TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Aplikácia metód Umelej Inteligencie pri vytváraní agenta hrajúceho strategické hry v reálnom čase

Diplomová práca

2015 **Martin ČERTICKÝ**

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Aplikácia metód Umelej Inteligencie pri vytváraní agenta hrajúceho strategické hry v reálnom čase

DIPLOMOVÁ PRÁCA

Študijný program:	Umelá inteligencia
Študijný odbor:	9.2.8 Umelá inteligencia
Školiace pracovisko:	Katedra kybernetiky a umelej inteligencie
Vedúci práce:	prof. Ing. Peter Sinčák, CSc.
Konzultant:	

Košice 2015 **Martin Čertický**

Abstract

Real-Time Strategy (RTS) games are a genre of video games representing an interesting, well-defined adversarial domain for Artificial Intelligence (AI) research. One of many subproblems that RTS players need to solve is the micromanagement of individual units (simple agents carrying out player's commands) during combat. Numerous multi-agent reactive control mechanisms have already been developed to maximize the combat efficiency of controlled units. Majority of these mechanisms make use of numeric parameters that need to be fine-tuned in order to achieve desired behavior. Due to a large number of these parameters, assigning them manually is inconvenient and training them by machine learning methods usually takes a long time (search space is too large). To reduce the search space and accelerate the training, we propose a simple reactive controller with only eight parameters. We implement it for a classic RTS game StarCraft: Brood War and train the parameters using genetic algorithms. Our experiments demonstrate an impressive combat performance after only a small number of generations.

Keywords: Evolution, Genetic Algorithms, Micromanagement, RTS Games, StarCraft

Abstrakt

Real-time strategické (RTS) hry predstavujú zaujímavú a širokú doménu pre výskum umelej inteligencie. Jeden z mnoha problémov, ktorým musí hráč pri hraní RTS hier čeliť je micromanagement jednotiek (jednoduchí agenti, vykonávajúci príkazy hráča) v boji. V minulosti bolo predstavených niekoľko multi-agentových mechanizmov, starajúcich sa o reaktívnu kontrolu jednotiek s cieľom maximalizovať ich efektivitu v boji. Veľká časť týchto controllerov využíva na nastavovanie svojich vlastností číselné parametre, vyžadujúce si ladenie pre zvýšenie efektivity. Vzhľadom na počty týchto parametrov je ich manuálne nastavovanie zložité, a nastavovať ich pomocou strojového učenia býva často zdĺhavé (kvôli veľkosti prehľadávacieho priestoru). Na zmenšenie prehľadávacieho priestoru a zrýchlenie učenia týchto parametrov predstavujeme jednoduchý controller s iba ôsmymi parametrami. Implementujeme ho v klasickej RTS hre StarCraft: Brood War a parametre trénujeme pomocou evolučných algoritmov. Naše experimenty ukazujú pôsobivé výsledky nášho agenta v boji už po niekoľkých generáciách učenia.

Kľúčové slová: Evolúcia, Evolučné Algoritmy, Micromanagement, RTS hry, StarCraft

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra kybernetiky a umelej inteligencie

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **9.2.8 Umelá inteligencia**

Študijný program: **Umelá inteligencia**

Názov práce:

Aplikácia metód Umelej Inteligencie pri vytváraní agenta hrajúceho strategické hry v reálnom čase

Aplication of methods of Artificial Intelligence for creation of an agent for real-time strategic games

Študent: **Bc. Martin Čertický**

Školiteľ: **prof. Ing. Peter Sinčák, CSc.**

Školiace pracovisko: **Katedra kybernetiky a umelej inteligencie**

Konzultant práce:

Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:

1. Vypracovať teoretický úvod do využitia umelej inteligencie v strategických hrách
2. Poskytnúť Prehľad súčasného stavu danej problematiky s dôrazom na evolučné algoritmy
3. Návrhnúť systému využitia evolučných algoritmov v real-time strategickej hre
4. Realizovať implementáciu a experimentálne overenie systému.
5. Vypracovať dokumentáciu podľa pokynov vedúceho.

Jazyk, v ktorom sa práca vypracuje: slovenský

Termín pre odovzdanie práce: 30.04.2015

Dátum zadania diplomovej práce: 31.10.2014

.....
prof. Ing. Peter Sinčák, CSc.
vedúci garantujúceho pracoviska

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY -11-

.....
prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Týmto prehlasujem, že som celú prácu vypracoval samostatne. Všetky použité zdroje v práci boli uvedené.

Košice, 27. apríla 2015

.....

Podpis

Poďakovanie

Rád by som vyjadril svoju úprimnú vďaku Michalovi Čertickému za jeho odbornú pomoc a neoceniteľné hodiny konzultácií. Moja vďaka patrí aj prof. Ing. Petrovi Sinčákovi, CSc. za prejavenú dôveru a odborné vedenie pri vypracovaní práce.

Predhovor

Ako človeka s dlhoročnými skúsenosťami s hraním počítačových hier ma fascinoval fakt, že rovnako ako pri skutočných hráčoch, aj umelo vytvorení zanechávajú v hre svoj vlastný rukopis. Žiadni dvaja agenti nie sú rovnakí, a teda každý programátor zanecháva v agentovi, ktorého vytvoril kus svojej osobnosti. Je zaujímavé sledovať strety týchto agentov v rôznych turnajoch umelých inteligencií, rovnako ako komunitu zúčastnených programátorov, ktorí sledujú svojho na ceste k víťazstvu.

Problematike agentov hrajúcich počítačové hry som sa venoval už vo svojej bakalárskej práci. Pri vytváraní agenta existuje obrovský počet aspektov, ktoré predstavujú výzvy pri definovaní ich správania. Či už je to snaha vytvoriť agenta kopírujúceho správanie ľudského hráča, alebo snaha vytvoriť neporaziteľného agenta.

Jeden zo spôsobov, ako čeliť týmto výzvam je využitie metód umelej inteligencie. V tejto práci sa snažím vytvoriť agenta, ktorý v konkrétnom aspekte hry značne predčí ľudského hráča. Tento aspekt, takzvaný micromanagement predstavuje schopnosť hráča efektívne ovládať jednotky vo svojej armáde. Keďže takéto ovládanie vyžaduje výnimočnú presnosť a rýchlosť, micromanagement je dlhodobo akýmsi meradlom sily hráčov na profesionálnej úrovni. Na rozdiel od ľudského hráča, umelo vytvorený agent nie je obmedzený na prácu s myšou a klávesnicou.

V práci definujem spôsob, ako pomocou metódy evolučných algoritmov vytvoriť agenta s čo najlepšou úrovňou micromanagementu. Metódu evolučných algoritmov som si zvolil najmä pre intuitívny spôsob jej činnosti, nie nepodobný skutočnej evolúcii, a pre veľkú flexibilitu pri výbere jednotlivých metód.

Spomínaný agent sa takisto zúčastní turnaja umelých inteligencií SSCAIT, kde je možné jeho výsledky porovnať s podobnými agentami z celého sveta.

Obsah

Zoznam obrázkov

Zoznam symbolov a skratiek

AI	Artificial Intelligence
API	Application Programming Interface
BWAPI	Brood War Application Programming Interface
CBR	Case-Based Reasoning
CSP	Constraint Satisfaction Problem
EA	Evolučné Algoritmy
IM	Influence Maps
PF	Potential Fields
RTS	Real-Time Strategy
SSCAIT	Student StarCraft Artificial Intelligence Tournament

Slovník výrazov

Auto-attack je controller používaný natívnou umelou inteligenciou implementovanou v StarCrafte na ovládanie jednotiek v boji.

Controller ako konečný automat (finite state machine) predstavuje agenta riadiaceho akcie našich jednotiek pomocou jednoduchých rozhodnutí počas hry.

Frame je jeden z množiny obrázkov tvoriacich pohyblivý obraz. V StarCrafte sa odohrá približne 23,81 frame-u za reálnu sekundu (pri najvyššej rýchlosti).

Konvergencia v prípade evolučných algoritmov vyjadruje stabilizáciu (sústreďovanie sa do podpriestoru) jedincov po určitom čase, ideálne v okolí riešenia v prehľadávacom priestore.

Mapa v našom prípade vyjadruje hernú plochu. Je to ploocha, na ktorej operujú agenti hrajúci RTS hru StarCraft.

Micromanagement je súbor jednoduchých príkazov, ktoré hráč/agent manuálne zadáva jednotlivým jednotkám počas hry s cieľom zvýšiť ich efektivitu v boji.

Scenario (scenár) je herná mapa, obsahujúca jednotky (prípadne budovy) dvoch hráčov stojacich proti sebe.

1 Formulácia úlohy

Úlohami našej diplomovej práce bolo:

1) Teoretický úvod do využitia umelej inteligencie v real-time strategických hrách.
2) Prehľad súčasného stavu danej problematiky s dôrazom na evolučné algoritmy.
3) Návrh systému využitia evolučných algoritmov v real-time strategickej hre.
4) Implementácia a experimentálne overenie systému.
5) Vypracovanie dokumentácie podľa pokynov vedúceho.

2 Úvod

Hráči sa pri hraní počítačových hier často stretávajú s potrebou adaptovať sa, promptne a efektívne reagovať na protivníkovu nepredvídanú stratégiu. Väčšina najznámejších metód umelej inteligencie už bola aplikovaná na hranie počítačových hier [8].

Rôzne herné žánre predstavujú takisto rôzne typy problémov pre hráčov (agentov). Táto práca je zameraná predovšetkým na Real-Time strategické hry (RTS). V RTS hrách, rovnako ako aj v iných vojnových simuláciách je hlavnou úlohou hráčov umiestňovať a manévrovať jednotky alebo budovy, ktoré majú pod kontrolou za účelom zabezpečenia oblastí na hernej ploche (mape), a/alebo zničenia nepriateľských jednotiek. RTS hry časom ukázali potenciál pre obrovské prehľadávacie priestory, s ktorými sa bežné prehľadávacie metódy umelej inteligencie nedokážu vysporiadať v dostatočne krátkom čase.

Pre účely našej práce si najprv potrebujeme definovať pojem agenta:

Podľa Russela a Norviga [11] je agent entita, ktorá vníma prostredie senzormi a racionálne ho ovplyvňuje efektormi. Ak hovoríme o živom agentovi, jedná sa o reálneho hráča. V našom prípade sa jedná o agenta hrajúceho hru StarCraft. Naším cieľom bude vytvoriť umelého agenta, takzvaného „bota“. V prípade umelého agenta sú rovnako senzory aj efektory funkcie nášho programovacieho prostredia BWMirror[1].

2.1 Kontext

Real-time strategické (ďalej RTS) hry slúžia ako zaujímavá doména pre výskum umelej inteligencie. Hráč je nútený riešiť ekonomické a strategické úlohy ťažbou surovín a výstavbou základní, zvyšovať svoju vojnovú silu výskumom nových technológií a trénovaním nových jednotiek, ktoré neskôr vedie do boja proti nepriateľom. V [36], S. Ontanón a spol. opísali prehľad existujúceho výskumu umelej inteligencie v RTS hre StarCraft.

RTS hry poskytujú vhodnú doménu pre riešenie plánovacích problémov, riešenie problémov s neurčitosťou, získavanie doménových znalostí, a takisto strojového učenia.

[1] https://github.com/vjurenka/BWMirror

2.2 Problém

Výskum v oblasti RTS hier je zväčša rozdelený do troch skupín podľa úrovne abstrakcie: high-level stratégia, middle-level taktika a low-level reaktívna kontrola jednotiek (hráčmi označovaná ako „micromanagement“).

Reaktívna kontrola, ako problém adresovaný v tejto práci sa snaží maximalizovať efektivitu ovládaných jednotiek počas boja tým, že vyberáme najvhodnejšie ciele a pozície pre jednotky na základe terénu a aktivity protivníkových jednotiek.

Pri rozhodovacom procese reaktívnej kontroly sa zväčša používajú dva prístupy: Centralizovaný prístup, kde pristupujeme k celej skupine jednotiek ako celku, alebo decentralizovaný prístup, kde ovládame každú jednotku samostatne. Napríklad, v [37] Churchill a Buro prezentovali alfa-beta prehľadávanie a Zhe a spol. v [23] použili Monte-Carlo plánovanie pre problém micromanagementu.

Centralizované prístupy sú zväčša s rastúcim počtom jednotiek výpočotovo príliš náročné, vzhľadom na obrovský prehľadávací priestor, ktorý StarCraft poskytuje (viac v 3.4).

Decentralizované prístupy sú používané častejšie, najmä kvôli lepšej schopnosti vysporiadať sa s väčšími počtami jednotiek. Prehľadávací priestor je značne zmenšený práve tým, že jednotky sú ovládané nezávisle na ostatných. Spoločnou črtou decentralizovaných prístupov je časté využívanie potenčných polí a influence máp, čím vznikajú pomerne jednoduché algoritmy controllerov.

