UNIVERZITA KOMENSKÉHO V BRATISLAVE

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Trénovanie agenta pre makromanažment v real-time strategických hrách pomocou učenia posilňovaním

Bakalárska práca

2014 **Osama Hassanein**

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Trénovanie agenta pre makromanažment v real-time strategických hrách pomocou učenia posilňovaním

Bakalárska práca

Študijný program: Informatika
Študijný odbor: 2508 Informatika
Školiace pracovisko: Katedra informatiky
Vedúci práce: Mgr. Viliam Dillinger

2014 **Osama Hassanein**

Čestné vyhlásenie

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím uvedených zdrojov.

V Bratislave .....

Poďakovanie

Touto cestou by som sa chcel poďakovať predovšetkým svojmu vedúcemu za usmernenie a užitočné rady pri robení tejto práce. Takisto by som sa chcel poďakovať rodine a priateľom za podporu, hlavne počas posledných dní pisania tejto práce. Nakoniec by som sa chcel poďakovať spoločnosti Summit Motors Slovakia za poskytnutie počítača, bez ktorého by sme nedokončili všetky experimenty uvedené v našej práci.

71955000

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Osama Hassanein
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Trénovanie agenta pre makromanažment v real-time strategických hrách pomocou učenia posilňovaním / *Agent training for macromanagement in real-time strategy games using reinforcement learning*

Cieľ:
1. Naštudujte problematiku učenia posilňovaním a dopredných neurónových sietí
2. Navrhnite, implementujte a natrénujte agenta pre makromanažment v real-time stratégii Starcraft:BrodWar.
3. Otestujte úspešnosť agenta vo vybraných pokusoch.

Vedúci: Mgr. Viliam Dillinger
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. PhDr. Ján Rybár, PhD.

Dátum zadania: 28.10.2013

Dátum schválenia: 28.10.2013 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Abstrakt

Cielom práce bolo navrhnúť a implementovať agenta založeného na učení posiľnovaním pre makromanažment v real-time strategických hrách.

Najprv sme navrhli agenta, ktorý túto úlohu rieši pomocou učiaceho algoritmu Advantage Learning a dopredných neurónových sietí. Knižnica BWAPI, nám potom umožnila vytvoriť agenta pre hru Starcraft, ktorý využíva stratégiu cannon rush na porazenie súpera.

Agenta sme trénovali v zápasoch proti natívnemu AI hry Starcraft, pričom sme vyskúšali viacere konfigurácie učiaceho algoritmu. Aj keď sa agentovi nepodarilo vždy nájsť optimálne riešenie, na konci trénovania bola jeho úspešnosť dostatočná na použitie v praxi.

Kľúčové slová: *Učenie posilňovaním, Advantage learning, Dopredné neurónové siete, Real-time stratégie, Makromanažment, Starcraft: Brood War, Cannon Rush*

Abstract

Our goal was to design and implement an agent based on Reinforcement Learning for macromanagment in Real-time strategy games.

We began by designing an agent that solves this problem using the Advantage Learning algorithm and Feedforward Neural Networks. We then used the BWAPI library to create an agent for Stacraft, which defeats its enemies using the Cannon Rush strategy.

We trained the agent in matches against Starcraft's native AI, while trying out various configurations of the learning algorithm. Even though the agent always wasn't able to find the most optimal solution, by the end of the training his success rate would be sufficient for practical use.

Kľúčové slová: *Reinforcement Learning, Advantage Learning, Feedforward Neural Networks, Real-time Strategy games, Macromanagment, Starcraft: Brood War, Cannon Rush*

Obsah

Úvod

Real-time strategické hry (Real Time Strategy - RTS) sú počítačové hry, v ktorých hráč ovláda svoje jednotky a ničí nimi jednotky súpera. Hráč musí popri tom získavať suroviny, aby mohol stavať budovy a výrábať ďalšie jednotky. Toto všetko sa odohráva v realnom čase, pričom hráči ovládajú svoje jednotky nezávisle od seba.

RTS hry si od hráča vyžadujú strategické zmýšlanie, rýchle reakcie a schopnosť sústrediť sa na viacero úloh naraz. Z pohľadu umelej inteligencie vytvárajú zložité nedeterministické prostredie, v ktorom treba riešiť množstvo netriviálnych úloh.

