VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

UMĚLÁ INTELIGENCE PRO HRANÍ HER

ARTIFICIAL INTELLIGENCE FOR GAME PLAYING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **VÁCLAV BAYER**
AUTHOR

VEDOUCÍ PRÁCE — **Doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2017

Abstrakt

Práce se zabývá metodami umělé inteligence aplikovanými pro hraní strategických her, ve kterých probíhá veškerá interakce v reálném čase (tzv. real-time strategic – RTS). V práci se zabývám zejména metodu strojového učení Q-learning založenou na zpětnovazebním učení a Markovovu rozhodovacím procesu. Praktická část práce je implementována pro hraní hry StarCraft: Brood War. Mnou navržené řešení, implementované v rámci pravidel soutěže SSCAIT, se učí sestavit optimální konstrukční pořadí budov dle hracího stylu oponenta. Analýza a vyhodnocení systému jsou provedeny srovnáním s ostatními účastníky soutěže a rovněž na základě sady odehraných her a porovnání počátečního chování s výsledným chováním natrénovaným právě na této sadě.

Abstract

The focus of this work is the use of artificial intelligence methods for a playing of real-time strategic (RTS) games, where all interactions of players are performed in real time (in parallel). The thesis deals mainly with the use of machine learning method Q-learning, which is based on reinforcement learning and Markov decision process. The practice part of this work is implemented for StarCraft: Brood War game. A proposed solution learns to make up an optimal order of buildings construction in respect to a playing style (strategy) of the opponent(s). The solution is proposed within the rules of the SSCAIT tournament. Analysis and evaluation of the proposed system are based on a comparison with other participants of the competition as well as a comparison of the system behavior before and after the playing of a set of the games.

Klíčová slova

umělá inteligence, StarCraft: Brood War, Q-učení, posilované učení, strategické hry v reálném čase, SSCAIT

Keywords

artificial intelligence, StarCraft: Brood War, Q-learning, reinforcement learning, Real-Time Strategy games, SSCAIT

Citace

BAYER, Václav. *Umělá inteligence pro hraní her*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Smrž Pavel.

Umělá inteligence pro hraní her

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Pavla Smrže, doc. RNDr., Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Václav Bayer
18. května 2017

Poděkování

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant, apod.).

Obsah

Kapitola 1

Úvod

Tato práce se zabývá uplatněním konceptů umělé inteligence pro hraní strategických her, ve kterých probíhá veškerá interakce v reálném čase (Real-Time Strategy, dále jen RTS). RTS hry spolu se svým vznikem v 90. letech, přinesly řadu nových zajímavých problémů a výzev, jenž do té doby žádný jiný žánr her nebyl schopen nabídnout, tudíž se současně staly oblíbeným testovacím prostředím v oblasti umělé inteligence.

Úkolem práce bylo zvolit si soutěž a v rámci zvolené soutěže implementovat herní strategii a nezbytná rozhraní pro zařazení do dané soutěže spolu s využitím konceptů umělé inteligence. Po dohodě s vedoucím byla zvolena soutěž *SSCAI Tournament* pro RTS hru *StarCraft* s aplikací metody strojového učení *Q-learning*.

1.1 Motivace

Komplexní dynamické prostředí, u kterého není k dispozici kompletní informace o současném stavu nebo o dynamice daného prostředí, je výzvou pro umělou inteligenci (AI). RTS hry zjednodušují dynamiku reálných situací (pohyb, ekonomiku, počasí apod.) do redukovaného a konečného prostředí. Prostředí RTS her může být však stále dostatečně komplexní k simulaci a studiu klíčových problémů, jako je plánování v reálném čase nebo rozhodování v rámci nejistoty. Z porozumění a nalezení řešení těchto problémů v RTS hrách může těžit nejen mnoho oblastí umělé inteligence, ale poznatky mohou být užitečné pro systémy analyzující přímo dané reálné situace.

RTS hry jsou podžánrem strategických her, mezi typické zastupitelé patří série her *Warcraft*, *StarCraft*, sérii *Age Of Empires*, nebo hra *Dune II* a další. Hraní RTS her, jako je například *StarCraft*, se díky jejich důmyslnému zpracování a perfektní vyváženosti stalo v mnoha zemích sportem. Hraní této hry na sportovní úrovni lze uznat rovnocenným způsobem výdělku ve srovnání s fyzičtějšími sporty. Profesionální hráči dostávají tituly celebrit s roční finanční odměnou ve stovkách tisíc dolarů placených sponzory [17]. Vydáním hry *StarCraft II* v roce 2009, se hned první den prodalo přes milión kopií, což opět rozšířilo soutěživost v RTS hrách po celém světě.

Díky zajímavým vlastnostem RTS her se tento typ her v posledních letech stal velice oblíbeným tématem pro výzkumníky v oblasti umělé inteligence. Za těchto okolností vzniklo pro významnou RTS hru StarCraft několik soutěží, které si mimo jiné také kladou za cíl zdokonalit vývoj agentů hry StarCraft k poražení profesionálních lidských hráčů. Tyto soutěže jsou:

- AIIDE [14] - první StarCraft AI soutěž, organizována Benem Weberem roku 2010.

- Computational Intelligence in Games (CIG) [8]
- SSCAI Tournament [12] - tato bakalářská práce se zaměří zejména na tuto soutěž.

Není to dlouhá doba, co se v oblasti herní umělé inteligence podařilo sestrojit počítačové programy (agenty), které byly schopny porazit profesionální hráče v mnoha hrách, jako jsou *Šachy* (Deep Blue [16]), *Texas Holdem poker* (Cepheus [27]) nebo dokonce hra *Go* (AlphaGo [7]). Silné herní systémy můžou mít také komerční využití v softwarovém průmyslu jako zábava nebo tréninkové nástroje pro hráče. Pokročilejší AI neposkytují jen lepší zážitek ve hrách pro jednoho hráče, ale také poskytují dobrý trénink hráčům v rostoucím odvětví eSportů a profesionálního hraní her, což je samo o sobě multi milionářský průmysl [18].

1.2 Stanovení cílů a struktura práce

Hlavním cílem této bakalářské práce je tvorba umělého systému pro RTS hru StarCraft. Pro tvorbu systému je použita metoda zpětnovazebního učení - Q-learning. Pro vývoj je k dispozici řada frameworků a nástrojů, jak pro hru StarCraft, tak pro integraci principů strojového učení.

Tento hlavní cíl lze rozdělit na následující podcíle:

- Podcíl 1 - definování problémů a výzev AI RTS her. Zvolení konkrétních problémových oblastí, kterými se budeme následně zabývat, zvláště pak s ohledem na hru StarCraft.
- Podcíl 2 - definování možných přístupů a následná volba vhodného přístupu k řešení zvolených problémů AI RTS her (StarCraft) z podcíle 1.
- Podcíl 3 - volba vhodných nástrojů pro vývoj.
- Podcíl 4 - sladění prvků hry StarCraft se zvoleným přístupem z podcíle 2. Jejich transformace a modelování do vhodné podoby k manipulaci ze strany strojového učení.
- Podcíl 5 - synchronizace frameworků a principů hry StarCraft spolu s jeho modelovaným systémem z podcíle 4, integrace principů strojového učení a následné otestování.

S ohledem na stanovené cíle je práce systematicky členěna do těchto kapitol:

- Kapitola 2 - seznámení se s definicí a principy RTS her spolu s úvodem do problematiky výzkumu AI v oblasti tohoto typu her. Kapitola také obsahuje úvod do strojového učení, stručný popis zvolené hry StarCraft a porovnání této hry s ostatními hrami z pohledu strojového učení.
- Kapitola 3 - kromě uvedení bližších informací o strojovém učení jako je Markovského rozhodovací proces apod. se hlavně zaměřuje zvolené podoblasti strojového učení - zpětnovazební učení, která bude pro realizaci AI uplatněna.
- Kapitola 4 - věnuje se návrhu tvorby systému pro hru StarCraft, zejména popisuje hlavní myšlenku aplikace metody Q-learning na specifickou problémovou oblast hry StarCraft a modelování hry do Markovova rozhodovacího procesu pro tuto metodu.
- Kapitola 5 - je zaměřena na popis implementace systému.
- Kapitola 6 - porovnává výsledný systém s ostanímy účastníky soutěže SSCAIT a provádí test systému.

Kapitola 2

Analýza

2.1 Real-time strategické hry

Real-time strategické hry se staly jedním z nejoblíbenějších žánrů her zhruba od začátků 90. let minulého století. Prvním průkopníkem tohoto typu her je hra *Dune II* (vydaná roku 1992), která již tehdy zahrnovala veškeré komponenty, které definují dnešní RTS hry.

Názvem RTS hry se označují hry, ve kterých hráč svým strategickým rozhodováním a taktikou ovlivňuje průběh hry tak, aby co nejlépe zoptimalizoval svůj stav nebo dosáhl zadaných úkolů. K dosažení daných úkolů je většinou potřeba jednotek, které vykonávají příkazy hráče, jenž jim v roli komandéra vydává povely. Tyto jednotky však nejsou zadarmo, k jejich produkci je potřeba mít k dispozici zdroje (např. ve formě peněz, zlata, jídla, dřeva apod.) specifikované danou hrou. Ačkoli je RTS her nepřeberné množství, princip ve většině případů ve výsledku zůstává stejný či podobný: co nejrychleji vytvořit těžební systém pro získání zdrojů, z těchto zdrojů postavit vojenský základ (armádu a technologie) a pomocí těchto sil zabrat více zdrojů a zároveň vyhladit nepřítelovy jednotky a základny.

Z teoretického pohledu hlavní rozdíly mezi RTS hrami a klasickými deskovými hrami, jako například *Šachy*, jsou:

- Tahy jsou vykonávaný souběžně, kde více jak jeden hráč může vykonat akci ve stejný čas. Tyto akce nejsou vykonány instantně, ale zaberou určitý čas.
- RTS hry jsou „real-time“, což znamená, že každý hráč má velmi krátký čas na vykonání rozhodnutí následujícího tahu, čím dříve se rozhodne, tím lépe pro něj. V porovnání s *Šachy*, kde hráči mají několik minut na rozmyšlenou jejich tahu, u hry StarCraft dochází k vykonání 24 snímků za sekundu, což znamená, že hráč může vykonat akci každých 42ms [25].
- U RTS her je hráč limitován v rozhledu mapy, může na mapě vidět jen již prozkoumané oblasti. Tento fakt je znám jako tzv. „fog-of-war“.
- Většina RTS her je nedeterministická, následující krok nemusí být jednoznačně určen. Některé akce mají šanci na úspěch, některé ale úspěchem skončit nemusí.
- Hlavním rozdílem je komplexnost RTS her, vyjádřena jak enormním množstvím stavového prostoru, tak i velkým množstvím možných akcí v každém stavu. Například u hry *Šachy* je stavový prostor stanoven zhruba na 10^{50}, u hry Texas Holdem poker okolo 10^{80} a u hry GO okolo 10^{170}. V porovnání se hrou StarCraft dochází k značnému rozdílu. Uvažme například mapu o velikosti 128×128. V jednom okamžiku

se na mapě může vyskytovat od 50 do 400 jednotek, kde každá jednotka může mít komplexní stav vyjádřený mnoha faktory. Na příkladu uvážení možných rozmnístění 400 jednotek (128×128 možných pozic pro každou jednotku) na mapě, se lze dobrat k číslu $16384^{400} \approx 10^{1685}$ [25].

