VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

ŘÍZENÍ ENTIT VE STRATEGICKÉ HŘE ZALOŽENÉ NA MULTIAGENTNÍCH SYSTÉMECH

STRATEGIC GAME BASED ON MULTIAGENT SYSTEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. PETR KNAPEK**
AUTHOR

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

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Knapek Petr, Bc.**
Program: Informační technologie Obor: Inteligentní systémy
Název: **Řízení entit ve strategické hře založené na multiagentních systémech**
Strategic Game Based on Multiagent Systems
Kategorie: Umělá inteligence

Zadání:

1. Prostuduje současný stav v oblasti řízení entit ve strategických hrách využívajících poznatků z oblasti multiagentnich systémů.
2. Z metod v oblasti strojového učení a plánování určete takové metody, které jsou vhodné k využití ve strategických hrách. Použití těchto metod má vést k zlepšování schopností počítačem řízených entit dosahovat lepších výsledků v těchto hrách.
3. Pro zvolené řešení navrhněte a implementujte algoritmy, které bude možné využít pro současné hry typu StarCraft.
4. Otestujte a zhodnoťte přínos vytvořeného díla jednak z hlediska racionalizace chování řízených entit ve srovnání s jejich původním chováním a také z hlediska možnosti jeho reálného použití v současných hrách.

Literatura:

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

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

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

Abstrakt

Tato práce se zaměřuje na návrh a implementaci systému podpory učení a plánování agentů schopných hry real-time strategických her typu StarCraft. Budou vysvětleny problémy ovládání herních objektů a protivníků počítačem a představeny obvykle používané způsoby řešení. Na základě analýzy je navržen a implementován nový systém využívající multiagentního přístupu k ovládání hry a metod strojového učení, který je schopný porážet protivníky a přizpůsobovat se novým výzvám.

Abstract

This thesis is focused on designing and implementing system, that adds learning and planning capabilities to agents designed for playing real-time strategy games like StarCraft. It will explain problems of controlling game entities and bots by computer and introduce some often used solutions. Based on analysis, a new system has been designed and implemented. It uses multi-agent systems to control the game, utilizes machine learning methods and is capable of overcoming oponents and adapting to new challenges.

Klíčová slova

Strategické hry, RTS, umělá inteligence, AI, strojové učení, posilované učení, neuronové sítě, Q-learning, agentní systémy, rozhodování a plánování, StarCraft, bot

Keywords

Stratic games, RTS, artificial intelligence, AI, machine learning, reinforcement lerning, neural networks, Q-learning, agent systems, decision-making and planning, StarCraft, bot

Citace

KNAPEK, Petr. *Řízení entit ve strategické hře založené na multiagentních systémech.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. František Zbořil, Ph.D.

Řízení entit ve strategické hře založené na multiagentních systémech

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci, návrh a implementaci systému vypracoval samostatně pod vedením Doc. Ing. Františka Zbořila, Ph.D. Konzultace poskytl i Bc. Lukáš Válek, který navrhl a vytvořil použitý multiagentní systém a poskytl základní implementaci systému pro StarCraft. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Knapek
22. května 2019

Poděkování

Tímto chci poděkovat vedoucímu práce, Doc. Ing. Františku Zbořilovi, Ph.D., za vedení, konzultace, čas a rady, které mi věnoval, a bez kterých by dokončení této práce nebylo možné. Rovněž bych rád poděkoval Bc. Lukáši Válkovi za spolupráci při vytvoření rozhraní multiagentního frameworku a úpravám v jeho rozhraní.

Obsah

Úvod

Hraní a vyhrávaní her počítačem řízenými procesy je programátorskou výzvou již několik desetiletí. Velké popularitě se tato disciplína těší nejvíce asi od roku 1996, kdy stroj s označením Deep Blue porazil tehdy úřadujícího světového velmistra v šachu, Garryho Kasparova.

Během let však museli programátoři čelit rychle rostoucí komplexitě pravidel a ovládání počítačových her, ve kterých se odehrává stále více věcí či pohybuje stále více entit najednou. I hráči se během let stali náročnějšími a zkušenějšími, často se specializují pouze na jednu hru, stále se v ní zdokonalují. Počítačoví oponenti jsou příliš statičtí, nemění své chování a nereagují na změny v hráčově strategii.

Cílem této práce je navržení systému učení a plánování určeného pro počítačem ovládané protivníky strategických her, mechanikami podobných hře StarCraft: Brood War, založeného na multiagentních systémech. Tento systém by měl zajistit přizpůsobivost, průběžné zlepšování a racionalizaci akcí agentů tak, aby byli schopni vyhrávat a při prohře se učit ze svých chyb. Implementací systému vzniká počítačem ovládaný protivník zvaný bot.

V první kapitole, 1, jsou popsány základní pojmy relevantní pro tuto práci a druhy strategických her s rozborem klíčových vlastností a aspektů nesoucích výzvy pro počítačem ovládaného protivníka. Rovněž je zde představena hra StarCraft: Brood War, jejímž principům se systém věnuje.

V kapitole 2 jsou popsány základní druhy herních umělých inteligencí a přístupy k jejich vytváření. Větší pozornost je věnována systémům založených na autonomních a racionálních agentech.

V kapitole 3 jsou uvedeny a částečně představeny metody rozhodování a strojového učení nejčastěji používané pro racionalizaci a plánování agentů, které jsou určeny pro tvorbu inteligentních herních protivníků pro zadaný druh her. V této kapitole je zmíněn způsob fungování jednoho z nejinteligentnějších botů současnosti, AlphaStaru od společnosti DeepMind.

V kapitole 4 je představen návrh systému založený na analýze získaných informací popsaných v předešlých kapitolách, multiagentním frameworku a jeho rozhraní, datech, která pro svou správnou činnost potřebuje a problémy, které řeší.

V následující kapitole, 5 je pak popsána implementace systému ve formě modelu bota pro StarCraft a použité knihovny.

V poslední kapitole, 6, jsou popsány průběh a výsledky testování hlavních částí systému.

Na projektu jsem spolupracoval s Bc. Lukášem Válkem, který navrhl a implementoval využívaný multiagentní systém. Rovněž vytvořil jednoduchého bota, který byl použit a rozšířen.

Kapitola 1

Strategické počítačové hry

V této kapitole jsou rozebrány základní principy a aspekty strategických počítačových her. Práce se věnuje především těm strategickým hrám, kde hráči soupeří ničením herních objektů ostatních. Jiné žánry nebudou brány v potaz. Strategické počítačové hry se někdy označují také jako herní simulace. Podle přístupu k simulačnímu času se dělí na „Real-time“, kde čas figuruje jako spojitá veličina a ubíhá volně, a tahové,„Turn-based“, s diskrétním časem o pevné velikosti kroku. Práce se zabývá především prvním druhem, rozebrán je však zběžně i typ druhý.

Cílem práce je návrh a implementace systému s podporou učení pro hry typu StarCraft: Brood War, v poslední části této kapitoly je proto popis této hry a rozbor základních herních mechanik.

1.1 Vlastnosti a historie obecných strategických her

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

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

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

V dnešní době však díky velkému výpočetnímu výkonu a rychlosti, omezenému počtu efektivních strategií a dostatečné kapacitě operační paměti pro uchování kompletní znalosti celého hracího pole při mnoha kombinacích příštích tahů zvládne počítač často zvítězit jen díky možnosti uvážení všech možností. V počítačových strategických hrách však takovou možnost nemají. Obecně musí ovládat více jednotek, a každá z nich je schopna provést více akcí. Herní plocha je typicky mnohem větší. Navíc běžně bývá část herního prostoru pokryta tmou, tzv. *„Fog of War“*, takže nelze vidět soupeřovy tahy (jako je tomu v šachu) – tedy nemá k dispozici úplná a přesná data. To musí pouze odhadovat z průběžně získávaných informací. V případě Real-time strategických her hraje značnou roli i čas. Veškeré výpočty se musí provádět opakovaně, mnohdy během každého snímku (reakce na podněty musí být okamžité), nemohou proto zabrat příliš mnoho času.

1.2 Real-time strategické hry

Real-time strategie, nebo taky „RTS“, jsou takové (obvykle bojové) strategické hry, kde se vývoj neřídí inkrementálně po střídavých tazích, ale všichni hráči řídí své postavy nebo jednotky zároveň. Typicky může každý hráč postavit více budov a jednotek různých druhů. Úkolem je pak zničení všech takovýchto objektů patřících oponentům nebo dosažení nějakých speciálních cílů, například obsazení části mapy či nashromáždění surovin.

1.2.1 Aspekty a fáze hry

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

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

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

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

1.2.2 Komplexita hry

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

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

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

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

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

1.3 Tahové strategické hry

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

1.3.1 Aspekty hry

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

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

Souboje mezi hráči mohou probíhat buď na velké strategické mapě sdílené s pohybem armád a stavbou budov. V některých případech jsou na strategické mapě reprezentovány celé armády jako pouze jedna entita a při střetu s nepřátelskou armádou se hra přepne na tzv. taktickou mapu. Na té již může probíhat bitva se zcela jiným pohledem. Heroes of Might and Magic [16] je typickým příkladem strategie s rozdělenou strategickou a taktickou mapou, kde se na obou hraje tahově. Hry ze série Total War mají strategickou mapu tahovou, ale bitva probíhá jako RTS.

1.3.2 Komplexita hry

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

1.4 StarCraft: Brood War

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

Obrázek 1.1: Zergské jednotky ničí základnu a dělníky Protossů

1.4.1 Popis hry

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

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

1.4.2 Herní suroviny a ekonomika

Během hry jsou důležité tři suroviny:

- **Minerály (Minerals)** – základní surovina v podobě modrých krystalů. Lze ji těžit již od úplného začátku hry a prakticky všechny jednotky a budovy ve hře potřebují minerály ke své produkci. Rozvinutí dostatečné těžby minerálů je od začátku hry hlavním ekonomickým cílem hráče a je potřeba použít minimálně desítky těžebních jednotek.

- **Vespene (Vespene gas)** – pokročilá surovina potřebná až pro lepší druhy jednotek a jejich vylepšení. Těžbou vespenu může hráč získat technologický náskok nad protivníky. Na počátku hry je potřeba nalézt gejzír vespenu, postavit na něm speciální budovu realizující těžbu a v ní následně „zaměstnat“ dělníky. Těžba vespenu je mno-

hem pomalejší vzhledem k nízkému počtu dostupných gejzírů a je potřeba s touto surovinou více šetřit.

- **Populační body (Supply)** – všechny jednotky ve hře obsadí svým vyrobením část hráčovy populace (často jednotky obsadí více populačních bodů než 1). Maximální možná velikost je 200 obsazených populačních bodů. Od počátku hry však hráč může mít jen mnohem méně jednotek a k tomuto číslu se může dostat pouze stavbou speciálních budov nebo jednotek, které mu další populaci poskytnou. Všechny tři herní frakce mají vlastní verzi populačních bodů.

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

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

1.4.3 Herní frakce

Ve StarCraftu lze hrát za tři zcela odlišné frakce:

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

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

- Protoss – Velmi vyspělá rasa s velmi odolnými, ale drahými jednotkami, které se trénují delší dobu než u ostatních ras. Spoléhají se na silné energetické zbraně a štíty, které se samy regenerují. Kombinují velmi silné jednotky pro boj z blízka a rychlé jednotky pro boj na dálku. Většina jejich budov musí být postavena pouze v okolí tzv. „Pylonů“, speciálních budov přidávajících populační body a generujících energii pro ostatní stavby. Dělník nemusí být přítomen u budovy a může se vrátit ki těžbě

– stavba budovy je vlastně jen teleport již hotové konstrukce na zvolené místo. U Protossů existuje velmi dobrá rovnováha mezi útokem a obranou, avšak kvůli menšímu počtu jednotek se jim může snadno stát, že je nepřítel obklíčí, hlavně v prvotních fázích hry.

Kapitola 2

Herní AI, rozhodování entit a strojové učení ve strategických hrách

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

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

2.1 Herní AI

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

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

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

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

2.1.1 Úplná informace

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

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

2.2 Základní typy herních AI

2.2.1 Statické skripty

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

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

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

Příklad řídící sekvence:

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

2.2.2 Konečné stavové automaty

Komplexnější způsob řízení herní AI [26]. Umožní každé jednotce volit svou následující akci podle jejího současného stavu [24]. Změna stavu však může být vyvolána nejen podnětem z okolí (v tomto případě ekvivalent abecedy přijímané konečném automatem), ale i změnou času. Tyto automaty musí být zcela deterministické.

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

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

2.2.3 Alfa-beta ořezávání, Min-Max

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

2.3 Systémy založené na agentech

Herní AI řízené agenty splňují obvykle všechny definice agentních systému. Agenti jsou autonomní, reaktivní, racionální a řízení záměrem a schopni vzájemné komunikace umožňující jim aplikovat sociálních schopností ke společnému plnění cílů [34].

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

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

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

2.3.1 Každá jednotka jako agent

První alternativou je použití agentů k reprezentaci všech hráčových jednotek [24]. Každá jednotka má k dispozici úplný popis možného chování a všechny možnosti k rozhodování. Tento způsob je potenciálně nejlepší po stránce reaktivnosti a efektivity celého systému, naopak nejhorší co se týče výpočetního výkonu. Důvodem je procházení logiky agenta opakovaně a často zbytečně. Příkladem je shluk dvaceti jednotek, kdy všechny hromadně útočí. Proces nalezení optimální pozice, optimálního cíle a použitých schopností může pro většinu (nebo dokonce všechny) z nich skončit s velmi podobným či zcela totožným výsled-

kem. V nejhorším případě jsme tedy dosáhli až dvacetinásobného zpomalení výpočtu, a to případně v každém snímku hry.

2.3.2 Centrálně řízené skupiny

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

2.3.3 Hybridní přístup

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

2.3.4 Zhodnocení agentních přístupů

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

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

- **Kiting (Pouštění draka)** – obdobná technika spočívající v použití jedné nebo několika málo jednotek jakožto návnady pro nepřátelské jednotky. Tyto návnady se pak neustále pohybují kolem spojenecké armády tak, aby na sebe zlákaly veškeré útoky a tak chránily zbytek armády, který se může soustředit na ničení. Pakliže nepřítel změní cíl, role se některých jednotek se změní, nová návnada okamžitě zahájí pohyb, předchozí se zastaví a začne útočit. Problémem bývá okamžitá detekce aktuálního

cíle nepřátelských vojsk a volba cílových pozic tak, aby nedocházelo ke kolizi, nepřítel nezměnil cíl, nedohnal návnadu a spojenecké síly mohly pálit.

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

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

Kapitola 3

Metody strojového učení, plánování a optimalizačních algoritmů ve strategických hrách

V této kapitole jsou představeny algoritmy a metody používané pro tvorbu racionálních a pokročilých herních umělých inteligencí, především pro StarCraft. Zdrojem informací o způsobech využití se stala hlavně revize botů soutěže SSCAIT (Student StarCraft Artificial Intelligence Tournament) od Michala Čertického a Davida Churchilla [35].

Metody a algoritmy jsou představeny jen zběžně, u všech jsou uvedeny i příklady potenciálního využití. Systém navržený v této práci některé z nich využívá. Jako jeden z nejčastějších přístupů je posilované učení [21] [30].

Na konci kapitoly je představen relativně nový bot od společnosti DeepMind, AlphaStar [5]. Jedná se o jednoho z nejlepších botů současnosti, schopného s přehledem porážet i profesionální hráče (dokonce i pěti vítězstvími z pěti her) a určeného pro StarCraft 2.

3.1 Evoluční algoritmy a dynamické skriptování

V umělé inteligenci je za evoluční algoritmy označen druh populačně orientovaných algoritmů pro optimalizaci a návrh [22]. Používá mechanismy inspirované přírodní evolucí, nazývané **genetické operátory**, jako je reprodukce, mutace, rekombinace a výběr (selekce). Populace je množinou kandidátních řešení problému, který se evoluční algoritmus snaží vyřešit. Kvalitu těchto řešení vyhodnocuje tzv. „fitness funkce“. Evoluční algoritmy se používají pro aproximaci řešení problémů a mnohdy si vedou dobře díky náhodnosti kandidátních řešení bez vazby na předpoklady nebo představy o řešení. V mnoha případech je limitujícím faktorem výpočetní náročnost – je potřeba potenciálně velkého množství iterací (nových generací), je nutné zavést ukončující podmínku.

Dynamické skriptování v herních umělých inteligencích obvykle znamená aplikaci evolučních algoritmů (zřídka pak jiných optimalizačních algoritmů) v kombinaci s herním botem založeném na statických skriptech (viz. 2.2.1). Kandidátními řešeními jsou v takovém případě posloupnosti příkazů potenciálně vedoucí k vítězství ve hře. Marc Ponsen, Héctor Muñoz-Avila, Pieter Spronck, a David W. Aha použili této metody k tvorbě bota pro hru Warcraft 2 [27], kdy se snažili o nalezení vhodných reakcí na strategie běžně používané lidskými oponenty. V případě Real-time strategií s sebou tato metoda přináší zásadní nevýhodu v nutnosti odehraní hry k ohodnocení kandidátního řešení na základě výsledku,

délky zápasu apod, stále je však použitelná pro optimalizace [33], potenciálně například fronty akcí manažerů v centralizovaném (agentním) systému [23].

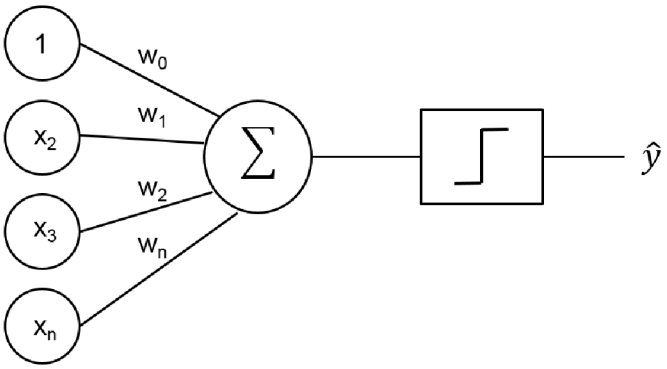
Data: Náhodná počáteční populace o velikosti n
každému jedinci z počáteční populace přiřaď hodnotu fitness;
repeat
  Selekce: vyber $n/2$ jedinců pro křížení („mating pool“);
  for *i in 0..n/2* **do**
    Náhodně vyber dva jedince za rodiče;
    Křížení: vytvoř dvojici potomků, každý s jistou pravděpodobností dědí některé geny;
    Mutace: se zvolenou pravděpodobností náhodně mutuj geny potomků;
    Nahrazení: některé slabé jedince z původní populace nahraď novými;
  end
until *ukončující kritérium*;

Algorithm 1: Pseudo kód algoritmu hledání strategií v posloupnosti stavů hráčů.