2.3 Náčrt riešenia

Controller nášho agenta sa stará o ovládanie jednotiek priraďovaním jednotlivých akcií na základe ich aktuálneho stavu. Predstavuje konečný automat, ktorý reaguje výhradne na herné podnety, bez predošlého predikovania zmeny stavov. Controller definuje niekoľko parametrov, ktoré je nutné ladiť pre dosiahnutie požadovaného a efektívneho správania.

V minulosti bolo použitých viacero metód umelej inteligencie na ladenie parametrov takýchto controllerov: Reinforcement Learning, použitie Bayesovských modelov, alebo evolučných algoritmov. V [36] je popísaných niekoľko výsledkov v tejto oblasti.

Avšak, v doterajšej praxi bol problémom práve vysoký počet parametrov controllera potrebných pre optimalizáciu, vytvárajúci príliš veľký prehľadávací priestor

(problém dimenzionality). Napríklad, Liu a spol. [27] použili EA na učenie 14 rôznych parametrov controllera (s jedincom kódovaným ako 60-bitový string), Ponsen [32] použil 20 atribútov v každom jedincovi na trénovanie 20 parametrov, a Sandberg a spol. [30] použili pri trénovaní svojho micromanagement controllera dokonca 21 parametrov.

V snahe vysporiadať sa s problémom dimenzionality, v našej práci používame Controller, pre ktorý trénujeme množinu len 8 parametrov. Tieto parametre trénujeme pomocou EA s populáciou 32 jedincov, ruletovej selekčnej metódy a uniformnou metódou kríženia a mutácie.

Táto práca je pokračovaním našej bakalárskej práce [39].

3 Real-Time Strategické hry a StarCraft

V tejto kapitole si opíšeme základy RTS hier, konkrétne hry StarCraft: Brood War, ktorá je jednou z najpopulárnejších hier tohto žánru. Táto hra bola 31.marca 1998 vytvorená a publikovaná firmou Blizzard Entertainment. Predstavíme si StarCraft aj ako prostredie umelej inteligencie.

3.1 Real-Time Strategické hry

RTS sa zväčša zaoberajú vojnovým konfliktom s jedným, alebo viacerými protivníkmi. Takisto ale obsahujú veľké rozhodovacie priestory v oblasti ekonomického vývoja, explorácie hernej plochy a výskumu v nedeterministickom, čiastočne pozorovateľnom prostredí. RTS hry sú populárne najmä kvôli rýchlemu spádu a nevyhnutnosti neustálych reakcií hráčov.

V RTS hrách účastníci operujú s rôznymi jednotkami a budovami za účelom zničenia jednotiek, ktorými disponuje protivník. Vo väčšine RTS hier je hráč nútený venovať dostatočnú pozornosť aj ekonomickým základom a správe surovín, bez ktorých by nebol schopný produkovať jednotky do svojej armády.

3.2 StarCraft ako počítačová RTS hra

Hra je sci-fi simuláciou vojnového konfliktu troch rôznych rás. Každá z týchto rás má k dispozícii množstvo jednotiek, ktoré nie sú ekvivalentné svojimi atribútmi, čo v konečnom dôsledku len zvyšuje jej komplexnosť a prehľadávací priestor pri výbere vhodných stratégií.

Každá rasa je závislá od troch typov surovín, ktoré sú potrebné pre produkciu jednotiek a budov v priebehu hry. Ekonomické zabezpečenie je teda jednou z hlavných úloh hráča/agenta v hre. Tieto suroviny delíme na minerály, plyn a populačné zdroje. Minerály sú potrebné na stavbu takmer všetkých jednotiek a budov, a sú priamo ťažené pomocou workerov z úložísk rôzne rozmiestnených po mape. Plyn hráč ťaží pre produkciu pokročilých jednotiek a budov. Ťažba neprebieha priamo, hráč potrebuje postaviť istý druh rafinérie na špeciálnom úložisku, z ktorej budú workeri ťažiť. Pre zvýšenie počtu populačných zdrojov je pre každú rasu potrebné buď postaviť špeciálne typy budov, alebo produkovať špeciálne jednotky.

Po získaní potrebných surovín ich hráč využíva na výskum nových technológií, trénovanie nových jednotiek a stavbu budov potrebných pre tieto činnosti. Na Obr. 1 je znázornený technologický strom budov jednej z rás StarCraftu.

Každá rasa v StarCrafte je istým spôsobom výnimočná, no napriek tomu sú sily všetkých 3 rás extrémne vyrovnané:

- Zerg je rasa insektoidov, operujúca lacnými, no relatívne slabými jednotkami. Vyznačuje sa extrémne rýchlou produkciou, nabádajúc hráča zaplaviť protivníka najmä početnou prevahou.
- Protoss, ako prastará rasa humanoidov, preferuje kvalitu oproti kvantite. Jednotky sú vyrábané podstatne pomalšie, ale vyznačujú sa veľkou silou a odolnosťou.
- Terran je rasa ľudí, poskytujúca akúsi strednú cestu medzi rasami Zergov a Protossov.

Pre účely našej práce sa zameriavame hlavne na multi-playerovú časť hry, teda hru viacerých hráčov, a nie hry jedného hráča proti počítaču. Cieľom hry je zničiť všetky protivníkove budovy na mape. Mapa je náhodne vybraná, nachádza sa na nej niekoľko úložisk surovín, ktoré v priebehu hry hráči čerpajú. Topológia máp je rôzna, pričom v našej práci okrem fázy učenia nášho agenta používame oficiálne mapy vytvorené Blizzardom pre multi-playerovú časť hry a niekoľko máp použitých v turnaji umelých inteligencií SSCAIT[2]. Učenie nášho agenta prebieha na špeciálne vytvorených mapách pre zabezpečenie čo najrýchlejšieho priebehu experimentov.

[2] http://www.sscaitournament.com/

Obr. 1 – Technologický strom jednej z rás v hre StarCraft

3.3 StarCraft ako prostredie z pohľadu umelej inteligencie

Pre účely nášho výskumu si definujeme StarCraft ako prostredie na základe vlastností, ktoré v svojej práci definovali Russel a Norvig [11]. Tieto vlastnosti sledujú niekoľko hlavných atribútov prostredia a v konečnom dôsledku ho definujú:

A. Plne pozorovateľné vs. Čiastočne pozorovateľné prostredie

Rozhodujeme, či je prostredie plne, alebo čiastočne pozorovateľné. O plne pozorovateľnom prostredí hovoríme v prípade, že agent má prehľad o kompletnom dianí na hernej ploche v danom čase, teda dokáže detekovať všetky aspekty relevantné pri ďalšom rozhodovaní. Pri plne pozorovateľnom prostredí si agent nepotrebuje udržiavať informácie o svete.

V našom prípade sa teda jedná o ***čiastočne pozorovateľné prostredie***. Jeden z dôvodov je aspekt hry s názvom „fog of war“. Jedná sa o akúsi hmlu, ktorá sa objavuje na miestach, ktoré sme v priebehu hry na hernej ploche preskúmali, ale nenechali sme na nich žiadnu z jednotiek, ani budov, teda o nich nemáme ďalej prehľad. Ďalší dôvod prečo hovoríme o StarCrafte ako o čiastočne pozorovateľnom prostredí je, že nemáme konštatne prehľad o akciách nášho protivníka v prípade, že ho v danom momente nesledujeme pomocou žiadnej jednotky, budovy alebo scouta.

B. Deterministické vs. Stochastické prostredie

O tom, či je prostredie deterministické, alebo nedeterministické (stochastické) rozhodujeme na základe miery „náhodnosti“ aspektov prostredia. Teda ak je následujúci stav prostredia ovplyvnený iba akciami nášho agenta, môžeme hovoriť o deterministickom prostredí. Ak je prostredie len čiastočne pozorovateľné, môže sa nám automaticky javiť ako stochastické, pretože v komplexnom prostredí nedokážeme predpovedať vývoj akcií objektov, o ktorých nemáme prehľad. To však samo o sebe neznamená, že prostredie skutočne obsahuje prvok náhody.

V našom prípade však hovoríme o ***nedeterministickom prostredí***, pretože sa v StarCrafte nachádza niekoľko aspektov, kde má v určitej miere úlohu aj náhoda. V prípade, že jednotka, útočiaca na diaľku vypáli projektil na inú jednotku, ktorá sa oproti nej nachádza na vyvýšenom mieste, alebo v prípade, že jednotke stojí v ceste nejaká prekážka (strom, značka atď.), má len 52,125% šancu, že zasiahne cieľ. V prípade, že

jednotky majú na seba priamy výhľad a nachádzajú sa v rovnakej výške, zásah nastáva s pravdepodobnosťou 99,609375%. Existuje teda pravdepodobnosť, že jednotka strelí „do vzduchu“.

Ďalší aspekt, ktorý je ovplyvnený náhodou v StarCrafte je takzvaný. „cooldown“. Cooldown nám definuje čas, ktorý prejde medzi jednotlivými útokmi jednotky (výstrel, úder, atď.) meraný v takzvaných frame-och za sekundu. V tomto prípade existujú náhodne malé rozdiely. Napríklad v prípade, že cooldown medzi jednotlivými výstrelmi je 50 frame-ov, je možné, že sa 2 frame-y pridajú, alebo 1 odoberie, teda výsledný cooldown je 49-52 frame-ov.

C. Epizodické vs. Sekvenčné prostredie

V epizodickom prostredí sú agentove akcie rozdelené na pravidelné ťahy. Teda v každej epizóde agent najprv vykoná akési vyhodnotenie a potom spraví jednu akciu. Ďalšia podmienka epizodickosti prostredia je, že ďalší priebeh (epizóda) nie je závislý na akciách, ktoré sme vykonali v predošlej epizóde.

Ani jedna z týchto podmienok v našom prípade nie je splnená, teda môžeme pokojne tvrdiť, že StarCraft ako ***prostredie je neepizodické*** (sekvenčné). Koniec-koncov, ako aj všetky real-time strategické hry.

D. Statické vs. Dynamické prostredie

Ak existuje možnosť zmeny prostredia v priebehu hry bez pričinenia nášho agenta, hovoríme o dynamickom prostredí. V statickom prostredí sa agent nemusí obávať nečakaných zmien, teda pri výbere ďalšej akcie nemusí brať do úvahy čas a dianie v prostredí. Na druhej strane v dynamickom prostredí sa agenta neustále pýtame akú akciu chce vykonať. Ak ešte nie je rozhodnutý, uvažujeme, že sa rozhodol nevykonať nič.

V hre StarCraft sa jedná jednoznačne o ***dynamické prostredie***. Hráč totiž nie je jediným aspektom zasahujúcim do hry a tým pádom ovplyvňujúcim prostredie v čase. V každej hre totiž náš agent čelí jednému, alebo viacerým agentom. Okrem toho sa v hre nachádzajú, aj neutrálne jednotky, ktoré neovláda ani jeden z hrajúcich agentov.

E. Diskrétne vs. Spojité prostredie

Prostredie je diskrétne, ak existuje konečný počet rozdielnych, neprekrývajúcich sa stavov, ktoré môže nadobudnúť, a vždy sa jednoznačne nachádzame v práve jednom z týchto stavov.

V našom prípade hovoríme o ***spojitom prostredí***, pretože náš aktuálny stav je určený aj spojitými premennými. Napríklad, životnosť jednotiek a budov je daná reálnymi číslami.

F. Single-agent vs. Multi-agent prostredie

Ako naznačuje nadpis, Single-agent prostredie je také, v ktorom operuje iba jeden agent.

V prostredí StarCraftu môžeme s istotou tvrdiť, že sa jedná o ***Multi-agent*** prostredie, pretože hrajú proti sebe vždy dvaja, alebo viacerí hráči.

3.4 StarCraft ako stavový priestor

Pri uvažovaní StarCraftu ako prehľadávacieho priestoru optimalizačného problému je vhodné opísať si jeho rozmery. Napriek tomu, že kvôli jeho komplexnosti nie je možné určiť tieto rozmery presne, existuje niekoľko faktov, ktoré nám o nich napovedajú, popísaných aj v [36].

Typická mapa v StarCrafte je štvorcová sieť, kde výška a šírka mapy sú merané vo štvorcoch 32 × 32 pixelov, takzvaných build tile-och. Veľkosť „kroku“ jednotky v StarCrafte je definovaná takzvanými walk tile-ami (8 × 8 pixelov). Rozmery typických máp sa pohybujú od 64 × 64 do 256 × 256 build tile-ov. Každý hráč môže v hre ovládať až 200 jednotiek (a neobmedzený počet budov). Každá rasa navyše definuje 30 až 35 vlastných typov jednotiek a budov, každá z nich s unikátnymi vlastnosťami.

Všetky tieto fakty napovedajú o tom, že StarCraft tvorí značnú výzvu, kde ľudský hráči sú stále na podstatne vyššej úrovni ako počítačoví agenti. Napríklad v hernom rebríčku iCCup[2], kde sú hráči a agenti hodnotení podľa celkového počtu bodov (E je najnižšie hodnotenie, A^+ a *Olympic* sú druhé najvyššie a najvyššie hodnotenie), najlepší agenti v rebríčku dosahujú hodnotenia medzi *D* a D^+, pričom priemerní hráči amatérskej úrovňe dosahujú hodnotenia medzi C^+ a *B*. Pre porovnanie, profesionálni hráči StarCraftu sú zväčša hodnotení známkami A^- a A^+.