Práca je zameraná na problém efektívnej alokácie surovín na produkciu budov a jednotiek. Tento aspekt RTS hier sa nazýva *makromanagment*. Pre tento problém existujú viacere riešenia, väčšina z nich však využíva napevno dané plány alebo jednoduché stavové automaty. Dali sme si preto za cieľ navrhnúť prispôsobivého agenta, ktorý hľadá dobré riešenie pre túto úlohu na základe situácií, do ktorých sa dostáva. Pri návrhu agenta sme preto zvolili metódu učenia posilnovaním, ktorá spočíva v skúšaní nových možností a učení sa na základe ich dôsledkov (Sutton and Barto, 1998).

Experimentálnu časť našej práce sme implementovali v poplárnej RTS hre Starcraft: Broodwar. Na implementáciu agenta používame voľne dostupnú knižnicu BWAPI, ktorá umožňuje priamo narábať s prostredím tejto hry. Náš agent využíva na porazenie súpera jednoduchú stratégiu *cannon rush*, v ktorej hráč využiva obranné veže na zničenie základne nepriateľa. Hlavnou úlohou agenta je naučiť sa, aké jednotky a budovy treba produkovať v rôznych situáciach, aby maximalizoval svoju úspešnosť.

V prvej kapitole uvádzame *Markovov rozhodovací proces*, ktorý slúži ako matematický model popísujúci herné prostredie. Druhá kapitola ukázuje ako *učenie posilňovaním* hľadá optimálne riešenie pre markovov rozhodovací proces bez znalosti modelu prostredia. V tretej kapitole popíšeme ako neurónové siete fungujú a ako ich dokážeme naučiť aproximovať funckiu. Štvrtá kapitola v krátkosti popisuje hru Starcraft: Broodwar a následne podrobnejšie vysvetľuje stratégiu cannon rush, ktorú náš agent využíva. Návrhu a implementácii agenta sa venujeme v piatej kapitole, kde ukážeme ako agent funguje a vymenujeme parametre, ktoré ovplyvňujú jeho chod.

Nami navrhnutého agenta sme nechali vo vybraných experimentoch hrať proti natívnemu AI hry, pričom sme skúšali rôzne konfigurácie parametrov učiaceho algoritmu a spôsoby odmeňovania. Návrh experimentov a ich výsledky prezentujeme v šiestej kapitole. Posledná kapitola je diskusia, v ktorej analyzujeme tieto výsledky.

Kapitola 1

Markovove rozhodovacie procesy

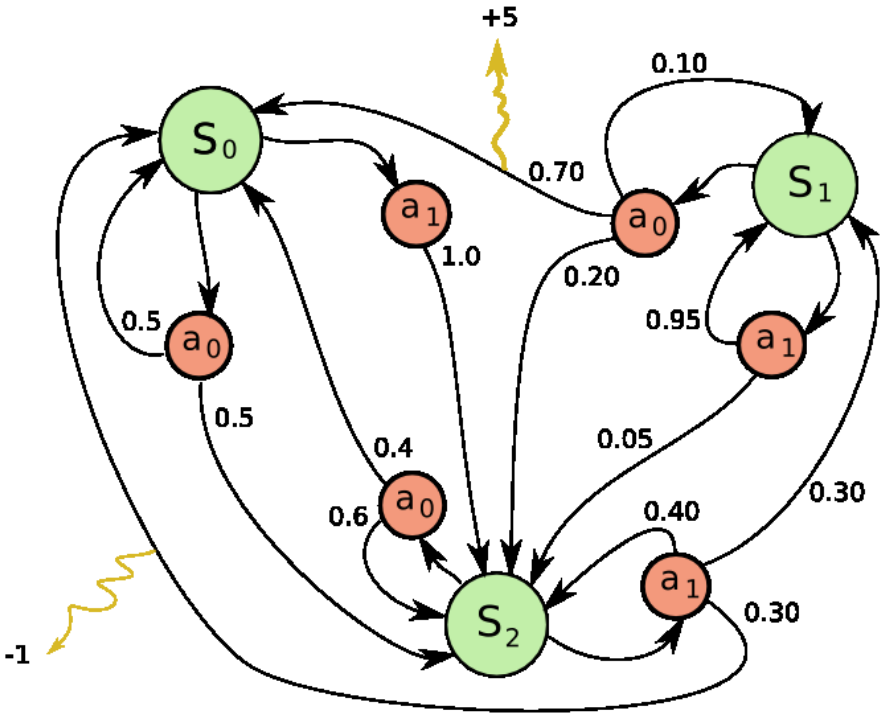
Markovov rozhodovací proces (Markov Decision Process - MDP) je model pre sekvenčný rozhodovací problém, ktorý zahŕňa *časové kroky*, *stavy*, *akcie*, *odmeny* a *prechodovú funkciu*.
V časovom kroku t sa nachádzame v stave s_t a zvolíme akciu a_t. Na základe prechodovej funkcie sa presúvame do nového stavu s_{t+1}, pričom dostávame odmenu r_{t+1}. Toto sa opakuje vo všetkých časových krokoch, pre ktoré je MDP definovaný (Puterman, 2005).