3.2 Neuronové sítě

Umělé neuronové sítě (anglicky *Artificial neural networks*, dále jen neuronové sítě) jsou výpočetní systémy inspirované biologickými neuronovými sítěmi a astrocyty, které dohromady tvoří mozky inteligentních organismů [13] . Takovéto systémy jsou schopny „učit se“ výkonu nějakých úkolů jejich opakovaným zkoumáním, obvykle aniž by byly naprogramovány na konkrétní pravidla těchto úkolů. Využívají se například ke klasifikaci dat nebo predikci. Podle dat z trénovací množiny, používané k učení, si samy tvoří závislosti a vyhledávají rozpoznávací znaky.

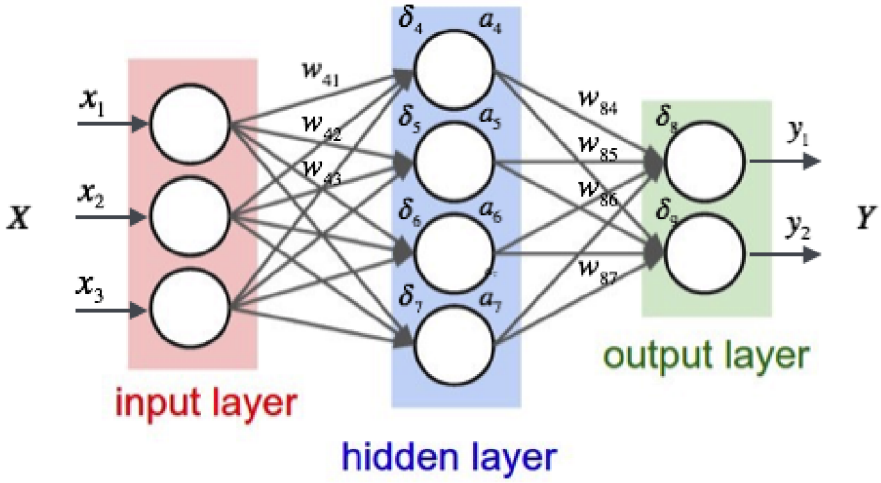
Neuronové sítě jsou složeny ze souboru propojených uzlů zvaných (umělé) neurony, které volně připomínají model neuronu biologického mozku. Jejich propojení slouží k přenosu signálu. Typickým zástupcem takovýchto neuronů je **Perceptron**, jehož model je na obrázku 3.1. Vektor hodnot $\vec{x}$ (rozšířený o implicitní jedničku) a váhový vektor $\vec{w}$ jsou vyhodnoceny bázovou funkcí a z ní se na základě aktivační funkce generuje výstupní signál y.



Obrázek 3.1: Model Perceptronu.

Neuronové sítě se liší podle topologie a s tím souvisejícími dostupnými algoritmy učení.

Topologie je dána vrstvami neuronové sítě a propojením jejich neuronů. Síť je typicky složena z jedné vstupní, jedné výstupní a žádné nebo několika skrytých vrstev o různých velikostech. Velikost vstupní a výstupní vrstvy je obvykle dána velikostí vstupních a výstupních vektorů. Propojení neuronů může být částečné nebo plné (výstup neuronu je propojen ke vstupu všech ostatních neuronů), mezi vrstvami pak může jít o jednosměrné (dopředné) nebo obousměrné, kdy vede k neuronům blíže vstupní vrstvě zpětná vazba od neuronů blíže výstupní vrstvě. Na obrázku 3.2 je příklad dopředné neuronové sítě se 2 skrytými vrstvami.



Obrázek 3.2: Dopředná neuronová síť se dvěma skrytými vrstvami.

Učení neuronových sítí je obvykle řazeno do kategorie tzv. učení s učitelem (*Supervised Learning*) [20]. Vyznačuje se nutností zařazení vstupních dat k očekávaným výstupům, často reprezentujícím jak hodnoty, tak identifikace kategorií – klasifikačních tříd. Samotný proces učení je potom upravování vah (důležitosti) vstupů jednotlivých neuronů tak, aby se při daném vstupu spíše aktivovaly jen neurony výstupní vrstvy odpovídající třídě vstupu. Velmi často používaným algoritmem je Backpropagation [14].

Ve strategických hrách se neuronové sítě začaly používat k mnoha účelům. Konvoluční neuronové sítě se využívají k analýze hry na základě její vizualizace, vícevrstvé perceptronové sítě pak například k volbě akcí agentů, analýze a klasifikaci nepřátelských sil či optimalizaci pomocí regrese. Mnoho takových použití si našly i v botech pro StarCraft [35]. Je na nich z velké části založený i AlphaStar, jeden z nejlepších inteligentních botů současnosti popsaná dále v kapitole 3.5.

3.3 Posilované učení

Posilované učení (*Reinforcement Learning*) je druh strojového učení soustředěného na volbu akcí agentů, umístěných v nějakém prostředí, které se snaží maximalizovat kumulativní odměnu za tyto akce [19]. Hlavním rozdílem od učení s učitelem (*Supervised Learning*) je, že nevyžaduje označené dvojice vstup/výstup, a suboptimální akce nemusí být vyřazeny z trénovací množiny. Místo přesného řešení se soustředí na vhodný poměr průzkumu neprozkoumaných cest a využívání získaných znalostí.

Prostředí agentů je pro posilované učení obvykle reprezentováno Markovskými rozhodovacími procesy (*Markov decision process*) [17]. Hlavním rozdílem mezi nimi a klasickými metodami je, že agenti nepotřebují mít kompletní a přesný matematický model okolí.

Pro posilované učení v počítačových hrách se často používá Q-learning [18]. Jedná se o algoritmus nevyžadující model okolí a jeho cílem je najít optimální rozhodovací politiku agenta – jakou akci zvolit za daných podmínek. Díky absenci nutnosti existence modelu je schopen nalézt stochastické řešení přechodu mezi stavy a nevyžaduje přizpůsobení stavu vnitřního model okolí. Právě díky této vlastnosti je vhodný pro hry, kde se podmínky často mění velmi rychle a mnoha způsoby (například díky akcím protivníků) [29].

Součástí procesu učení Q-learningu je agent, množina stavů S a množina akcí A proveditelných v těchto stavech. Vykonáním akce $a \in A$ se agent přesune z jednoho stavu do druhého. Na základě stavů je mu za tento přechod poskytnuta odměna ve formě numerické hodnoty. Agent se snaží o maximalizaci své odměny, tu si pamatuje – buď ve formě tabulek (obrázek 3.3) nebo učením neuronové sítě.

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	327	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Obrázek 3.3: Tabulka odměn akcí v určitých stavech pro Q-learning.

Do této tabulky se ukládá nakumulovaná hodnota odměny vzniklá váženým průměrem všech získaných odměn za tento přechod. Uplatňuje se strategie „*Good start*“, neboli „dobrý začátek“, kdy odměny získané dříve nesou větší váhu, než ty pozdější („discount rate“, neboli

sleva). Odměna je získávána funkcí reprezentovanou touto tabulkou, která se průběžně aktualizuje vzorcem 3.1 [7].

$$Q(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma maxQ'(s',a') - Q(s,a)] \tag{3.1}$$

V tomto vzorci:

- $Q(s,a)$ reprezentuje funkci odměny za akci a ve stavu s.
- $R[s,a]$ reprezentuje konkrétní odměnu za akci a ve stavu s.
- α je koeficientem učení.
- γ je míra slevy (váhy odměny).
- $maxQ'(s',a')$ je maximální očekávaná hodnota odměny za všechny možné akce v příštím stavu s', do kterého se akcí a ve stavu s dostaneme.

Posilované učení je pro strategické hry velmi efektivní metoda učení podávající dobré výsledky a poskytující možnosti nepřetržitého učení i mimo přípravnou fázi. [25] Nejvíce je využívaná v centrálně řízených multiagentních systémech [35].

3.4 Optimalizace hejnem částic a mravenčí kolonií

Algoritmy Particle swarm optimalization i Ant colony optimalization se řadí mezi optimalizační (aproximační) algoritmy, inspirované pohybem organismů v přírodě, a používané ke hledání cest. Ve strategických hrách je nutné řídit větší množství herních entit, z nichž každá musí prohledávat herní mapu k nalezení cesty ke svému cíli. Tento úkol se označuje jako „*pathfinding*“ a běžně se řeší algoritmy pro prohledávání stavového prostoru, například A*. Při větším množství zároveň hledajících objektů se však celý proces stává výpočetně náročným.

Algoritmus mravenčích kolonií (ACO) je inspirován pohybem mravenců při hledání potravy. Mravenci si značí cesty pomocí feromonů, přičemž intenzita vylučování závisí na kvalitě nebo velikosti zdroje potravy. Mravenci následují nejsilnější cesty a ještě více je zvýrazňují, naopak u méně významných tras se feromony pouze vypařují.

Algoritmus hejna částic (PSO) je inspirován pohybem ptactva. Trasa každého jedince je částečně individuální, avšak má na ni vliv pohyb zbytku hejna.

ACO se ve strategických hrách používá především pro průběžné hledání cest, kdy feromon na mapě zůstává delší dobu a nemusí se v každém snímku počítat znovu.

PSO používá mnoho botů k řízení pohybu větších skupin bojových jednotek [35]. Tento algoritmus funguje dobře i pakliže je použit v systémech založených na agentech [24].

3.5 DeepMind AphaStar

V lednu 2019 představila britská společnost DeepMind, která se dlouhodobě věnuje vývoji umělé inteligence nejen pro počítačové hry, nového bota pro hru StarCraft 2 [5]. Již dříve se jí podařilo pokořit [15] hry jako Go, Dota2 nebo Quake 3 Arena, nicméně StarCraft je díky obrovskému množství možných akcí považován za mnohem komplexnější a delší dobu se nedařilo na tuto výzvu úspěšně odpovědět.

I velmi inteligentní boti nedokázali vyhrát nad profesionálními hráči, mnohdy i když měli na své straně výhody jako zjednodušenou herní mapu, nebyli limitováni pohybem obrazovky po mapě či měli informace navíc. Tohle vše AlphaStar změnil, když se mu podařilo bez jakýchkoliv podobných výhod porazit profesionální hráče, a to i deset her v řadě.

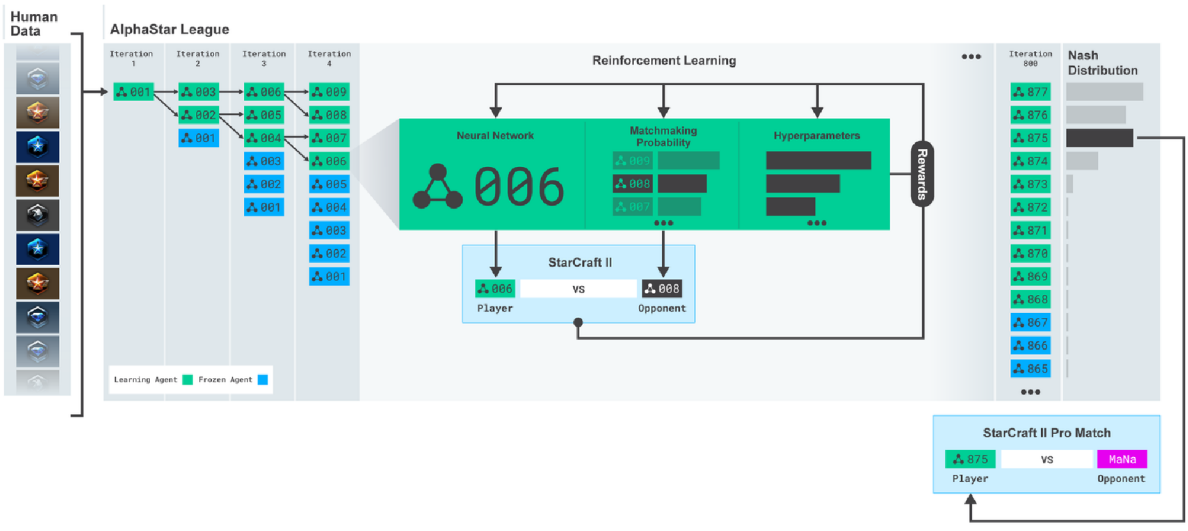
Projekt AlphaStar vznikl na platformě PySC2, což je knihovna umožňující propojení externích botů se hrou StarCraft 2 vzniklou za podpory společnosti Blizzard. Ta rovněž dodávala data k učení ve formě záznamů z her odehraných hráči z celého světa. K učení poskytla záznamy s herním časem neuvěřitelných 200 let, které AlphaStar všechny analyzoval.

Architektura AlphaStaru je založena na kombinaci:

- Posilovaného učení,
- Učení s učitelem,
- Multiagentního systému učení.

Jeho chování je určováno složitou konvoluční neuronovou sítí, která přijímá všechna data o hře bez jakéhokoliv předzpracování. Jedná se o obrovské množství informací o pozicích a stavech všech herních prvků a podobě mapy. Výstupem sítě je posloupnost akcí, které má bot ve hře provést.

Multiagentní systém učení neznamená v jeho podání soubor vzájemně komunikujících, autonomních entit se sociálními schopnostmi, ale množinu větví chování, z nichž se každá označuje pojmem agent. Když AlphaStar analyzoval všechny prvotní záznamy, jeho naučené chování se stalo jeho prvním agentem. Pokaždé, když se narazilo na nové chování protivníka nebo bylo potřeba specializovat se, byl vytvořen nový agent – nová větev chování. Každá tato větev obsahuje jiný model konvoluční neuronové sítě a jiné váhy algoritmů posilovaného učení. AlphaStar si potom podle různých parametrů hry vybírá, na základě speciálního modelu posilovaného učení, agenty, kteří jsou aktivní a volí jeho akce.



Obrázek 3.4: Zjednodušená architektura volby agentů AlphaStaru.

Zajímavostí je, že AlphaStar nevítězí nad protivníky větším množstvím akcí (jeho průměrné AMP – Actions Per Minute – je typicky nižší), ale jeho rozhodnutí jsou přesnější.

Kapitola 4

Navržený systém

Práce je věnována vypracování návrhu založeného na multiagentních systémech s podporou metod učení a plánování (či také rozhodování) agentů, kteří budou díky tomuto schopni hrát bojové strategické hry typu StarCraft.

Tento systém byl navržen na základě možností použitého multiagentního frameworku, na dostupných datech získatelných z her a použitelných pro učení agentů, a také na informacích o již existujících systémech pro hru StarCraft popsaných v kapitole 3.

Nejprve je zde popis fungování agentního systému a jeho částí rozšiřitelných o racionálnější chování či metody učení. Následuje popis a rozbor dat, která lze získat pro hru StarCraft nebo jí podobné, a které vymezují možnosti učení v takovýchto hrách.Samotný návrh systému a metod vzniklých na základě těchto zjištění je v poslední části této kapitoly.

Celý návrh je realizován přímo pro StarCraft, ale dostatečně obecně pro možnosti použití ve hrách se stejnými nebo podobnými mechanikami. Důvodem tohoto výběru je, kromě zadání práce, které k této alternativě přímo vybízí, i její popularita – lze sehnat velká množství analyzovatelných dat nebo porovnat implementaci s jinými již vytvořenými systémy.

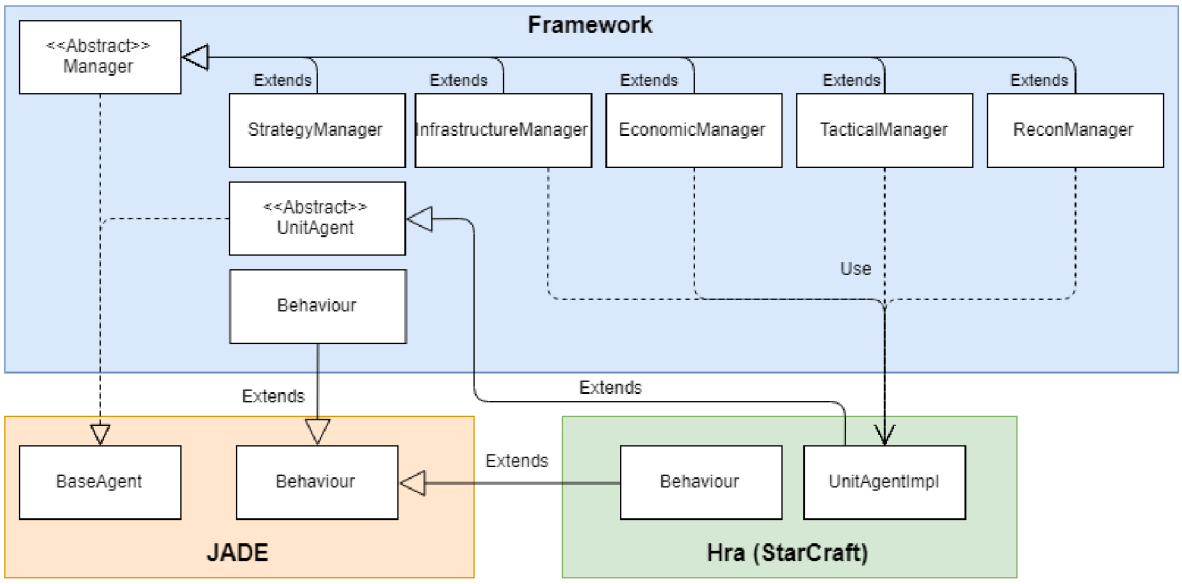
4.1 Framework pro tvorbu agentů

Souběžně s touto prací vyvíjel Bc. Lukáš Válek framework určený k tvorbě multiagentních systémů pro hraní strategických her. Ten poskytl vhodné rozhraní pro externí implementaci algoritmů rozhodování a učení agentů. Kromě toho byla možná spolupráce a domluva na rozšíření těchto možností. V této kapitole je popsán princip fungování systému vytvořeného tímto frameworkem, rozhraní pro dodání schopnosti ovládat hru a možnosti ovlivnění rozhodování agentů.