RTS hry díky své komplexnosti poskytují značnou příležitost výzkumníkům v oblastech umělé inteligence, jako jsou multiagentní systémy, plánování, optimalizace apod. Získané poznatky v tomto testovacím prostředí lze uplatnit například ve skutečných vojenských i civilních operací a mnoha dalších oblastech.

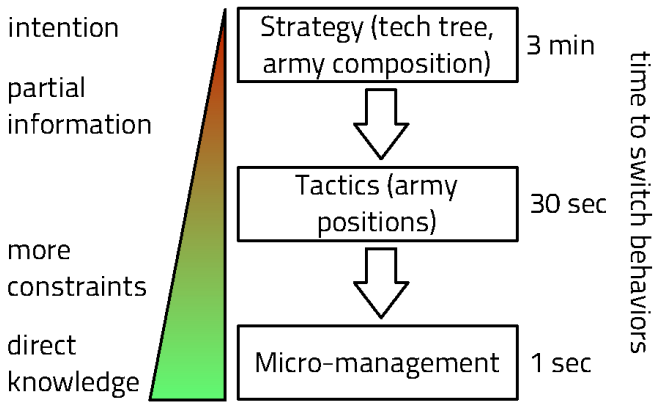
2.2 Problémové oblasti AI RTS her

RTS hry jsou z hlediska umělé inteligence zajímavé díky svému velkému množství problémů. Výzkum AI RTS her se zabývá následujícími problémy [15]:

- Správa zdrojů.
- Rozhodování s neurčitostí.
- Prostorové a časové uvažování.
- Spolupráce a kooperace systému (v případě více než jednoho systému).
- Modelování oponenta a učení se z jeho chování.
- Provádění plánování v reálném čase.

Zatímco ve většině z těchto výzev byla z pohledu výzkumu odvedena spousta práce, některé z nich jsou stále skoro nedotčené (zejména spolupráce a kooperace systémů). Nicméně kvalitní systémy hrající RTS hry by se měly zabývat většinou z výše uvedených problémů. Pokrýt všechny tyto zmíněné problémy v oblasti výzkumu RTS AI však není jednoduché, je vhodnější se zaměřit na určitou část problémových oblastí, kde dochází k jejich propojení. K definování těchto oblastí se využívá následujících přístupů [18] (viz. Obrázek 2.1):

- Strategie - nejvyšší úroveň abstrakce, která odpovídá nejširšímu strategickému rozhodnutí, jenž hráč může v RTS hrách vykonat. Strategie ovlivňují hru jako celek a jsou složeny z vysokoúrovňových dlouhodobých akcí. Nalezením optimální protistrategie proti strategii oponenta je klíčovým problémem v RTS hrách. Podproblémy, kterými se strategie zabývá, jsou:
  - Znalost a učení
  - Modelování oponenta a předpověď jeho chování
  - Styl hraní (vyvážení mezi ekonomickou expanzí a bojovou agresí)
  - Složení armád
  - Konstrukční pořadí budov
- Taktiky - implementují strategii a hlídají splnění strategických cílů. Časová náročnost je obvykle do půl minuty. Taktiky jsou zaměřeny na ovládání skupin jednotek, zvláště vojenských na bitevním poli. Podproblémy taktik jsou:
  - Průzkum



Obrázek 2.1: Dělení RTS AI podle úrovně abstrakce a jejich vlastností spolu s časovými nároky jednotlivých úrovní. Tyto uvedené časové nároky odpovídají době trvání ve hře StarCraft. Převzato z [19].

  - Časování útoků a pozic
  - Rozmístění budov

- Micromanagement - zabývá se specifickými akcemi na úrovni jednotek. Akce na této úrovni určují, jak se jednotky budou pohybovat a útočit. Reaktivní ovládání je implementací taktik, stará se o naplnění cílů taktik. Dělení tohoto přístupu na jednotlivé podproblémy:
  - Akce jednotek
  - Hledání cesty
  - Analýza terénu

2.3 StarCraft: Brood War

Real-time strategickou hru StarCraft: Brood War lze zařadit do žánru military science fiction. Po vydání této hry v roce 1998 se společnost Blizzard těšila velkému úspěchu, dokonce se StarCraft zařadil mezi jednu z nejprodávanějších her pro osobní počítače v historii [31].

Hra funguje na principu získávání nezbytných zdrojů (konkrétně minerály a plyn) pro stavbu budov, jak vojenských, tak užitkových [31]. Budovy umožnují produkci jednotek a výzkum pro zlepšení jejich vlastností. Zdroje pro produkci jednotek, stavbu budov nebo zkoumání vylepšení jsou ale ve hře limitovány a tak vzniká priorita zabrat co největší možné množství zdrojů a znemožnit k nim přístup nepříteli. Umožňuje-li to tedy populační limit a množství dostupných surovin, za pomocí produkce vojenských jednotek v kombinaci s výzkumy ke zdokonalení a posilnění těchto jednotek, se hráč snaží oddělit nepřítele od surovin a následně díky této surovinové převaze vojenskou silou dosáhnout vítězství.

Hráč může ovládat během hry až 200 jednotek a neomezené množství postavených budov. Jednotkám jsou přidělovány různé aktivity, jako je průzkum, obrana, útok, stavba, opravování a sběr surovin. Hráč se během zadávání těchto povelů a rozkazů jednotkám,

musí být také schopný orientovat v mapě, tak aby věděl, kde strategicky postavit budovy a rozmístit jednotky pro výhodu v boji. Do těchto strategických rozhodování je také třeba počítat s možností bitvy s nepřítelem, kde hráč musí manévrovat s jednotlivými jednotkami a současně jim zadávat příkazy. Všechny tyto faktory spolu vytváří hru StarCraft jednoznačnou výzvou, ve které jsou lidé stále lepší než stroje.

Hráči mají pro hraní na výběr ze tří možných ras. Jeden z důležitých aspektů hry je ten, že všechny rasy jsou velice dobře vyvážené, i když se navzájem velice liší. Tyto rasy se dělí následně:

- Protoss - velmi vyspělá a inteligentní rasa. Disponuje silnými a odolnými jednotkami, které však také mají vyšší nároky na zdroje.
- Zerg - rasa stvořená již zmíněnou rasou Protoss. Její jednotky jsou sice slabší a mají nejnižší výdrž a odolnost v porovnání s ostatními rasami, avšak na druhé straně jsou jednotky velice levné. Styl hraní za tuto rasu spočívá v přečíslení nepřítele.
- Terran - rasa s lidskými rysy. Jsou zlatou střední cestou mezi ostatními dvěma frakcemi.

Každá rasa se vyznačuje svými specifickými jednotkami, budovami, vylepšeními, příběhem a výčtem možných herních strategií. Víceméně, každá z uvedených frakcí obsahuje 30 až 35 rozdílných jednotek a budov, kde většina z nich má určitý počet speciálních schopností. Vzhledem k rozdílnosti každé rasy v těchto mnoha ohledech, je vhodné zvolit pro vývoj umělého systému jen jednu frakci a na tu se více zaměřit. Pro účely této bakalářské práce byla vybrána rasa Protoss. Práce je tedy zaměřená na tuto frakci, její jednotky a budovy. Avšak hlavní principy začlenění strojového učení do hry by však měly zůstat stejné, nebo minimálně velmi podobné pro obě zbývající frakce.

2.4 Soutěž SSCAI Tournament

Pro porovnání výsledného systému s ostatními systémy byla zvolena soutěž SSCAI Tournament [12]. Tato soutěž je hostována universitami České vysoké učení technické v Praze a Universita Komenského v Bratislavě v této oblasti je velmi známa.

Soutěž byla spuštěna poprvé roku 2011 a slouží jako testovací prostředí převážně pro studenty zaujaté problematikou umělé inteligence. Disponuje svými webovými i facebookovými stránkami a aktivní komunitou. Zahrnuje jasné pravidla, velké množství účastníků a základní tutoriály. Po přihlášení AI do soutěže je AI následně zařazena do 1v1 soubojů proti ostatním zúčastněným systémům. Veškeré probíhající souboje jsou veřejně streamovány ve službě Twitch.

K vývoji umělého systému je na výběr ze dvou jazyků, Java nebo C++. Tato volba se odvíjí od zveřejněných frameworků, které jsou poskytnuty pro zpřístupnění možnosti interakce s rozhraním hry StarCraft. Do těchto frameworků se řadí BWAPI [11] (The Brood War Application Programming Interface) pro jazyk C++ a BWMirror. Frameworky JNIBWAPI a BWMirror slouží jako Java wrapper k frameworku BWAPI. My použijeme k vývoji jazyk Java spolu s frameworkem BWMirror, vývojové prostředí je Intellij IDEA.

Následně je třeba mít nainstalováno spolu se hrou StarCraft také program Chaoslauncher, který slouží jako most mezi běžícím klientem AI v IDE a hrou. Díky pomocí tohoto programu dojde při spuštění programu v IDE k jeho následné projekci do hry.

Vývoj probíhá pod operačním systémem Windows 7, je možné použít i Windows XP, Linuxové distribuce použít nelze, jelikož program Chaoslauncher není pro tento typ OS oficiálně podporován.

Dalšími důležitými nástroji jsou framework Atlantis pro snadnější implementaci principů hry StarCraft a knihovna BURLAP pro implementaci strojového učení v jazyce Java. Atlantis poskytuje abstrakci instrukcí z BWAPI a BWMirror ve formě zhotovených manažerů, kteří spravují určité okruhy hry jako je např. manažer pro stavbu, manažer pro produkci jednotek, manažer pro průzkum atd. Tento framework je použit k získání abstrakce nad akcemi jednotek z důvodu snazšího modelování hry pro strojové učení. Po následné modelaci jednotek, je třeba právě tyto jednotky integrovat do konceptu a celkového principu frameworku Atlantis, stejně tak i principy strojového učení. V tomto okamžiku je potřeba zkoordinovat chování všech tří frameworku: BWMirror a Atlantis pro hru StarCraft spolu s BURLAP pro strojové učení. Této problematice se věnuje podrobněji sekce 4.2.