Z teoretického hľadiska, prehľadávací priestor pri riešení optimalizačného problému v StarCrafte dosahuje obrovské rozmery. Napríklad, uvažujme rozmery mapy 128 × 128 (najčastejšie používaný rozmer pri hre dvoch hráčov). V každom momente hry sa na mape pri hre dvoch hráčov môže nachádzať až 400 jednotiek, z ktorých každá má svoj aktuálny stav popísaný viacerými údajmi – zostávajúce životy, energia, poloha, akcia, ktorú vykonáva atď. Takáto mapa vytvára obrovský počet možných stavov (omnoho väčší ako pri bežných hrách ako Šach alebo Go).

Napríklad, uvažujúc *iba* polohy všetkých jednotiek na mape (pre 128 × 128 možností pre každú jednotku a 400 existujúcich jednotiek), dostávame $16384^{400} \approx 10^{1685}$ stavov. Pri pridaní ďalších faktorov ovplyvňujúcich priebeh hry dostávame mnohonásobne väčšie čísla.

4 Súvisiaca práca

Táto kapitola popisuje súčasný stav problematiky umelej inteligencie v počítačových hrách s dôrazom na metódu evolučných algoritmov, ktorú v našej práci využívame.

Metóda evolučných algoritmov (EA) má obrovský počet aplikácií v hernom priemysle. Či už sa jedná o vývoj hier samotných [4], [5], [22], alebo o snahu optimalizovať agentov hrajúcich jednotlivé hry.

Pri skúmaní aplikácií EA na pôde hernej umelej inteligencie sme našli množstvo publikácií venujúcich sa tejto problematike. Rozdelíme si teda rôzne typy hier na základe určitej taxonómie, podobne ako uviedli Aha, Molineaux a Ponsen v [1]. Hry a im prislúchajúci výskum v oblasti EA sme rozdelili z tradičného pohľadu hráčov na 6 kategórií:

4.1 Klasické doskové hry

Klasické doskové hry ako šach a dáma predstavujú diskrétne, deterministické prostredie s dvoma agentami (hráčmi), ktorí doň zasahujú v epizodických ťahoch. Klasickým doskovým hrám sa venovalo niekoľko výskumných článkov, počnúc výskumom Arthura Samuelsa o učení sa pri hraní dámy [9]. De Jong a Schultz sa zaoberali výskumom GINA, ktorý si zapamätával čiastočné herné stromy pri hraní hry Othello [3]. Veľmi populárnou témou v tejto oblasti výskumu bol šach. Napríklad, Kerner opísal metódu učenia sa vyhodnocovať abstraktné vzory [7].

Z pohľadu EA sa doskovými hrami zaoberal napr. Cunningham [17], ktorý používal EA na generovanie víťaznej stratégie pre doskovú hru Othello. Shah a kol. predstavili EA pri hraní klasickej hry Go-Moku, kde sa okrem zvyšovania úspešnosti hráča snažili minimalizovať čas potrebný na jednotlivé herné rozhodnutia [16].

4.2 Adventúry

Pri dobrodružných hrách (adventúrach) sa umelá inteligencia používala najmä pri automatickom generovaní obsahu. Fairclough a Cunningham opísali metódu OPIATE [5], ktorá používa plánovač založený na CBR (Case-Based Reasoning) a CSP (Constraint Satisfaction Problem). Ten generuje vývoj udalostí, pričom zabezpečuje také plynulé správanie sa postáv, ktoré je v súlade s koherentným príbehom hry. Taktiež Daz-Agugo a kol. popisujú svoj znalostný prístup [4], ktorým extrahujú obmedzenia s užívateľom

interaktívne zadanej špecifikácie, používajú ich pri výbere spomedzi preddefinovaných prípadov a ich adaptácii, a nakoniec ním vytvárajú čitateľnú príbehovú zápletku za použitia techník z oblasti generovania prirodzeného jazyka.

Použitie EA v adventúrach a RPG hrách (Role-Playing Games) popísali napr. Linden a kol., hovoriac o generovaní herných úrovní pomocou EA [15]. Liapis a spol. hodnotili jednotlivé aspekty herných úrovní pomocou EA, konkrétne ovládateľnosť v úrovniach, exploráciu a vyrovnanosť úrovní [14].

4.3 Real-time individuálne hry

Real-time individuálne hry, ako napríklad bojové simulácie z prvého pohľadu (first-person shooters) nám z pohľadu EA poskytujú mnoho príkladov využitia. Učenie agentov hrajúcich takéto hry s cieľom optimalizácie ich výkonu predstavili Cole [20], Alcázar a kol. [21]. Ďalším využitím EA v real-time individuálnych hrách bola ich aplikácia na vytváranie zaujímavých úrovní pre hráčov, predstavené Cardamone-om a kol. [22].

4.4 Real-time manažérske hry

Single-playerové manažérske hry sa zväčša venujú problematike plánovania akcií agenta v nedeterministickom prostredí. Systém MAYOR [6] zaznamenáva plánovacie chyby v hre SimCity, kde má náš agent na starosti správu mesta. MAYOR monitoruje očakávané plány a vytvára/učí sa kauzálny model prostredia, ktorý mu umožní predísť ďalším chybám v plánovaní, pričom jeho cieľom je zvýšiť percento úšpešne vykonaných plánov.

4.5 Diskrétne ťahové stratégie

Komplexné ťahové stratégie ako Freeciv (open-sourcový klon Civilizácie), vyžadujú od hráča/agenta aby riešil niekoľko osobitných pod-úloh.

Príkladom použitia EA pri riešení takýchto úloh je výstavba mesta, popísaná Watsonom a kol. [19], alebo výber vhodnej lokácie pre výstavbu miest pomocou EA, použitá Arnoldom a kol. [18].

4.6 Real-time stratégie

RTS hry pokrývajú pravdepodobne najväčší počet publikácií zaoberajúcich sa využitím metód umelej inteligencie v počítačových hrách. Práve hra StarCraft slúžila

vďaka svojej popularite často ako prostriedok implementácie rôznych metód umelej inteligencie do agentov, hrajúcich tieto hry.

Aha a kol. vytvorili CBR systém CaT [1] na výber ofenzívnych, alebo defenzívnych stretégií, Weber a Mateas [10] použili CBR na výber build-orderu (poradia produkcie jednotlivých budov) v hre Wargus (open-source klon hry WarCraft 2). Cadena a Garrido prezentovali prístup kombinujúci CBR a Fuzzy logiku pri riešení strategického a taktického manažmentu v StarCraft-e [2].

Problematiku micromanagementu adresovali Parra a Garrido aplikáciou rozhodovacieho modelu vytvoreným pomocou Bayesovských sietí [24], alebo Zhe a kol. použitím Monte-Carlo plánovania [23].

Z pohľadu EA je zaujímavý výskum Tavares-a a spol. [26], v ktorom simulovali „swarm“ inteligenciu, ktorej parametre boli ladené pomocou EA. Chih-Sheng Lin sa v [25] zaoberal vytváraním efektívnych formácií armády pre taktické účely v boji pomocou EA.

Vzhľadom na to, že v našej práci používame EA pri adresovaní problému micromanagementu, spomenieme si existujúci výskum v tejto oblasti: Ponsen sa v [32] v hre Wargus venoval autonómnemu agentovi schopného pomocou offline učenia vytvárať vlastné taktiky a stratégie.

Asi najčastejší spôsob adresovania problému micromanagementu pomocou EA je využitie takzvaných potenčných polí (PF), resp. influence máp (IM). Či už je to substitúciou hernej mapy IM [27], vyhodnocovaním okolia jednotiek pomocou IM [28], vyhľadávaním protivníkovej armády pomocou PF [30], či v niektorých prípadoch použitím viackriteriálnych EA v spojení s PF [29].

Ďalším príkladom využitia EA v RTS hrách je algoritmus, slúžiaci na evolúciu topológií neurónových sietí slúžiacich na riadenie micromanagementu agenta, prezentovaný v [31].

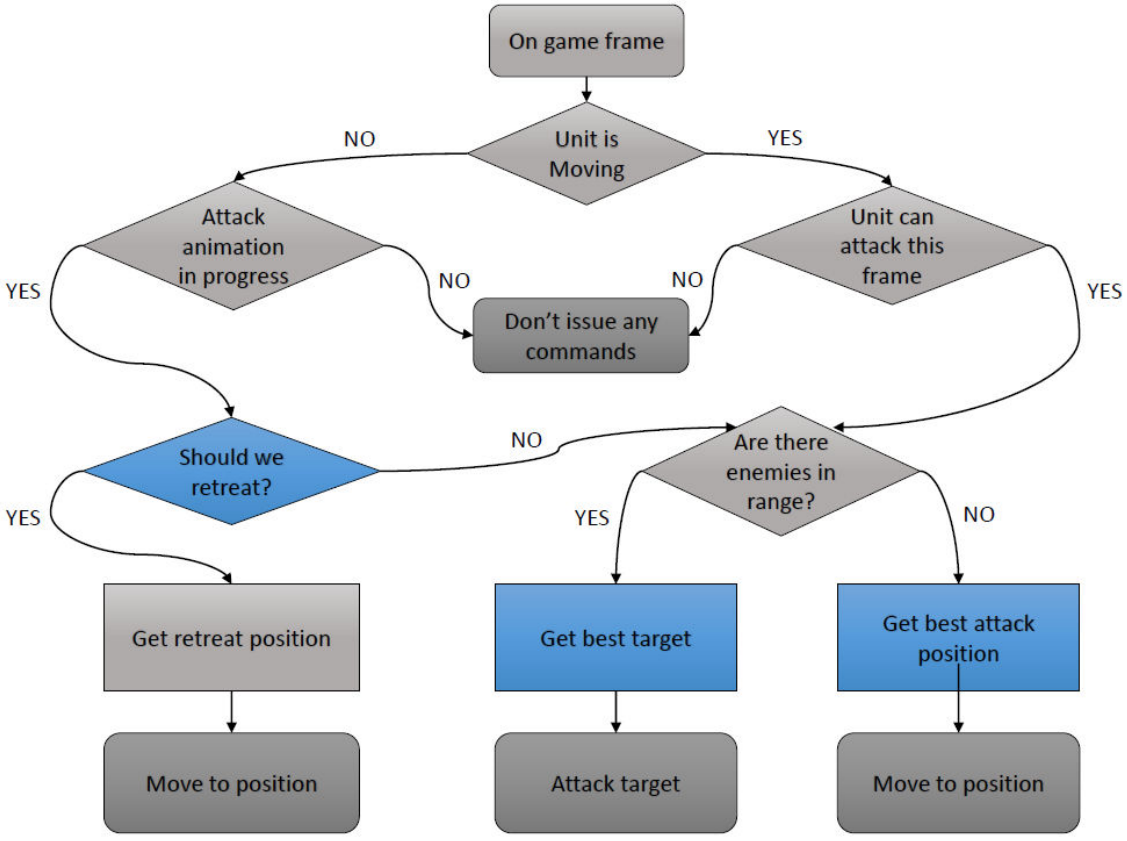
5 Micromanagement Controller ako konečný automat

Controller volaný na každom frame je svojim správaním zhodný s konečným automatom, popisujúcim aktuálny stav každej jednotky. Každej jednotke je Controllerom priradený jeden z dvoch príkazov, *attack(Unit enemy)* alebo *move(Position pos)*.

V následujúcich kapitolách si popíšeme celý rozhodovací proces nášho Controllera, rovnako ako všetky dôležité metódy, ovplyvňujúce správanie jednotiek.

5.1 Štruktúra Controllera

Na Obr. 2 je zobrazený celý rozhodovací proces nášho Controllera, ktorý spravuje micromanagement jednotiek ovládaných našim agentom v boji. Jednotlivé metódy, ktoré Controller používa sú popísané v 5.2.



Obr. 2 – Štruktúra Micromanagement Controllera

5.2 Činnosť Controllera

Prvá kontrola stavu jednotky zisťuje, či jednotka na aktuálnom frame má priradený príkaz *move*.

Ak áno, Controller sa pýta, či môže jednotka v danom frame útočiť (v StarCrafte je pre každú jednotku definovaný počet frame-ov, ktoré musia uplynúť prv, než môže jednotka znova zaútočiť – takzvaný weapon cooldown). V prípade, že útočiť nemôže, nepriraďujeme jednotke žiadny príkaz, jej stav sa v aktuálnom frame nezmení. Ak útočiť môže, a zároveň sa v jej dostrele nachádzajú jednotky nepriateľa, Controller vyberá najvhodnejší cieľ pre útok (5.2.1) a priraďuje jednotke príkaz *attack*. V prípade, že v dostrele jednotky neexistuje ani jeden cieľ (jednotka, alebo budova nepriateľa), Controller vyberá najvhodnejšiu pozíciu na útočenie (5.2.2) z okolia jednotky a priraďuje jej príkaz *move*.