4.1.1 Multiagentní systém a možnosti rozhraní frameworku

Architektura frameworku je založena na knihovně JADE [8] a jím vytvářený systém používá hybridní přístup vytváření agentů a dělby jejich zodpovědností 2.3 Z obrázku 4.1 je patrné, že se jedná o dva hlavní druhy agentů:

- **Manažeři** – každý z nich reprezentuje jeden z makro elementů hry, jako jsou například ekonomika nebo řízení boje.
- **UnitAgenti** – jednotkoví agenti, každý z těchto agentů zastupuje jediného vojáka, dělníka či budovu.



Obrázek 4.1: Ilustrační diagram – architektura agentního systému ve frameworku.

Manažeři se pak podle zodpovědnosti specializují na tyto:

- **StrategyManager** – strategický manažer, provádí analýzu hry na nejvyšší úrovni, tedy analyzuje nepřátelské a vlastní tahy, a na jejím základě provádí dlouhodobá rozhodnutí.
- **InfrastructureManager** – infrastrukturní manažer, stará se u produkci jednotek a výstavbu budov. Zajišťuje zdroje a přerozděluje nově vytvořené entity ostatním manažerům.
- **EconomicManager** – ekonomický manažer, hlídá hráčův příjem surovin, ovládá dělníky a řídí ekonomické expanze.
- **TacticalManager** – taktický manažer, ovládá veškeré jednotky či budovy určené k boji, volí krátkodobé cíle útoku a porovnává vojenské síly hráče a soupeře.
- **ReconManager** – průzkumný manažer, obstarává nové informace o herní mapě a akcích soupeře, k tomuto účelu si žádá a následně pak ovládá jednotky k tomu učené.

Jednotkoví agenti jsou téměř zcela abstraktní entity reprezentující individuální funkční objekty konkrétní hry. Framework nezná její rozhraní pro jejich ovládání a vyžaduje implementaci uživatelem. Jedná se vlastně o jistou podobu překladu příkazů frameworku na příkazy hry.

Manažeři ovládají jim přidělené jednotkové agenty. Mohou se navzájem žádat o jejich poskytnutí či vytvoření zcela nových. Rovněž se domlouvají i o dalších svých potřebách, komunikují s jednotkovými agenty a sdílí si znalosti.

Tyto znalosti mají uloženy v tzv. „**KnowledgeBase**“, tedy databázi znalostí. Ta reprezentuje informace o mapě, stavu surovin a veškerých herních objektech všech hráčů, které umí manažeři rozpoznat. Vše včetně pozic, které jsou často aktualizovány. Každý manažer by měl mít k dispozici informace relevantní pro jeho činnost. Databáze znalostí je navržena pro snadné vyhledávání a v systému je využívána k získávání dat pro rozhodování agentů a

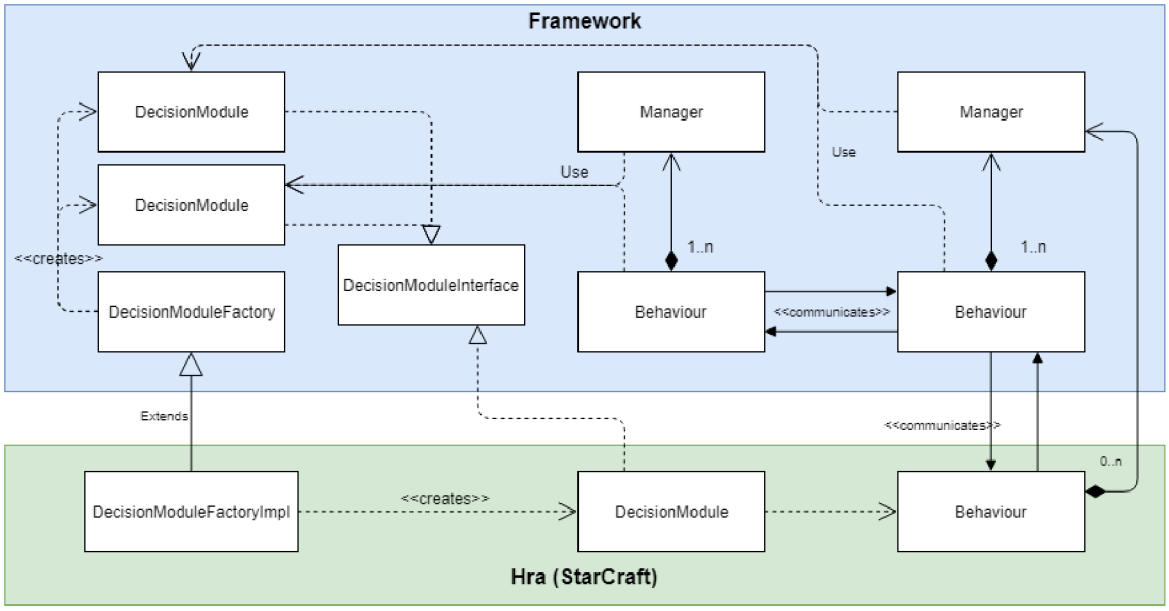
pro tvorbu jejich stavů. Manažeři udržují svůj aktuální stav ve speciálních objektech, které jsou průběžně aktualizovány. Stav každého manažera je jiný a obsahuje jiná data.

Veškerá komunikace mezi agenty je realizována pomocí tzv. modulů chování (`Behaviour`), které vychází přímo z JADE fungují na bázi zasílání ACL zpráv [6]. Framework již mnoho takových modulů obsahuje, ty obstarávají i jiné než sociální stránky agentů. Tyto moduly jsou hlavním prostředkem rozšiřitelnosti manažerů pro chování konkrétní hry bez modifikace jejich interního kódu – uživatel frameworku si vytvoří nový takovýto modul a manažerovi ho předá.

Jelikož je ve své podstatě framework obecný, nepracuje interně s žádnými konkrétními informacemi o žádné hře. Vyžaduje specifikaci akcí, herních entit a hráčových strategií. Toto rozhraní je realizováno načítáním konfiguračních souborů ve formátu JSON. Akce musí mít jedinečný identifikátor, mohou mít definovanou cenu a jejich výsledkem může být uzel technologického stromu. Uzly stromu jsou herní entity mající svou cenu a jisté závislosti – pro konstrukci uzlu musí hráč typicky vlastnit instance jiných uzlů a získáním nového se mu mohou odemknout další možnosti. Strategie jsou pak definované uzly tohoto stromu, které se systém snaží získat v předepsaném množství. Tyto strategie jsou primární reprezentací cílů manažeru v systému.

4.1.2 Rozhodování a plánování agentů

Na obrázku 4.2 je znázorněna architektura rozhodování a plánování manažerů. K tomu slouží tzv. rozhodovací moduly (`DecisionModule`). Ty jsou pro tuto práci nejdůležitější částí frameworku, právě zde se realizuje veškeré plánování a úpravy výchozího chování. Každý manažer potřebuje pro svou funkčnost rozhodovací modul s jemu určeným rozhraním (každý druh manažera definuje vlastní rozhraní). Tato rozhraní ovlivnila návrh celého systému této práce definováním možností učení.



Obrázek 4.2: Architektura modulů chování a rozhodování v multiagentním frameworku.

Existující metody z rozhraní rozhodovacích modulů jsou využívány přímo manažery, výjimečně jejich moduly chování. Pakliže chce uživatel vytvořit nový druh rozhodnutí, musí

implementovat i nový modul chování, který bude tyto nové metody volat, a přiřadit ho správnému manažerovi. Tímto jsou zajištěny poměrně velké možností rozšíření chování a plánování manažerů.

Fungování rozhodovacích modulů je v zásadě řešení problémů představených fungováním manažera nebo odpovídání na otázky o průběhu či podobě hry:

- Rozhodnutí o vykonání či nevykonání akce nebo úlohy.
- Volba nejvhodnější reakce nebo akce.
- Hledání ideálního času pro akci.
- Optimalizování parametrů akcí nebo predikce nejvhodnější hodnoty (například počet jednotek).
- Klasifikace herních prvků (obecných i konkrétních pro danou hru).

Framework obsahuje výchozí chování pro některé z těchto metod, se kterým je schopen StarCraft ovládat, nicméně nedává velký smysl. Pro konkrétní hru je zde mnoho prostoru pro zlepšení. Některé tyto rozhodovací úlohy byly kvůlu menší strategické hodnotě pro výsledek hry vyřešeny pouze jednoduchým nebo výchozím chování, a to do takové míry, aby pro StarCraft nebo podobné hry fungovaly:

- Volba cíle taktického manažera – odhadem poměru sil armád bota a jeho nepřítele se rozhoduje o útoku či obraně základny. Pakliže jsou síly vyrovnané nebo je bot silnější, útočí. Odhad sil je realizován součtem populačních bodů jednotlivých armád v jeho databázi znalostí, závisí tedy na vyrovnanosti hry samotné. Tento model je pro práci dostačující.
- Volba cíle útoku taktického manažera – pakliže bot útočí na nepřítele, náhodně si vybírá nepřátelskou budovu či základnu ke zničení. Ty hledá ve své databázi znalostí.
- Zaslání žádosti o nové jednotky taktickým manažerem – žádá se téměř stále, o to intenzivněji pakliže má nepřítel vojenskou výhodu. Bot žádá o všechny jednotky dostupné skrze splněnou část technologického stromu a v počtu dostupných budov schopných výroby konkrétních jednotek, opět získaných z databáze znalostí.
- Volba cíle prohledávání mapy průzkumným manažerem – prochází počáteční pozice hráčů v náhodném pořadí. Tyto pozice jsou získány nástrojem BWMA popsaným v kapitole o implementaci (5.4). Zápasy jeden na jednoho se typicky hrají v mapách pro dva až čtyři hráče, kde nemá díky náhodnosti pozicí volba jiného než náhodného postupu velký strategický význam.
- Doplnění herních objektů poskytujících populační body v infrastrukturním manažerovi – lze použít ke korekci plánovaných akcí tohoto manažera, pakliže současné nároky na populační body překročí jejich aktuální počet. Tento proces je v infrastrukturním manažerovi řešen důmyslněji, zde byla pouze implementována pojistka nutící bota postavit příslušnou budovu v případě nouze.
- Žádost o postavení budov expanzí ekonomického manažera – opět vyřešeno důmyslnějším způsobem, avšak použito jako ošetření stavů, kdy se o expanzi žádalo příliš brzy pomocí hlídání času.

Na jiné se tato práce soustředila mnohem více a budou rozebrány detailněji v následujících kapitolách o návrhu, implementaci a jejich efektivita podléhala testování:

- Žádost o vytvoření nových dělníků pro ekonomického manažera – množství dělníků hráče ve hře je informací získatelnou z dostupných záznamů vítězných her a jejich počet má velký dopad na výsledek hry. Dělníci mají své surovinové nároky a reprezentují investici s velkou návratností, avšak pouze v jisté míře. Pakliže je postaveno příliš mnoho, a to moc brzy, bot plýtvá dostupnými zdroji a dokonce ani nemá pro dělníky žádné volné sloty k získávání zdrojů (z jednoho zdroje může těžit pouze jeden). Pro určení vhodného počtu dělníků, a následné vyvození žádosti o jejich vytvoření, je použita dopředu natrénovaná neuronová síť.

- Klasifikace nepřátelské strategie strategickým manažerem – rozpoznání jedné z předem definovaných strategií pro konkrétní hru s využitím informací o nepřátelských jednotkách a budovách z databáze znalostí. Tato informace má pro bota značnou strategickou hodnotu a znamená jeho schopnost reakce na tahy oponenta. Ke klasifikace se používají neuronové sítě.

- Volba vlastní strategie proti použité strategii oponenta ve strategickém manažerovi – důsledek předchozího bodu, se kterým společně tvoří hlavní reaktivní část systému. Na základě klasifikované strategii oponenta a znalostí předešlé efektivity vlastních strategií proti ní jsou zvoleny cíle pro zbytek systému, především pak pro infrastrukturního manažera (jaké budovy a jednotky jsou k vítězství potřeba). Používá se neuronová síť.

- Volba herního objektu k produkci v infrastrukturním manažerovi – na základě vlastněných uzlů technologického stromu, cílů reprezentovaných zvolenou strategií, žádostech od ostatních agentů a mnoha dalších parametrů se vybírají následující akce. Tento proces je asi nejsložitější v celém frameworku a má největší dlouhodobé dopady, přímý vliv na výsledek hry. Tato volba je v systému realizována na základě modelu posilovaného učení.

4.2 Data pro učení a analýzu

Pro jakýkoliv systém využívající strojového učení je potřebná možnost získání trénovacích a testovacích dat. V případě počítačových her se však většinou jedná o zcela uzavřené systémy, do kterých nelze sběr potřebných dat efektivně doplnit. Výjimkou jsou například open-source projekty jako OpenRA [11], které nejsou příliš početné.

Mnoho her typu StarCraft a podobných však již jistý takový systém sběru dat obsahuje, jen je na první pohled ukrytý, těžce analyzovatelný a nemívá žádné možnosti rozšíření. Jedná se o systém záznamů her, tzv. „replays“. Ty se ukládají, většinou v binární podobě, k pozdějšímu shlédnutí hráči.

Jelikož tyto záznamy musí být schopné kompletně replikovat celý průběh hry, musí obsahovat záznamy o všech herních událostech a akcích hráčů. Problémem je pouze získání správných informací ve správném formátu – ideálně takovém, aby byla použitelná v kombinaci s multiagentním frameworkem a korespondovala se stavy manažerů.

4.2.1 Existující nástroje pro analýzu, výstupy

Pro analýzu záznamů her StarCraftu byly již vytvořeny nástroje, které jsou schopny extrahovat a převést mnoho informací o průběhu hry z binárních dat do textové podoby. Mezi takové nástroje řadíme například **ScExtractor** [28], **bwrepanalysis** [2] nebo **bwrepdump** [3].

Pro usnadnění získávání dat z binární podoby byl použit výstup analyzátoru **bwrepdump**. Tento nástroj je vlastně botem využívajícím otevřeného systému StarCraftu postavený na knihovně BWAPI, která je podrobně popsána v kapitole o implementaci 5.2. Místo reakcí na herní události z vlastních her je však tento bot postavený jen na principu spouštění záznamů cizích her a výpis informací. S předem určeným intervalem vzorkuje záznam hry (s formátem REP (`.rep`)) a data ukládá do tří druhů výstupních souborů:

- **RGD** (`.rgd`) – vzorkované informace o stavech příjmu surovin (včetně populačních bodů a včetně součtu od začátku hry) a všech herních událostech, jako je vytvoření, objevení a zničení jednotek (odpovídajícím metodám popsaných v kapitole o implementaci 5.2) včetně času, pozice, typů jednotek apod.
- **ROD** (`.rod`) – všechny příkazy zadané hráčem skrze rozhraní hry (úkoly přiřazené herním entitám, pohyby herní obrazovky po mapě atd.) včetně jejich času a pozice.
- **RLD** (`.rld`) – pohyby jednotek mezi regiony mapy (které nástroj sám vytvoří a v hlavičce nadefinuje).

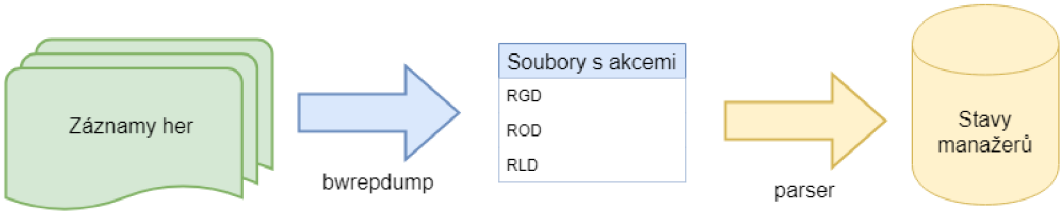
Vzhledem k zaměření práce nebyla data z ROD a RLD souborů nikterak zajímavá, jelikož jsou zaměřená na micromanagement a ovládání jednotek, které byly v navrženém systému řešeny jinak, nikoliv na plánování, učení a chování agentů. Z RGD souborů však práce těží ve velké míře.

4.2.2 Automatická analýza a konverze dat

Pro převod takto získaných dat do podoby použitelné multiagentním frameworkem byl navržen samostatný systém hromadné analýzy záznamů. Data v požadovaném formátu není možné díky specifikám multiagentního frameworku a jeho metodám učení získat přímo.

Tento systém ovlivnil volby i implementace způsobů a metod učení, a to díky vymezení dat, která lze k tomuto účelu získat. Vysvětlení činnosti přímo ovlivňující strukturu a původ dat je rovněž základem pro pochopení možnosti experimentů a změn jejich vstupních parametrů.

Jak bylo zmíněno v kapitole 4.1.1, ve hře jsou botovi předem nadefinovány strategické cíle reprezentující výčet herních entit provázaných s jejich individuálním množstvím, kterých se pak snaží manažeři dosáhnout.



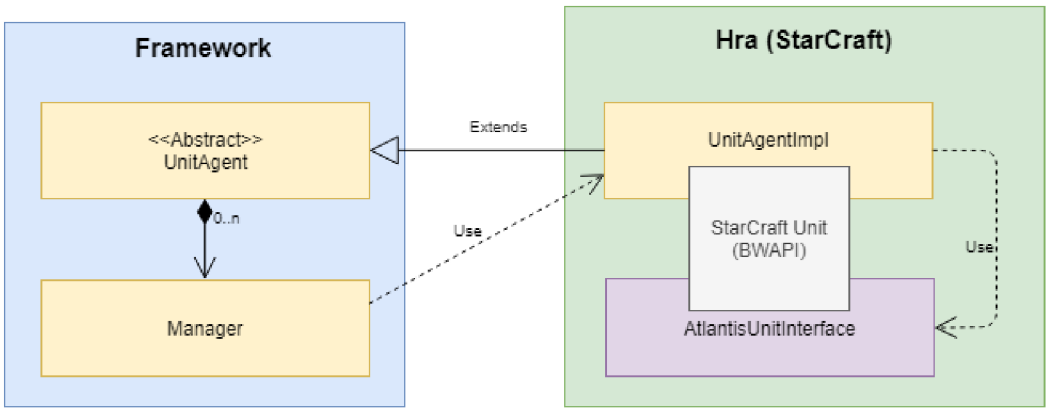
Obrázek 4.3: Proces získávání dat ze záznamů her.