1.1 Formálna definícia

Markovov rozhodovací proces charakterizujeme ako päticu $(N, S, \mathcal{A}, R, T)$ kde:

- N je množina časových krokov
- S je množina stavov
- $\mathcal{A}(s)$ je množina dostupných akcií pre stav $s \in S$
- $T(s, a, s')$ je prechodová funkcia, ktorá určuje s akou pravdepodobnosťou prejdeme zo stavu s do stavu s' pri voľbe akcie a pre každý stav $s \in S$ a $s' \in S$ a akciu $a \in \mathcal{A}(s)$
- $R(s, a, s')$ je odmeňovacia funkcia, ktorá určuje akú odmenu dostaneme za prechod zo stavu s do stavu s' ak zvolíme akciu a pre každý stav $s \in S$ a $s' \in S$ a akciu $a \in \mathcal{A}(s)$

Množina $\mathcal{A} = \bigcup_{s \in S} \mathcal{A}(s)$ je množina všetkých akcií definovaných v MDP.
Prechodová funkcia a odmeňovacia funkcia v MDP neberú do úvahy iné stavy než aktuálny stav, a teda ide o model sekvenčného rozhodovacieho procesu, ktorý spĺňa *markovovú vlastnosť* (Markov Property) (Sutton and Barto, 1998).

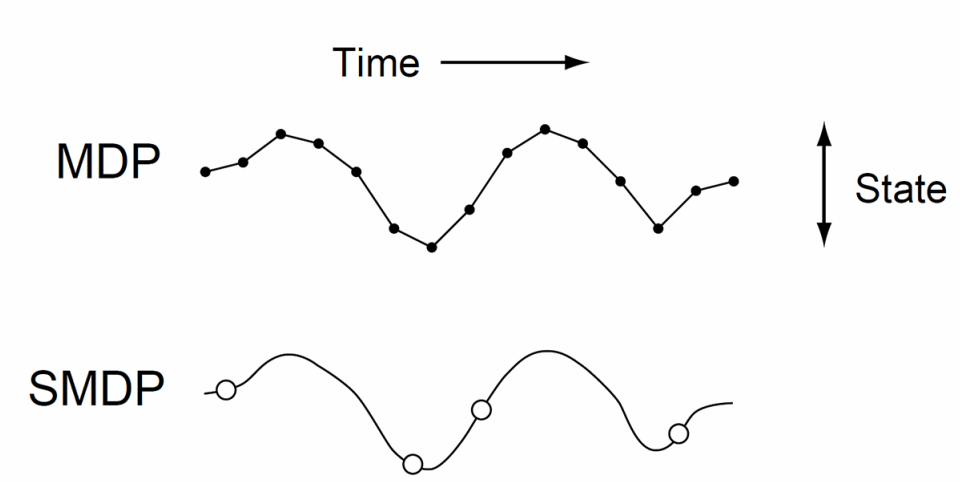


Obr. 1.1: Diagram popisuje MDP s 3 stavmi (zelené vrcholy) a 2 akciami (červené vrcholy). Cena hrány vedúcej z akcie určuje pravdepodobnosť prechodu do nového stavu a žltá šipka určuje odmenu (Prebrané z wikipedia.org `http://en.wikipedia.org/wiki/Markov_decision_process`).

1.2 Semi-markovove rozhovacie procesy

Semi-markovov rozhodovací proces (Semi-Markov Decision Process - SMDP) rozširuje formálnu definíciu klasického MDP o čakaciu funckiu $\tau(s,a)$ pre každý stav $s \in S$ a akciu $a \in \mathcal{A}(s)$. Čakacia funkcia určuje počet časových krokov, po ktorých nastane ďalši rozhodovací krok, ak v stave s zvolíme akciu a. Rozhodovacie kroky sú časové kroky, v ktorých SMDP volí akciu, v iných časových krokoch akcie nevolí (Tijms, 2003). V klasickom MDP je hodnota čakacej funkcie 1 pre všetky stavy a akcie, a teda každý časový krok je aj rozhodovací krok.