Ak jednotka na danom frame má priradený príkaz *attack*, a zároveň neprebieha animácia útoku jednotky (animáciu útoku jednotiek pre plynulosť ich správania v boji neprerušujeme), jednotka má dve možnosti: pokračovať v útočení na aktuálny cieľ, alebo sa presunúť na výhodnejšiu pozíciu (volaním metódy *move*). V prípade, že prebieha animácia útoku jednotky Controller sa pomocou metódy *shouldWeMove* (*5.2.3*) rozhoduje, či je pre jednotku výhodnejšie ustúpiť, pokračovať v útoku, alebo sa presunúť na inú pozíciu.

Rozhodnutie, či ustúpiť, alebo nie (volaním metódy *shouldWeMove*) je učené našim EA, avšak samotná pozícia na ktorú jednotku presúvame je pevne zakódovaná (5.2.3).

5.2.1 Výber cieľa

Táto sekcia popisuje proces výberu najvhodnejšieho cieľa pre jednotku. Cieľ je vyberaný spomedzi všetkých protivníkových jednotiek v dostrele ovládanej jednotky. Zavádzame hodnotiacu funkciu $Score_e$, ktorá vracia najlepšie hodnotený objekt typu *Unit*, neskôr použitý ako argument pre metódu a*ttack*. Hodnotiaca funkcia $Score_e$ je potom definovaná ako:

$$\boldsymbol{Score_e = (D_e.p_1) - (HP_e.p_2) + (L_e.p_3)} \quad (1)$$

kde D_e je poškodenie (enemy damage), ktoré je jednotka protivníka schopná vyvinúť jedným útokom (pred použitím vynásobená tromi, pre zdôraznenie jej dôležitosti

bez nutnosti normalizácie) a HP_e je súčet zostávajúcich životov (enemy hit points) a štítov jednotky protivníka. Premenná L_e (enemy lethal) sa rovná 100 v prípade, že súčet zostávajúcich životov a štítov jednotky protivníka je menší ako poškodenie, ktoré je naša jednotka schopná spôsobiť. V opačnom prípade, hodnota premennej L_e sa rovná 0, teda:

$$L_e \begin{cases} \mathbf{100} & \boldsymbol{if} \quad \boldsymbol{D_f \geq HP_e} \\ \mathbf{0} & \boldsymbol{if} \quad \boldsymbol{D_f < HP_e} \end{cases}$$

(2)

Každá premenná v hodnotiacej funkcii je navyše prenásobená vlastným parametrom (p_1, p_2, p_3), ktorých ladenie pomocou EA je hlavným cieľom našej práce.

5.2.2 Výber pozície útoku

Ďalšia funkcia, ktorej parametre trénujeme je funkcia hodnotiaca pozície v okolí ovládanej jednotky. Funkcia vracia objekt typu *Position*, obdobne ako pri výbere cieľa neskôr poskytnutý ako argument pre metódu *move*. Vrátená hodnota walk tile-u *t* predstavuje najvýhodnejšiu pozíciu na presun našej jednotky. Hodnotíme walk tile-y v okolí našej jednotky (popísané v kapitole 3.4) hodnotiacou funkciou v tvare:

$$\boldsymbol{Score_t = D_{from} . p_4 - D_{to} . p_5 - L_f . p_6 - C . p_7}$$

(3)

kde premenná D_{from} predstavuje celkové poškodenie, ktoré Controllerom ovládaná jednotka z danej pozície môže spôsobiť, a premenná D_{to} súčet poškodení protivníkových jednotiek s dosahom na túto pozíciu. Premenná L_f v tejto funkcii je podobná ako pri funkcii na výber cieľa *scoreTarget* – v prípade, že súčet poškodení jednotiek s dosahom na danú pozíciu je väčší ako zostávajúce životy ovládanej jednotky, nadobúda hodnotu 100, inak nadobúda hodnotu 0. Teda ak počet jednotiek protivníka s dosahom na hodnotenú pozíciu je *n*:

$$L_f \begin{cases} \mathbf{100} & \boldsymbol{if} \quad \boldsymbol{\sum_{i=1}^{n} D_{ei} \geq HP_f} \\ \mathbf{0} & \boldsymbol{if} \quad \boldsymbol{\sum_{i=1}^{n} D_{ei} < HP_f} \end{cases}$$

(4)

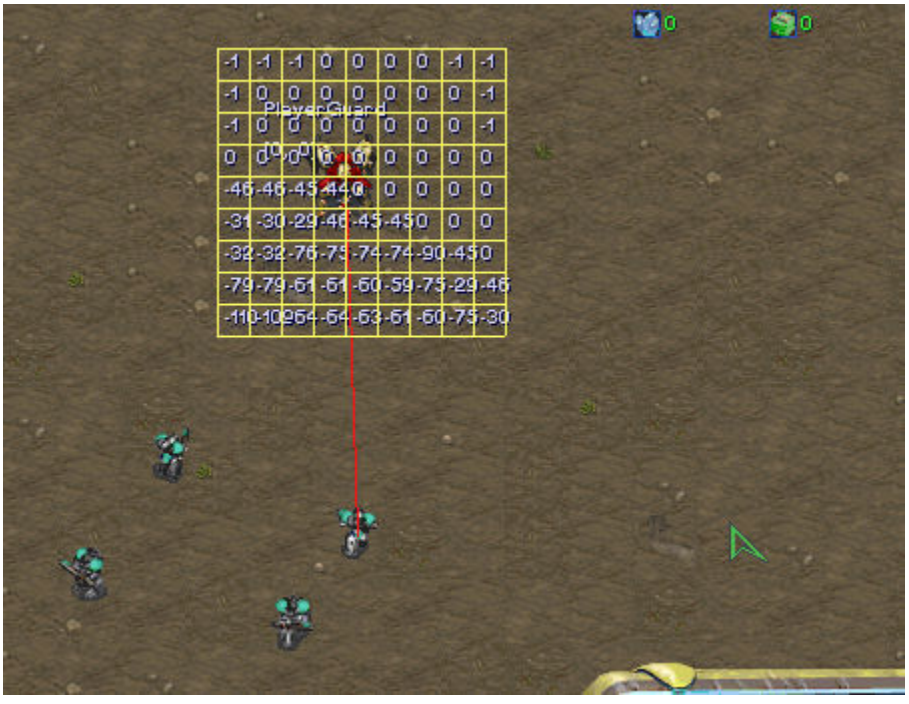
kde HP_f je súčet zostávajúcich životov a štítov našej jednotky (friendly hit points).

Premenná *C* (crossing path of unit) bola zavedená pre zabránenie kríženiu ciest jednotlivých jednotiek. V prípade, že cesta ovládanej jednotky na hodnotenú pozíciu kríži cestu inej presúvajúcej sa jednotke (alebo prechádza cez inú jednotku), táto premenná nadobúda hodnotu 50, v opačnom prípade zostáva nulová. Jedná sa teda o penalizáciu pozícií, ktoré by v prípade výberu spôsobovali nežiadúce kolízie jednotiek.

Opäť, všetky premenné v hodnotiacej funkcii sú prenásobené učenými parametrami (p_4, p_5, p_6, p_7).

Počas práce na systéme sa ukázalo, že veľkosť okolia, ktoré Controller pre každú jednotku prehľadáva s cieľom vybrať vhodnú pozíciu výrazne ovplyvňuje správanie ovládaných jednotiek. Avšak postupné zvyšovanie veľkosti tohto okolia ukázalo, že rovnako ovplyvňuje aj výpočtovú náročnosť nášho systému. Preto sme ako riešenie zvolili dynamickú veľkosť prehľadávaného okolia:

Náš agent pravidelne kontroluje dĺžku každého herného frame-u. V prípade, že táto dĺžka neprekročila zadanú hranicu 55 milisekúnd, agentovi je dovolené ďalej zväčšovať okolie, ktoré Controller prehľadáva. Agent zväčšuje okolie dovtedy, kým sa nedostane na zadanú hranicu, resp. kým veľkosť prehľadávaného okolia nedosiahne maximálnu hranicu (veľkosť mapy).



Obr. 3 – Screenshot agenta prehľadávajúceho pozície v okolí jednotky

5.2.3 Ústup

Posledná metóda, ktorá používa parametre trénované našim EA rozhoduje, či je pre ovládanú jednotku potrebné ustúpiť na bezpečnejšiu pozíciu. Na základe jednoduchej funkcie vracia hodnotu *true*, alebo *false*:

$$shouldWeMove = \begin{cases} true & if \quad \sum_{i=1}^{n} D_i \geq HP_f . p_0 \\ false & if \quad \sum_{i=1}^{n} D_i < HP_f . p_0 \end{cases} \quad (5)$$

kde D_i je protivníkova jednotka aktuálne útočiaca na ovládanú jednotku a HP_f je súčet zostávajúcich životov a štítov ovládanej jednotky a n je počet jednotiek protivníka aktuálne útočiacich na ovládanú jednotku.

Rovnako ako v predchádzajúcich prípadoch, aj v tomto prípade násobíme relevantnú premennú vo funkcii (HP_f) učeným parametrom (p_0).

Kým rozhodnutie Controllera kedy jednotke priradiť príkaz na ústup je ovplyvnené učeným parametrom, samotný výber pozície, na ktorú sa má jednotka pri ústupe presunúť je jednoznačne zakódovaný. Narozdiel od 5.2.2, Controller vypočíta ústupový vektor nasmerovaný smerom od najväčšej hrozby zo strany protivníka.

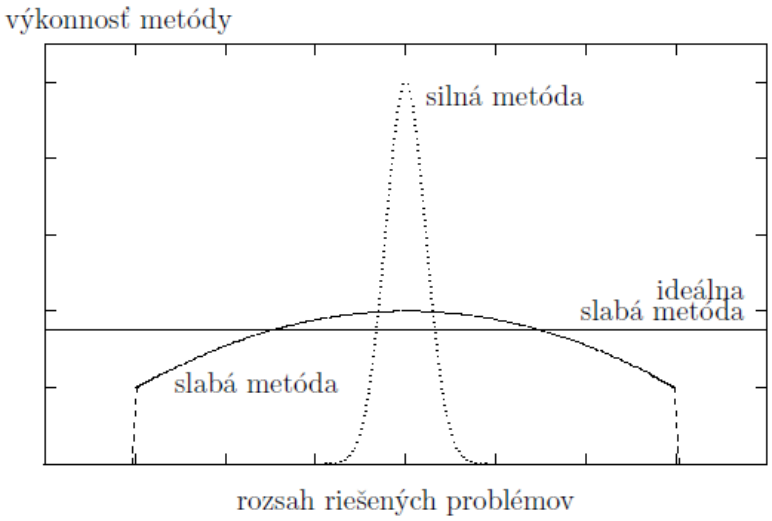
6 Evolučné algoritmy

Ako sme už popísali v 3.4, StarCraft nám ponúka prehľadávací priestor obrovských rozmerov, pričom je obtiažne nájsť vhodné doplňujúce heuristické informácie, ktoré by nám pomohli v jeho prehľadávaní. Takisto je zložité určiť relevantné súvislosti medzi jednotlivými stavmi hry užitočné pre nájdenie požadovaných riešení.

Pre podobné prípady je vhodné použiť slabú prehľadávaciu metódu:

Slabé metódy sú charakterizované malým objemom znalostí, ktoré vyžadujú o riešených úlohách. Pretože slabé metódy kladú iba málo podmienok, ktoré musia byť splnené riešenými úlohami, je možné tieto metódy použiť pre riešenie širokej škály problémov.

Ideálna slabá metóda by dokázala vyriešiť všetky úlohy, pričom každú z nich by riešila s rovnakou výkonnosťou (presnosťou, rýchlosťou, atď.). Keďže reálna slabá metóda (napríklad EA) predsa len kladie nejaké podmienky na riešené úlohy, nedokáže ich už vyriešiť všetky, ale iba nejakú (aj ked značne veľkú) podmnožinu týchto úloh. Navyše, nie je rovnako vhodná pre riešenie každej úlohy, takže jej výkonnosť pri riešení rôznych úloh kolíše [34].