4.3 Návrh micromanagementu

Z příkladů předešlé kapitoly, části 4.1.2, je patrné, že většina rozhodnutí manažerů spadá do tzv. makra, tj. plánování z dlouhodobého hlediska a nebo pro mnoho entit najednou. Tato práce se zabývá především makrem. Ovšem systém by se neobešel bez efektivního způsobu ovládání jednotek. Především těch určených k boji či průzkumu.

Pro tento účel je využita Atlantis [1], knihovna využívající konečných automatů k řízení bota pro StarCraft. Obsahuje systém dělby zodpovědností velice podobný tomu používaného multiagentním frameworkem z předešlé kapitoly 4.1, tedy systém manažerů. Zde se však nejedná o autonomní a komunikující agenty, pouze o statické automaty neposkytující žádné možnosti zásahu do jejich rozhodování či plánování akcí bez přepsání velkého množství zdrojového kódu.

Vyžití Atlantis si žádalo navržení vhodného rozhraní, jelikož se musí o kontrolu nad jednotkami dělit s agentním frameworkem. Znázorněno na obrázku 4.4, manažer dává příkazy jednotkám pod svou kontrolou, ty je vykonávají skrze konkrétní implementaci pro StarCraft. Ta si uchovává referenci objekt herní jednotky, která je zároveň v potřebné podobě uložená i v Atlantis. Implementace jednotkového agenta pak může ve svých metodách volat korespondující metody Atlantis.



Obrázek 4.4: Rozhraní mezi agentním frameworkem, herním botem a Atlantis.

4.4 Klasifikace nepřátelských strategií

Strategický manažer během celé hry analyzuje všechny herní entity (uzly strategického stromu) vlastněné jak hráčem reprezentovaným frameworkem, tak nepřítelem. K odhadnutí záměrů právě tohoto oponenta se v navrženém systému přistupuje jako ke klasifikačnímu problému, který je řešen vícevrstvými neuronovými sítěmi.

Získávání klasifikačních tříd je v takovémto případě obtížné. Jako řešení se nabízí například shlukování, avšak tento přístup s sebou nese úskalí a vyžaduje optimalizace, na které nemusí být během hry čas:

- Volba množství výsledných shluků – nutnost časté změny klasifikačního modelu a počtu výstupů, opětovné učení.
- Dimenzionalita dat – shluky přes jiné množství jinak korelujících vstupních dat znamená jiný počet vstupů sítí.

- Různá strategická hodnota různých dimenzí – například (od nejnižší po nejvyšší) jednotek, budov a vylepšení. Taktéž častá změna korelací.

V navrženém systému je tento problém řešen výchozím nastavením multiagentního frameworku – ruční definicí běžně používaných strategií získaných online [9]. Výstupem klasifikace je pak jedinečný identifikátor nejpravděpodobnější strategie. Lze říci, že touto ruční definicí tříd může dojít ke snížení jejich přesnosti, na druhou stranu je nesporná výhoda jednoduchosti procesu a strategie může definovat i dobrý hráč, který v nich vidí smysl ihned nebo je píše s nějakým konkrétním úmyslem.

Díky relativně malému množství kombinací jednotek a budov, a to hlavně ze začátku hry, je přesnost klasifikace potenciálně nízká, zatímco výběr a zpracování vhodných vstupních dat jsou obtížné a zdlouhavé. Maximalizace zisku v této části systému však může potenciálně ovlivnit dva další modely, její důležitost je proto obrovská. Pravou výzvou je zde volba učících dat.

4.4.1 Architektura modelu

Pro klasifikaci bylo zvoleno použití vícevrstvých neuronových sítí 3.2.

Vstupním vektorem pevné délky jsou normalizované počty jednotek a budov, v případě rozšíření i úrovně vylepšení. Hodnoty jsou v intervalu $[0.0, 1.0]$. Ve výchozí konfiguraci se jedná o počty či úrovně všech entit příslušejících oponentově rase, bez ohledu na jejich korelaci či vlivu na výsledek, avšak s možností redukce dimenzionality k získání přesnosti.

Výstupním vektorem jsou míry příslušnosti vstupního vektoru k jednotlivým klasifikačním třídám. Velikost výstupního vektoru je rovna počtu strategií konkrétní rasy, které jsou definovány v konfiguračním souboru. Hodnoty jsou opět v intervalu $[0.0, 1.0]$. Výsledná třída je určena jako index vektoru s maximální hodnotou.

Vstupní a výstupní vrstvy mají velikost totožnou se vstupním a výstupním vektorem. Navržená neuronová síť má jednu skrytou vrstvu o velikosti vstupního vektoru. Jedná se o síť dopřednou, složenou z perceptronů. K učení je využito metody Backpropagation. Nalezení vhodné velikosti skryté vrstvy a koeficient učení jsou součástí experimentů.

Využití sítě s architekturou závislou na počtu strategií definovatelnou uživatelem a možnou změnou velikosti vstupního vektoru znamená, že s úpravou těchto parametrů se síť stává nepoužitelnou a je potřeba nový model znovu přenastavit a znovu naučit.

Rovněž je nutné vytvořit nový model pro každou rasu nepřítele kvůli odlišnosti typů i počtů použitých jednotek a vypsaných strategií (tento fakt však nemusí nutně znamenat nevýhodu pro výsledky, dedikované sítě mohou klasifikovat přesněji).

4.5 Volba vlastní strategie

Strategický manažer po klasifikaci oponentovy strategie volí vhodnou reakci v podobě strategie vlastní, takové, která se dříve ukázala jako vítěznou. Tu následně nařizuje infrastrukturnímu manažerovi, který ji plní stavbou vhodných budov a jednotek.

K tomuto úkolu se systém opět staví jako ke klasifikačnímu problému, avšak již od začátku se předpokládá nízká přesnost této části. Závisí na datech o strategiích použitých oběma hráči za celou hru získaných ze záznamů her a tato data jsou vždy pouze odhadem, nikdy přesnou informací o úmyslu hráče. Jelikož nelze přesně odhalit ani přesné dvojice navazujících tříd, vstup i výstup má několik aktivních prvků a u modelu doufáme alespoň ve vytvoření silnějších vazem mezi některými vstupními a výstupními třídami. Výstup této

části má velký vliv na rozhodování infrastrukturního manažera a bez něj pozbývá smyslu klasifikace oponentovy strategie popsaná v předešlé kapitole 4.4.

4.5.1 Architektura modelu

Pro klasifikaci se používá neuronových sítí složených pouze ze vstupní a výstupní vrstvy. Jedná se o síť perceptronů, k učení je využita metoda Backpropagation.

Velikosti vstupní i výstupní vrstvy je ekvivalentní počtu definovaných strategií pro rasu oponenta a hráče. Při učení je každá použitá strategie označena na vstupu hodnotou 1.0, nepoužitá hodnotou 0.0. Výstupem je pak vektor příslušností k jednotlivým třídám reprezentujícím strategie hráče, za výsledek se volí index s maximální hodnotou.

Tato architektura vyžaduje nový model při změně známých strategií a také dedikovaný model pro každou dvojici herních ras.

4.6 Volba budov, jednotek a vylepšení k produkci

Infrastrukturní manažer z multiagentního frameworku obsahuje rozhodovací modul, který mu vybírá akce k vykonání. Ty reprezentují budovy ke konstrukci nebo jednotky k výcviku. S jeho pomocí lze optimalizovat jejich výběr a měnit pořadí konstrukce, a to podle velkého množství parametrů. Tento proces je bez debat nejsložitějším v každém botovi a vhodnému řešení se věnuje největší množství prací [35].

Výstupem tohoto rozhodovacího modulu je identifikátor akce k provedení. Má dva vstupy. Prvním je množina akcí proveditelná v aktuálním stavu, vymezená podmínkami a závislostmi definovanými v technologickém stromu hry. Druhým je pak současný stav manažera obsahující následující data, která má uložena ve své databázi znalostí (nebo je možné je snadno vypočítat):

- Počet základen (expanzí).
- Poměr počtu dělníků a základen.
- Počet tzv. „facilities“ – budov schopných produkce jednotek.
- Množství zbývajících volných míst pro jednotky, pakliže hra používá systém populačních bodů. V případě StarCraftu lze tento princip uplatnit.
- Současně zvolenou strategii pro konstrukci (volenou strategickým manažerem pomocí klasifikace z předešlých kapitol).
- Výčet všech vlastněných uzlů technologického stromu uložených jako seznam objektů obsahujících identifikaci uzlu a následně známý počet takovýchto uzlů.

Ve výchozím chování frameworku tento modul pouze posílá seznam akcí předem definovaný ve statické frontě (který je nutné pro konkrétní hru definovat, pokud není implementován jiný, praktičtější systém). Tímto způsobem je možné hru hrát, ale není nikterak reaktivní a závisí pouze na zkušenostech uživatele frameworku.

Pro zlepšení byl navržen systém založený na posilovaném učení, v základní podobě algoritmem Q-learning 3.3. Posilované učení funguje na základě stavů, prováděných akcí a odměn za tyto akce.

V každém možném stavu má bot k dispozici pouze omezený počet akcí. Každá je jinak dobrá a jinak vhodná. Po provedení některé z nich se ocitne ve stavu jiném. Pokud ho tato akce přivedla blíže k cíli reprezentovaného zvolenou strategií, odměna musí být větší, pokud ne, tak nízká či záporná.

Pro efektivní použití tohoto systému učení a rozhodování je potřeba vyřešit několik aspektů.

4.6.1 Mohutnost stavového prostoru

Při pohledu na data uchovávaná ve stavu infrastrukturního manažera a jejich množin hodnot lze velice rychle usoudit, že může vzniknout obrovské množství kombinací. Vezměme si například stavy během boje, kdy se může během krátkého časového intervalu několikrát změnit množství existujících hráčových jednotek různých druhů, či se změní počet dělníků v době, kdy jich má hráč již velké množství (ve StarCraftu běžně až 50 a více).

Velká množina stavů je pro posilované učení potenciálně nechtěná, hlavně pokud je způsobená daty s menší důležitostí pro použitý model. Vazby mezi těmito stavy vzniklé odměňováním akcí jsou slabší, což přináší složitější rozhodování a méně efektivní učení. Příliš mohutná množina stavů rovněž implikuje složitější vyhledávání a větší paměťové nároky (před vhodným zpracováním dat k učení bylo běžné setkat se se stovkami tisíc až několika miliony stavů). Mnohem častěji vznikají i takové stavy, se kterými se ještě rozhodovací modul nikdy nesetkal, jev taktéž vedoucí ke snížení efektivity učení.

S tímto problémem se systém potýká dvěma způsoby:

1. Diskretizací atributů – převodem kvantitativních dat na diskrétní intervaly.
2. Vynecháváním méně relevantních dat.

Agentní framework poskytuje podporu diskretizace dat předpřipravenou třídou reprezentující úrovně hodnot (`VERY_LOW`, `LOW`, `MEDIUM`, `HIGH`, `VERY_HIGH`, `ULTRA_HIGH`) podporovanou ve svých metodách. Diskretizace se používá k redukci oboru hodnot atributů stavu. Konkrétně jde o atributy jako poměr dělníků a základen, množství facilities a zbývající úroveň populačních bodů.

Přináší však i další obrovskou výhodu – nerovnoměrným rozdělením intervalu lze změnit strategickou hodnotu jednotlivých úrovní. Pro výsledek hry je například mnohem podstatnější postavení jisté budovy určené k produkci speciálního typu jednotek, pokud nevlastní žádnou, než když vlastní například 6. Obdobné je to s úrovní populačních bodů – pokud hráči již téměř dochází, úrovně se mění rychleji, než když mu jich zbývá poměrně hodně. Prudké změny některých těchto úrovní z vysokých na nízké pak značí například postavení nové základny, kdy se prudce sníží poměr dělníků, nebo při boji, kdy se rychle uvolní velké množství populačních bodů. To potenciálně umožní rychlejší reakce a okamžité zahájení produkce potřebnějších věcí.

Za vynechávání dat se v tomto návrhu nepovažuje ignorování některých atributů stavu, ale spíše jejich částí. Příkladem je například ignorováním jednotek nebo některých budov v seznamu vlastněných uzlů. Tímto způsobem se při učení lépe vytváří závislosti mezi strategicky důležitějšími uzly technologického stromu. Redukce stavového prostoru se dosahuje zmenšením počtu stavů, kdy hráč vlastní stále stejné budovy, ale pouze produkuje nebo přichází o jednotky.

Konkrétní funkce pro diskretizaci hodnot a výčty ignorovaných dat jsou předmětem optimalizací učení, v kapitole o experimentech jsou uvedeny srovnání testovaných alternativ a potenciálně nejlepší kombinace.

4.6.2 Funkce pro výpočet odměn

Výpočet odměny za provedenou akci v závislosti na současném nebo budoucím stavu je náročný úkol, jelikož do vzorce zasahuje velké množství vstupních parametrů. Je to část práce pravděpodobně nejvíce náchylná na změnu hodnot a další úpravy, proto se s ní nejvíce experimentovalo. Zde jsou uvedeny cíle pro implementaci co možná nejvýhodnějšího způsobu odměňování – aspekty, které by měly být hodnoceny jak pro konkrétní akce, tak obecně.

- Produkce dělníků – lépe ohodnocena při nízkém poměru k základnám, hodnocena záporně při dostatku.
- Produkce jednotek – lépe ohodnocena při vysokém stavu volných populačních bodů.
- Stavba jakéhokoliv uzlu podle jeho proveditelnosti.
- Stavba uzlu podle jeho relevance k současně zvolené strategii.
- Produkce čehokoliv jiného, než budov poskytujících populační body, když je jejich úroveň nízká.
- Provedení jakékoliv akce podle výsledku hry – vítězné tahy mají větší hodnotu.
- Provedení akcí v závislosti na délce hry – dlouhá hra s těsnou prohrou je hodnocena jinak, než rychlé vítězství.
- Postavení uzlu umožňujícího vytváření jednotek, pokud je takových málo či nejsou zatím žádné.
- Nejlépe hodnocené akce musí být ty, které vedou ke splnění aktuálního cíle reprezentovaného zvolenou strategií.

Multiagentní framework usnadňuje výběr a hodnocení akcí několika příhodnými způsoby, díky kterým lze hodnotící funkci poněkud zjednodušit:

- Nedovoluje výběr akcí, které nejsou vzhledem k aktuálnímu stavu technologického stromu splnitelné.
- Nedovoluje stavbu dělníků či bojových jednotek, pakliže o ně nikdo nežádal.
- Nedovoluje stavbu jednotek při příliš nízké hladině populačních bodů (nižší, než je cena pro danou jednotku).
- Nedovoluje stavby ekonomických expanzí, pokud o ně nežádá ekonomický manažer.

Kapitola 5

Implementace

V rámci tohoto projektu bylo nutné implementovat herní umělou inteligenci (bota) schopnou hrát hru podobnou StarCraftu založenou na multiagentním systému se schopnostmi učení a volbou cílů. Zvolena byla varianta přímo bota pro StarCraft Brood War, realizovaného pomocí knihovny BWAPI.

V této kapitole jsou popsány základní prvky implementace, jako jsou knihovny pro ovládání hry, multiagentní podporu i učení. Jednotlivé kapitoly obsahují jak popis těchto komponent, tak většinou i hlavní třídy včetně popisu zodpovědností. Díky dostupným knihovnám není implementace rozsáhlá.

Na úvod je však popsána implementace a způsob fungování nástroje určeného k analýze a zpracování dat dostupných ze záznamů her StarCraftu. Bez tohoto nástroje by byla implementace algoritmů učení a rozhodování nepoužitelná.

V dalších kapitolách jsou pak popsány konkrétní algoritmy a třídy reprezentující výsledné chování a které používají získaná data jako vstupy.

5.1 Replay Data Converter

Tento nástroj je implementován, podle specifikací z kapitoly 4.2, za účelem analýzy a extrakce dat ze záznamů her v podobě:

- Přijímané neuronovými sítěmi – vstupní i výstupní vektory typu `double[]` stavu jednotek a použitých strategií.
- Posloupnosti stavů agentů a prováděných akcí – hlavně infrastrukturního manažera, viz. 4.6.
- Stavy surovin a dělníků závislé na čase.
- Dodatečné informace o hře – použité strategie, výsledek a délka hry.

Vývoj tohoto nástroje zabral v několika iteracích větší množství času a změny jeho výstupu byly používány k přímému ovlivnění kvality učení. Tento nástroj lze díky tomu označit za součást systému učení a práce na něm měla dopad na celou jeho podobu.

Tento nástroj nelze použít pro jinou hru, pokud nemá podobný formát pro ukládání záznamů her, avšak po úpravě implementace související se vstupy existují možnosti znovupoužitelnosti.

5.1.1 Formát dat

K popisu získávání informací je potřeba vysvětlit formát dat souboru RGD používaného tímto nástrojem a získaného pomocí bwrepdump (popsáno v návrhu systému v kapitole 4.2). Na začátku souboru se nachází vždy hlavička s informacemi o herní mapě, hráčích, jejich rasách, jménech apod. Hlavička je relevantní jen pro zjištění identifikátoru hráče z řádků ve formátu: `$PLAYER_ID, $PLAYER_NAME, $START_LOC`.

Po hlavičce následuje chronologický seznam prvků ze vzorkování stavů surovin a zápis všech herních událostí. Podrobný popis lze nalézt na stránce nástroje bwrepdump [3], projekt však nemá zcela úplnou dokumentaci a informace je potřeba dohledávat v implementaci. Každý prvek je na samostatném řádku a začíná informací o aktuálním herním čase (reprezentovaném pomocí počtu snímků od začátku hry) a identifikátorem hráče. Z tohoto seznamu jsou pro práci důležité následující elementy:

- `R,$MINERALS,$GAS,$GATHERED_MINERALS,$GATHERED_GAS,$SUPPLY_USED,$SUPPLY_TOTAL` – reprezentuje vzorek stavu surovin hráče v čase.
- `Created,$UNIT_ID,$UNIT_TYPE,($POS_X,$POS_Y),$CDR_HASH,$REGION_HASH` – informace o vytvoření zcela nové jednotky hráčem.
- `Morph,$UNIT_ID,$UNIT_TYPE,($POS_X,$POS_Y)` – informace o vytvoření jednotky transformací jiné jednotky.
- `Discovered,$UNIT_ID,$UNIT_TYPE` – informace o objevení jednotky jednoho hráče druhým.
- `Destroyed,$UNIT_ID,$UNIT_TYPE,($POS_X,$POS_Y)` – událost oznamující zničení jednotky.