Popri SMDP prebieha paralelne *prirodzený proces*. SMDP modeluje prostredie len v rozhodovacích krokoch. Prirodzený proces modeluje stavy a odmeny, pre každý časový krok $t \in N$. Stav s' pre $T(s,a,s')$ je stav, v ktorom sa SMDP bude nachádzať v nasledujúcom rozhodovacom kroku. Odmena $R(s_i,a,s_{i+1})$ v SMDP je kumulovaná odmena za časové kroky medzi aktuálnym rozhodovacím krokom i a tým, čo po ňom nasleduje (Puterman, 2005).



Obr. 1.2: MDP vyberá akciu v každom časovom kroku. SMDP vyberá akciu len v rozhodovacích krokoch, stavy sa však môžu meniť v každom časovom kroku (Sutton et al., 1999)

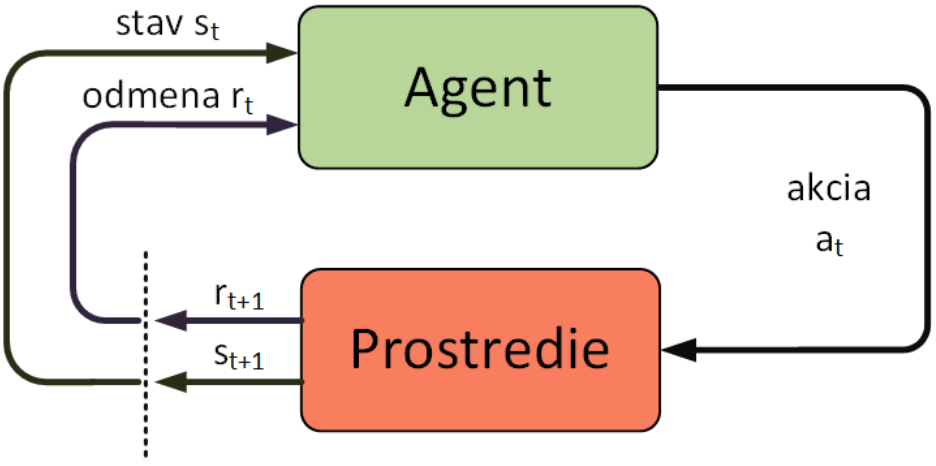
1.3 Stratégia

Riešením MDP (aj SMDP) je *stratégia* (Policy), ktorá pre každý stav $s \in S$ určuje akú akciu $a \in \mathcal{A}(s)$ treba zvoliť. Označujeme ju π (Russell and Norvig, 2003). Stratégia je pre každý stav $s \in S$ definovaná ako pravdepodobnosť $\pi(s,a)$ s akou zvolíme akciu $a \in \mathcal{A}(s)$ (Sutton and Barto, 1998).

Kapitola 2

Učenie posilňovaním

Učenie posilňovaním (Reinforcement Learning - RL) rieši úlohu hľadania optimálnej stratégie (optimal policy) v MDP. V RL rozlišujeme *agenta* a *prostredie*. Agent volí akcie a učí sa stratégiu. Prostredie určuje stav a dáva agentovi číselnú odmenu. Agent nepozná prechodovú funkciu prostredia $T(s,a,s')$ ani odmeňovaciu funkciu $R(s,a,s')$. Cieľ agenta v RL je určený odmeňovacou funckiou $R(s,a,s')$. Úlohou agenta je maximalizovať *celkovú odmenu*, ktorú z prostredia dostane (Sutton and Barto, 1998).



Obr. 2.1: Diagram interakcie agenta a prostredia v RL (Sutton and Barto, 1998)

V rozhodovacom kroku t dostáva agent reprezentáciu aktuálneho *stavu* prostredia s_t, na základe ktorého zvolí a vykoná *akciu* a_t. V nasledujúcom kroku $t+1$, agent dostáva od prostredia číselnú *odmenu* r_{t+1} a nový stav prostredia s_{t+1} (Sutton and Barto, 1998). Toto sa opakuje pre každý rozhodovací krok t.

Agent sa musí popri voľbe akcií učiť maximalizovať celkovú získanú odmenu.