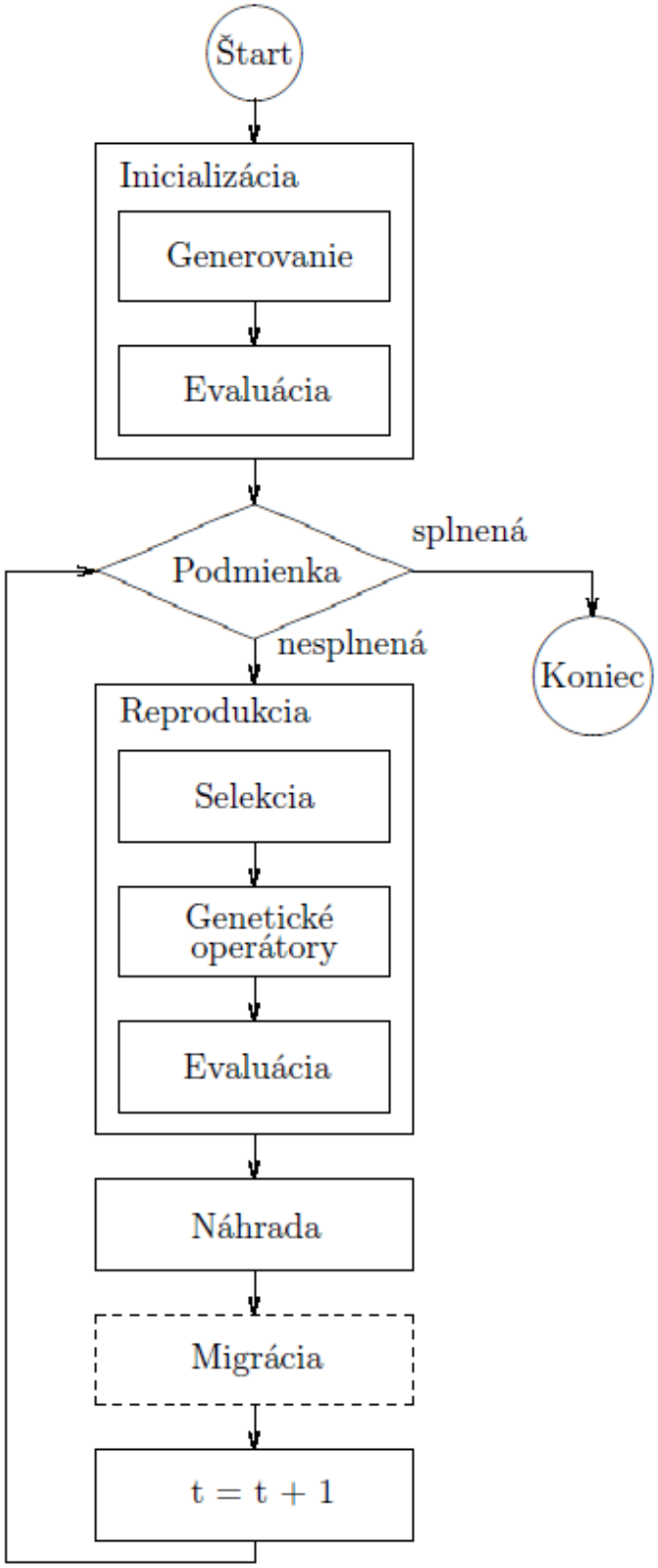


Obr. 4 – Slabé vs. silné prehľadávacie metódy podľa [34]

Evolučné algoritmy sú vďaka svojím nízkym nárokom na počet informácií o riešenej úlohe považované za typického reprezentanta slabých prehľadávacích metód.

6.1 Štruktúra algoritmu

Štruktúra EA použitého v našej práci je takmer zhodná so štruktúrou algoritmu na Obr. 5.



Obr. 5 – Všeobecná štruktúra evolučného algoritmu podľa [34]

Prvotná populácia je vytvorená náhodne (hodnoty parametrov nastavené na náhodné čísla z rozsahu <0, 1>), následne sú všetky jedince ohodnotené.

Podmienka ukončenia nášho algoritmu je určená počtom generácií, ktoré boli našim algoritmom preskúmané.

Blok reprodukcie slúži na vytváranie nových generácii jedincov. Jednotlivé časti tohoto bloku sú popísané v nasledujúcich sekciách.

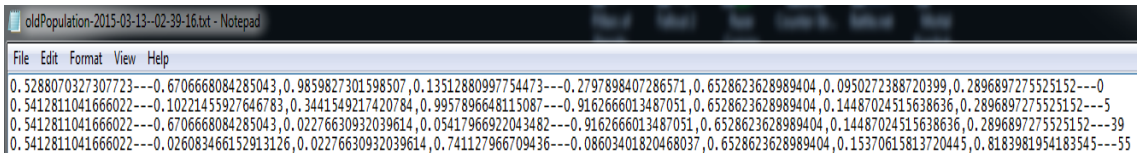
6.1.1 Kódovanie / Genotyp

EA skúma naraz viacero kandidátov (bodov v prehľadávacom priestore) riešenia. Každý z týchto kandidátov je označovaný ako jedinec, a všetky jedince skúmané algoritmom v určitom čase vytvárajú populácie. Pre zabezpečenie efektívnej manipulácie s jedincami je nutné zvoliť si vhodný spôsob reprezentácie. Použitá reprezentácia definuje priestor prehľadávania a zároveň aj stanovuje:

- veľkosť priestoru prehľadávania,
- jeho vzťah k priestoru kandidátov riešení,
- požiadavky na genetické operátory.

Dôležitými vlastnosťami reprezentácie sú takisto dĺžka a usporiadanie jednotlivých prvkov (atribútov) jedinca. Oboje v zásade môžu byť dvojakého typu [34]:

- pevné,
- variabilné.



Obr. 6 – Príklad niekoľkých jedincov použitých v našom EA

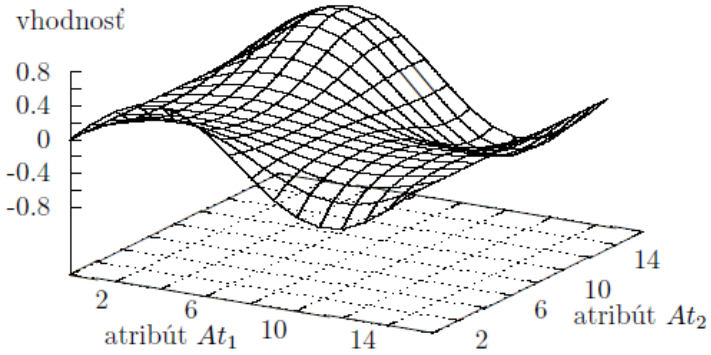
Vzhľadom na charakter nášho problému sme zvolili dĺžku aj usporiadanie prvkov jedinca pevné:

Jedinec (Obr. 6) je reprezentovaný vektorom 8 reálnych čísel z intervalu <0, 1>, pričom každé z týchto čísel reprezentuje jeden z parametrov použitých v metódach

Controllera popísaných v 5.2. Je teda prirodzené, že voľba variabilnej dĺžky, či usporiadania jednotlivých prvkov jedinca neprichádza do úvahy.

6.1.2 Evaluácia

Každý z jedincov má priradenú svoju individuálnu vhodnosť (fitness), charakterizujúcu jeho kvalitu z pohľadu cieľa EA (teda nájdenia riešenia problému). Vnútorná štruktúra a vhodnosť jedinca definujú jeho polohu v prehľadávacom priestore. Jednotlivé jedince v prehľadávacom priestore (kde každý z nich reprezentuje jedného kandidáta riešenia) vytvárajú hyperplochu označovanú ako plocha vhodnosti (Obr. 7). Hľadanie riešenia je teda nespojitý pohyb po tejto ploche, s cieľom nájsť jej globálny extrém, predstavujúci optimálne riešenie [34].



Obr. 7 – Príklad plochy vhodnosti podľa [34]

V našej práci sme sa stretli s pomerne veľkou mierou náhodnosti spôsobenou charakterom hry StarCraft, čo môže v prípade ignorácie výrazne negatívne ovplyvniť schopnosti učenia EA. Táto náhodnosť je spôsobená do istej miery nedeterministickým charakterom hry (3.3), a najmä často sa líšiacim správaním natívnej umelej inteligencie agenta StarCraftu, použitého ako protivníka pri našom učení.

Počas fázy učenia nášho algoritmu hra končí vtedy, keď jeden z hráčov príde o všetky svoje jednotky, alebo hra dosiahne preddefinovaný časový limit. Na konci každej hry je vypočítaný súčet zostávajúcich životov a štítov všetkých našich jednotiek, a jednotiek protivníka (pričom životy a štíty jednotiek protivníka majú zápornú hodnotu):

$$Score = \sum_{i=1}^{n} HP_{fi} - \sum_{i=1}^{n} HP_{ei}$$

Celková vhodnosť každého jedinca je určená ako priemerné skóre jedinca po odohratí piatich hier, s cieľom minimalizovať vplyv vyššie spomínanej náhodnosti.

6.1.3 Selekčný mechanizmus

Selekcia slúži na vytvorenie skupiny rodičov z aktuálnej populácie jedincov. Počet rodičov, ktorých je potrebné vybrať z aktuálnej populácie je daný počtom potomkov, ktorí majú byť vygenerovaní, a zvolenými genetickými operátormi.

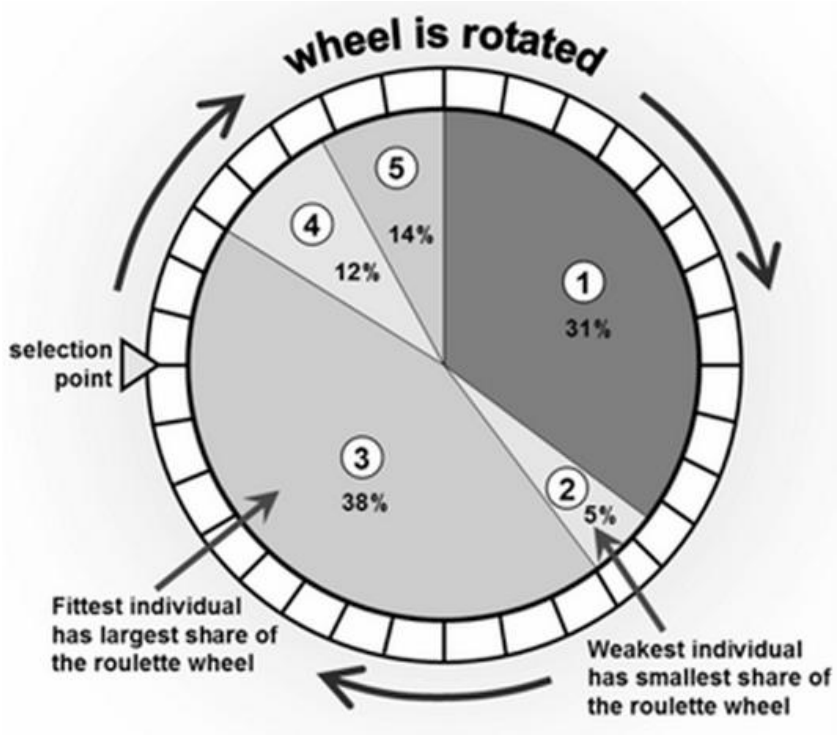
Výber jedincov sa realizuje na základe šance na reprodukciu jednotlivých jedincov. Tá je určená podľa vhodnosti jednotlivých jedincov a zvolenej selekčnej metódy. Proces vzorkovania jedincov do úlohy rodičov typicky delíme na:

- vzorkovanie s explicitnými pravdepodobnosťami,
- vzorkovanie s implicitnými pravdepodobnosťami.

V prvom prípade proces selekcie pozostáva z dvoch fáz: určenia cieľovej pravdepodobnosti selekcie jedincov a výberu jedincov do úlohy rodičov. Pre každú z týchto fáz existuje niekoľko metód, pričom ich možno vzájomne kombinovať. Selekcia s explicitnými pravdepodobnosťami sa v literatúre často prezentuje ako jedna metóda - ruletový výber:

V minulosti boli často používanými metódy založené na analógii s ruletou. Pri týchto metódach sa vychádzalo z jednoduchého princípu:

Na rulete sa vyznačilo toľko úsekov, koľko bolo jedincov v populácii. Veľkosť každého úseku korešpondovala s očakávaným počtom rodičov, prislúchajúcim danému jedincovi (a teda celková dĺžka obvodu rulety sa zhodovala s požadovaným počtom rodičov). Ruleta sa náhodne rozkrútila a po jej zastavení bol za rodiča vybraný ten jedinec, ktorému prislúchal ten dielik rulety, na ktorý ukazoval jazýček. Tento proces bol opakovaný toľkokrát, koľko bolo neobsadených miest rodičov.



Obr. 8 – Ruletová selekčná metóda

Metódy vzorkovania s implicitnými pravdepodobnosťami majú spoločné to, že v popise ich činnosti sa nevyskytuje slovo „pravdepodobnosť“, avšak je možné ju určiť na základe toho, ako sú jednotlivé jedince selektované. Do tejto triedy selekčných metód patria tri skupiny: turnaje, orezanie a náhodný výber [34].

V našej práci sme ako selekčný mechanizmus zvolili jednoduchý ruletový výber (Obr. 8) jedincov do úlohy rodičov. Po priradení vhodnosti všetkým jedincom bola ich vhodnosť premapovaná škálovaním tak, aby sme predišli záporným hodnotám vhodnosti. Teda premapovaná vhodnosť každého jedinca sa rovnala súčtu jeho pôvodnej vhodnosti a absolútnej hodnoty vhodnosti najhoršieho jedinca v populácii.

Pre zlepšenie schopnosti nášho EA konvergovať k riešeniu sme obohatili selekčný mechanizmus o elitistické správanie: najlepší jedinec každej populácie sa so 100%-nou pravdepodobnosťou stane rodičom (a teda aj súčasťou ďalšej generácie, ako vyplýva z 6.1.4).