5.1.2 Získávání dat

Soubory RGD jsou čteny řádek po řádku a z každého se snadno dedukuje typ informace. Omezením nástroje je vyčítání dat pouze ze záznamů her, v nichž jsou přítomni pouze dva hráči a jeden z nich je Protoss. Až na záznam o stavu surovin je mnoho informací z událostí spíše doplňkových a v zásadě jsou nejvíce relevantní jen tyto čtyři informace:

1. Čas.
2. Identifikátor hráče.
3. Typ události.
4. Typ jednotky reprezentovaný hodnotou proměnné `UNIT_TYPE`.

Je nutné poznamenat, že hodnoty `UNIT_TYPE` nejsou kompatibilní s konstantami třídy `UnitType` z BWAPI. Nejen, že jsou tam znaky „_“ zaměněny za mezery, dokonce některé entity mají i jiné jméno (`Protoss_High_Templar` nahrazen pouze za `Protoss_Templar`) či mají některá slova začínající malými písmeny i proti konvenci zbytku. Tento fakt zanášel během vývoje mnohé chyby do celého procesu a vyžadoval ruční přepis mnoha konstant.

Pro každého hráče se udržují mapy aktuálních budov, jednotek a vylepšení. V těch se inkrementují hodnoty počtů těchto entit při `Created` nebo `Morph`. Dekrementují se v případě `Destroyed`. Jelikož je sběr těchto dat závislý na herních událostech a ne vzorkování, nesbírá

se v závislosti na herním čase. V případě změny na straně hráče s rasou Protoss se vygeneruje při každé z těchto událostí záznam aktuálního součtu všech jeho herních objektů, který se po průchodu celým souborem používá pro generování historie stavů pro posilované učení.

Pro hráče reprezentujícího Zerga nebo Terrana se pak ještě navíc hlídají veškeré objekty odhalené hráčem s rasou Protoss. Tyto informace se ukládají do mapy odhalených jednotek, kde se inkrementují událostí `Discovered` a opět dekrementují při `Destroyed`. Jedná se obecně o podmnožinu všech objektů vlastněných nepřítelem, která se využívá při dedukci jeho strategií.

Rovněž je při každém novém záznamu o počtech surovin aktualizována hodnota pro každého hráče uložená v příslušném vektoru.

5.1.3 Výstup stavů a akcí infrastrukturního manažera

Jak bylo zmíněno v předchozí kapitole, pro hráče s rasou Protoss jsou ukládány informace o všech jeho aktuálních herních objektech, při změně v těchto datech je spuštěno generování nového záznamu odpovídajícímu formátu stavu infrastrukturního manažera agentního systému. Při tomto procesu se pro záznam:

1. Je vytvořena kopie mapy jeho herních objektů.
2. je vytvořena se kopie mapy jeho vyzkoumaných technologií.
3. Je spočten počet základen hráče sumou staveb typu `Protoss_Nexus`.
4. Je spočten počet staveb schopných výcviku jednotek – `Protoss_Gateway`, `Protoss_Stargate` a `Protoss_Robotics_Facility`.
5. Je vypočítán poměr dělníků (počet jednotek typu `Protoss_Probe`) a již získaného počtu základen vzorcem $\frac{workers}{max(1,bases)}$.
6. Je získána hodnota zbývajících populačních bodů z aktuálního stavu surovin hráče.
7. Vytvoří nový záznam o příslušnosti ke hráčem zvolené strategii, který je prozatím nastaven na výchozí hodnotu -1.
8. K záznamu je připojen identifikátor akce či herní události, která vedla k novému stavu.

Zde je však v implementaci jeden rozdíl. Většina z těchto hodnot (kromě počtů herních objektů, úrovní technologií a počtu základen) jsou ve frameworku a v botovi používány ve formátu `ValueLevel`, nikoliv jako numerická hodnota. Nástroj však na výstup dává čísla. Důvodem je kompatibilita přepočtu, kdy by se vzorec pro získání úrovně mohl na více místech lišit, je proto pouze v botovi a ten hodnoty transformuje po načtení.

Po průchodu všemi daty záznamu hry se všechny vytvořené stavy převedou do výstupní podoby. Tato část generuje dva soubory:

- Záznam stavů (soubor `.state`) – JSON pole jednotlivých stavů o velikosti n.
- Posloupnost identifikátorů provedených akcí mezi stavy o délce $n - 1$.

Při učení jsou pak tyto soubory načteny, jejich hodnoty převedeny na úrovně stavů a podle identifikátorů vyhledány příslušné akce. Správná posloupnost je řízena následujícím vzorcem: při stavu n je provedena akce ze řádku n k získání stavu $n + 1$.

Jelikož můžou být během hry některé herní objekty zničeny, je nutné generování nových stavů i tehdy. V takovém případě však k novému stavu nevedla žádná hráčova akce a posilované učení potřebuje akci specifikovat. Z tohoto důvody byla vytvořena speciální akce s identifikátorem 666, která se ve výstupu nástroje objevuje, avšak v učení se na ni nereaguje.

5.1.4 Dedukce strategií

V nástroji byly vytvořeny dva velmi podobné procesy k dedukci (či také klasifikaci) použité strategie – jeden pro hráče a jeden pro jeho protivníka. V obou jsou využity podobné algoritmy porovnávání strategií uložených ve formátu JSON s počty herních entit hráčů, viz. algoritmus 2. Tento algoritmus se napřed pokusí najít ve stavech všechny možné strategie zcela přesně a až pokud tohle nelze zcela splnit, tak s jistou podobností. Při nalezení shody jsou všechny doposud nezařazené stavy označené jako vedoucí k této strategii a tato strategie se dále nehledá – od tohoto bodu se hledají jen další, navazující. Změnou důležitosti strategií s více či méně součástmi lze ovlivnit množství dat k učení a jejich priority.

Je možné, že pro některé stavy není nalezena strategie žádná. Zde se hráčovým stavům nastaví neutrální strategie použitelná pro obecné chování, zatímco takováto oponentova data se zcela odstraní z výstupu.

```
Data: Strategie pro hráčovu nebo oponentovu rasu
Data: Všechny stavy
Data: Zařazené stavy
Data: Nezařazené stavy
Data: Použité strategie
foreach stav ze všech stavů do
    přidej stav do nezařazených;
    nastav zvolenou strategii na -1;
    foreach strategie ze známých strategií která není mezi použitými do
        if jednotky ve stavu odpovídají strategii s podobností 1.0 then
            if zvolená strategie je -1 OR strategie má větší počet požadavků než
              zvolená strategie then
                nastav zvolenou strategii na aktuálně porovnávanou;
            end
        end
    end
    if zvolená strategie není -1 then
        všem stavům v nezařazených změň ID strategie na zvolenou strategii;
        všechny nezařazené stavy přemísti do zařazených;
        přidej strategii do seznamu použitých;
    end
end
sniž pořadovanou úroveň podobnosti;
opakuj cyklus, ale tentokrát vyhledávej s nezařazených stavech místo všech stavů;
```

Algorithm 2: Pseudo kód algoritmu hledání strategií v posloupnosti stavů hráčů.

Po dokončení tohoto vyhledávání se všem stavům hráče reprezentující hodnoty pro infrastrukturního manažera nastaví příslušné ID strategie a mohou se vypsat na výstup, do souboru se stavy.

Pro hráčova oponenta se tento proces liší. Vstupy se negenerují na základě vytvoření nové jednotky či zničení již existující, ale při objevení některé z jeho stávajících. V takovémto okamžiku je vytvořena kopie seznamu jeho objevených herních objektů i se stavem všech jeho existujících objektů. Strategie se posuzuje na základě všech, avšak na výstup se zapisují pouze objevené. Tato část procesu je nesmírně důležitá z následujících důvodů. Nepřátelský hráč má jistý motiv, o kterém má představu během hry pouze on a při analýze záznamu hry jej lze vyvodit pouze ze všech jím vlastněných entit. Druhá strana nemá potenciálně nikdy všechny informace a tento motiv usuzuje jen z toho, co má k dispozici. Predikce je tedy již od začátku oslabena neúplnými trénovacími daty ve snaze naučit se klasifikovat právě takovéto informace.

Výstupní soubor nepřátelské strategie je označen příponou `.strategy`, který je ve formátu CSV. Každý jeho řádek obsahuje počty všech nepřátelských entit či úrovní vylepšení, poslední hodnotou na řádku je identifikátor strategie, tedy klasifikační třída, ke které tento výstupní vektor náleží.

5.1.5 Dodatečná data

Poslední částí výstupu je soubor označení příponou `.summary` ve formátu JSON, který obsahuje data použitá v posilovaném učení infrastrukturního manažera a predikci strategií:

- Vítěze hry – ve formátu RGD souboru neexistuje vždy stoprocentní postup pro získání informaci o vítězi. Existuje několik variant, jako například detekce události opuštění hry některým hráčem, či prohledání stavů herních objektů a detekce nulového počtu budov (hráč bez budov prohrává). Pakliže tyto metody zklamou, je stále potřeba hru nějak označit. Z tohoto důvodu byla hodnota tohoto pole nastavena třístavově – vyhrál jeden, druhý nebo není známo.
- Délku hry (v podobě počtu snímků záznamu) – získána z posledního prvku v posloupnosti herních událostí z RGD souboru.
- Seznamy použitých strategií hráče i jeho protivníka – kopie vektorů získaných během dedukce strategií z předešlé kapitoly 5.1.4 a slouží k predikci vhodných strategií v reakci na oponentovy.

5.2 BWAPI

K vytvoření programu schopného hrát StarCraft Broodwar bylo použito **BWAPI** – Brood War Application Programming Interface [35]. Jedná se o open-source framework v podobě dynamické knihovny pro C++, ovšem existuje i verze pro Javu (**BWMIRROR**) realizovaná pomocí JNI – Java Native Interface. Jednou ze zásadních nevýhod této knihovny je podpora pouze pro 32-bitovou verzi Javy, což znamenalo omezení výběru použitelných knihoven.

BWAPI obsahuje kompletní soubor všech tříd potřebných pro ovládání StarCraftu. A to nejen entit ovládaných hráčem, ale i hry samotné – například změny systémového nastavení. Primárně používaná část je však pouze kontrola nad hráčovými jednotkami a shromažďování informací o herní mapě a nepříteli.

Implementací vzniká tzv. **bot** – samostatně běžící aplikace komunikující přes BWAPI se spuštěnou instancí hry. Hlavní třídu bota je nutné vytvořit implementací `DefaultBWListener` třídy BWAPI, která poskytuje ovládací rozhraní a zachytává herní události.

Na herní události rozhraní reaguje invokací příslušné metody:

- `onStart()` – metoda volaná při startu hry. Perfektní místo pro inicializaci agentů, všech modulů a knihoven, analýzu mapy a načtení konfigurací.
- `onFrame()` – metoda volaná v při každém novém snímku (viz. **??**). Jediná metoda umožňující agentům provádění akcí ve hře. Tímto funguje jako synchronizační bod pro jinak paralelně běžící výpočty agentů.
- `onUnitDiscover(Unit)` – vyvolána vždy, kdy přijde hráč do kontaktu s libovolnou herní jednotkou. A to jak svou, tak protivníkovou. Metoda slouží k detekci a ukládání informací ze hry do databáze znalostí agentů. Vstupním parametrem metody je instance třídy `Unit` reprezentující objevenou jednotku.
- `onUnitCreated(Unit)` – vyvolána pokaždé, když hráč začne s výstavbou budovy nebo výcvikem jednotky. Parametrem je vlastní jednotka (opět instance třídy `Unit`).
- `onUnitComplete(Unit)` – vyvolána při dokončení stavby nebo výcviku hráčovy budovy či jednotky. Parametrem je vlastní jednotka (opět instance třídy `Unit`).
- `onUnitDestroy(Unit)` – vyvolána při zničení jednotky či budovy libovolného hráče. Zničená jednotka je parametrem. Ideální místo pro mazání záznamů jednotek z databáze znalostí.
- `onEnd(boolean)` – metoda indikující konec hry. Jejím parametrem je jediný `boolean` značící, zda bot ve hře vyhrál nebo prohrál. Metoda dává botovi prostor uložit si po záznam hry. Ten může být zpracován nástroji pro analýzu a konverzi dat, následně použit jako zpětná vazba naučeným modelům. Tímto způsobem se boti mohou učit z vlastních chyb.

Zajímavým jevem je, že při vytvoření většiny entity hráče se zavolají jak metoda `onCreate()`, tak metoda `onDiscover()`. S totožným parametrem. Pořadí volání však občas zanáší chyby, jelikož agenti mohou manipulovat s vytvořenou jednotkou dříve, než jsou přidány do databáze znalostí. To si žádá ošetření a občas vícenásobné přidávání vlastních jednotek mezi znalosti – znalost se nepřidá znovu, pokud už existuje, může se však aktualizovat (ačkoliv má ve stejném snímku stejné vlastnosti).

Instance třídy `Unit` reprezentující jednotky mohou mít svou pozici. Pozic jsou v BWAPI celkem dva druhy – pozice v pixelech (třída `Position`) a pozice v herních „dlaždicích“ (třída `TilePosition`). V případě dlaždic je pozice vlastně indexem této dlaždice na osách X i Y a její počátek vyjádřený v pixelech je tento index vynásobený číslem 32. Objekty svou pozici mohou měnit s každým snímkem hry. Pakliže nepřátelská jednotka zmizí hráčovým jednotkám z dohledu, její pozice přestává být platnou. Při znovuobjevení jednotky se provede aktualizace.

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

5.3 Implementace bota

Testovací bot byl implementován tak, aby byl schopen hrát pouze za Protossy. Jak bylo popsáno v kapitole o StarCraftu, každá rasa má svá specifika, tomu třeba při implementaci čelit. Jako Protoss čelí bot například těmto výzvám:

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

5.4 BWMA

K analýze mapy se používá knihovna distribuovaná společně s BWAPI, která je rovněž implementována i v Javě, jménem BWMA - Brood War Map Analyzer. Ta slouží k analýze mapy. Spouští se hned po začátku hry a po skončení analýzy poskytuje hráči údaje (ve svých vlastních třídách, nikoliv ve třídách BWAPI), jako například:

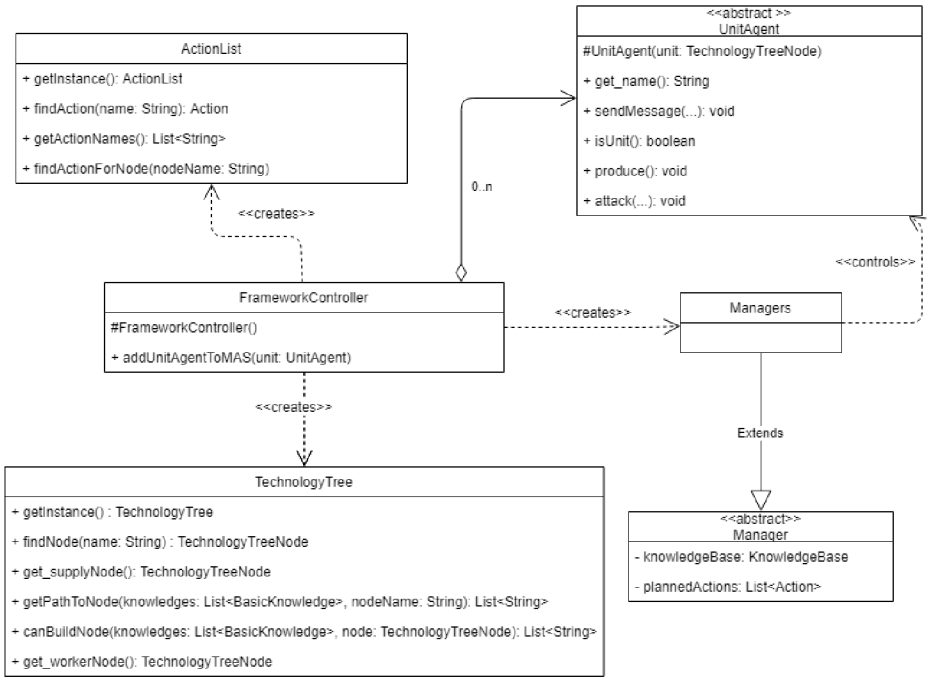
- Potenciální počáteční pozice všech hráčů (ne však informace o tom, kde kdo opravdu je).
- Pozice k potenciální expanzi.
- Úzká hrdla – tzv. *ChokePointy* – místa vhodná k obraně a útoku.
- Pozice surovin.

Implementovaný bot využívá především informace o počátečních pozicích hráčů pro průzkum a potenciální pozice expanzí.

5.5 Multiagentní podpora

K dodání multiagentní podpory botovi byl použit framework popsaný v kapitole 4.1. Na obrázku 5.1 je diagram popisující vztahy tříd realizujících rozhraní StarCraftu a frameworku.

Ihned po startu hry bot vytvoří instanci třídy `FrameworkController`, která se následně postará vytvoření instancí jednotlivých manažerů, stejně jako inicializaci jejich modulů



Obrázek 5.1: Základní třídy nutné pro dodání multiagentní podpory botovi.

chování a učení. Tato třída se rovněž stará o načtení technologického stromu, seznamu proveditelných akcí a dostupných strategií z příslušných JSON souborů. Ty jsou uloženy do tříd, které jsou k dispozici agentům i jejich modulům (pro chování i rozhodování). Popis formátu těchto konfiguračních souborů je detailně popsán v dokumentaci frameworku.

Důležitou funkčností poskytovanou třídou `FrameworkController` je přidávání vlastních, nově definovaných chování a modulů učení. Rovněž slouží jako vstupní bot nových instancí `UnitAgentImpl` (nových agentů reprezentujících jednotky) do multiagentního systému. Tento kontrolér je přiřadí manažerům a umožní jim komunikaci.