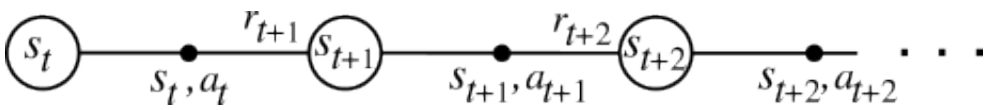
2.1 Predikcia budúcich odmien

Nech agent uplatňuje stratégiu π pri voľbe akcií počnúc od rozhodovacieho kroku t. Prostredie nám potom dáva nekonečnú postupnosť odmien $r_{t+1}, r_{t+2}, r_{t+3}, \ldots$, kde r_i je odmena získaná v rozhodovacom kroku i (Russell and Norvig, 2003).

Utilitu postupnosti odmien počítame ako súčet diskontovaných odmien (Marsland, 2009):

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

kde koeficient $\gamma \in [0,1]$ je *faktor diskontovania*. Čím vyšší faktor γ zvolíme tým väčší význam pripisujeme budúcim odmenám. Faktor γ slúži takisto na zaručenie konečnej sumy pre utilitu. Pri hľadaní optimálnej stratégie sa snažíme maximalizovať utilitu postupnosti odmien (Russell and Norvig, 2003).



Obr. 2.2: Diagram ilustruje postupnosť stavov, akcií a odmien v RL (Sutton and Barto, 1998)

2.1.1 Ohodnocovanie stavov

Hodnotu stavu s pre stratégiu π definujeme ako očakávanú utilitu postupnosti odmien $V^{\pi}(s)$, ktorú agent dostane, ak začína v stave s a ďalej uplatňuje stratégiu π. Hodnota $V^{\pi}(s)$ vyjadruje výhodnosť stavu s pre agenta, ktorý uplatňuje stratégiu π (Harmon and Harmon, 2000).

Nech E_{π} označuje očakávanú utilitu postupnosti odmien pre stratégiu π, kde t je ľubovolný rozhodovací krok, potom platí (Sutton and Barto, 1998):

$$V^{\pi}(s) = E_{\pi}(R_t | s_t = s) = E_{\pi}(\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} | s_t = s)$$

Ohodnocovacia funkcia $V^{\pi}(s)$ nám pre každý stav $s \in S$ udáva predikciu budúcich odmien získaných stratégiou π. *Optimálna ohodnocovacia funkcia* $V^*(s)$ je taká pre ktorú platí (Sutton and Barto, 1998):

$$\forall s \in S : V^*(s) \geq V^{\pi}(s) \text{ pre každú stratégiu } \pi$$

Optimálna stratégia π^* je stratégia, ktorej ohodnocovacia funkcia je optimálna. Môže existovať viac optimálnych stratégií (Sutton and Barto, 1998).

2.1.2 Ohodnocovanie akcií

Q-hodnotu akcie a v stave s pre stratégiu π, definujeme ako očakávanú utilitu postupnosti odmien $Q^{\pi}(s,a)$, ktoré agent dostane, ak začína v stave s, zvolí akciu a a ďalej uplatňuje stratégiu π. Hodnota $Q^{\pi}(s,a)$ vyjadruje výhodnosť akcie a v stave s pre agenta, ktorý uplatňuje stratégiu π (Harmon and Harmon, 2000).

Podobne ako pre ohodnocovanie stavu platí (Sutton and Barto, 1998):

$$Q^{\pi}(s,a) = E_{\pi}(R_t|s_t = s, a_t = a) = E_{\pi}(\sum_{i=0}^{\infty} \gamma^i r_{t+i+1}|s_t = s, a_t = a)$$

Q-funkcia $Q^{\pi}(s,a)$ nám pre každú dvojicu stavu $s \in S$ a akcie $a \in A(s)$ udáva predikciu budúcich odmien získaných stratégiou π. *Optimálna Q-funkcia* $Q^*(s)$ je taká pre ktorú platí (Sutton and Barto, 1998):

$$\forall s \in S, a \in \mathcal{A}(s) : Q^*(s,a) \geq Q^{\pi}(s,a) \text{ pre každú stratégiu } \pi$$

O optimálnej Q-funkcii pre akcie a optimálnej ohodnocovacej funkcii pre stavy platí (Engelbrecht, 2007):

$$V^*(s) = max_{a \in A(s)} Q^*(s,a) \text{ pre každý stav } s \in S$$

Optimálna stratégia π^* je stratégia, ktorej Q-funkcia je optimálna. Môže existovať viac optimálnych stratégií (Sutton and Barto, 1998). Z optimálnej Q-funkcie dostaneme optimálnu stratégiu, keď v každom stave s volíme akciu a s najvyššou Q-hodnotou $Q(s,a)$ (Harmon and Harmon, 2000).