Souhrnný soupis nezbytných systémových požadavků pro vývoj AI pro hru StarCraft:

- StarCraft: Brood War, verze 1.16.1 [1]
- balíček map pro StarCraft [13]
- ChaosLauncher [13]
- BWAPI [11]
- BWMirror [5]/JNIBWAPI
- 32 bit JRE (Java Runtime Environment) [10]
- Intellij IDEA [9]/Eclipse IDE [3]
- Mircrosoft Windows XP nebo Windows 7

Volitelné:

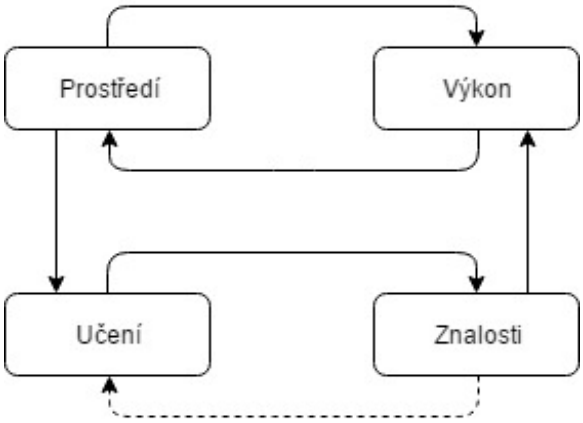
- BURLAP [2]
- framework pro vývoj SC AI - Atlantis [4]

2.5 Strojové učení

Strojové učení lze definovat jako schopnost inteligentního systému měnit své znalosti tak, že při příštím vykonávání stejné, nebo podobné úlohy je tento daný systém schopen vykonat úlohu efektivněji [33].

Tato podoblast umělé inteligence se tedy zabývá technikami a algoritmy, které umožnují umělému systému „učit se“. Učení v tomto případě znamená změnu vnitřního stavu systému, která zefektivní schopnost přizpůsobit se změnám okolního prostředí [28]. Učení může být ovlivněno již naučenými znalostmi, které ovlivňují zpětný proces učení, tento vztah lze vidět na obrázku 2.2, kde jej symbolizuje přerušovaná čára. Obrázek znázorňuje princip interakce mezi učením, znalostí, výkonem a prostředím. Učení ovlivňuje znalosti a tyto znalosti ovlivňují výkon, který je zároveň ovlivňován i zkušenostmi z prostředí. Čím víc je výkon navýšen, tím lépe pro daný systém.

V následujících odstavcích je vysvětleno do jakých podoblastí se strojové učení dělí. V této bakalářské práci se zaměříme hlavně na zpětnovazební učení a jeho metodu Q-učení.



Obrázek 2.2: Znázornění principu interakce mezi učením, znalostí, výkonem a prostředím

Učení s učitelem. Hlavní myšlenkou je definovat funkci tak, aby k určitým vstupním datům našla správný výstup. Jak již název napovídá, tento typ strojového učení si lze představit na příkladu žáka a učitele. Na základě otázek učitele se žák snaží, díky svým znalostem (vstup), správně odpovědět (výstup) na danou otázku. Učitel je schopný říci žákovi, zda odpověď na jeho otázku byla správná, či nikoli. Neuronové sítě na tomto principu získávají zpětnou vazbu, kde pomocí těchto výsledků a případné velikosti chyby upravují hodnoty vah tak aby v následných kolech byla tato chyba zredukována.

Učení bez učitele. Proti předešlému případu nemáme k dispozici posudek, zda odpověď (výstup) na danou otázku je správná, nebo ne, tudíž výstup se nevyhodnocuje. U neuronových sítí je se vstupem dostupná i sada vzorů, které si neuronová síť sama organizuje a třídí.

Zpětnovazební učení. Zpětnovazební učení [26] je oblast strojového učení, jenž má za úkol se chovat tak, aby bylo dosaženo maximální hodnoty odměny. Učícímu se objektu není řečeno jakou akci má vykonat, jako ve většině podob strojového učení, místo toho má za úkol zjistit, která akce navrací maximální odměnu, tím že bude zkoušet jednotlivé akce. V nejvíce zajímavých a vyzývavých případech se ocitáme ve chvíli, kdy akce nemusí ovlivnit jen okamžitou odměnu ale i následující situaci a díky tomu veškeré následující odměny.

2.6 Multiagentní systémy

Vznik a vývoj agentních systémů navazuje na distribuovanou umělou inteligenci (Distributed Artificial Intelligence, DAI). V DAI se vyskytují autonomní jednotky (Actors), které jsou schopné řešit některé problémy. Podle těchto jednotek se vyvinul pojem agenti (Agents). Systém, ve kterém se nachází více volně propojených agentů, kteří vzájemně spolupracují v zájmu dosažení společného cíle, se nazývá Multiagentní systém (Multi-agent system, MAS). Stejně tak jako je u týmové práce, skupina agentů je díky paralelnímu přístupu schopna řešit efektivně problém, taktéž lze očekávat snížení nároků ke komunikaci, protože jednotliví členové, podílející se na řešení daného problému, si nepředávají dál všechny zjištěné informace, ale především své závěry (na rozdíl od centrálně řízeného distribuovaného systému) [24].

Charakteristické rysy multiagentního systému:

- Data jsou decentralizovaná.
- Výpočty jsou prováděny asynchronně.
- Neexistuje globální řízení systému.
- Každý agent má jen částečnou a nekompletní informaci pro řešení systému.

Prvky multiagentních systémů:

- Agenti - hlavní aktéři v multiagentních systémech. Agenti se v každém okamžiku nalézají v určitém stavu, označovaném jako lokální stav, který obsahuje veškerou jim dostupnou informaci, kterou je možné označit jako znalost. Vývoj systému v diskrétním čase pak popisuje globální stav.
- Prostředí - vše, s čím agent přichází do styku a je relevantní pro řešenou úlohu.

Kapitola 3

Zpětnovazební učení

Zpětnovazební učení je oblast strojového učení, ve které se agent musí učit, na základě pokusů a omylů, optimálním akcím v konkrétních situacích, voleným tak, aby bylo získáno maximální celkové odměny [26]. Skrz mnoho iterací slabého učení s učitelem, zpětnovazební učení může objevit nové a chytřejší řešení problémů než bylo řešení předešlé. Je relativně snadné jej aplikovat pro novou doménu, jelikož vyžaduje jen popis situace v podobě definování možných akcí a odměnových hodnot. Nicméně, v tak komplexním prostředí RTS her jako je StarCraft, zpětnovazební učení vyžaduje chytré abstraktní vyjádření akcí a stavů. Tato technika je využívána v RTS hrách pro rozhodování na úrovni taktik častěji, než pro rozhodování na úrovni strategií, jelikož na úrovní strategií se potýká s širokým záběrem problémové oblasti hry a také s opožděnou odměnou, které komplikují použití zpětnovazební učení. Avšak, i přes tento fakt, je práce zaměřena na aplikaci zpětnovazebního učení pro strategické rozhodování, zejména k volbě konstrukčních pořadí budov v souvislosti s daným stavem hry.

3.1 Markovův rozhodovací proces

Markovské rozhodovací procesy (MDP) [29] jsou využívány v potřebě modelovat rozhodování za podmínek, kdy jsou výsledky z části náhodné a z části kontrolovány uživatelem. Markovův rozhodovací proces se skládá ze čtyř částí:

- stavového prostoru S,
- funkce A, která poskytuje možné akce pro každý stav,
- přechodová funkce T
- a užitková funkce R.

Náhodná proměnná označující stav v čase t je S_t, a aktuální stav v čase t je s_t. Stav v čase $t+1$ záleží na stavu v čase t a na akci a_t provedenou v čase t. Tato závislost je vyjádřena přechodovou funkcí T, neboli $T(s_t, a_t) = T_{s+1}$, jenž značí stav v čase $t+1$. Díky faktu, že počet stavů je konečný, přechody mohou být vyjádřeny pravděpodobností, tudíž $T(s, a)$ může vrátit vzorek z pravděpodobnostního rozložení přes S.

Pro vyjádření pravděpodobnosti provedení akce a ve stavu s pro transformaci s do stavu s' můžeme použít:

$$p(s'|s, a) = Pr(T(s, a) = s'). \tag{3.1}$$

Tato množství se nazývají přechodové pravděpodobnosti. Obdobně při poskytnutém aktuálním stavu s a akci s, společně s jakýmkoli následujícím stavem s', očekávaná hodnota další odměny je

$$r(s, a, s') = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s']. \tag{3.2}$$

Tato množství, $p(s'|s, a)$ a $r(s, a, s')$ vyjadřují důležité aspekty dynamiky konečného MDP. Přechodové pravděpodobnosti jsou obvykle nazývané modelem prostředí. Pro některé metody zpětnovazebního učení musí být model znán, avšak pro metodu Q-učení není model explicitně uveden.

3.2 Rozhraní agent-prostředí

Jak Sutton zmiňuje v [26], objekt, který se ve strojovém učení učí a rozhoduje, je nazýván *agent*. Vše co se nachází mimo agenta je *prostředí*. Pro lepší představu můžeme vidět na obrázku 3.1 agenta interagujícího s prostředím tím způsobem, že v každém časovém kroku t (krok ze sekvence diskrétních časových kroků $t = 0, 1, 2, 3, ...$) vykonává *akci*, $A_t \epsilon A(S_t)$, kde $A(S_t)$ je set akcí dostupných ve stavu S_t. Prostředí na tyto akce reaguje dvěma způsoby.

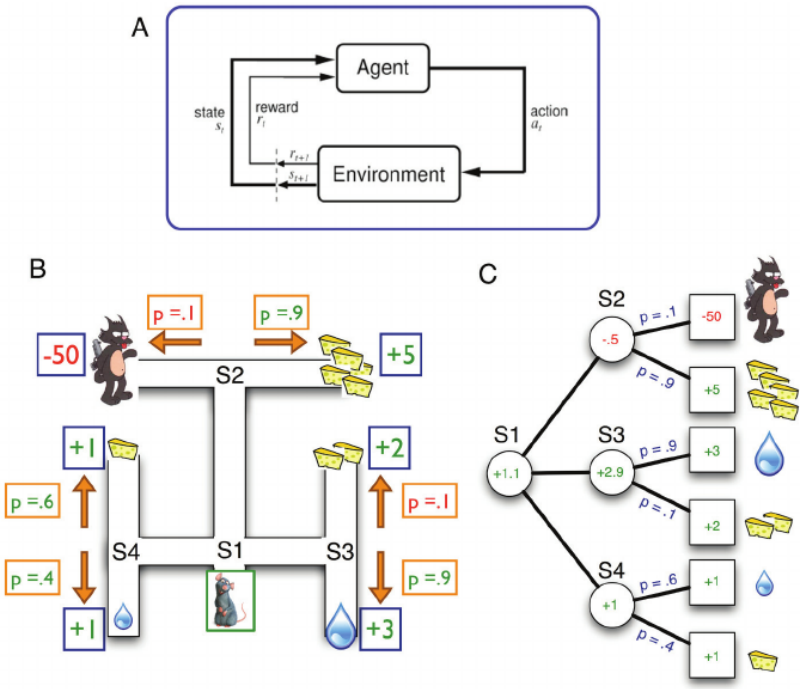
- První způsob je, že prostředí následnou reakcí na akci mění sebe sama a vytváří tím nové situace pro agenta. Tyto situace nazýváme *stavy*, $S_t \epsilon S$ kde S je množina možných stavů, a na základě této informace se agent rozhoduje, jakou akci vykoná. Po vykonání akce se tedy ocitne v novém stavu S_{t+1}.
- Druhým způsobem předává agentovi zpětnou reakcí *odměnu* R_{t+1}, číselnou hodnotu, na jeho zvolenou akci. Ve zpětnovazebním učení je cílem agenta maximalizovat celkové množství získané odměn, to znamená, že nedochází k maximalizaci jen dočasné odměny ale hlavně maximalizaci odměny z dlouhodobého hlediska.