6.1.4 Náhrada

Cieľom náhrady je vyformovať nasledujúcu generáciu z jedincov, ktoré sú aktuálne k dispozícii. Jedná sa o dve skupiny jedincov – jedince pôvodnej populácie a nové jedince (potomkovia) vytvorené z rodičov pôvodnej populácie. Existuje niekoľko

spôsobov náhrady, pričom najjednoduchší je takzvaná náhrada jednogeneračného charakteru, kde sa nová populácia skladá výhradne z potomkov vytvorených z jedincov pôvodnej populácie. Pri zložitejších realizáciách náhrady vytvárame novú populáciu nielen z potomkov, ale aj z členov aktuálnej populácie. V prípade, že počet miest v následujúcej generácii je menší ako počet jedincov, ktoré sú k dispozícii, dochádza k súťaži medzi jedincami. Organizácia súťaže môže byť dvojaká:

- oddelená súťaž,
- spoločná súťaž.

Pri oddelenej súťaži je presne dané koľko miest v novej generácii bude obsadených potomkami a koľko miest pôvodnými jedincami.

Pri spoločnej súťaži sa vyberajú jedince do novej generácie spoločne zo skupiny potomkov aj pôvodných jedincov. Nie je vopred stanovený počet miest, ktoré obsadia jedince z jednotlivých skupín [34].

V našej práci sme zvolili variant oddelenej súťaže, kde nová populácia je tvorená výhradne z potomkov a ich rodičov. Teda jedince, ktoré neboli vybrané do úlohy rodičov zaniknú a tie, ktoré vybrané boli sa automaticky stávajú aj členmi novej generácie. Napriek jednoduchosti vybraného spôsobu náhrady sa ukázal ako dostatočne efektívny z hľadiska celkovej konvergencie algoritmu.

6.1.5 Genetické operátory

Cieľom genetických operátorov je poskytnúť nástroj, umožňujúci zužitkovať štrukturálne elementy, z ktorých sú vytvorení rodičia, pre vytvorenie nových potomkov (v priestore prehľadávania). Keďže cieľom tvorby nových potomkov je získanie nových vzoriek na ploche vhodnosti, aplikácia genetických operátorov na štruktúru rodičov musí generovať potomkov, ktorých štruktúra je odlišná od štruktúry ich rodičov (či už sa jedná o odlišnosť menšiu, alebo väčšiu).

Najčastejší spôsob delenia genetických operátorov je založený na počte rodičov, potrebných pre vygenerovanie nových jedincov [34]:

- asexuálne operátory – potomok má iba jedného rodiča,
- sexuálne operátory – potomok má dvoch rodičov,
- panmiktické operátory – potomok má viac ako dvoch rodičov.

V práci sme použili sexuálny operátor kríženia – uniformné kríženie, pri ktorom je maska generovaná náhodným spôsobom. Každý potomok teda preberá náhodný počet atribútov od jedného rodiča, kým druhý potomok zdedí tieto atribúty od druhého rodiča – vznikajú tak 2 zrkadlové jedince. Hlavným dôvodom výberu tejto metódy bolo zachovanie poradia jednotlivých atribútov nášho jedinca a jednoduchá implementácia.

Zvolili sme tiež uniformný spôsob mutácie, kde pri úspešnej mutácii jedinca je jeden z jeho atribútov nastavený na novú, náhodnú hodnotu z intervalu <0, 1>. Testovali sme niekoľko variánt pravdepodobnosti mutácie, nakoniec ale učenie prebiehalo s 10%-nou pravdepodobnosťou mutácie jedincov.

6.1.6 Udržiavanie rôznorodosti populácie

Počas činnosti EA sa populácia jedincov v jednotlivých generáciách vyvíja. Jedince reprezentujúce kandidátov riešenia sa presúvajú do sľubnejších častí prehľadávacieho priestoru (charakterizovaných lepšou vhodnosťou) a postupne sa v nich sústreďujú. Výsledkom je snaha jedincov obývať iba jeden podpriestor tvoriaci malú časť celkového prehľadávacieho priestoru.

V prípade, že algoritmus skonverguje skôr než dokáže lokalizovať hľadané riešenie (populácia skonverguje k lokálnemu extrému), hovoríme o predčasnej konvergencii [34].

Existuje niekoľko spôsobov ako sa s takouto konvergenciou vysporiadať, v našej práci sme použili jeden z najjednoduchších, takzvanú elimináciu duplikácií:

V prípade, že sa pri vytváraní novej populácie objaví jedinec, ktorý sa v populácii už nachádza (zhodou náhod, pri aplikovaní genetických operátorov), na tohto jedinca je aplikovaná takzvaná hypermutácia (mutácia jedinca s 50%-nou pravdepodobnosťou).

6.2 Učenie

Prvým krokom pri inicializácii učenia v našej práci bola kontrola, či bol algoritmu poskytnutý súbor s počiatočnou populáciou, alebo nie. V prípade, že program tento súbor našiel, použil ako prvotnú populáciu jedince, ktoré v ňom boli uložené. V opačnom prípade sa vytvorila populácia o veľkosti 32 jedincov, kde každému z nich boli atribúty nastavené na náhodné hodnoty z intervalu <0, 1>.

Algoritmus bol otestovaný s viacerými variantami veľkosti populácie (od veľkosti 8 jedincov až po veľkosť 100), pričom uspokojivé výsledky sme dosahovali už pri spomínaných 32 jedincoch.

Po načítaní počiatočnej populácie bolo potrebné pre každého jedinca určiť jeho vhodnosť (podľa 6.1.2). Po určení vhodnosti jedincov bola uložená priemerná vhodnosť populácie, vhodnosť všetkých jedincov bola premapovaná škálovaním tak, aby žiaden nenadobúdal záporné hodnoty a jedince boli zoradené podľa svojej vhodnosti.

Ďalším krokom nášho učenia bol proces selekcie, pri ktorej bola do úlohy rodičov vybratá polovica aktuálnej populácie. Podmienkou bolo, že jeden jedinec nesmie obsadiť obidve miesta rodičov pri vytváraní nových potomkov (jeden jedinec sa ale môže zúčastniť na procese kríženia viac ako raz). Takisto bolo potrebné zabezpečiť, aby sa najlepší jedinec aktuálnej populácie aspoň jedenkrát stal rodičom (elitistický charakter algoritmu opísaný v 6.1.3). Nakoniec, v snahe udržať rôznorodosť populácie prebehla kontrola, či sa v množine potomkov už nenachádza rovnaký jedinec ako ten, ktorý vznikol krížením (6.1.6). V prípade, že takáto duplikácia nastala, na nového potomka bola aplikovaná hypermutácia s pravdepodobnosťou 50%.

Po získaní novej populácie jedincov spojením množiny rodičov a potomkov bola na túto populáciu aplikovaná mutácia s pravdepodobnosťou 10%. Ako sme popísali v 6.1.5, mutácia jedinca prebieha reinicializáciou jedného z jeho atribútov, teda jeho nastavením na novú hodnotu z intervalu <0, 1>.

Nová populácia bola následne zapísaná do súboru a algoritmus prešiel do ďalšej generácie, kde sa celý proces opakoval. Pre dosiahnutie uspokojivej konvergencie nášho algoritmu bolo pri všetkých experimentoch (7.4) dostatočných 15 generácií.

7 Riešenie a experimenty

V tejto kapitole opisujeme jednotlivé experimenty, použité scenaria, návrh hodnotenia výsledkov experimentov a učenie parametrov nášho agenta. Definujeme aj vzniknuté zmeny správania agenta, ukážeme si prostredie, v ktorom sme agenta implementovali, rovnako ako prostriedky, ktoré sme pri tom použili.

7.1 Implementácia

Pri programovaní nášho agenta sme použili programovacie prostredie BWAPI[3], resp. jeho Java wrapper – BWMirror [38].

BWAPI [12] je open-sourcový C++ framework na vytváranie inteligentných modulov pre hru StarCraft: Broodwar. Boti naprogramovaní pomocou BWAPI môžu získavať informácie o hráčoch a jednotkách v hre, rovnako ako rozdávať príkazy jednotlivým jednotkám, ktoré majú pod kontrolou.

Aj keď BWAPI umožňuje agentovi prístup k herným informáciám, odhaľuje mu iba viditeľné časti hry, ktoré by rovnako dokázal spracovať aj živý hráč. Teda jednotka, ktorá je zahalená fog of war-om je pre nášho agenta neviditeľná. Doteraz jedinou výhodou, ktorú majú k dispozícii BWAPI boti oproti ľudským hráčom, s ktorou sme sa stretli je, že prostredie (a teda aj agent) má prístup k jednotlivým ID číslam jednotiek, teda dokáže rozlíšiť 2 rovnako vyzerajúce jednotky v hre.

Keďže BWAPI vypína štandardné GUI StarCraftu, ľudský používateľ sa dostáva len do úlohy pozorovateľa. Napriek tomu, po zmene nastavení môžeme hocikedy v priebehu hry zasiahnuť a zadávať agentovi príkazy manuálne, spolu s botom (čo sa ukázalo byť veľmi užitočným pri debugovaní).

BWMirror API je Java wrapper BWAPI. Obsahuje všetky triedy, konštanty a enumerácie v Java objektoch, poskytujúc úplne rovnakú funkcionalitu ako pôvodné C++ BWAPI.

[3] https://code.google.com/p/bwapi/

7.2 Návrh experimentov

Controller, ktorý ovláda nášho agenta sme trénovali pre ovládanie jedného konkrétneho typu jednotiek – Dragoonov. Rasa Protossov (za ktorú hrá náš agent) je ako sme opísali v 3.2 typická armádami tvorenými menším počtom silných jednotiek, čo nám poskytuje výhodu v prípade, že tieto jednotky sú ovládané efektívnym spôsobom. Dragoon, ako jednotka s veľkou odolnosťou a dostrelom je ideálna na získanie takejto výhody. Ostatné jednotky rasy Protossov sú zväčša špecializované na boj zblízka, pri ktorom má ovládanie Controllerom podstatne menšie využitie.

Vzhľadom na to, že micromanagement agenta nie je s jednou množinou parametrov rovnako efektívny proti všetkým zloženiam armády, bolo v našej práci potrebné trénovať Controller jednotlivo pre niekoľko typických zložení armád. Vybrali sme 3 najčastejšie používané scenaria, pričom protivníkova armáda je v každom z nich zraniteľná (kvôli typu jednotiek, ktoré obsahuje) voči inému správaniu:

A. Dragoons vs. Zealots scenario

V prvom scenariu náš agent ovládal iba 2 jednotky typu Dragoon v boji proti 6 jednotkám typu Zealot (Obr. 9). Zealot je základná jednotka rasy Protossov, jedna z vôbec najčastejšie používaných jednotiek StarCraftu. Hlavnou nevýhodou týchto jednotiek je ich dosah, ktorý ich obmedzuje výhradne na boj zblízka. Hráči sa s týmto scenariom stretávajú často, preto bola táto kompozícia armád prvou voľbou pre naše experimenty. V scenariu má protivník podstatnú prevahu, či už v samotnom počte jednotiek, alebo v ich cene potrebnej na ich produkciu:

2 jednotky Dragoon stoja 250 minerálov, 100 plynu a 4 jednotky populačných zdrojov.

6 jednotiek Zealot stojí hráča 600 minerálov a 12 jednotiek populačných zdrojov.

Je teda zjavné, že v prípade ovládania oboch armád rovnakým spôsobom by bola protivníkova armáda vo výhode, napriek tomu sa náš agent po určitom čase dokázal s touto armádou vysporiadať pomerne bez problémov. Pre zvýraznenie rozdielu jednotiek v dosahu bolo našim jednotkám nastavené vylepšenie, zvyšujúce ich dosah (takzvaný singularity charge upgrade). Samotné experimenty sú popísané v 7.4.1.

Obr. 9 – Dragoons vs. Zealots scenario

B. Dragoons vs. Marines scenario

Druhé scenario zvolené pre naše experimenty proti jednotkám nášho agenta postavilo jednotky typu Marine, predstavujúce lacné, často používané jednotky s krátkym dosahom patriace pod rasu Terran (Obr. 10). Na rozdiel od Zealot-ov, tieto jednotky nie sú obmedzené na boj zblízka, no napriek tomu majú dosah menší ako jednotky našej armády. V scenariu má protivník opäť prevahu v zdrojoch potrebných na získanie použitej armády:

4 jednotky Dragoon stoja 500 minerálov, 200 plynu a 8 jednotiek populačných zdrojov.

12 jednotiek Marine stojí hráča 600 minerálov a 12 jednotiek populačných zdrojov.

Aj v tomto scenariu bol pre zvýraznenie rozdielu jednotiek v dosahu bolo našim jednotkám nastavené vylepšenie, zvyšujúce ich dosah (takzvaný singularity charge upgrade).