2.2 On-line učiace algoritmy

Agent si pamätá ohodnocovaciu funkciu $V(s)$, ktorá je aproximáciou optimálnej ohodnocovacej funkcie $V^*(s)$. On-line učiaci algoritmus túto aproximáciu zlepšuje počas toho ako agent prechádza prostredím (Harmon and Harmon, 2000).

Pre optimálnu ohodnocovaciu funkciu $V^*(s)$ platí *Bellmanová rovnosť* (Engelbrecht, 2007):

$$V^*(s_t) = \max_{a_t \in \mathcal{A}(s_t)} \Big\{ \sum_{s_{t+1} \in S} T(s_t, a_t, s_{t+1})(R(s_t, a_t, s_{t+1}) + \gamma^{\Delta t} V^*(s_{t+1})) \Big\}$$

kde hodnota Δt je počet časových krokov, ktoré ubehli medzi posledným a aktuálnym časovým krokom.

V uvedených učiacich algoritmoch je Bellmanová rovnosť základom pre vzťahy, ktoré upravujú aproximáciu ohodnocovacej funkcie. Agent podľa nich v rozhodovacom kroku $t+1$ upravuje ohodnotenie $V(s_t)$ pre stav s_t, v ktorom sa nachádzal v predošlom rozhodovacom kroku (Harmon and Harmon, 2000).

Bellmanová rovnosť platí rovnako aj pre optimálnu Q-funkciu.

2.2.1 Q-learning

Agent v stave s_t zvolí akciu $a_t \in \mathcal{A}(s_t)$ s najvyššou Q-honotou $Q(s_t, a_t)$. Agent sa potom presúva do nového stavu s_{t+1} a naučí sa novú hodnotu $Q(s_t, a_t)$ podľa vzorca (Engelbrecht, 2007):

$$Q(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma^{\Delta t} \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a))$$

kde hodnota α je *rýchlosť učenia* (learning rate).

2.2.2 Advantage learning

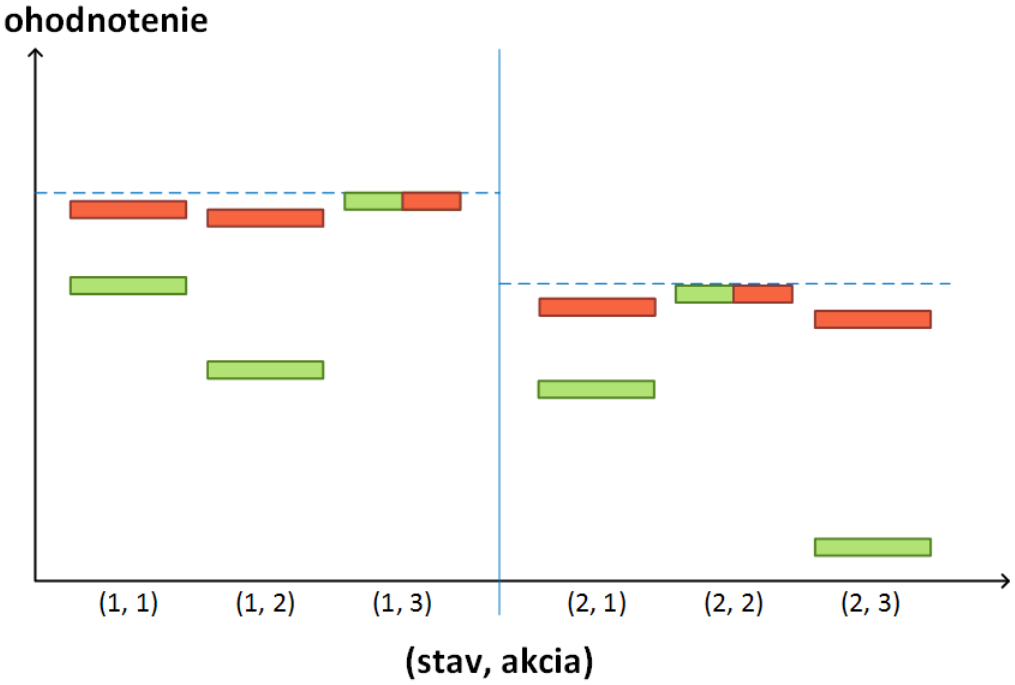
Agent v sa advantage learningu učí aproximovať *advantage funkciu* $A(s,a)$ namiesto Q-funkcie, ktorá pre akciu a v stave s počíta advantage hodnotu. Advantage hodnota $A(s,a)$ vyjadruje výhodnosť výberu akcie a oproti aktuálne najlepšie ohodnotenej akcie v rovnakom stave s. Aproximovanú hodnotu stavu s definujeme ako $V(s) = \max_{a \in \mathcal{A}(s)} A(s,a)$.