3.3 Q-learning

Při učení se optimální strategii v interakci s prostředím se může stát, že T a R funkce mohou být neznámé, jelikož v posilovaném učení je model obvykle neznámý [23]. K řešení tohoto problému se používá dvou přístupů. *Modelový* přístup se nejdříve snaží naučit model a potom použít metody dynamického programování k sestavení optimální strategie s ohledem na vytvořený model. Druhou možností je *bez modelový* přístup, který se soustředí na učení se hodnotám stavů (nebo Q-hodnotám, viz.dále) přímo a sestavuje optimální strategii na základě těchto hodnot.

My se v této kapitole zaměříme na metodu Q-learning [29][30][26][20] jenž je metoda zpětnovazebního učení, která nepotřebuje model prostředí a ve které lze uplatnit *bezmodelového* přístupu. Metoda je *off-policy* tzn., že algoritmus se učí optimální hodnotu funkce nezávisle na strategii, kterou se při učení řídí. Princip metody spočívá v tom, že se neohodnocují stavy ale akce a vykonané ve stavu s v čase t a současně se determinuje optimální strategie pro MDP. Zmíněné ohodnocení stavů se značí $Q(s_t, a_t)$, vzorec potom je:

$$Q(s,a) \leftarrow \underbrace{Q(s,a)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \left[\overbrace{\underbrace{r}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_{a'} Q(s',a')}_{\text{max future value}}}^{\text{expected discounted reward}} - \overbrace{Q(s,a)}^{\text{old value}} \right] \tag{3.3}$$



Obrázek 3.1: Demonstrace interakce agenta s prostředím ve zpětnovazebním učení, část A. Agent reaguje s prostředím vybíráním akcí, na které získává odpověď v podobě stavů a odměny. Na obrázku v části B můžeme tuto interakci vidět na příkladu, kde myš hledá cesty v bludišti a získává různé odměny. Znaménkové hodnoty (+ a −) jsou odměnové veličiny v odlišných koncových bodech. Hodnoty p vyjadřují pravděpodobnost přechodu do cílových bodů. Stavy jsou značeny od $S1$ do $S4$. Třetí část obrázku, C, zobecňuje předešlou část B do série propojených stavů. Převzato z [21].

kde r_{t+1} je obdržená hodnota, parametr α určuje stupeň učení ($0 < \alpha \leq 1$), a γ je discount faktor ($0 \leq \gamma < 1$), který určuje vliv ohodnocení následujících stavů s' na ohodnocení stavu s.

S ohledem na Q-hodnoty, při dosažení stavu s, algoritmus Q-učení typicky vybírá „greedy“ akci a. Tato akce a má nevyšší hodnotu $Q(s, a)$ s pravděpodobností $1 - \varepsilon$ a vybírá náhodnou akci s pravděpodobností $\varepsilon (0 < \varepsilon < 1)$. Obyčejně ε je nastavena na velmi nízkou hodnotu (tj. blízko nule), což způsobuje, že greedy akce je většinou vybrána. Tato metoda výběru mezi greedy akcí a náhodnou akcí je nazývána ε-greedy metoda a povoluje Q-learningu přeskakovat lokální maxima.

Kapitola 4

Návrh systému pro hru StarCraft

V předešlých kapitolách bylo zmíněno co to je strojové učení a jakou jeho metodu použijeme k vývoji umělého systému. V této kapitole jsou teoretické poznatky s dřívějších kapitol uplatněny v praxi pro návrh systému pro hru StarCraft. Jsou zde podrobněji popsány nástroje uvedené v sekci 2.4, které jsou zapojeny do procesu vývoje umělého systému. Kromě definice nástrojů, je zde popsáno modelování hry StarCraft do Markovského rozhodovacího procesu, kterému byla věnována kapitola 3.1.

4.1 Volba strategie

V této práci není definována striktní strategie hry StarCraft a tvořený systém tudíž není zaměřen jen na jednu strategii. Záměrem této práce je totiž použití metody Q-learning za účelem dedukce ideálního konstrukční pořadí budov a dynamické volby těchto pořadí za běhu hry.

Tato řešená problémová oblast RTS her se tedy nachází na nejvyšší úrovni rozhodování, kde se od taktik přechází ke strategiím, ale tyto rozhodnutí jsou vykonávány na úrovni jednotek. To znamená, že rozhodnutí nevykonává agent v podobě manažera, který by své rozhodnutí pak přidělil dělníkovi ve hře, rozhodnutí jsou totiž vykonávána přímo na úrovni agentů dělníku, jejichž počet je pro demonstraci multiagentního rozhodování v tomto systému implicitně definován na dva dělníky. Pro takové použití metody strojového učení je potřeba vytvořit odpovědné rozhraní mezi hrou a strojovým učením. Řeč je o modelování hry do MDP, konkrétně agenta dělníka, jehož funkcionalita a konkrétní modelované prvky jsou popsány v sekci 4.3.

I přes to, že akce agenta dělníka jsou limitovány kvůli úspoře stavového a akčního prostoru (popsáno více v následném textu), jsou zvoleny tak, aby bylo možné dobrat se k produkci vojenských jednotek jako je *Zealot* či *Dragoon* a také možnosti jejich vylepšení pomocí výzkumu. Ze známých strategií a taktik pak tato kombinace umožňuje aplikovat následující:

- Zealot rush - spočívá ve vysílání vln jednotek *Zealot*. Při správném načasování a síle vln, se jedná o velmi efektivní a pro nepřítele nebezpečnou taktiku.
- Protoss mix - navazuje na taktiku *Zealot rush*. Jde o kombinaci jednotek *Zealot* s jednotkami *Dragoon*. Cílem je díky ekonomičtějšímu stylu hry dosáhnout vylepšení jednotek *Dragoon* pro větší dostřel. Kombinace těchto jednotek je velice silná.

- Zealot hell - opět navazující taktika k taktice *Zealot rush*. Cíl spočívá v expanzi a stavby nového centra, pro znásobení těžby surovin, jelikož si taktika klade za cíl maximálně vylepšit jednotky *Zealot*, jak pro obranu a útok, tak i pro zrychlení pohybu.

Zmíněné taktiky nejsou nijak detailně implementovány, o produkci jednotek a micromanagement se stará výchozí chování nástroje Atlantis (viz. 4.2). Tyto taktiky jsou uvedeny hlavně pro demonstraci potenciálního průběhu hry při optimálním rozhodování agentů dělníků. Zkušenější hráč StarCraftu dokáže usoudit, že se jedná o základní taktiky, které otevírají hru do náročnějších a pokročilejších fází hry.

4.2 Úrovně akcí z pohledu nástrojů

Problémové oblasti RTS her byly kategorizovány do tří úrovní: strategie, taktiky a micromanagement. Akce tedy mohou nabývat mnoha úrovní, od elementárních, nízko úrovňových *micro* akcí, přes taktické akce na střední úrovni rozhodování, až po vysokoúrovňové *macro* akce pro volbu strategických tahů. Kapitola o strojovém učení uvádí, že agenti interagují s prostředím volbou akcí, jenž mají být vykonány. Tyto akce musí být pro interakci s agentem modelovány v MDP. Pro proces modelování je vhodné určit jaké akce, a na jaké úrovni, budou modelovány, současně s ohledem na možnosti nástrojů, které jsou k dispozici.

Micro akce v *micromanagementu* ovládají pohyb jednotek a jsou důležité zejména v bitvách. V těchto situacích velmi záleží na každém pohybu každé jednotky, zvláště pak při větším počtu jednotek, kde se cení každá ušetřená setina sekundy. V příkladu dvou proti sobě bojujících armád o stejných velikostech, kde pohyb jednotek jedné armády je optimalizován určitou metodou, algoritmem či technikou, kdežto druhá armáda má defaultní neoptimalizované chování hry StarCraft, lze spatřit rozdíl již během prvních pár vteřin konfliktu. Neoptimalizované chování způsobuje často zbytečné prodlevy kvůli neideální kooperaci jednotek. Například při pohybu armády se jednotky s různou pohybovou rychlostí nejčastěji řadí do lajny, kde jsou právě jednotky s vyšší pohybovou rychlostí limitovány pomalejšími. Při enormním množství akcí vykonaném během krátkého okamžiku, právě v těchto detailech, lze získat v optimalizovaném chování s každým kooperovaným pohybem časovou výhodu. Celkový součet takto ušetřeného času se razantně projeví ve výsledku boje. Na téma modelování *combat* systémů pro RTS je zaměřeno spousta výzkumných prací, například Young [32] ve své práci prezentuje *combat* systém, jenž se učí na základě imitace her expertů, Churchill [18] popisuje systém SparCraft, který je založen na mnoha algoritmech jako je Alfa-Beta, UCT (Upper Confidence Bound) a další. Tato bakalářská práce se však hlavně zaměřuje na volbu akcí pro ovládání hry jako celku, tzn. kooperaci jednotek jak vojenských, tak i nevojenských, a to převážně na úrovni strategií a taktik, tudíž *combat* systém zde nebude probírán.

Pro získání přístupu k akcím hry jsou k dispozici následující frameworky: BWAPI, BWMirror a Atlantis. Frameworky BWAPI a BWMirror jsou základní nástroje k získání přístupu k rozhraní hry StarCraft, tudíž poskytují akce na nejnižší úrovni, jsou typickými zástupci *micromanagementu*. Tyto frameworky nabízí opravdu elementární akce rozhraní hry StarCraft. Mezi tyto akce lze uvést na ukázku:

- `move(Position target)` - přesune jednotku na danou pozici,
- `stop()` - přikáže jednotce zastavit,
- `gather(Unit target)` - pošle jednotku těžit daný cíl.

Akce jsou opravdu elementární a nejsou nijak ošetřeny, v případě zadání nesprávného parametru, například u funkce `gather(Unit target)`, kdy cílová jednotka nebude zdroj, který může být těžen, dojde k chybě. Aby se předešlo těmto komplikacím, je třeba vybudovat kontrolu o úroveň výš. Další problém je obrovské možné množství akcí nabízených těmito nástroji, je vhodné jej zredukovat do vyšší míry abstrakce pro snazší manipulaci. Řešení obou problémů nabízí framework Atlantis. Atlantis byl vytvořen za účelem vyhnutí se stavby umělého systému hry StarCraft od úplného začátku. Visuální porovnání těchto nástrojů na základě úrovně rozhodování můžeme vidět na obrázku 4.1. Atlantis disponuje širokým rozsahem úrovní možných akcí, od micromanagementu, až po strategické úkony. Zahrnuje komandéry a manažery, kteří spravují celé okruhy jako je produkce jednotek, stavba budov, průzkum, ovládání jednotek ve shlucích atd.