Obr. 10 – Dragoons vs. Marines scenario

C. Dragoons vs. Dragoons scenario

Posledné scenario, ktoré sme v experimentoch použili simulovalo boj rovnakých jednotiek, obe armády sa skladali výhradne z jednotiek typu Dragoon (Obr. 11). Toto scenario bolo predstavené pre získanie schopnosti agenta vysporiadať sa s jednotkami s rovnakým dosahom ako majú jednotky jeho armády.

Rovnako ako v predošlých scenariach, aj tu má protivník prevahu voči armáde nášho agenta (5 jednotiek Dragoon našej armády proti 6 jednotkám armády protivníka).

Obr. 11 – Dragoons vs. Dragoons scenario

7.3 Spôsob hodnotenia experimentov

Vzhľadom na obtiažnosť hodnotenia samotného micromanagementu agenta bolo potrebné definovať spôsob, akým budú jednotlivé experimenty hodnotené. Na konci každého experimentu sme porovnali efektivitu nášho Controllera s dvomi ďalšími agentami:

A. Natívna umelá inteligencia, implementovaná v hre StarCraft (auto-attack)

B. UIAlberta bot, úspešný agent vyvinutý na University of Alberta [35], ktorý sa úspešne zúčastnil na niekoľkých turnajoch umelých inteligencií

7.4 Priebeh experimentov

V tejto kapitole popisujeme jednotlivé výsledky učenia nášho Controllera pre každé popísané scenario, zároveň si definujeme vzniknuté vzory správania, ktoré z tohto učenia vyplynuli.

Po inicializácii počiatočných parametrov (popísanej v 6.2) bolo pre každé scenario odohratých 5 hier pre každého jedinca z generácie (spolu 160 hier na jednu

generáciu). Učenie parametrov bolo ukončené vždy po 15 generáciách, pričom konvergencia bola zväčša najsilnejšia do piatej generácie. Po skončení učenia nášho Controllera bolo odohratých 5 hier pre oboch agentov slúžiacich na porovnanie výsledkov (použité bolo priemerné skóre agentov z týchto 5 hier).

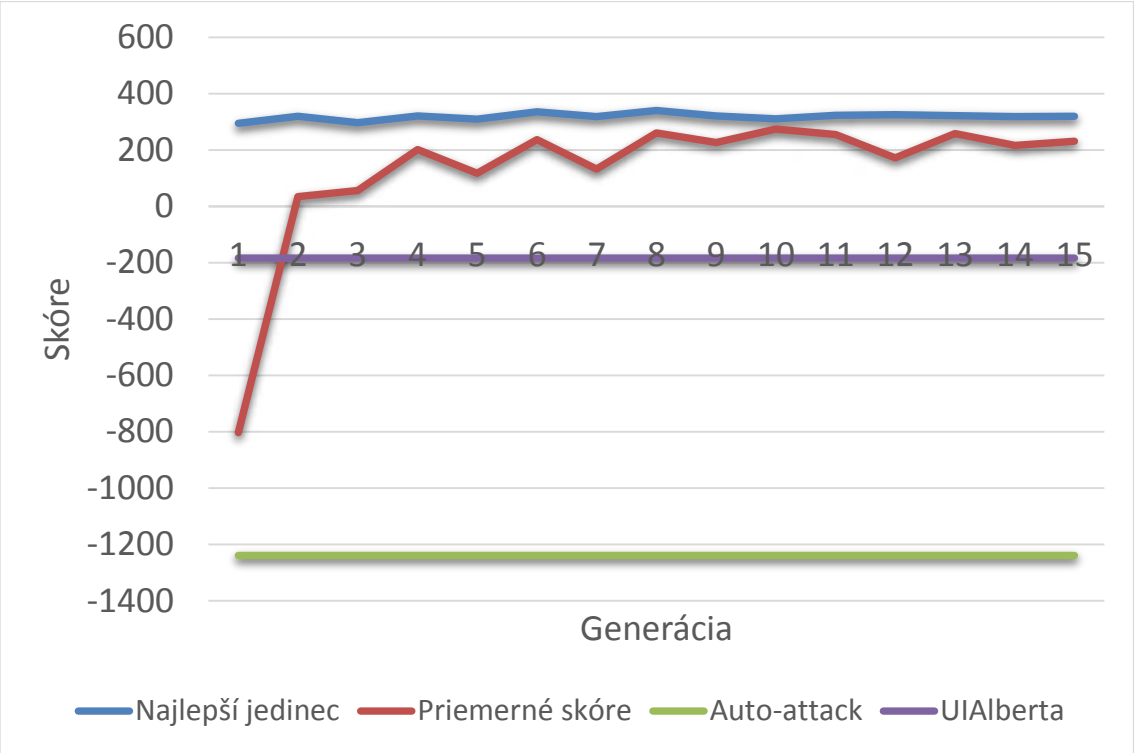
Z výsledkov je vidieť, že efektivita Controllera sa pre každé scenario líši, čo je spôsobené obmedzeniami, spojenými s pravidlami StarCraftu.

7.4.1 Dragoon vs. Zealot scenario

V boji proti jednotkám obmedzeným na boj zblízka majú jednotky typu Dragoon značnú výhodu. V tomto scenariu armáda protivníka napriek veľkej prevahe v počte jednotiek (7.2) nedokázala po niekoľkých generáciách nielen zničiť žiadnu z našich jednotiek, dokonca ich ani významne poškodiť.

Pri sledovaní správania našich jednotiek po niekoľkých generáciách bolo možné spozorovať vzor v ich správaní, takzvaný „kiting“. Toto správanie jednotiek sa vyznačuje využívaním výhody v dosahu jednotiek. Jednotky sa sťahujú z dosahu protivníka dovtedy, kým nie je možné zaútočiť a opäť sa presunúť skôr, než jednotka protivníka je schopná poškodiť ovládanú jednotku. Je nutné podotknúť, že takéto správanie vyžaduje podstatne viac času na boj ako pri bežných stretoch.

Ako je vidieť na Obr. 12, Controller s naučenými hodnotami parametrov po niekoľkých generáciách bez problémov prekonal oboch agentov, s ktorými bol porovnávaný. V tomto scenariu sa dokonca podarilo dosiahnuť optimálne riešenie (vzhľadom na skóre hry), teda hra sa niekoľkokrát skončila bez poškodenia našich jednotiek.



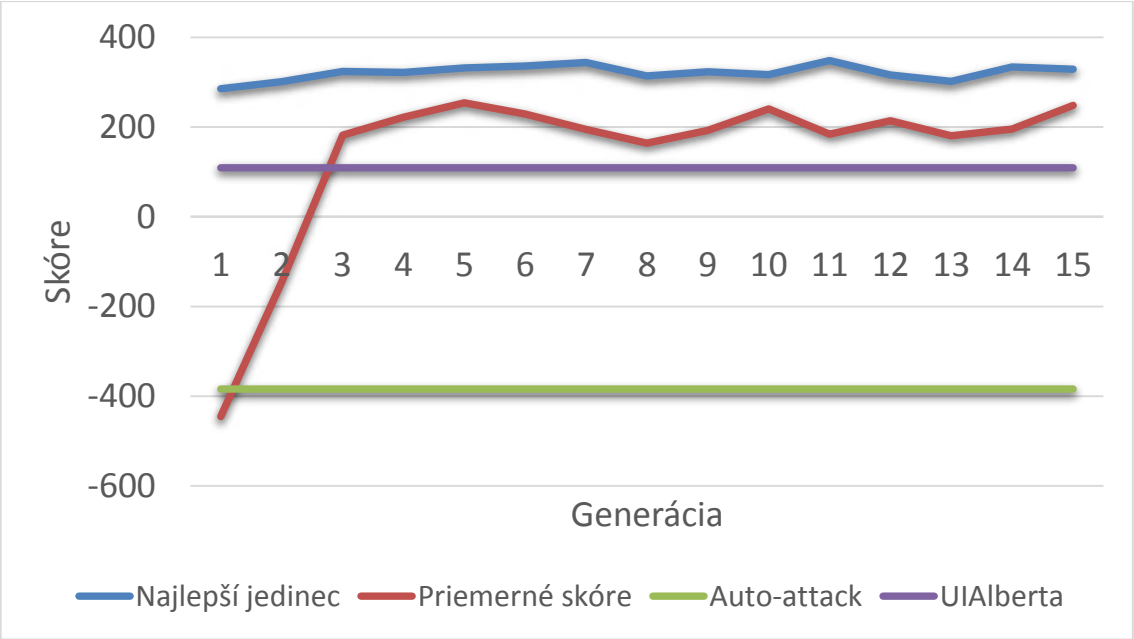
Obr. 12 – Graf výsledkov Dragoons vs. Zealots scenaria

7.4.2 Dragoon vs. Marine scenario

Jednotky typu Marine predstavujú priemerného reprezentanta jednotiek StarCraftu s ohľadom na dosah. Preto sme predpokladali, že výsledky sa budú od prvého scenaria značne líšiť. Napriek tomu sa krivka konvergencie značne podobá prvému scenariu.

Ovládané jednotky sa po niekoľkých generáciách začali správať vyhýbavo, osvojili si kiting, popísaný v 7.4.1. Toto správanie takisto spôsobilo, že hry trvali dlhšie ako pri bežných stretoch – čas hier bol priamo úmerný skóre nášho agenta.

Pri porovnávaní výsledkov nášho Controllera s výsledkami natívnej umelej inteligencie StarCraftu a botom UIAlberta je vidieť síce menšie rozdiely ako v predošlom scenariu, no napriek tomu náš agent oboch prekonáva výrazným spôsobom.



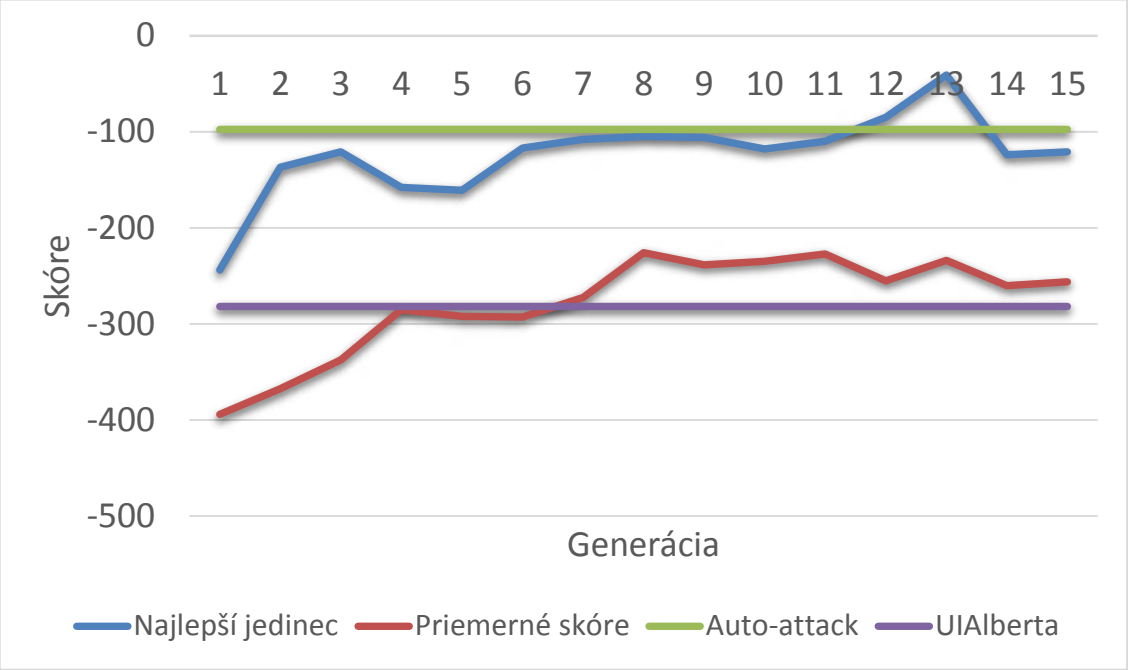
Obr. 13 – Graf výsledkov Dragoons vs. Marines scenaria

7.4.3 Dragoon vs. Dragoon scenario

Toto scenario bolo predstavené s cieľom nastaviť parametre Controllera tak, aby sme dosiahli maximálnu efektivitu v boji proti jednotkám s rovnakým dosahom. Vzhľadom na obmedzenia spojené s pravidlami StarCraftu, jednotky ovládané našim Controllerom dosahovali horšie výsledky ako jednotky používajúce bežný spôsob boja, implementovaný v StarCrafte. Tento jav je spôsobený tendenciou nášho Controllera využívať v boji pohyb jednotiek, a získať tak výhodu oproti armáde protivníka.

Jednotky špecializujúce sa na boj na diaľku majú okrem cooldown-u (definovanom v 3.3) aj čas, kedy prebieha animácia útoku. V prípade jednotky typu Dragoon je to akési otváranie a zatváranie kanónu, slúžiaceho na streľbu, ktoré musí prebehnúť vždy, keď je jednotka medzi útokmi presúvaná na inú pozíciu. Toto spôsobuje, že pri boji jednotiek rovnakého typu je ich nadmerné presúvanie na iné pozície často škodlivé pre celkový vývoj boja.