Novú hodnotu pre $A(s_t, a_t)$ počíta agent podobe ako v Q-learningu, keď je v stave s_{t+1} podľa vzorca (Baird, 1999):

$$A(s_t, a_t) = (1-\alpha)A(s_t, a_t) + \alpha\left\{V(s_t) + \frac{(r_t + \gamma^{\Delta t}V(s_{t+1})) - V(s_t)}{\Delta t K}\right\}$$

kde hodnota K je *časový koeficiant* (time unit scaling factor). Čím vyššia hodnota časového koeficientu tým, výraznejší je rozdiel medzi advantage hodnotami pre každý stav.

Rozdiely medzi Advantage hodnotami sú $1/K$-násobne väčšie ako medzi Q-hodnotami. Advantage learning je vďaka tomu odolnejší voči chybám pri aproximácii Q-funkcie (Baird, 1999).



Obr. 2.3: Graf porovnáva ohodnotenia dvojíc stavov a akcií Q-learning (červené čiary) a Advantage learning (zelené čiary) (Baird, 1999)

2.3 Metódy explorácie

Explorácia je výber akcií nezávisle od ohodnocovacej funkcie. Explorácie dáva učiacim algoritmom príležitosť zlepšiť aproximáciu optimálnej ohodnocovacej funkcie. Opakom je exploitácia, pri ktorej agent volí akcie na základe ohodnocovacej funkcie. Pri hľadaní optimálnej stratégie je potrebné vyvážiť mieru explorácie a exploitácie. Mieru explorácie je možné počas učenia meniť (Russell and Norvig, 2003).

2.3.1 ε greedy explorácia

V rozhodovacom kroku t agent zvolí akciu a_t :

- náhodne s pravdepodobnosťou ε_k
- najlepšie ohodnotenú s pravdepodobnosťou $1-\varepsilon_k$

kde hodnota $\varepsilon_k \in [0,1]$ je miera explorácie (exploration rate) v rozhodovacom kroku t (Busoniu et al., 2010).

2.3.2 Boltzmannová explorácia

Nech $P_t(a,s)$ je pravdepodobnosť s akou agent v stave $s \in S$ zvolí akciu $a \in \mathcal{A}(s)$ v rozhodovacom kroku t (Busoniu et al., 2010):

$$P_t(s,a) = \frac{e^{Q(s,a)/\tau_t}}{\sum_{a' \in \mathcal{A}(s)} e^{Q(s,a')/\tau_t}}$$

Hodnota $\tau_t \in (0,\infty)$ je teplota (temperature) v rozhodovacom kroku t. Agent pri $\tau_t \to 0$ volí najlepšie hodnotenú akciu a pri $\tau_t \to \infty$ volí akcie rovnomerne náhodne (Busoniu et al., 2010).

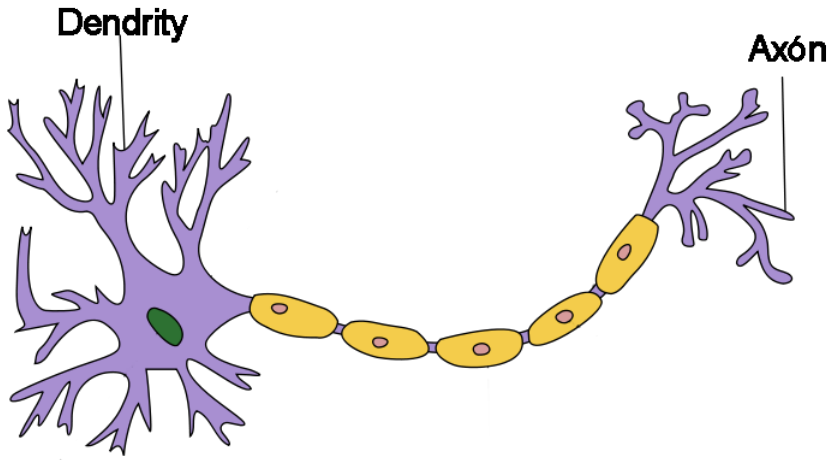
Kapitola 3

Neurónové siete

Neurónová sieť (Neuron network - NN) je výpočtový model založený na princípoch fungovania nervovej sústavy živých organizmov (Russell and Norvig, 2003). Využíva sa ako funkčný aproximátor, ktorý je schopný učiť sa napodobňovať funkcie na základe vzorov. Neurónová sieť dokáže efektívne generalizovať funkčné hodnoty aj pre neznáme vstupy (Marsland, 2009).