Pro lepší představu poslouží názorné příklady funkcí z Atlantis frameworku:

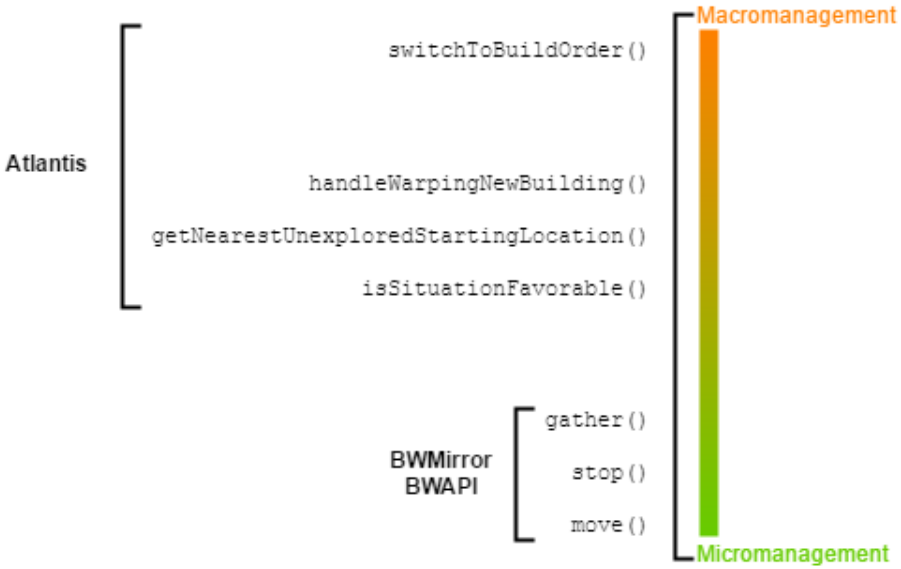
- `handleWarpingNewBuilding(AUnit newBuilding)` - užitečná funkce pro rasu Protoss, jelikož způsob stavby budov pro tuto rasu se liší od ostatních frakcí. Stavební manažer nepoužívá v tomto případě stavební frontu.
- `getNearestUnexploredStartingLocation()` - průzkum nejbližší neprozkoumané startovní pozice, za účelem odhalení nepřítele.
- `handleWarpingNewBuilding(AUnit unit)` - vyhodnotí bojovou situaci kolem dané jednotky a vrátí informaci, zda se jednotka má zapojit do boje, nebo by se měla stáhnout v případě, že nepřítel je silnější.
- `switchToBuildOrder(String order)` - tato akce, jenž mění strategický plán stavby budov, se liší od výše zmíněných akcí, které se vyskytovaly na úrovni *micro* akcí a *taktik*. Je to typický zastupitel nejvyšší možné úrovně - rozhodování na úrovni strategií. Touto funkcí původně framework Atlantis přepínal již pevně definované konstrukční pořadí budov, které byly načítány ze souboru při startu hry. Cílem této práce je nahradit tuto funkcionalitu tak aby se pořadí konstrukce budov měnilo dynamicky dle potřeby.

Vzhledem k vhodnosti výše zmíněných akcí pro splnění účelů této práce, se framework Atlantis jeví jako ideální nástroj pro vývoj systému, lépe řečeno je ideálním základem pro integraci metody strojového učení a potřebných rozšíření, jenž umožní jednotné fungování systému s ohledem na danou metodu. Je možné použít celý koncept nástroje Atlantis, téměř veškerou mechaniku a principy jako výchozí chování našeho systému, tedy až na manažery, kteří svým způsobem spravují a zaštiťují námi modelované jednotky (viz. 4.3). V tomto případě to je zejména stavební manažer, který bude zcela vypuštěn a nahradí jej nový manažer přizpůsobený k chování modelovaných jednotek ovlivněných strojovým učením.

4.3 Modelování Probe jednotky do MDP

Hru StarCraft lze dle účelnosti jednotek dělit do tří okruhů:

- Budovy - jejich účel se dále dělí na dva další směry, *výzkum* a *produkce jednotek*. Každá budova poskytující produkci jednotek s určitými vlastnostmi a speciálními schopnostmi, většinou umožňuje i možnost výzkumu, pomocí kterého lze vlastnosti a schopnosti jednotek vylepšit.



Obrázek 4.1: Grafické znázornění úrovní akcí jednotlivých frameworků.

- Jednotky vojenské - jsou nezbytností k dosažení vítězství. Rozlišující se na pozemní jednotky a létající. Kromě speciálních schopností se svým chováním navzájem moc neodlišují.
- Jednotky užitkové - pro účel této práce nejpodstatnější okruh. Oproti vojenským jednotkám, se jednotky v této skupině navzájem výrazně odlišují. Například, je-li možno jednotku *Shuttle* uvážit jako užitkovou a nevojenskou, pak při porovnání s dělníkem dochází k značnému rozdílu. *Shuttle* slouží jako letecký transportér pro ostatní jednotky, kdežto *Probe* (dělník) je prospěšný v získávání surovin, tudíž možnost užitku je značně rozdílná. Toto porovnání ale není v této skupině jedinečné, jelikož *Probe* se svým chováním také dělí na dvě role. *Probe* jednotku totiž lze považovat za sběrače nebo stavitele. Zatímco sběrač se stará jen o dodávání surovin, tedy jeho aktivita je cesta od budovy *Nexus* (centrum) ke zdroji (minerály, plyn) a zpět, stavitel musí být chytřejší, aby věděl, kterou budovu je zrovna třeba postavit v souvislosti s herní strategií a také v souvislosti s dostupnými surovinami. Modelovaná jednotka bude právě druhý případ, *Probe*-stavitele.

Zvoleným kandidátem pro modelování do MDP z výše popsaných okruhů je tedy *Probe*-stavitel. Za zmínku stojí, že ačkoli se na první pohled nejedná o významnou jednotku, opak je pravdou. Stavitel produkuje budovy, které následně produkují vojáky a výzkum, průběh hry se tedy odvíjí hlavně na základě akcí tohoto typu jednotek.

4.3.1 Reprezentace stavu

K redukci stavového prostoru, je pro funkce, které reprezentují mnoho hodnot, použito dělení do úrovní. Například suroviny se dělí do 18 úrovní, kde úroveň 1 znamená 0 jednotek dané suroviny a úroveň 18 znamená 4000 a více jednotek dané suroviny. U funkcí pro vyjádření počtu užitkových, vojenských jednotek a dostupného populačního limitu (kapacity) je dělení podobné, každá úroveň vyjadřuje zastoupení 5 jednotek, přičemž úroveň 1 opět vyjadřuje 0 jednotek, tudíž například počet 12 jednotek vojáků *Zealot* je zastoupen úrovní

4. Pro vyjádření počtu určitého množství budov se již úrovně nepoužívají, jelikož se nepředpokládá vysokých čísel (maximálně u budov *Pylon* se lze dobrat průměrného počtu dvou desítek během pokročilé fáze hry, ale jedná se spíše o výjimku). Funkce vyjadřující reprezentaci stavu pro jednotku *Probe*-stavitel jsou následující:

- Úroveň minerálů
- Úroveň plynu
- Úroveň kapacity
- Počet kasáren (*Gateway*)
- Počet základen (*Nexus*)
- Počet dolů na plyn (*Assimilator*)
- Počet budov *Pylon*
- Počet kováren (*Forge*)
- Počet věží (*Cannon*)
- Počet budov *Cybernetics Core*
- Úroveň našich bojových jednotek
- Úroveň našich padlých jednotek
- Úroveň zabitých nepřátelských jednotek

Poslední tři funkce seznamu pro vyjádření počtu jednotek se možná napohled zdají zbytečné, zejména funkce „Úroveň našich bojových jednotek“, jelikož tento počet se odvíjí hlavně od počtu kasáren, avšak frekvence produkce vojenských jednotek je také spjata s dostupnými surovinami, které jsou ovlivněny počtem těžících základen a dolů. Stejně tak funkce „Úroveň zabitých nepřátelských jednotek“ může mít souvislost nejen s počtem našich vojáků, kteří se snaží toto číslo maximálně navýšit, ale také s počtem věží na obranu základny. Podobných vyjádření a funkcí může být daleko více, pro účely práce postačí tyto.

4.3.2 Reprezentace akcí

Mnohokrát bylo zmíněno, že rozhodování probíhá na úrovni strategií, tomu budou odpovídat i akce *Probe*-stavitele, které se nacházejí na vysoké úrovni abstrakce. Tyto akce zahrnují lokaci vhodného místa pro stavbu, alokaci potřebných zdrojů, pohyb dělníka na určité místo, zahájení stavby apod. Použitím těchto vysoko abstraktních akcí vede k redukci velikosti Q-tabulky. Seznam možných akcí pro *Probe*-stavitele:

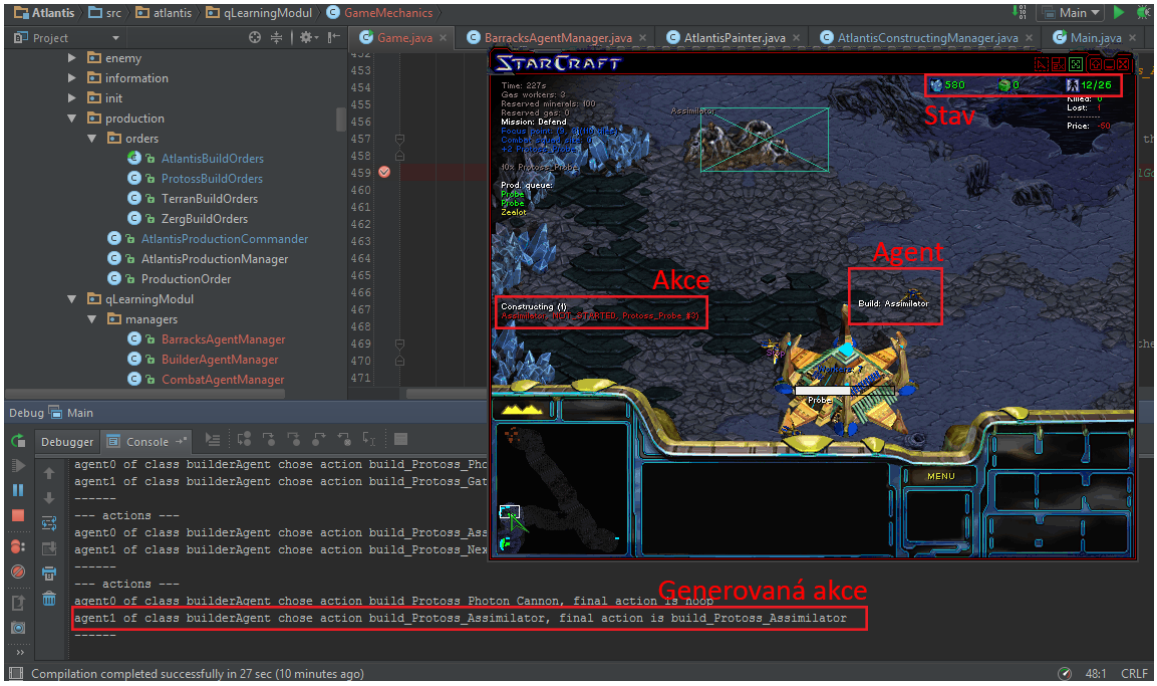
- Postav kasáreny (*Gateway*)
- Postav základnu (*Nexus*)
- Postav důl na plyn (*Assimilator*)
- Postav budovu *Pylon*

