 04. 2016).

[87] Angel Camilo Palacios Garzón. „Q-Learning in RTS Game's Micro-Management“. Dipl. Universitat de Barcelona, 2015.

[88] StarCraft AI Wiki contributors. *StarCraft AI Benchmarks*. StarCraft AI Wiki. URL: http://www.starcraftai.com/wiki/StarCraft_AI_Benchmarks (cit. 20. 04. 2016).

[89] Florian Richoux. *AIIDE 2015 survey - AIUR*. URL: https://dl.dropboxusercontent.com/u/23817376/Starcraft/AIIDE/2015/survey/Aiur.txt (cit. 20. 04. 2016).

[90] *BWAPI-MonoBridge-Setup*. URL: https://github.com/suegy/bwapi-mono-bridge2/wiki/MonoBridge-Setup (cit. 29. 04. 2016).

[91] Johan Hagelbäck a Stefan J. Johansson. „Using Multi-agent Potential Fields in Real-time Strategy Games“. In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*. AAMAS '08. Estoril, Portugal: International Foundation for Autonomous Agents a Multiagent Systems, 2008, s. 631–638. ISBN: 978-0-9817381-1-6. URL: http://dl.acm.org/citation.cfm?id=1402298.1402312.

[92] Antonín Komenda, Peter Novák a Michal Pechoucek. „Decentralized Multi-agent Plan Repair in Dynamic Environments“. In: *CoRR* abs/1202.2773 (2012). URL: http://arxiv.org/abs/1202.2773.

[93] David Sislak et al. „Multi-Agent Simulation of En-Route Human Air-Traffic Controller“. In: *Proceedings of the Twenty-Fourth Innovative Appications of Artificial Intelligence Conference*. Toronto, Canada: AAAI Press, 2012, s. 2323–2328. ISBN: 978-1-57735-568-7.

[94] Chek Tien Tan a Ho-lun Cheng. „A Combined Tactical and Strategic Hierarchical Learning Framework in Multi-agent Games“. In: *Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games*. Sandbox '08. Los Angeles, California: ACM, 2008, s. 115–122. ISBN: 978-1-60558-173-6. DOI: 10.1145/1401843.1401865. URL: http://doi.acm.org/10.1145/1401843.1401865.

[95] Tristan M. Behrens, Koen V. Hindriks a Jürgen Dix. „Towards an environment interface standard for agent platforms“. In: *Annals of Mathematics and Artificial Intelligence* 61.4 (2010), s. 261–295. ISSN: 1573-7470. DOI: 10.1007/s10472-010-9215-9. URL: http://dx.doi.org/10.1007/s10472-010-9215-9.

[96] Andreas Schmidt Jensen, Christian Kaysø-Rørdam a Jørgen Villadsen. „Interfacing Agents to Real-Time Strategy Games“. In: *Proceedings of the 13th Scandinavian Conference on Artificial Intelligence (SCAI 2015)*. Ed. S. Nowaczyk. IOS Press, 2015, s. 68–77. ISBN: 978-1-61499-588-3.

[97] Johan Hagelbäck. email interview. 2015.

[98] Johan Hagelbäck. „Potential-Field Based navigation in Starcraft“. In: *2012 IEEE Conference on Computational Intelligence and Games (CIG)*. 2012, s. 112–117.

[99] Aurélien Lermant. email interview. 2015.

[100] H. Penny Nii. *Blackboard Systems*. Výzk. zpr. Stanford University, 1985.

[101] *SSCAIT Report 05*. StarCraft Artificial Intelligence Tournament. 2015. URL: https://www.youtube.com/watch?v=hnMG4hHEnw0 (cit. 02. 05. 2016).

[102] Igor Dimitrijevic. email interview. 2015.

[103] Christian Thurau, Gerhard Sagerer a Christian Bauckhage. „Imitation learning at all levels of Game-AI“. In: *International Conference on Computer Games, Artificial Intelligence, Design and Education*. 2004, s. 402–408.

[104] Henrik Alburg et al. „Making and Acting on Predictions in StarCraft: Brood War“. Bachelor of Science Thesis. University of Gothenburg, 2014.

[105] *A customized AI system for StarCraft Brood War based on Ualberta-Bot*. National University of Singapore. URL: https://github.com/Pansophy350/PansophyBot (cit. 19. 04. 2016).

[106] Filip Bober. *TerranUAB repository*. Poznan University of Technology. URL: https://github.com/filipbober/scaiCode (cit. 19. 04. 2016).

[107] Francisco Liwa et al. *NUS-BOT wiki*. National University of Singapore. URL: https://code.google.com/archive/p/nus-bot/ (cit. 19. 04. 2016).

[108] Chris Coxe. email interview. 2015.

[109] *BWMirror API - Documentation*. URL: `http://bwmirror.jurenka.sk/` (cit. 16. 04. 2016).

[110] Ben G. Weber, Michael Mateas a Arnav Jhala. „Building Human-Level AI for Real-Time Strategy Games“. In: *AIIDE Fall Symposium on Advances in Cognitive System*. 2011.

[111] Adam Heinermann. *Support multithreading*. BWAPI repository issue comment. 13. srp. 2013.

[112] *Copy constructors*. URL: `http://www.javapractices.com/topic/TopicAction.do?Id=12` (cit. 30. 04. 2016).

[113] Pankaj Kumar. *Command Design Pattern in Java*. JournalDev. 18. čvc 2013. URL: `http://www.journaldev.com/1624/command-design-pattern-in-java-example-tutorial` (cit. 18. 04. 2016).