Preto výsledky experimentov na tomto scenariu ukazujú, že náš Controller dosahuje iba podobné výsledky ako natívna umelá inteligencia implementovaná v StarCrafte, ktorá jednotkami v boji takmer nepohybuje. Náš Controller počas učenia postupne obmedzil pohyb jednotiek na minimum a menili sa iba parametre ovplyvňujúce výber cieľa pre útočenie.



Obr. 14 – Graf výsledkov Dragoons vs. Dragoons scenaria

7.5 Vyhodnotenie experimentov

Výsledky nášho agenta prekonali agentov, s ktorými boli porovnávané vo väčšine prípadov. Controller dosahoval najlepšie výsledky v stretoch s jednotkami s krátkym a stredným dosahom. Pri stretoch s jednotkami s rovnakým dosahom sme dosiahli výsledky podobné ako natívna umelá inteligencia implementovaná v StarCrafte.

Tieto experimenty nám poskytli možnosť implenetovať adaptačný mechanizmus, ktorý na základe zloženia protivníkovej armády vyberie najvhodnejšie parametre pre Micromanagement Controller nášho agenta počas hry a umožní mu tak vysporiadať sa s protivníkom efektívnym spôsobom.

Princíp tejto adaptácie sa podobá na rozhodovací proces v [33], kde počas hry agent pravidelne sleduje zloženie protivníkovej armády. Na základe tejto informácie sa agent dokáže rozhodnúť, ktorú hernú variantu má zvoliť pre dosiahnutie čo najväčšej efektivity.

V našom prípade teda agent sleduje všetky jednotky v protivníkovej armáde a vypočíta ich priemerný dosah. V prípade, že armádu ako celok definuje ako armádu

s krátkym, resp. stredným dosahom, volí náš agent parametre pre boj s týmto typom armády. V opačnom prípade (ak agent stojí proti armáde s veľkým dosahom) jeho Controller používa parametre pre boj s touto armádou, podobne ako pri experimente *Dragoons vs. Dragoons*.

8 Hodnotenie riešenia

Úlohou našej práce bolo navrhnúť systém používajúci EA pri učení agenta hrajúceho Real-Time Strategické hry. Práca tiež obsahuje prehľad aplikácií umelej inteligencie v počítačových hrách, vrátane prehľadu využitia EA v tejto problematike.

Jednou z výziev práce bolo adresovať problém dimenzionality StarCraftu ako prehľadávacieho priestoru optimalizačného problému umelej inteligencie. Tento problém je spôsobený veľkým počtom parametrov v riadiacich mechanizmoch bežne používaných na ovládanie jednotiek v real-time strategických hrách.

Oproti typickým spôsobom riešenia micromanagementu v RTS hrách naše riešenie ponúkalo Controller ovplyvňovaný iba 8 učenými parametrami. Implementovali sme ho do klasickej RTS hry StarCraft a spomínané parametre trénovali pomocou EA.

Po vyhodnotení experimentov sa ukázalo, že naše riešenie dosahuje výnimočne dobré výsledky v dvoch testovaných scenariach (v treťom sa jednalo o výsledky veľmi podobné agentom, s ktorými sme náš Controller porovnávali). Výsledky boli závislé na type jednotiek, ktoré ovládal protivník. „Udri a uteč“ správanie nášho agenta sa ukázalo byť extrémne efektívne proti jednotkám s krátkym a stredným dosahom. Pričom proti jednotkám s väčším dosahom sa ukázalo byť neúspešné a po niekoľkých generáciách bolo potlačené a nahradené statickejším prístupom. Samotný Controller sa ukázal byť efektívnym už po niekoľkých generáciách EA.

9 Záver

Prvý cieľ práce, teoretický úvod využitia umelej inteligencie v real-time strategických hrách, je popísaný v kapitole 3. Hovorí o strategických hrách ako o prostredí z pohľadu umelej inteligencie, napríklad prehľadávacom priestore optimalizačných problémov.

Početné využitia umelej inteligencie v real-time strategických hrách nájdeme popísané v kapitole 4, pričom v každej podkapitole bol kladený dôraz na využitie evolučných algoritmov v problematike.

Kapitoly 5 a 6 sa venujú samotnému návrhu nášho systému. Kapitola 5 definuje nami použitý Micromanagement Controller ako Finite-state machine, s dôkladným popisom jeho najdôležitejších metód a ich činnosti. Kapitola 6 obsahuje teoretické základy evolučných algoritmov, vrátane popisu ich implementácie do nášho systému. Koniec tejto kapitoly popisuje celý učiaci proces algoritmu použitého v našej práci.

Štvrtý cieľ, implementácia a experimentálne overenie systému je popísaný v kapitole 7. Po návrhu samotných experimentov a určení spôsobu ich hodnotenia, je každý z nich popísaný v samostatnej podkapitole. Na záver kapitoly sa venujeme zhodnoteniu experimentov a software-ovému opisu prostriedkov použitých v našej práci.

Dokumenty potrebné na vypracovanie piateho cieľa (používateľská a systémová príručka) sa nachádzajú v prílohách.

Okrem cieľov práce definovaných v zadaní (kapitola 1) počas práce na systéme vzniklo niekoľko podcieľov, ktoré v zadaní diplomovej práce definované neboli:

- zavedenie dynamickej veľkosti okolia prehľadávaného Controllerom (5.2.2)
- vytvorenie adaptačného mechanizmu agenta (7.5)
- publikovanie článku o diplomovej práci na študentskú konferenciu SCYR
- prihlásenie nášho agenta do turnaja umelých inteligencií SSCAIT [13]

Zoznam použitej literatúry

[1] D. W. Aha, M. Molineaux, M. Ponsen, "Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game," Case-Based Reasoning Research and Development, Lecture Notes in Computer Science Volume 3620, 2005, 5-20.

[2] P. Cadena, L. Garrido, "Fuzzy Case-Based Reasoning for Managing Strategic and Tactical Reasoning in StarCraft," Advances in Artificial Intelligence, Lecture Notes in Computer Science Volume 7094, 2011, 113-124.

[3] K. De Jong, A. C. Schultz, "Using experience-based learning in game playing," Proceedings of the Fifth International Conference on Machine Learning, 1988, 284-290.

[4] B. Daz-Agudo,P. Gervs, F. Peinado, "A case based reasoning approach to story plot generation," Proceedings of the Seventh European Conference on Case-Based Reasoning, 2004, 142-156.

[5] C. R. Fairclough, P. Cunningham, "AI structuralist storytelling in computer games," Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education, 2004, 5.

[6] M. J. Fasciano, "Everyday-world plan use," Technical Report TR-96-07, The University of Chicago, Computer Science Department, 1996.

[7] Y. Kerner, "Learning strategies for explanation patterns: Basic game patterns with applications to chess," Proceedings of the First International Conference on Case-Based Reasoning, 1995, 491-500.

[8] S. Ontanón, K. Mishra, N. Sugandh, A. Ram, "Case-Based Planning and Execution for Real-Time Strategy Games," Lectuire Notes in Computer Science Volume 4626, 2007, 164-176.

[9] A. Samuel, "Some studies in machine learning using the game of checkers," IBM Journal of Research and Development, 3(3), 1959, 210-229.

[10] B. G. Weber, M. Mateas, "Case-Based Reasoning for Build Order in Real-Time Strategy Games," In Proceedings of the Fifth Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE), 2009, 106-111.

[11] S. Russel, P. Norvig, "Artificial intelligence: A modern approach," In Prentice Hall, Engelwood Cliffs, NJ, 1995.

[12] BWAPI, An API for interacting with StarCraft: Broodwar (1.16.1) [online], 2015. Dostupné na internete: <https://code.google.com/p/bwapi/>

[13] SSCAIT, Student StarCraft AI Tournament [online], 2015. Dostupné na internete: <http://sscaitournament.com>

[14] A. Liapis, G. N. Yannakakis, J. Togelius "Towards a Generic Method of Evaluating Game Levels," In Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2013, 30-36.

[15] R. van der Linden, R. Lopes, R. Bidarra, "Procedural generations of dungeons," IEEE Transactions on Computational Intelligence and AI in Games, 2013, 78-89.

[16] S. Shah, D. Singh, J. Shah, "Using Genetic Algorithm to Solve Game of Go-Moku," In Special Issue of International Journal of Computer Applications (0975-8887) on Optimization and On-chip Communication, No.6., 2012, 28-31.

[17] T. Cunningham, "Using the Genetic Algorithm to Evolve a Winning Strategy for Othello," Stanford University, California, 2003.

[18] F. Arnold, B. Horvat, A. Sacks, "FreeCiv Learner: A Machine Learning Project Utilizing Genetic Algorithms," Georgia Institute of Technology, Atlanta, 2004.

[19] I. Watson, D. Azhar, Y. Chuyang, W. Pan, G. Chen, "Optimization in Strategy Games: Using Genetic Algorithms to Optimize City Development in FreeCiv," University of Auckland, 2008.

[20] N. Cole, "Using a Genetic Algorithm to Tune First-Person Shooter Bots," Evolutionary Computation (CEC2004), 2004, 139-145.

[21] A. I. Esparcia-Alcázar, A. Martinez-Garcia, A. Mora, JJ. Mercelo, P. Garcia-Sánchez, "Controlling bots in a First Person Shooter Game using Genetic Algorithms," Evolutionary Computation, 2010, 1-8.

[22] L. Cardamone, G. N. Yannakakis, J. Togelius, P. L. Lanzi, "Evolving Interesting Maps for a First Person Shooter," Applications of Evolutionary Computation, 2011, 63-72.

[23] W. Zhe, K. Nguyen, R. Thawonmas, F. Rinaldo, "Using Monte-Carlo Planning for Micro-management in StarCraft," In Proceedings of the 4th annual Asian GAME-ON Conference on Simulation and AI in Coputer Games (GAMEON ASIA 2012), Kyoto, Japan, 2012, 33-35.

[24] R. Parra, L. Garrido, "Bayesian Networks for Micromanagement Decition Imitation in the RTS Game StarCraft," Tecnólogico de Monterrey, Campus Monterrey, México, 2013, 433-443.

[25] C. Lin, "Emergent Tactical Formation Using Genetic Algorithm in Real-Time Strategy Games," Technologies and Applications of Artificial Intelligence (TAAI), 2011, 325-330.

[26] A. Tavares, H. Azpúrua, L. Chaimowicz, "Evolving swarm intelligence for task allocation in real time strategy game," In Proceedings of the SBGames 2014, 99-108.

[27] S. Liu, S. J. Louis, C. Ballinger, "Evolving Effective Micro Behaviors in RTS Game," At Dept. of Computer Science and Engineering, University of Nevada, Reno, 2014, 1-8.

[28] S. Liu, S. J. Louis, M. Nicolescu, "Comparing Heuristic Search Methods for Finding Effective Group Behaviors in RTS Game," Dept. of Computer Science and Engineering, University of Nevada, Reno, 2013, 1371-1378.

[29] Jørgen Bøe Svendsen, "Micromanagement in StarCraft using Potential Fields tuned with a Multi-Objective Genetic Algorithm," Norwegian University of Science and Technology, 2012.

[30] T. W. Sandberg, "Evolutionary Multi-Agent Potential Field based AI approach for SSC scenarios in RTS games," IT University of Copenhagen, 2011.

[31] I. Gabriel, V. Negru, D. Zaharie, "Neuroevolution based Multi-agent System for Micromanagement in Real-Time Strategy Games," In Proceedings of the Fifth Balkan Conference in Informatics, 2012, 32-39.

[32] M. Ponsen, "Improving Adaptive Game AI with Evolutionary Learning," Faculty of Media & Knowledge Engineering, Delft University of Technology, 2004.

[33] M. Čertický, M. Čertický, "Case-Based Reasoning for Army Composition in Real-Time Strategy Games," In Proceedings of Scientific Conference of Young Researchers (SCYR 2013), 2013, 70-73.

[34] M. Mach, "Evolutionary algorithms: elements and principles," In Elfa, Košice, ISBN 978-80-8086-123-0, 2009.

[35] D. Churchill, M. Buro, "Build Order Optimization in StarCraft," In Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2011, 14-19.

[36] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, M. Preuss, "A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft," In IEEE Transactions on Computational Intelligence and AI in games, 5 (4), 2013, 1-19.

[37] D. Churchill, M. Buro, "Fast Heuristic Search for RTS Game Combat Scenarios," In Proceedings of AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2012, 112-117.

[38] BWMirror API – Java wrapper for BWAPI [online], 2015. Dostupné na internete: < https://http://bwmirror.jurenka.sk/>

[39] M. Čertický, "Aplikácia case-based reasoningu v real-time strategických hrách," Technická Univerzita v Košiciach, 2013.

Prílohy

Príloha A: CD médium, ktoré obsahuje:

- Diplomovú prácu v elektronickej podobe
- Používateľskú príručku v elektronickej podobe
- Systémovú príručku v elektronickej podobe
- Projekt solution programu (zdrojové kódy)
- Spustiteľnú verziu programu

Príloha B: Používateľská príručka

Príloha C: Systémová príručka

Príloha D: Článok v anglickom jazyku