- Postav kovárnu (*Forge*)
- Postav věž (*Cannon*)
- Postav budovu *Cybernetics Core*
- Nekonej žádnou operaci

4.3.3 Odměny

Aby bylo dosaženo správného chování agentů v průběhu hry, je nutno je správně motivovat pomocí odměn. Na téma volby optimální strategie udělování odměn je publikováno nepřeberné množství publikací a článků a je testováno mnoho přístupů v různorodých prostředích. Například Matingon ve své práci [22] porovnává více přístupů udělování odměn agentovi, který se v podobě myši snaží v bludišti najít cestu k sýru (gól). V tomto případě, se myš může pohybovat jen čtyřmi směry a stavový prostor taktéž není veliký v porovnání s řešeným problémem v případě dělníka v RTS hrách, kde současně s širokým stavovým a akčním prostorem vzniká problémů celá řada. Například udělením odměny agentovi za to, že postavil kasárny, není nijak užitečné, jelikož v příští minutě hry bude třeba postavit další, jelikož máme přebytek surovin a první kasárna nestíhá produkovat vojáky. Nicméně v opačném nemůže být každá druhá postavená budova kasárnami. Určitě je dobré agenta motivovat gólem typu prohra či výhra celkové hry, ale tato informace agentovi moc neřekne, jelikož v průběhu hry bylo provedeno mnoho akcí v mnoho stavech. Jako řešení těchto problémů byly definovány odměnové funkce, které berou v potaz více faktorů v době konání zvolené akce, jako jsou čas a prostředky. Tyto dynamické funkce jsou:

- Supply - vrací kladnou odměnu agentovi vždy, když dochází volná kapacita (méně než produkuje jedna budova *Pylon*) a agent se snaží o stavbu budovy *Pylon* pro rozšíření této kapacity.
- Army production - tato funkce klade za cíl stavby kasáren v určitých časových intervalech. Důležitým bodem je dosáhnutí počtu dvou až tří kasáren v rané fázi hry, každá další postavená kasárna je odměněna jen v případě, že se v době volby stavby nachází přebytek surovin.
- Protection - tato funkce se stará o vybudování ochrany v základně pomocí věží - *Photon Cannon*. Má-li agent k dispozici alespoň minimální přebytek surovin (hlavní je mít suroviny pro produkci vojenských jednotek), je tato akce pro stavbu budovy *Photon Cannon* ohodnocena kladně. Zde je třeba ale brát zřetel na dosud nezmíněný aspekt hry, některé budovy, včetně budovy *Photon Cannon*, vyžadují existenci ostatních budov určitého typu, tzn. budovu *Photon Cannon* lze stavět jen v případě, že je již postavena budova *Forge*. V případě splnění těchto podmínek agent při volbě akce stavby budovy *Photon Cannon* je odměněn kladnou hodnotou.
- Research - tato funkce se snaží o řešení problémů nároků budov na existenci ostatních budov. Jestliže se agent snaží o stavbu budovy, která doposud není postavena a zároveň by umožnila přístup stavby ostatních budov, dojde ke kladnému ohodnocení akce. Jestliže je ale budova již postavena a agent se snaží o opětovnou stavbu stejné budovy, akce v tomto případě není odměněna, jelikož na splnění podmínky k povolení stavby dalších budov jiného typu, stačí existence jen jedné dané budovy. Funkce platí pro budovy *Forge* a *Cybernetics Core*.



Obrázek 4.2: Ukázka modelovaných prvků hry StarCraft a běhu systému.

Kapitola 5

Implementace

Tato kapitola popisuje implementaci výsledného systému, problémy a překážky, které byly v průběhy objeveny a nalezené řešení těchto problémů.

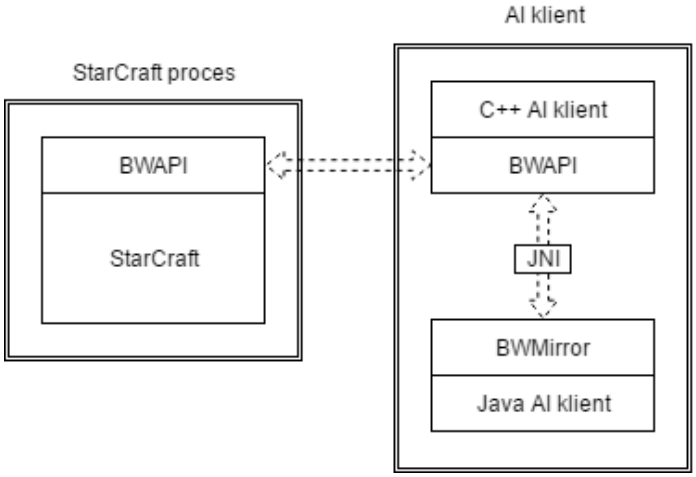
5.1 Popis nástrojů

Zvolený jazyk pro vývoj systému je Java, nelze tedy přímo použít BWAPI framework určený pro C++, jenž poskytuje rozhraní pro používání StarCraft instrukcí. Pro přístup k tomuto API v jazyce Java je třeba použít framework BWMirror, který slouží jako Java wrapper. BWMirror je silně spjat s JNI (JNI), což je rozhraní umožňující propojit kód běžící na virtuálním stroji Javy s nativními programy a knihovnami napsanými v jiných jazycích – v tomto případě to je BWAPI napsané v C++. Skladbu těchto frameworků znázorňuje obrázek 5.1.

Výchozím chováním systému je chování frameworku Atlantis, který operuje na rozhraní frameworku BWMirror a je vybudován na principu konečného automatu. Atlantis v každé iteraci smyčky konečného automatu deleguje práci jednotlivým komandérům, kteří dále rozdělují práci manažerům. Manažeři se starají o specifické okruhy hry. Struktura vazeb na nejvyšších dvou úrovních:

- Worker commander
  - Worker manager
- Combat commander
- Production commander
  - Supply manager
  - Production manager
  - Construction manager
- Scout manager

Ze zmíněné struktury nástroje Atlantis je patrné, že konečný automat již řeší problematiku konstrukce a budování budov. Funkcionalita pro konstruování budov a ovládání dělníků stavitelů je pokryta mnoha manažery a nachází se napříč celým rozsahem nástroje Atlantis, nicméně z pohledu této práce nejdůležitějšími manažery jsou *worker*, *construction*



Obrázek 5.1: Skladba StarCraft frameworků pro vývoj AI.

a *production* manažer. Princip spolupráce těchto manažerů spočíval v přidávání budov požadovaných ke konstrukci (konstrukční pořadí byly nahrávány ze souboru) do společné fronty a v průběhu rozdělování úkolů všem herním jednotkám, byli budovám přiděleni optimální dělníci (většinou nejblíže se vyskytující dělníci bez probíhající aktivity). Nicméně tento princip je nahrazen naším řešením. Pro splnění cílů této bakalářské práce je záměr použít výchozí chování nástroje Atlantis a integrovat modul strojového učení, který je implementován v rámci frameworku BURLAP. Tento modul strojového učení komunikuje s nástrojem Atlantis pomocí nově vytvořeného manažera `BuilderAgentManager`, který provádí projekci rozhodování agentů do akcí jednotek (*Probe*) hry StarCraft.

5.2 Start systému

První operací při startu systému je vytvoření instance třídy `QLController`, která zahrnuje veškeré potřebné funkce pro běh metody Q-learning. Po vytvoření instance dochází ke spuštění její funkce `prepareSCWorld()`, která inicializuje a definuje všechny potřebné hodnoty pro běh metodu Q-learning.

```
public void prepareSCWorld(){
   gridGame = new Game();
   domain = gridGame.generateDomain();

   final State s = Game.initialState();
   JointRewardFunction rf = new Game.getJointRewardFunction(domain, -1, 10, 5,
       false);
   TerminalFunction tf = new Game.getTerminalFunction(domain);
   SGAgentType builderAgentType = Game.getBuilderAgentType();

   w = new SCWorld(domain, rf, tf, s);

   MultiAgentQLearning a0 = new MultiAgentQLearning(...);
   MultiAgentQLearning a1 = new MultiAgentQLearning(...);

   w.join(a0);
```

```
    w.join(a1);
    ...
}
```

Po vytvoření instance třídy `Game`, je následně spuštěna metoda této třídy `generateDomain()`. Tato metoda `generateDomain()` generuje doménu, tzn. přeloží třídy `BuilderAgent` (třída agenta dělníka, MDP prvky popsány v 4.3) a `Goal` (třída definující agentovy góly, odměnové funkce byly zmíněny v 4.3) do stavů vyjádřených MDP, definuje pravděpodobnostní funkce a akce, které agenti mohou používat pro přechod mezi těmito stavy a také stanoví terminálové funkce. V průběhu generování domény je zvolena mechanika třídy `GameMechanics` jako výchozí chování pro zpracování společných akcí všech agentů pro tuto doménu (viz. 5.4). Po vygenerování domény je nastaven počáteční stav funkcí `initialState()` třídy `Game`, pro tento stav jsou vytvořené dvě instance třídy `AgentBuilder` a definovány osobní a univerzální góly, kterými je možno ovlivňovat agentovo chování (viz. 5.4). Třída `JointRewardFunction` spouští metodu `getJointRewardFunction()` a nastavuje parametry tak, že cena akcí, které nepovedou do jakéhokoli gólu jsou trestány hodnotou -1, kdežto úspěšné akce vedoucí do personálního gólu jsou odměněny hodnotou 10 a pro dosažení univerzálního gólu to je hodnota 5, poslední parametr funkce určuje, že agentovy akce, které neprovádí žádnou operaci, nejsou trestány zápornou hodnotou. Po vygenerování domény, získání odměnových a terminálových funkcí je vytvořen svět (třída `SCWorld`), ve kterém se budou agenti pohybovat. Vytvořením instance třídy `MultiAgentQLearning` probíhá inicializace Q-learning agenta, těmto agentům je nastaven *discount factor* na hodnotu 0.95, *learning rate* na hodnotu 0.1 a pro *learning policy* je hodnota ϵ nastavena na 0.1. Tyto instance jsou připojeny do světa třídy `SCWorld`, kde dojde k utvoření relací s již existujícími agenty (instance `BuilderAgent`) ve vygenerované doméně. Po inicializaci hodnot a nastavení počátečních hodnot modulu strojového učení je spuštěn cyklus hry modulu téhož modulu a běh systému vrací zpět k výchozímu chování frameworku Atlantis.