5.5.1 Třída `UnitAgentImpl`

Tato třída je implementací abstraktní třídy `UnitAgent` reprezentující herní entitu ze StarCraftu. Vedle konfiguračních souborů tvoří největší část rozhraní mezi multiagentním frameworkem a hrou. Obsahuje implementaci velkého množství abstraktních metod převádějící obecné příkazy frameworku na konkrétní chování. Mezi nejdůležitější patří například:

- Stavba budov.
- Produkce jednotek.
- Zadávání příkazů jednotkám.

Implementace těchto akcí vyžaduje velké množství kontrol, zda jsou příkazy botovi známy a jejich parametry jsou definovány v technologickém stromu a seznamu povolených akcí. Pakliže lze akci provést, obvykle je potřeba manažery informovat příslušnou návratovou hodnotou, podle dokumentace.

Dalšími důležitými metodami je zjišťování informací o hře. Jelikož je framework obecný, žádá si implementaci mnoha takových metod. Jedná se o obrovskou škálu informací o mapě,

jednotkách, budovách, surovinách či hráči. Příkladem mohou být metody zjišťující stav surovin, zdraví, stav, pozice a vlastník jednotek či budov. Pro usnadnění přístupu si instance udržuje referenci na instanci reprezentované herní entity – třída `Unit` z BWAPI.

Nejčastěji zjišťovanou vlastností je její typ reprezentovaný třídou `UnitType`, která obsahuje výčet všech herních jednotek. Tento typ je používán ke klasifikaci entity po jejím prvním objevení, ke kontrolám, zda může provádět požadované akce a často i k identifikaci vazeb v technologickém stromu.

5.6 Ovládání micromanagementu

Jak bylo popsáno v návrhu systému, v kapitole 4.3, k ovládání jednotek během průzkumu nebo boje je používána knihovna Atlantis.

Jelikož má Atlantis vlastní reprezentace herních entit a jejich vlastností definované třídou `AUnit`, je potřeba je vytvořit instance metodou `createFromUnit(Unit)` přijímající parametrem reprezentaci z BWAPI. Pro usnadnění přístupu se tento proces odehrává ve třídě `UnitAgentImpl`, jejíž instance si následně referenci na nově vzniklý objekt ponechá. Tímto slouží jako rozhraní mezi agentním frameworkem, BWAPI i Atlantis.

Implementace Atlantis umožňuje statické volání metod jejích manažerů, přijímajících jako parametr instance `AUnit`. Poté si již Atlantis obstará vše sama. Tato volání se realizují právě v příslušných metodách třídy `UnitAgentImpl`, kam by se za jiných okolností dala vlastní implementovaná funkcionalita.

Systém manažerů frameworku není, bohužel, s Atlantis zcela kompatibilní, jelikož využívá jiný způsob žádání o jednotky (agenti si kontrolu nad jednotkami předávají a komunikují o ní, v Atlantis si ji objekty „prostě vezmou“). Z tohoto důvodu se neinvokují systémy Atlantis jako celky, ale jen části jejich interního chování. Statická implementace, jindy označitelná za *„antipattern“*, by se zde dala označit jako výhodná.

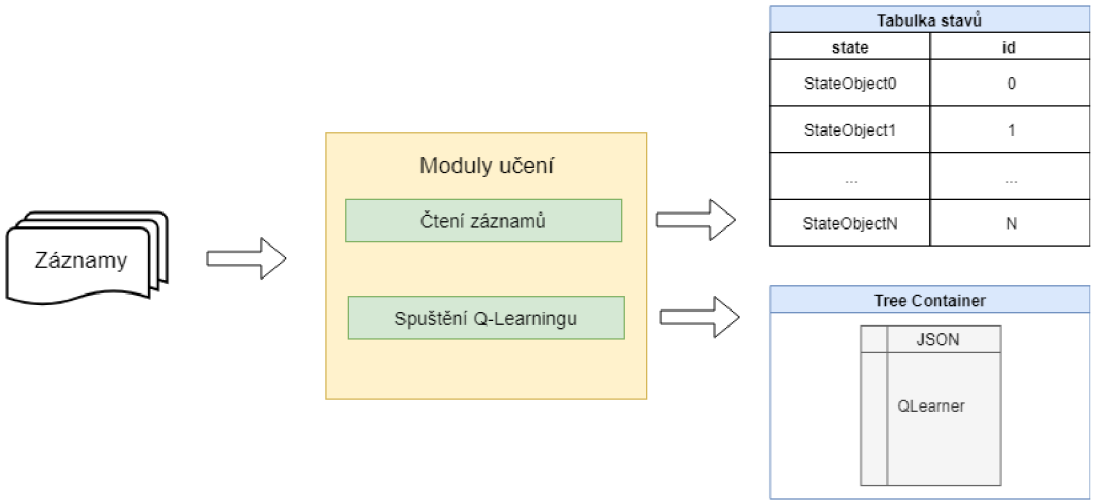
5.7 Posilované učení

K implementaci posilovaného učení sloužícího k volbě akcí infrastrukturního manažera, jak bylo popsáno v kapitole 4.6, byla použita knihovna java-reinforcement-learning, která je implementována jako doplňující materiál ke knize "Reinforcement Learning: An Introduction"[31] a obsahující algoritmy jako Q-Learning, R-Learning, SARSA a Actor-Critic. Je velmi pohodlná a má podporu pro serializaci naučených modelů do formátu JSON.

Učení je a jeho podpora jsou implementovány třídou `Learning`, která obsahuje dodatečnou metodu `main()` pro možnost spustit bota v režimu pouze pro učení (režim pro hraní skrze BWAPI funguje bez metody `main()`). V takovém případě se bot pokusí načíst existující naučený model. Pakliže neexistuje, vytvoří nový.

Součástí bota je i tabulka známých stavů infrastrukturního manažera reprezentovaná třídou `StateTable`. Ta se, stejně jako naučený model, načítá při spuštění bota, a nebo vytváří nová, pokud žádná neexistuje. Rovněž ve formátu JSON. Tato tabulka obsahuje veškeré stavy známé modelu, a to již v upravené podobě, kde jsou data diskretizovaná a bez filtrovaných informací. Důvodem její existence je požadavek knihovny na identifikaci známých stavů pomocí hodnoty Integer. U každého stavu je pak možno dohledat, zda byl použit a s jakým identifikátorem.

Při spuštění v režimu učení a po načtení či vytvoření všech komponent se vyhledají ve složce `data/replays` trojice souborů se stejným jménem a příponami `.action`, `.state`



Obrázek 5.2: Části modulu posilovaného učení.

a `.summary` obsahující informace získané ze záznamů her nástrojem vytvořeným pouze pro tuto práci a popsaným v kapitole 5.1.

Soubor `.state` obsahuje chronologicky seřazené stavy hry zapsaných ve formátu JSON (pouze jako seznam reprezentovaný JSON polem). Zapsání každého nového stavu ze záznamu hry je dáno vytvořením či ztrátou jednotky nebo budovy vlastněných hráčem. Po načtení se stavy diskretizují a vynechají se ignorovaná data, pokud jsou ve stavech přítomna.

Soubor `.action` obsahuje chronologickou posloupnost (každá na novém řádku) botovi známých akcí reprezentovaných unikátním identifikátorem z `ActionList` konfiguračního souboru bota za celý průběh hry.

Vztah mezi těmito soubory je následující:

- Počet stavů je o 1 větší, než počet akcí.
- Ve stavu s indexem `n` byla provedena akce na řádku `n` a výsledkem byl stav s indexem `n + 1`.

Zajímavostí zde je, že bot nemá žádnou akci definovanou jako zničení vlastní jednotky. Během hry, kde se čerpá z naučeného modelu pouze rozhodnutí, není tato akce relevantní, ale při učení je. Bez jejího zapsaní by se totiž mezi stavy ztratila návaznost – model by nevěděl, jak se dostal do nového stavu. Za tímto účelem byla akce zničení jednotky definována identifikátorem reprezentovaným konstantou symbolických `666`.

Soubor `.summary` obsahuje výsledek a délku hry, uložené ve formátu JSON. Tyto dva údaje jsou použity při hodnotící funkci. Z některých záznamů nejde však zjistit, někdy ani vydedukovat analýzou zbývajících budov, kdo vyhrál, proto byla zavedena třístavová logika – hráč vyhrál, prohrál nebo je výsledek neznámý.

Mezi vstupy učení lze zařadit i výstupy funkcí pro diskretizaci dat stavu. Tyto funkce musí mít identickou implementaci po celou dobu existenci modelu – během učení u rozhodování se podle něj za hry. Změnou se naruší konzistence modelu a přestane dávat smysluplné údaje.

Soubory s trénovací množinou se prochází stav po stavu. Při každém načteném stavu se musí prohledat celá tabulka známých stavů pro získání identifikace stavu či vytvoření nového záznamu a identifikace, pokud ještě není znám. I s použitím hashovacích funkcí a zjednodušením stavů tento proces stále reprezentuje velkou část času celého učení.

Pro každý stav se najde odpovídající akce ze hry. Tato dvojice je vstupem hodnotící funkce, která akci podle vybraných atributů stavu, výsledku a doby trvání hry ohodnotí. Tato trojice je společně se stavem získaným provedením akce vstup modelu posilovaného učení.

Při inicializaci jsou botovi předány nové rozhodovací moduly obsahující referenci na třídu `Learning`. V té se načte naučený model a ten je pak k dispozici v rozhodovacím modulu.

5.8 Neuronové sítě

Bot používá neuronových sítí ke klasifikaci strategií nepřítele a volbě vlastních strategií, viz. kapitola 4.4.

K implementaci těchto neuronových sítí byla použita knihovna **Neuroph** [10] implementovaná v Javě a C++. Původním plánem bylo použití mnohem pokročilejší knihovny Deeplearning4j [4], avšak ta je, stejně jako většina zajímavějších knihoven v dnešní době, podporovaná pouze ve 64-bitové verzi Javy, zatímco BWAPI pouze ve 32-bitové.

Neuronové sítě jsou uloženy ve třídě `Learning`. Tato třída umí vytvořit sítě nové, načíst existující naučené modely ze souborů nebo je po naučení uložit. Samotná knihovna Neuroph poskytuje velice dobrou podporu k ukládání naučených modelů v binární podobě.

Pro vytvoření nové sítě je potřeba nové instance třídy `MultiLayerPerceptron` se zadanými počty neuronů v jednotlivých vrstvách. Pro novou síť se instancí třídy `BackPropagation` vytvoří nový učící model s koeficientem učení a požadovaným počtem iterací učení pro každý prvek trénovacích dat jakožto parametry.

Po vytvoření nové či načtení existující neuronové sítě jsou načítána data z trénovací množiny získané skrze implementovaný nástroj z kapitoly 5.1, který se stará o správný formát dat. O celé učení se potom stará model knihovny `BackPropagation`. Po naučení jsou vypsány statistiky a celá síť se uloží.

Při inicializaci se botovi přiřadí nové rozhodovací moduly obsahující referenci na třídu `Learning`, skrze kterou se sítě načtou a volají.

Kapitola 6

Testování a dosažené výsledky

V této kapitole jsou popsány vstupy, průběh a výstupy testování navrženého a implementovaného systému. Testování probíhalo nejen po dokončení, ale i průběžně při implementaci. Právě v té fázi poskytlo mnoho námětů ke scénářům experimentů a vedlo k mnoha úpravám. Ty jsou zde popsány také, avšak nikoliv v samostatné kapitole, nýbrž vždy u části postupu, kde vznikly.

V první části lze nalézt deklarace cílů testování, popis metodiky a zvolených testovacích dat.

Následuje popis jednotlivých, na sebe navazujících fází experimentů, u každé s přihlédnutím a zhodnocením všech zúčastněných metod pro učení a plánování.

V poslední části se pak jedná o zhodnocení výsledků navrženého systému učení a výčet námětů k potenciálnímu zlepšení.

6.1 Cíle experimentů

Systém pro strategickou hru lze hodnotit z několika úhlů pohledu. Primární je, samozřejmě, schopnost výhry a míra vlivu implementovaných metod učení a plánování. Kritérií je však více a i tyto dva uvedené příklady mohou být testovány ve více fázích proti jiným protivníkům.

Testování bylo zaměřeno především na:

1. Schopnost modelu hrát StarCraft jako takový, bez naučených modelů.
2. Schopnost porazit výchozí počítačové protivníky ve hře.
3. Schopnost zvítězit i nad těžšími soupeři, například jinými inteligentními boty.
4. Schopnost poučit se ze svých proher.
5. Efektivita a přesnost metod učení a klasifikace, jejich přínos.
6. Přínos multiagentního přístupu – nejen jako systému rozdělování odpovědnosti, ale i samostatnost agentů a vliv jejich komunikačních schopností souvisejících se systémem učení.

Testování bylo zaměřeno na hry jeden proti jednomu, tedy „1V1“ (duel), což zaručovalo větší vyrovnanost her a napodobení podmínek turnajů či soutěží.

6.2 Testovací data a postup testování

Z internetových zdrojů [32] byla získána testovací data v podobě několika tisíc záznamů her, ve kterých soupeřili zkušení lidští hráči, převedených nástrojem bwrepdump vzorkovaných do podoby použitelné jako vstup pro nástroj popsaný v kapitole 5.1 (kde je možné nalézt celý postup procesu získávání vzorků dat k učení).

Z těchto záznamů bylo 2027 her typu Protoss (rasa hraná implementovaným botem) proti Zergům (vždy reprezentující oponenta). Z těchto záznamů bylo přes 200 vyřazeno pro špatný formát nebo porušení podmínky přítomnosti přesně dvou hráčů, následně několik nejkratších s nejmenší hodnotou pro zaokrouhlení velikosti. Celkem tedy bylo vyřazeno 227 záznamů a testovací data obsahovala 1800 her.

V každém kole testování jednotlivých modelů učení se soubory náhodně (s uniformním rozložením pravděpodobnosti) rozdělily do tří množin:

- Data pro učení – velikost 60% původní množiny.
- Validační data – použitá pro optimalizování koeficientu učení neuronových sítí, velikost 20% původní množiny.
- Testovací data – pro zhodnocení přesnosti modelu, velikost 20% původní množiny.

Při testování učení neuronových sítí se při každé změně parametru provedlo vyhodnocení desetkrát. Pokaždé se znovu náhodně vygenerovaly jednotlivé množiny dat, vnitřními nástroji knihovny optimalizoval koeficient učení a na testovací množině vypočítala přesnost modelu.

Při testování posilovaného učení hraním her se testovalo dvěma způsoby:

1. Bez učení v průběhu hraní.
2. S učením v průběhu hraní.

V obou případech se testovalo odehráním stovky her a vyhodnocením procenta výher buď za všechny, nebo po desítkách. Pakliže bylo učení během hry vypnuté, s každou změnou parametrů se odehrála přesně jedna stovka her. Ve druhém případě by takový vzorek byl potenciálně nedostačující, celkový počet her byl tedy násobkem stovek, vždy vyhodnocovaný opět po celých stovkách nebo desítkách.

Během experimentů byly vynechány hry, kdy se bot zasekl vlivem špatného ovládání jednotek, protože na tento aspekt hry se práce nesoustředila. Taková hra byla průměrně každá třináctá.

6.3 Přesnost klasifikace nepřátelských strategií

Tato část experimentování je zaměřena na přesnost klasifikace nepřátelských strategií neuronovou sítí ve strategickém manažerovi. Tato klasifikace nemá přímý vliv na výsledek hry, je však součásti procesu výběru strategie bota, jej96 vliv se testuje v dalších fázích.

Při pohledu na trénovací data lze jednoznačně říci, že pro některé třídy existuje mnoho podobných prvků bez možnosti jednoznačného ručního přiřazení ke kterékoliv ze tříd. Tento problém lze popsat příkladem vstupního vektoru [0,0], který může být cestou ke strategii splněné při [1,0] nebo k jiné, splněné při [0,1]. Dalším faktorem je účast některých jednotek či budov ve všech strategiích. Příkladem jsou dělníci, budovy ekonomické expanze či uzly

technologického stromu blízko kořenu, které se musí postavit vždy. Z těchto dvou důvodu se nepředpokládalo dosažení vysoké přesnosti.

Základní parametry experimentu:

- Všechny jednotky a budovy Zergů tvoří dohromady vstupní vektory o velikosti 35 prvků.
- Bylo specifikováno 8 Zergských strategií – 8 klasifikačních tříd.
- Použita byla síť s jednou skrytou vrstvou o velikosti totožné se vstupní vrstvou.
- Data ve vstupních vektorech byla normalizována do intervalu $[0.0, 1.0]$.

Byly navrženy celkem 3 testovací scénáře, které byly spuštěny zároveň. Každý se stejně zvolenými vzorky dat, pouze jinou neuronovou sítí a vstupním vektorem:

1. Základní – podle základních parametrů experimentu.
2. Vynechání nulových prvků – redukce velikosti vstupních vektorů vynecháním prvků, které jsou ve všech datech nulové, celkem tedy o 9 (na 26). O tento počet uzlů se zmenšily i vstupní a skrytá vrstva neuronové sítě.
3. Vynechání společných prvků – z vektorů již zmenšených v předchozím kroku se stejným způsobem vynechaly další 4 uzly (a zmenšila se síť), které se vyskytovaly ve všech strategiích a obvykle ve stejném nebo podobném počtu. Z herní logiky jde především o dělníky, jednotky poskytující supply, nejobyčejnější druh jednotek (Zergling) a speciální Zergské stavby neplnící jinou funkci než rozšíření zastavitelného území.

Po deseti pokusech byly průměrné přesnosti klasifikace strategií následující:

ID strategie	0	1	2	3	4	5	6	7
Průměrný počet prvků	6754	6257	10205	7771	7259	4058	3144	4398
Základní přesnost (%)	22,57	20,19	31,46	17,51	24,38	13,92	14,04	18,26
Přesnost scénáře 2 (%)	53.93	52.99	58.97	44.19	51.41	43.51	46.33	48.71
Přesnost scénáře 3 (%)	78.03	78.54	79.66	77.25	78.73	77.61	78.18	76.59

Z experimentů plyne, že redukce vstupních dat s nižší důležitostí má velký pozitivní vliv na přesnost klasifikace. Zároveň je možné vyvodit několik dalších závěrů:

- Strategie s nejvíce prvky v trénovacích datech má nejvyšší průměrnou přesnost, i když se nejedná o nejobecnější strategii.
- Některé strategie mají nižší průměrnou přesnost i přes vyšší počet trénovacích dat.
- Přírůstek přesnosti po vynechání nulových prvků byl jen nepatrně vyšší, než po vynechání nejčastějších prvků, ačkoliv jich bylo mnohem více (menší přírůstek v poměru na jeden vynechaný prvek).
- Vynechání nejčastějších prvků výrazně snížilo odchylky v průměrných přesnostech.

Praktická využitelnost scénáře 2 je sporná, jelikož v jiných trénovacích datech by vynechané prvky nemusely být nulové. Takováto síť má rovněž omezenou uplatnitelnost pro kontinuální učení během hraní po nasazení bota, kdy by síť neuměla reagovat v případě, že by se takový prvek objevil. Jelikož ale multiagentní framework vyžaduje ruční specifikace

strategií (nelze do nich automaticky přidávat další druhy jednotek), tento přístup se pro další experimenty ponechal.

Výsledky scénáře 3 ukazují na to, že vynechání jednotek vyskytujících se při všech strategiích, vede k nejlepším výsledkům. Bohužel v případě StarCraftu nelze vynechat více takovýchto objektů, jelikož by se nedaly v datech rozpoznat podobné strategie specializované na útok a na obranu (více budov a méně jednotek či naopak).

Vzhledem k nízkým očekáváním a faktu, že se strategie vyhodnocují velmi často (může dojít ke změně i několikrát za sekundu při objevování nepřátelské základny), byl výsledek označen za uspokojivý a všechny další experimenty byly prováděny s podobou klasifikace ze scénáře 3.

6.4 Diskretizace dat a hodnotící funkce pro posilované učení

Pro všechny experimenty s modelem posilovaného učení byly použity totožné funkce diskretizace vstupních dat a hodnotící funkce $Q(s, a)$ 3.3 generující odměnu pro akci na základě stavu.

Diskretizace dat byla nutná pro převedení numerických hodnot na úrovně požadované stavem infrastrukturního manažera a délka hry ze statistik. Konkrétně se jedná o hodnoty volných populačních bodů (freeSupply), poměru dělníků a základen (workerRatio), počet staveb schopných produkce jednotek a délka hry. Funkce jsou reprezentovány následující tabulkou a hodnoty byly odvozeny na základě pozorování během vývoje bota:

	freeSupply	workerRatio	facilities
VERY_LOW	< 4	< 6	< 1
LOW	< 6	< 10	< 2
MEDIUM	< 10	< 16	< 5
HIGH	< 20	< 24	< 8
VERY_HIGH	< 35	< 35	$>= 8$
ULTRA_HIGH	$>= 35$	$>= 35$	nepoužito

Takováto podoba funkcí vykazovala nejen vysokou schopnost redukce stavového prostoru pro učení Q-learningem, ale i strategickou hodnotu. Bot se například okamžitě snažil (v kombinaci se správně nastavenou hodnotící funkcí) o postavení alespoň jedné budovy k produkci jednotek, o druhou už pak méně. Hodnoty pro volné populační body mohou přijít někomu, kdo StarCraft zná, jako vysoké – například když budovy přidávající populační body dodávají 8 jednotek, úroveň VERY_LOW pro hodnotu nižší než 4 zabírá polovinu tohoto intervalu. Tyto hodnoty jsou dány interní reprezentací BWAPI přizpůsobené jednotkám zabírajícím polovinu zobrazeného populačního bodu a jsou vynásobené dvěma oproti hodnotám, které vidí hráč na obrazovce.

Hodnotící funkce posilovaného učení byla navržena na základě pozorování a zkoušení podle požadavků popsaných v kapitole **??** následovně:

Data: Počáteční stav
Data: Akce
Data: Jednotka produkovaná akcí
Data: Herní čas a výsledek
Data: Stavy surovin hráče
stanov počáteční hodnotu odměny na 5.0;
if *vítězem je hráč* **then**
| odměna *= multiplikátor času výhry
else
| odměna *= multiplikátor času prohry
end
spočítej multiplikátor Pylonu;
if *akce je postavení Pylonu* **then**
| odměna *= multiplikátor Pylonu
else
| odměna /= multiplikátor Pylonu
end
if *akce je postavení expanze a hráč již expanzi má* **then**
| odměna *= 0.1
end
if *akce je postavení obranné stavby* **then**
| odměna *= odměna * 2.0 / počet_obranných_staveb + 1;
| // Menší hodnota obrany pokud již nějaká existuje, na začátku hry je hodnota
| ale vysoká
end
if *akce je postavení facility* **then**
| odměna *= multiplikátor facility
end
if *pokud je akce postavení budovy* **then**
| odměna *= 2.0 // Větší strategická hodnota budov než jednotek
end
if *pokud je akce výzkum vylepšení* **then**
| odměna *= 4.0 // Větší strategická hodnota vylepšení než budov a jednotek
end
if *pokud je akce produkce dělníka* **then**
| odměna /= 2.0 // Ve hře se jich staví vleké množství, na začátku je to
| vyváženo omezenými možnostmi produkce a jsou voleni časteji
end
if *pokud je akce produkce jednotky* **then**
| odměna *= počet_volných_populačních_bodů_hráče / 2 // Když je volných
| hodně bodů, je potřeba produkovat hodně armády
end
if *akce je součástí zvolené strategie* **then**
| odměna *= 5.0 // Nejdůležitější faktor
end

Kde multiplikátory nabývají následujících hodnot:

	čas výhry	čas prohry	Pylon	Facility
VERY_LOW	nepoužito	nepoužito	5.0	3.0
LOW	1.5	0.5	2.5	2.0
MEDIUM	1.3	0.8	1.2	1.5
HIGH	1.1	0.9	0.7	1.0
VERY_HIGH	nepoužito	nepoužito	0.1	0.5
ULTRA_HIGH	nepoužito	nepoužito	0.1	0.5

6.5 Redukce stavového prostoru posilovaného učení

I po diskretizaci některých částí stavu infrastrukturního manažera se počet jedinečných stavů vygenerovaných z trénovacích dat pohyboval běžně ve stovkách tisíc. V takovém počtu byla data kvůli paměťovým nárokům stále nepoužitelná. Všechny stavy bylo potřeba udržovat v tabulce známých stavů a tabulka Q-learningu pro uchovávání hodnocení akcí také nabývala enormních rozměrů. Aplikace nebyla schopna běhu a docházelo k dosažení limitu 2GB operační paměti pro 32-bitovou verzi Javy vynucenou BWAPI. Při tomto limitu však nebyla načtena stavů ani polovina.

Pro redukci bylo nutné vybrat data s největší diverzitou a přitom potenciálně nižší strategickou hodnotou pro rozhodování. Bylo rozhodnuto vynechat informace o počtu některých hráčem vlastněných jednotkách a budovách.

Data o počtech jednotek mají pro infrastrukturního manažera nejmenší strategickou hodnotu a suverénně nejvyšší diverzitu. O stavbě dalších jednotek rozhoduje na základě žádostí manažerů a volných populačních bodů, proto nebudou příliš chybět. Stejně tak počty dělníků jsou již vyjádřeny v poměru dělníků a základen. Hodnota zbývajících populačních bodů také znamená snížení důležitosti budov, které je poskytují.

Vynecháním těchto dat se počet stavů podařilo dostat do intervalu od 10 000 do 20 000, což byla více než dostatečná redukce.

Nevýhodou tohoto vynechávání je, že pokud agent vykoná akci vedoucí ke zvýšení počtu některého z vynechaných objektů, dostane se do stavu, ve kterém už je. Tímto může docházet k neúměrnému posílení hodnot tvorby jednotek (což je i důvodem pro snížení odměny pro tyto akce patrné z předchozí kapitoly).

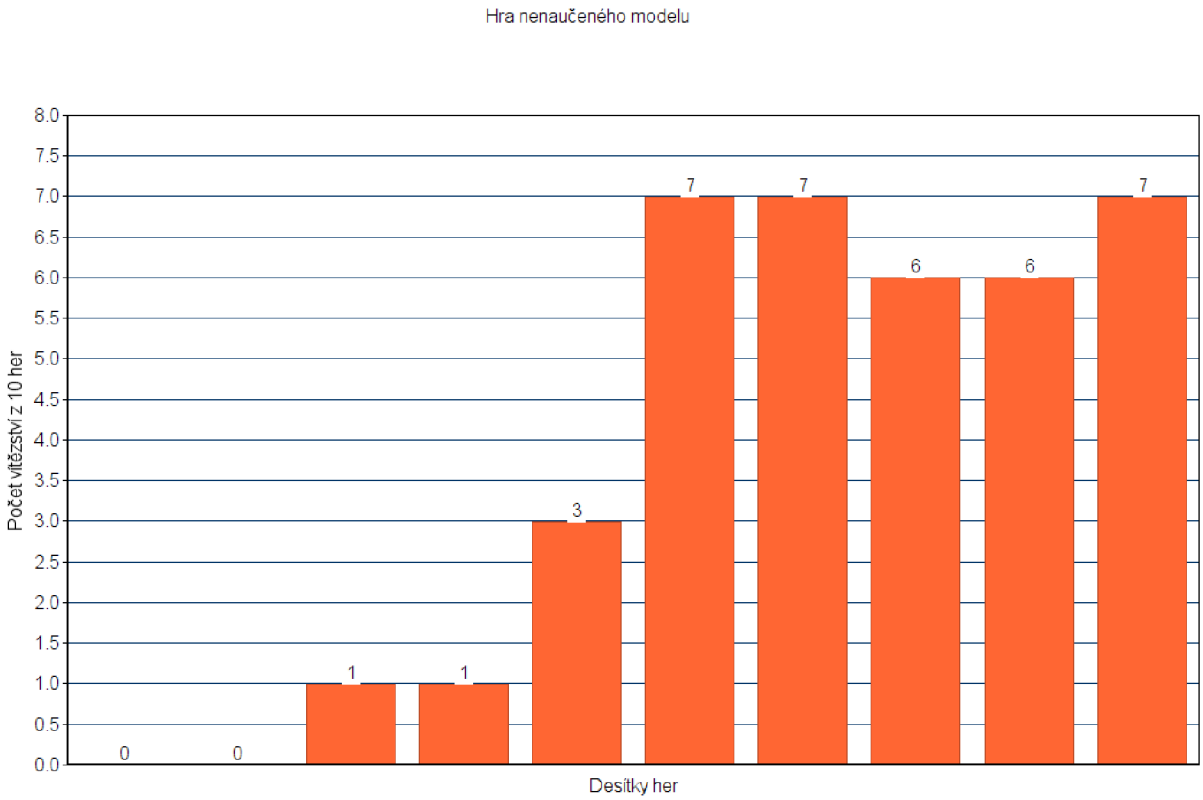
6.6 Hra bez volby strategií a naučených modelů

Tento experiment měl prokázat, zda je systém posilovaného učení v kombinaci s daty poskytovanými multiagentním frameworkem (jako například seznam možných akcí vzhledem k technologickému stromu) schopen smysluplné hry a bez jakýchkoliv předešlých znalostí se naučit vyhrát.

Bot odehrál 100 her proti výchozímu Zergskému protivníkovi. Po každé hře byl uložen její záznam, ten byl analyzován nástrojem pro získávání dat k učení, model se před odehráním další hry učil. Výsledky jsou vyobrazeny v následujícím grafu:

Bot byl schopen díky omezením ze strany manažera hrát v mezích pravidel hry a jeho akce byly korektní, avšak neměly žádný smysl nebo cíl. Po 24 hrách se botovi povedlo poprvé zvítězit a jeho úspěšnost rostla. Podařilo se mu však naučit pouze jednu strategii, kterou neustále opakoval. Ta byla velice jednoduchá a spoléhala na zničení soupeře dříve,

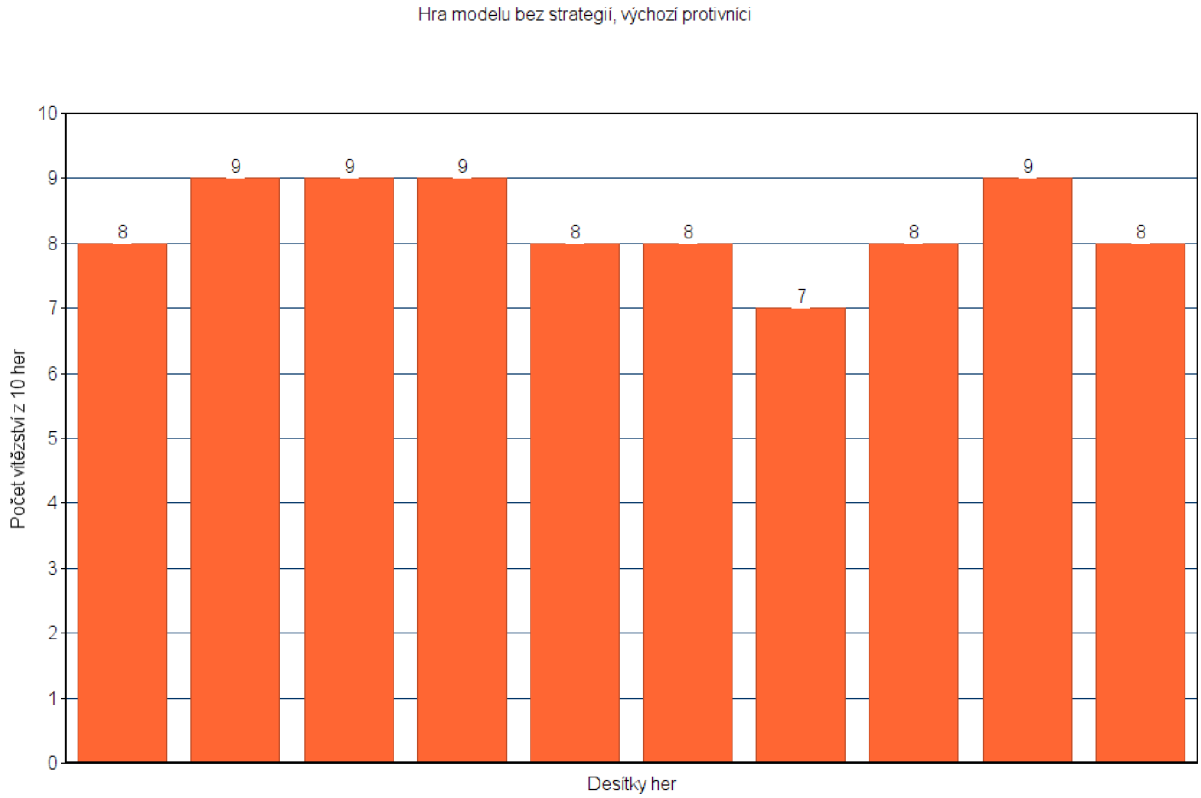
než vyrobí pokročilejší jednotky. Od poloviny experimentu byly prakticky všechny prohry způsobeny právě tím.



Obrázek 6.1: Graf počtu výher bez naučeného modelu.

6.7 Výchozí protivníci s naučenými modely posilovaného učení a bez volby strategií

Po naučení vstupních dat ze záznamů her lidských hráčů podle popisu v kapitole 6.2 se výchozí Zergští protivníci stali pro bota mnohem slabším soupeřem. Všechny prohry byly způsobeny pouze případy, kdy oponent volil produkci jednotek schopných útoku aniž by byly pro bota viditelné. Ten na to kvůli absenci výběru strategií nebyl schopen reagovat. Mnohdy se mu povedlo zničit nepřátelskou základnu, ale poté stejně prohrál.



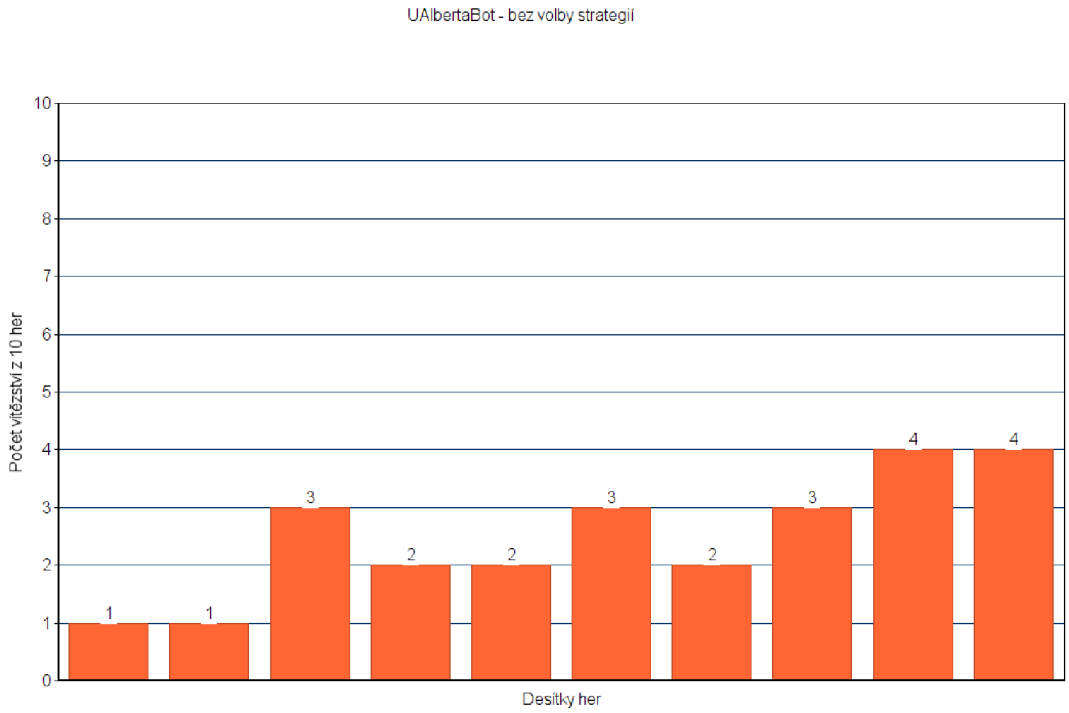
Obrázek 6.2: Graf počtu výher naučeného modelu proti výchozím oponentů, bez volby strategií.