Framework BWAPI zpřístupňuje velké množství eventů hry StarCraft, kterým je možno naslouchat pomocí rozhraní `BWEventListener`. Nástroj Atlantis implementuje třídou `Atlantis` právě toto rozhraní, tudíž má přístup k eventům hry. Při volání eventu `onStart()`, kdy dochází ke startu nové hry, dojde k připojení systému do hry, analýze mapy a inicializaci všech potřebných proměnných. Po ukončení eventu `onStart()`, je ve smyčce v průběhu hry volán event `onFrame()`. Tento event je volán při každém novém snímky hry, k čemuž dochází 24× za vteřinu. Každý z těchto snímků vyjadřuje aktuální stav hry, veškeré akce hry jsou vykonávány právě zde. Nástroj Atlantis v tomto eventu provádí průchod konečným automatem (třída `AtlantisGameCommander`), který přiděluje všem možným jednotkám ve hře akce, které se mají provést.

5.3 Synchronizace procesů

V tomto okamžiku je třeba rozhodnout, jakým způsobem by bylo vhodné integrovat jeden průběh (fáze) modulu strojového učení (vygenerování akcí neboli přechod agentů z aktuálního stavu do nového stavu), pro získání akcí a následné provedení těchto akcí konečným automatem nástroje Atlantis. Během testování možných řešení nastal hlavní problém, a to že průběh konečného automatu není náročný na čas a stihne proběhnout ve vymezeném čase jednoho snímku (42ms), kdežto fáze implementované metody Q-learning značně přesa-

huje časový rozsah jednoho snímku (jednotky sekund). Rozdíl v časových nárocích je tedy značný. Odzkoušeny a testovány byly postupně tři přístupy:

1. Sekvenční zpracování fáze metody Q-learning a konečného automatu. Rozdíl v časových nárocích byl již popsán, hra působí trhaně a snímek přesahuje časové limity definované soutěží SSCAIT, řešení je tedy nevhodné.

2. Paralelní zpracování konečného automatu a fáze metody Q-learning, kde při každém novém snímku je generován nový proces pro průběh fáze metody Q-learning, který je vykonáván paralelně s ostatními procesy již běžících předešlých fází metody. Řešení je sice rychlé, ale není vhodné, jelikož porušuje zásady zmíněné metody, navíc dochází ke konfliktům při vícenásobném přístupu do Q-tabulky.

3. Opět paralelní zpracování konečného automatu a fáze metody Q-learning, kde pro fáze metody je v nových snímcích generován nový proces, ale zpracování procesů probíhá sekvenčně. Nově vygenerovaný proces je uložen do fronty a proběhne, až jakmile jsou zpracovány všechny předešlé procesy ve frontě. Tento přístup je vhodný, jelikož nedochází k žádným konfliktům, jediným problémem je zdlouhavé zpracování fází metod, tudíž se ve frontě kupí doposud nevykonané vygenerované procesy.

Jako nejvhodnější řešení byl zvolen přístup č.3. Problém s narůstající frontou generovaných procesů byl částečně vyřešen definováním intervalu, kdy je možné generovat nový proces pro fázi metody a přidat ho do fronty čekajících procesů. Interval byl nastaven na limit 20 herních sekund, což zhruba odpovídá časové náročnosti rozhodování na úrovni taktik a strategií. Doposud popisovaný problém by se dal krátce vyjádřit kódem:

```
public ExecutorService es = Executors.newSingleThreadExecutor();
private AtlantisGameCommander gameCommander;
private QLController qlController = QLController.getInstance();

@Override
public void onFrame() {
   ...
    if (AtlantisGame.getTimeFrames() %
      atlantis.qLearningModul.Game.GAME_RUN_FRAME_COLDOWN == 0){
        es.submit(qlController::runStage);
    }

    gameCommander.update();
   ...
}
```

Konečným automatem je zde `gameCommander`, k jeho spuštění slouží funkce `update()`. Spuštění fáze metody Q-learning dochází voláním funkce `runStage()` třídy `QLController`.

5.4 Průběh fáze metody Q-learning

Po zahájení práce fáze modulu strojového učení voláním funkce `runStage()`, dochází ke kontrole, zda je aktuální stav terminálním stavem, či nikoli. V kladném případě je hra ukončena a uložena do seznamu odehraných her, v záporném případě fáze pokračuje dál. Agentům (`AgentBuilder`) jsou v každé nové fázi zaktualizovány statické proměnné indikující aktuální

stav hry, tudíž agenti kromě svých soukromých hodnot jako je ID, jméno, ID přidělné herní jednotky pro vykonání volených akcí atd., sdílí stejnou informaci o daném stavu. Agenti následně volí akci na základě algoritmu metody Q-learning. Jelikož však agenti provádí volbu asynchronně a kromě své vlastní volby nevědí, jaké akce zvolili ostatní agenti, je potřeba provést kontrolu společných akcí, zda současný stav umožňuje provést daný výběr. Při současné volbě akcí může dojít k mnoha kolizím. Kolize může nastat pro volbu budov, pro které není možné alokovat dostatek surovin pro započetí stavby, nebo u budov kterých nejsou splněny požadavky pro zahájení stavby (např. není kde stavět, nebo budova požaduje existenci jiné budovy, která doposud nebyla postavena). Zmíněnou kontrolu společných akcí provádí třída `GameMechanics`, která sadou metod, na základě stavových hodnot hry, detekuje, ke kterým kolizím dochází, sestavuje sady kolizí, vzájemným porovnáváním různých kombinací se snaží najít optimální řešení a sestavuje finální sadu akcí. U ukací, které jsou vyhodnoceny jako konfliktní, se akce nahrazuje operací „noop“, neboli „no-operation“. Samozřejmě každý agent je schopen si sám zvolit akci, kdy nebude provádět žádnou operaci, tato akce je ohodnocena nulovou hodnotou. Jestliže však agent zvolí stavební akci, která se vyhodnotí jako kolizní a neprojde kontrolou třídy `GameMechanics`, je zaměněna za „noop“ operaci. Rozdíl mezi takto přidělenou akcí a přímým zvolením akce pro nic nědělání, nastává v ohodnocení, jelikož se již nejedná o hodnotu nulovou, nýbrž zápornou. Tato mechanika dává v příští volbě akce agentovi popud k tomu, aby se za podobných stavových podmínek takto vyhodnocené akci vyhnul. Zvolené akce jsou následně obdrží `AgentBuilderManager`, který sestavuje konstrukční příkazy a přiděluje vhodné herní jednotky pro vykonání těchto příkazů. Tento manažer má funkci mostu mezi modulem strojového učení a principy frameworku Atlantis, u kterých zajišťuje kominukaci a bezproblémový soulad v průběhu hry.

Jakmile jsou akce zvoleny a kolize úspěšně vyřešeny, agent si výslednou akci nastaví do svých soukromých hodnot (ne do statických, kvůli rozlišení následných akcí agentů). Agenti v této části fáze, postupně provádí průzkum ohodnocení budoucích akcí, děje se tak na základě společných statických informací, ke kterým si každý agent připočte soukromou dočasnou proměnnou. Jsou-li například ve hře postaveny dvě budovy *Photon Cannon* a agent zvolí akci ke stavbě právě této budovy v budoucím tahu, pak při průzkumu má každý agent informaci o aktuálním stavu hry, tedy, že jsou již postaveny dvě věže, avšak jen ten agent, který zvolil stavbu věže zná informaci, že v budoucím tahu budou tři a při správném splnění podmínek odměnové funkce bude ohodnocen kladnou odměnou. Tyto odměnové funkce byly popsány v podkapitole 4.3.3. Není podmínkou, že se agent musí snažit dosáhnout kritérií odměnových funkcí, tato snaha je určena při definici počátečního stavu v podobě gólů, který může nabývat dvou typů:

- Univerzální - tento gól platí pro veškeré agenty v rámci hrané hry. Typ gólu je nastaven na hodnotu 0, což znamená, že je dostupný pro všechny agenty ve hře.
- Personální - gól je přidělen agentovi na základě shody agentova ID a hodnoty typu gólu. Gól může být použit pro rozdělení rolí jednotlivých agentů, například jeden agent stavitel by mohl stavět jen vojenské budovy, jiný by mohl rozšiřovat základny a místa pro těžbu. V této práci není tento typ gólu použit.

Agent po volbě akce provede průzkum každého definovaného gólu a zjišťuje, zda splňuje dané kritéria odměnových funkcí v rámci přidělených gólů, v pozitivním případě je ohodnocen kladnou odměnou, která je připočtena do celkové odměny agenta v průběhu hry.

Jak již bylo řečeno v sekci 5.3 fáze strojového učení je volána ve smyčce do té doby dokud není hra ukončena nebo pokud se v průběhu fáze nenarazí terminální stav. V takových případech se čeká na doběhnutí aktuální fáze metody Q-learning, po které je rovněž

ukončena běžící hra metody téže metody a následně je přidána mezi ostatní do té doby vykonané hry.

Kapitola 6

Zhodnocení systému

Kromě definice pravidel daných soutěží SSCAIT, které je třeba dodržet pro vývoj systému, je zde možnost volby porovnat systém s ostatními účastníky soutěže ve formě turnaje, který je konán jednou ročně. Turnaje jsou spuštěny dva, jeden pro studenty a druhý pro ostatní účastníky soutěže, oba probíhají v měsíci Leden, z důvodu tohoto časného zahájení se nebylo možné turnaje zúčastnit, jelikož v té době byl systém teprve ve vývojové fázi. Zápasy mezi přihlášenými systémy však probíhají každodenně i po ukončení turnaje.

Volba účastníků hraných zápasů probíhá docela náhodně, tomu je kromě jiných faktorů zapříčiněno hlavně možností hlasování sledujících pro volbu budoucích účastníků zápasu. Tento problém v kombinaci s velkým množstvím účastníků velice komplikuje sběr statistik a analýzu chování systému.

Po průzkumu popisů přihlášených systémů a částečném sledování hraných her, lze říci, že většina systémů je založena na konečných automatech a pevně daných strategiích. V nejlepších případech se strategie u některých systémů občas změní za jinou pevně danou strategii. Značnou výhodou těchto systémů je jejich rychlost, jelikož hrané strategie a taktiky nejsou zpomaleny učícím se procesem. Nevelké množství systémů, které jsou schopné se učit, jsou vyvíjeny v týmech nebo v rámci magisterských a diplomových prací a většinou jsou zaměřeny na jednotlivé problémové oblasti hry StarCraft, nejčastěji volenou oblastí je micromanagement hry. Komplexnější a rozsáhlejší systémy mohou být vyvíjený i mnoho let, například [6].

6.1 Experiment

Obecně lze říci, že pokud je vytvořený systém schopen porazit NPC (non-player character), které je zabudováno ve hře StarCraft, jedná se o kvalitní systém. Z časových důvodů byl pro vyhodnocení systému proveden lokální experiment, který spočívá v odehrání sady zápasů našeho systému proti NPC hry StarCraft. Ve čtrnácté hře byl systém schopen porazit NPC.

Z důvodu vyváženosti byla pro soupeře zvolena rasa Protoss pro StarCraft NPC. Volby akcí mého systému byly v prvních pěti hrách spíše náhodné a nedávaly žádný ucelený smysl. Můj systém byl poražen obvykle v průměrném čase 5-6 minut. Začátkem hry č. 5 systém v budoucích hrách začal systematicky volit v prvním kole akcí (jedním kolem je myšleno současná volba akcí všech agentů, tedy dvě současné akce pro dva agenty) stavbu budovy *Pylon*, od hry č. 8 začal volit v prvních třech počátečních kolech akci stavbu kasáren. Ve hře č. 11 byl systém schopen aktivně odolávat nepříteli a občas se dostal i do protiútoku, až k hranicím nepřátelské základny. Hra trvala zhruba třináct minut a systém

Hra č. 1 - skóre 1522				
Pořadí	Agent číslo	Čas (ms)	Chtěná akce	Výsledná akce
1	1	54	noop	noop
2	2	54	Cybernetics Core	noop
3	1	108	Nexus	Nexus
4	2	108	Forge	noop
5	1	175	Nexus	Nexus
6	2	175	Nexus	Nexus
7	1	248	Cybernetics Core	noop
8	2	248	Gateway	Gateway
9	1	313	Gateway	Gateway
10	2	313	noop	noop
11	1	335	Forge	Forge
12	2	335	Assimilator	Assimilator
13	1	352	noop	noop
14	2	352	Nexus	Nexus
15	1	358	Forge	noop
16	2	358	noop	noop
17	1	363	Gateway	noop
18	2	363	Photon Cannon	noop
19	1	366	Pylon	Pylon
20	2	366	Forge	noop

Obrázek 6.1: Konstrukční pořadí mého systému pro hru č. 1

aktivně produkoval jednotky díky postaveným třem kasárnám a pěti budovám *Pylon* v ranné fázi hry. Ve hře č. 14 bylo dosaženo důležitého milníku, systém porazil soupeře aktivním tlakem vysílaných vojenských jednotek. Vývoj stavebního pořadí lze pozorovat porovnáním Obrázku 6.1 a Obrázku 6.2. Systém se dokázal přizpůsobit hernímu stylu NPC hry StarCraft a po 14 odehraných hrách jej porazit, systém tedy jeví známky učení.

Hra č. 14 - skóre 90346				
Pořadí	Agent číslo	Čas (ms)	Chtěná akce	Výsledná akce
1	1	35	Pylon	Pylon
2	2	35	noop	noop
3	1	110	Gateway	Gateway
4	2	110	Forge	noop
5	1	204	Nexus	Nexus
6	2	204	Gateway	Gateway
7	1	313	Pylon	Pylon
8	2	313	noop	noop
9	1	383	Pylon	Pylon
10	2	383	noop	noop
11	1	469	Nexus	Nexus
12	2	469	Gateway	Gateway
13	1	560	Assimilator	Assimilator
14	2	560	Photon Cannon	noop
15	1	650	Assimilator	Assimilator
16	2	650	Pylon	Pylon
17	1	824	Cybernetics Core	noop
18	2	824	Cybernetics Core	noop
19	1	911	Gateway	Gateway
20	2	911	Pylon	Pylon
21	1	955	Nexus	Nexus
22	2	955	Cybernetics Core	noop
23	1	1107	Forge	Forge
24	2	1107	noop	noop
25	1	1198	noop	noop
26	2	1198	Pylon	noop
27	1	1527	Cybernetics Core	Cybernetics Core
28	2	1527	Photon Cannon	noop

Obrázek 6.2: Konstrukční pořadí mého systému pro hru č. 14

Kapitola 7

Závěr

Úkolem bakalářské práce bylo navrhnout, implementovat a ohodnit systém pro hraní her s aplikací konceptů umělé inteligence. Hra a implementační podmínky byly dány zvolenou soutěží. Po dohodě s vedoucím práce byla vybrána soutěž *SSCAI Tournament* pro RTS hru *StarCraft* s aplikací metody strojového učení *Q-learning*.

Pro splnění podcílů práce bylo potřeba seznámit se a porozumět problémovým oblastem RTS her. S ohledem na tyto oblasti, mnou navržené řešení využívá přístupu rozhodování na strategické úrovni a učí se sestavit optimální konstrukční pořadí budov dle hracího stylu oponenta. Pro implementaci tohoto systému byly zvoleny nástroje BWAPI, BWMirror a Atlantis pro ovládání hry StarCraft a framework BURLAP pro snadnou aplikaci metody Q-learning. Pro uplatnění této metody byla hra modelována do MDP a framework Atlantis byl přízpůsoben znalostem získaných z procesů učení. Ověření schopnosti učení proběhlo na sadě odehraných her se systémem hry StarCraft. Mnou vytvořený systém se dokázal přizpůsobit hernímu stylu NPC hry StarCraft a po 14 odehraných hrách jej porazit.

Mé řešení nemusí být konečné. Vzhledem k bohaté škále problémových oblastí RTS her, se nabízí se mnoho uplatnění metody Q-learning pro rozhodování na úrovni strategií a taktik. Kromě volby vhodné stavební strategie by systém mohl například dedukovat soupeřovu válečnou strategii a následně se jí dokázat přizpůsobit.

Literatura

[1] Blizzard Entertainment: StarCraft. http://eu.blizzard.com/en-gb/games/sc/, (Accessed on 04/06/2017).

[2] BURLAP. http://burlap.cs.brown.edu/, (Accessed on 04/06/2017).

[3] Eclipse Downloads. https://www.eclipse.org/downloads/eclipse-packages/, (Accessed on 04/06/2017).

[4] GitHub - Ravaelles/Atlantis: StarCraft BroodWar AI Java Framework based on BWMirror 2.5. https://github.com/Ravaelles/Atlantis, (Accessed on 04/06/2017).

[5] GitHub - vjurenka/BWMirror: BWMirror API source files. https://github.com/vjurenka/BWMirror, (Accessed on 04/06/2017).

[6] Home · davechurchill/ualbertabot Wiki · GitHub. https://github.com/davechurchill/ualbertabot/wiki, (Accessed on 05/18/2017).

[7] How Google's AlphaGo Beat Lee Sedol, a Go World Champion - The Atlantic. https://www.theatlantic.com/technology/archive/2016/03/the-invisible-opponent/475611/, (Accessed on 05/13/2017).

[8] IEEE CIG StarCraft AI Competition – Sejong University. https://cilab.sejong.ac.kr/sc_competition/, (Accessed on 04/21/2017).

[9] IntelliJ IDEA the Java IDE. https://www.jetbrains.com/idea/, (Accessed on 04/06/2017).

[10] Java SE Runtime Environment 7 - Downloads | Oracle Technology Network | Oracle. http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html, (Accessed on 04/06/2017).

[11] Releases · bwapi/bwapi · GitHub. https://github.com/bwapi/bwapi/releases, (Accessed on 04/06/2017).

[12] [SSCAIT] Student StarCraft AI Tournament 2017. http://sscaitournament.com/index.php?action=presskit, (Accessed on 04/06/2017).

[13] [SSCAIT] Student StarCraft AI Tournament 2017. http://sscaitournament.com/index.php?action=tutorial, (Accessed on 04/06/2017).

[14] StarCraft AI Competition. http://www.cs.mun.ca/~dchurchill/starcraftaicomp/, (Accessed on 04/21/2017).

[15] Buro, M.: Real-Time Strategy Games: A New AI Research Challenge. In *IJCAI*, editace G. Gottlob; T. Walsh, Morgan Kaufmann, 2003, s. 1534–1535.
URL http://dblp.uni-trier.de/db/conf/ijcai/ijcai2003.html#Buro03

[16] Campbell, M.; Hoane, A. J.; Hsu, F.-h.: Deep blue. *Artificial intelligence*, ročník 134, č. 1-2, 2002: s. 57–83.

[17] Churchill, D.; Buro, M.: Build Order Optimization in StarCraft. In *AIIDE*, editace V. Bulitko; M. O. Riedl, The AAAI Press, 2011.
URL http://dblp.uni-trier.de/db/conf/aiide/aiide2011.html#ChurchillB11

[18] Churchill, D. G.: Heuristic Search Techniques for Real-Time Strategy Games. 2016, doi:10.7939/R3HT2GN96.
URL https://doi.org/10.7939/R3HT2GN96

[19] De, D.; Grenoble, L. D.; Informatique, S.; aj.: Bayesian Programming and Learn- ing for Multi-Player Video Games Application to RTS AI.

[20] Jaidee, U.; Munoz-Avila, H.: CLASSQ-L: A Q-Learning Algorithm for Adversarial Real-Time Strategy Games. 2012.
URL
https://www.aaai.org/ocs/index.php/AIIDE/AIIDE12/paper/view/5515/5734

[21] Ludvig, E. A.; Bellemare, M. G.; Pearson, K. G.: A Primer on Reinforcement Learning in the Brain. In *Computational Neuroscience for Advancing Artificial Intelligence*, IGI Global, s. 111–144, doi:10.4018/978-1-60960-021-1.ch006.
URL https://doi.org/10.4018%2F978-1-60960-021-1.ch006

[22] Matignon, L.; Laurent, G.; Le Fort-Piat, N.: Reward function and initial values: better choices for accelerated goal-directed reinforcement learning. *Artificial Neural Networks–ICANN 2006*, 2006: s. 840–849.

[23] Neto, G.: From single-agent to multi-agent reinforcement learning: Foundational concepts and methods. *Learning theory course*, 2005.

[24] Netrvalová, A.: Úvod do problematiky multiagentních systémů. *ZČU v Plzni, FAV, KIV*, 2005.

[25] Ontanon, S.; Synnaeve, G.; Uriarte, A.; aj.: A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in Games*, ročník 5, č. 4, dec 2013: s. 293–311, doi:10.1109/tciaig.2013.2286295.
URL https://doi.org/10.1109%2Ftciaig.2013.2286295

[26] Sutton, R.; Barto, A.: *Reinforcement Learning: An Introduction*. A Bradford book, Bradford Book, 1998, ISBN 9780262193986.
URL https://books.google.co.uk/books?id=CAFR6IBF4xYC

[27] Tammelin, O.; Burch, N.; Johanson, M.; aj.: Solving Heads-Up Limit Texas Hold'em. In *IJCAI*, 2015, s. 645–652.

[28] VOLNÁ, E.: Základy softcomputingu. 2012.

[29] Watkins, C. J. C. H.: *Learning from Delayed Rewards*. Dizertační práce, King's College, Cambridge, UK, May 1989.
URL http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[30] Watkins, C. J. C. H.; Dayan, P.: Q-learning. *Machine Learning*, ročník 8, č. 3-4, may 1992: s. 279–292, doi:10.1007/bf00992698.
URL https://doi.org/10.1007%2Fbf00992698

[31] Wikipedie: *StarCraft — Wikipedie: Otevřená encyklopedie*. 2016, [Online; navštíveno 15. 03. 2017].
URL https://cs.wikipedia.org/w/index.php?title=StarCraft&oldid=14422334

[32] Young, J.; Hawes, N.: Learning Micro-Management Skills in RTS Games by Imitating Experts. In *AIIDE*, 2014.

[33] Zbořil, F.: Základy umělé inteligence, studijní opora. *FIT VUT v Brně*, 2006.

Přílohy

Příloha A

Přiložené CD se zdrojovými kódy programu