6.8 Pokročilejší protivníci

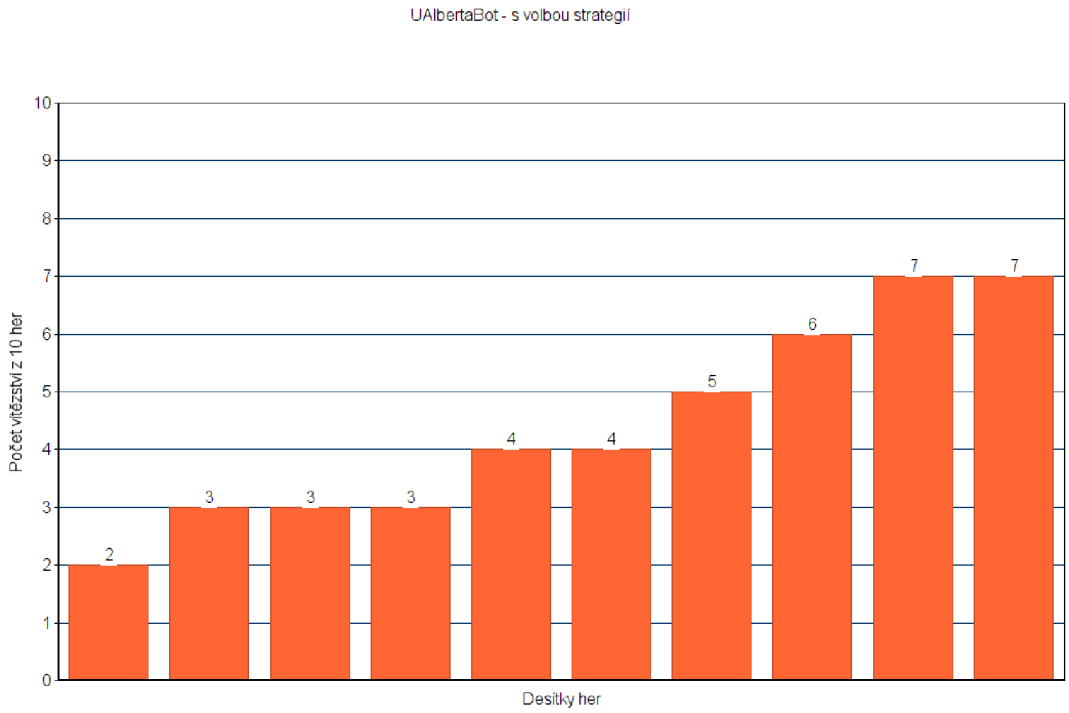
Pro experimenty se podařilo sehnat [35] [12] spustitelné verze dvou open-source botů upravených pro hru za Zergy. Jedním z nich byl UAlbertaBot, který je vyvíjený jedním ze zakladatelé soutěže SSCAIT [35] a stal se základem pro velké množství dalších, mnohem pokročilejších botů. Druhým z nich byl HITA.

Bot vyvíjený v této práci odehrál s oběma dvakrát sto her. V první stovce byla volba strategií vypnutá, ve druhé zapnutá. Na začátku každé etapy her bot začínal s modelem naučeným pouze pro prvotní testovací data. Po každé hře byl uložen a analyzován záznam, na nových datech se bot učil.

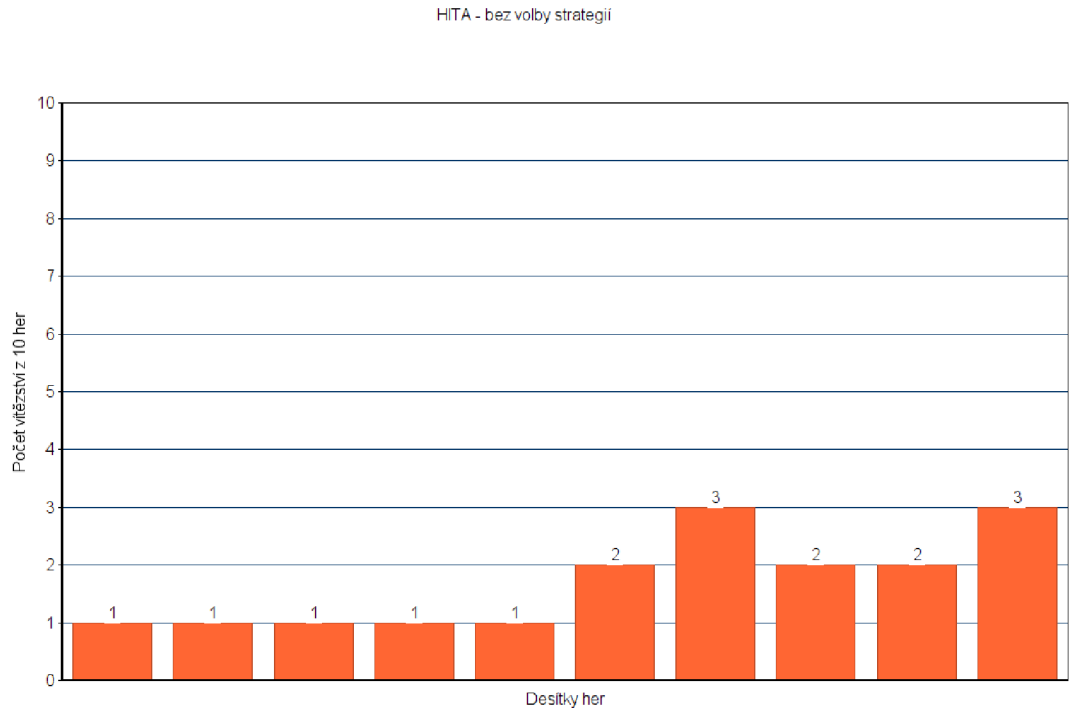
Nepředpokládalo se vysoké procento výher, protože protivníci byli vysoce specializovaní pro StarCraft a vyvíjení mnohem delší dobu, zatímco implementovaný systém je založený na obecném frameworku a s velmi nízkou podporou micromanagementu. Pozornost byla věnována spíše míře zlepšení.



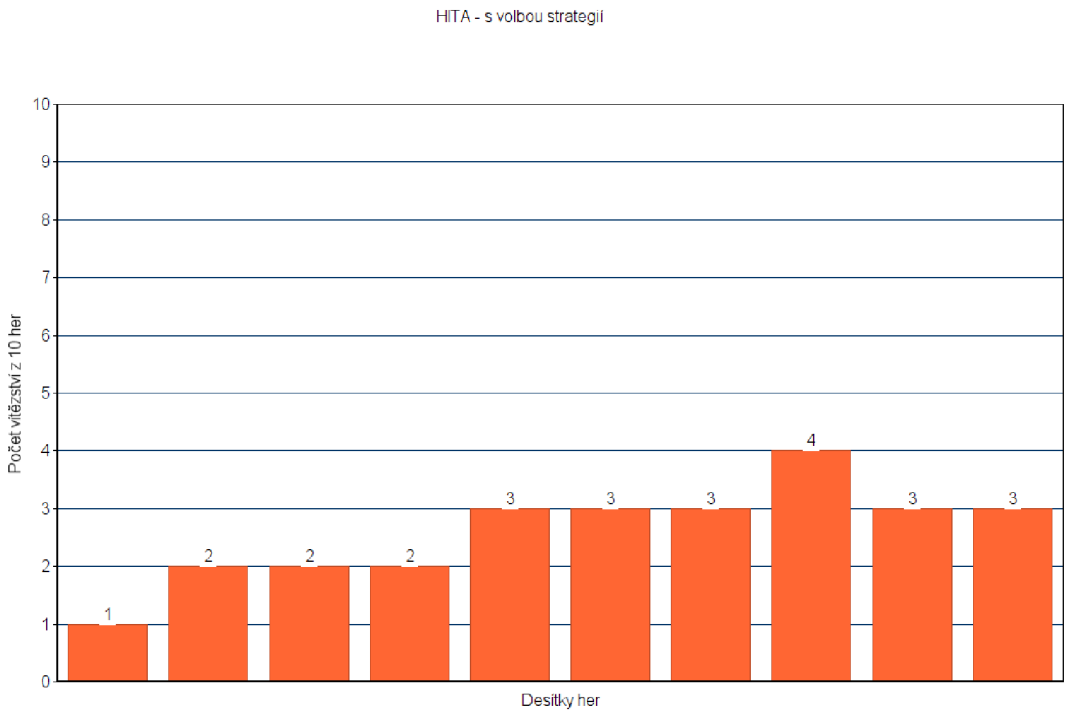
Obrázek 6.3: Graf počtu výher proti botovi UAlbertaBot, bez volby strategií.



Obrázek 6.4: Graf počtu výher proti botovi UAlbertaBot, s volbou strategií.



Obrázek 6.5: Graf počtu výher proti botovi HITA, bez volby strategií.



Obrázek 6.6: Graf počtu výher proti botovi HITA, s volbou strategií.

6.9 Zhodnocení

Z experimentů vyplývá, že vyvinutý systém prokázal:

- Schopnost naučit se z větší trénovací množiny.
- Během hraní se průběžně učit a dosahovat lepších výsledků.
- Do jisté míry rozeznávat strategie nepřítele a na základě této klasifikace upravovat své vlastní strategie a volby.
- I přes obecnost platformy je schopen konkurovat specializovanějším botům.

Naopak mezi hlavní nevýhody se řadí:

- Nutnost ukládání záznamů her a učení se z proher až po konci hry.
- Nutnost ručních specifikací strategií.
- Výběr a analýza dat pro učení z trénovací množiny vyžadují velké množství času a předzpracování.

Navržený systém těžil z multiagentního přístupu, a to hlavně během volby akcí infrastrukturního manažera. Jím poskytované omezující podmínky ve formě povolených akcí v závislosti na technologickém stromu a především na žádostech ostatních agentů v systému usměrňovaly akce správným směrem i bez složitější analýzy.

Zásadními problémy systému je například systém ovládání jednotek, který často způsobil prohru nebo nízká přizpůsobivost na neznámé stavy ve hře, kde se objeví.

Systém by bylo možné vylepšit implementací dalších způsobů učení pro jiné manažery frameworku a jejich důmyslnější propojení.

Kapitola 7

Závěr

V této práci byl navržen a implementován systém podpory učení a plánování autonomních agentů v multiagentním systému pro hraní strategických počítačových her. Tento systém funguje pro hru StarCraft: Brood War, avšak je zaměřen na dostatečně obecné aspekty hry, aby byl po změně vstupních dat použitelný na jiné hry s podobnými mechanikami.

Tento systém vznikl na základě analýzy metod strojového učení a rozhodování racionálních agentů používaných pro strategické hry a rozboru získatelných dat použitelných jako trénovací vzorky. Soustředil se na rozhodování s větším významem pro hru jako celek, ne na příkazy jednotlivým herním objektům a jednotkám.

Návrh byl do velké míry ovlivněn také schopnostmi používaného multiagentního systému vyvinutého Bc. Lukášem Válkem, implementovaném na platformě JADE, který byl použit. Implementace systému navrženého v této práci je založena na jím vyvinutém jednoduchém herním systému, který byl vylepšen a rozšířen o metody učení. Tento multiagentní systém funguje na principu centralizovaných manažerů, z nichž každý ovládá specializovanou část hry.

Metody rozhodování systému se soustředí kolem tří úkolů. První z nich je klasifikace nepřátelských strategií pomocí vícevrstvých perceptronových neuronových sítí využívajících k učení algoritmu Backpropagation. Za druhé pak s využitím stejné architektury neuronových sítí, volí vlastní strategie, které se dříve ukázaly jako odpovídající odpověď na oponentovy. Největší částí systému je model posilovaného učení algoritmem Q-learning rozhodující o stavbě budov a produkci jednotek založený na stavu hry a zvolené hráčově strategii.

Systém prokázal schopnosti porazit původní, výchozí počítačem řízené protivníky ve hře StarCraft, konkurenceschopnost specializovaným inteligentním botům. Největším pozitivem systému je prokazatelné zlepšování výsledků s více odehranými zápasy.

Literatura

[1] *Atlantis – knihovna pro tvorbu StarCraft botů. [Online; navštíveno 06.03.2019].*
URL https://github.com/Ravaelles/Atlantis

[2] bwrepanalysis – analyzátor záznamů her pro StarCraft. [Online; navštíveno 01.03.2019].
URL https://code.google.com/archive/p/bwrepanalysis/

[3] bwrepdump – analyzátor záznamů her pro StarCraft. [Online; navštíveno 01.03.2019].
URL https://github.com/syhw/bwrepdump

[4] Deeplearning4j – knihovna strojového učení v Javě. [Online; navštíveno 02.02.2019].
URL https://deeplearning4j.org/

[5] DeepMind AlphaStar. [Online; navštíveno 02.03.2019].
URL https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/

[6] FIPA ACL Message Specification. [Online; navštíveno 19.03.2018].
URL http://www.fipa.org/specs/fipa00061/SC00061G.pdf

[7] Freecodecamp - Q-learning. [Online; navštíveno 20.04.2018].
URL https://medium.freecodecamp.org/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc

[8] JAVA Agent Development Framework. [Online; navštíveno 19.03.2018].
URL http://jade.tilab.com/

[9] Liquipedia – Strategie pro StarCraft: Brood War. [Online; navštíveno 22.03.2019].
URL https://liquipedia.net/starcraft/Build_Orders_(BGH)

[10] Neuroph – knihovna pro neuronové sítě v Javě. [Online; navštíveno 03.02.2019].
URL http://neuroph.sourceforge.net/

[11] OpenRa - open source game engine. [Online; navštíveno 11.02.2019].
URL http://www.openra.net/

[12] SC AI Portal. [Online; navštíveno 12.04.2019].
URL http://www.starcraftai.com/wiki/Main_Page

[13] Wikipedia - Artificial neural network. [Online; navštíveno 17.03.2018].
URL https://en.wikipedia.org/wiki/Artificial_neural_network

[14] *Wikipedia - Backpropagation. [Online; navštíveno 17.03.2018].*
URL https://en.wikipedia.org/wiki/Backpropagation

[15] *Wikipedia - Deepmind. [Online; navštíveno 22.02.2019].*
URL https://en.wikipedia.org/wiki/DeepMind

[16] *Wikipedia - Heroes of Might and Magic. [Online; navštíveno 17.01.2018].*
URL https://en.wikipedia.org/wiki/Heroes_of_Might_and_Magic

[17] *Wikipedia - Markov decision process. [Online; navštíveno 17.03.2018].*
URL https://en.wikipedia.org/wiki/Markov_decision_process

[18] *Wikipedia - Q-learning. [Online; navštíveno 17.03.2018].*
URL https://en.wikipedia.org/wiki/Q-learning

[19] *Wikipedia - Reinforcement learning. [Online; navštíveno 17.03.2018].*
URL https://en.wikipedia.org/wiki/Reinforcement_learning

[20] *Wikipedia - Supervised learning. [Online; navštíveno 17.03.2018].*
URL https://en.wikipedia.org/wiki/Supervised_learning

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

[22] *Bidlo, M.: Podklady k předmětu EVO - Aplikované avoluční algoritmy. [Online; navštíveno 28.1.2018].*
URL https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FEVO-IT%2Flectures&cid=12093

[23] *Churchill, D.; Buro, M.: Build Order Optimization in StarCraft. [Online; navštíveno 14.02.2018].*
URL https://pdfs.semanticscholar.org/dfd9/1e739bd979c08485a75fd11c501a6ec05118.pdf

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

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

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

[27] *Ponsen, M.; Muñoz-Avila, H.; Spronck, P.; aj.: Automatically Generating Game Tactics through Evolutionary Learning. [Online; navštíveno 2.12.2017].*
URL https://www.aaai.org/ojs/index.php/aimagazine/article/viewFile/1894/1792

[28] *Robertson, G.: ScExtractor – analyzátor záznamů her pro StarCraft. [Online; navštíveno 01.03.2019].*
URL https://github.com/phoglenix/ScExtractor

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

[30] *Sethy, H.; Patel, A.; Padmanabhan, V.: Real Time Strategy Games: A Reinforcement Learning Approach. [Online; navštíveno 4.12.2017].*
URL https://www.sciencedirect.com/science/article/pii/S187705091501354X

[31] *Sutton, R. S.; Barto, A. G.: Reinforcement Learning: An Introduction. [Online; navštíveno 01.03.2019].*
URL http://incompleteideas.net/book/bookdraft2017nov5.pdf

[32] *Uriarte, A.: Testovací data pro StarCraft boty. [Online; navštíveno 02.02.2019].*
URL http://nova.wolfwork.com/dataMining.html

[33] *Watson, I.; Azhar, D.; Chuyang, Y.; aj.: Optimization in Strategy Games:Using Genetic Algorithms to Optimize City Development in FreeCiv. [Online; navštíveno 14.02.2018].*
URL https://www.cs.auckland.ac.nz/research/gameai/projects/GA%20in%20FreeCiv.pdf

[34] *Zbořil, F.: Podklady k předmětu AGS - VI. Multiagentní systémy. [Online; navštíveno 6.1.2018].*
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FAGS-IT%2Flectures%2FAGS16_07.pdf&cid=10764

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

Příloha A

Instalace a spuštění

Spuštění bota vyžaduje:

1. Vývojová platforma JDK 8 a vyšší, 32-bitová verze.
2. Nainstalovanou knihovnu BWAPI (verzi 4.2.0) a ChaosLauncher [35].
3. Nainstalovanou legální kopii hry StarCraft: Brood War verze 1.16.1.
4. Načtení projektu ve vývojovém prostředí IntelliJ Idea libovolné verze vyšší než 14.
5. Nainstalovaný Maven.

Po instalaci všech částí stačí ve vývojovém prostředí projekt spustit. V odevzdaném balíku jsou již poskytnuty veškeré potřebné konfigurační soubory nutné pro fungování bota a multiagentního frameworku.

Spuštění nástroje na analýzu a získání dat ze záznamů je potřeba pouze vývojového prostředí IntelliJ Idea a SDK jazyka Kotlin.

A.1 Manuál k nástroji na analýzu záznamů her

Po otevření projektu stačí spustit hlavní třídu.

Nástroj bere vstupy ze složky `input` a ukládá výsledné soubory do složky `output`. Je rovněž potřeba dodat strategie do složky `strategies`.