3.1 Biologický neurón

Biologický neurón sa skladá z *tela*, *dendritov* a *axónu*. Dendrity vnímame ako vstup a axóny ako výstup. Dva neuróny sú navzájom prepojené, keď sa dendrit jedného neurónu napojí na axón druhého. Takéto prepojenie sa nazýva *synapsia*. Každý neurón môže byť prepojený s viacerými neurónmi naraz. Neuróny slúžia na prenos a šírenie signálu, ktorý buď zosilňujú alebo tlmia (Engelbrecht, 2007).



Obr. 3.1: Schéma biologického neurónu

3.2 Umelý neurón

Umelý neurón je zjednodušený matematický model biologického neurónu. Neurón prijíma na vstupe signál, ktorý buď zosilňuje alebo zoslabuje pomocou číselných váh. Konečný výstup určuje aktivačná funkcia, ktorá počíta funkčnú hodnotu pre sčítané signály (Engelbrecht, 2007).

Neurón je samostatne schopný aproximovať akúloľvek linearne seprovateľnú funkciu $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}$, kde vstup je vektor $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ a výstup je hodnota y. Výstup neurónu sa nazýva *aktivácia* (Russell and Norvig, 2003). Funkcia $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}$ je lineárne separovateľná ak je možné rozdeliť *n-rozmerný priestor* jej vstupov *n-rozmernou nadrovinou* na 2 množiny X_a a X_b, tak aby platilo $\forall x_a \in X_a : f(x_a) < k \wedge \forall x_b \in X_b : f(x_b) \geq k$ pre ľubovoľnú hraničnú hodnotu $k \in \mathbb{R}$ (Engelbrecht, 2007).

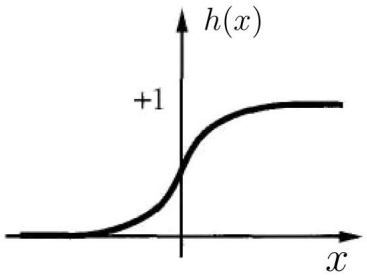
3.2.1 Formálna definícia

Definícia neurónu v sebe zahŕňa (Russell and Norvig, 2003):

- číselné váhy w_i prislúchajúce každému vstupu x_i
- bias hodnota θ
- aktivačná funkcia f

Aktivačná funkcia je zobrazenie $f : \mathbb{R} \to \mathbb{R}$. Vhodné aktivačné funkcie sú (Engelbrecht, 2007):

- sigmoida $f(x) = \frac{1}{1+e^{-x}}$
- linerárna funkcia $f(x) = x$



Obr. 3.2: Grafu sigmoidy $f(x) = \frac{1}{1+e^{-x}}$. Charakterizuje ju prudký rast v okolí 0, pričom na zvyšku číselnej osi má funkčnú hodnotu veľmi blízku 0 pre záporné čísla a veľmi blízko 1 pre kladné čísla

3.2.2 Výpočet výstupu

Postup pre výpočet aktivácie neurónu y pre vstupný vektor **x** (Russell and Norvig, 2003):

1. prenásobíme vstupné hodnoty váhami a sčítame

$$net = \sum_{i=0}^{n-1} x_i w_i$$

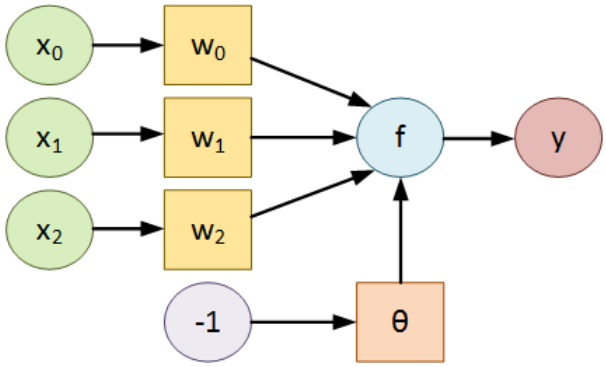
2. vypočítame funkčnú hodnota rozdielu súčtu vstupov a bias hodnoty pre aktivačnú funkciu

$$y = f(net - \theta)$$

Pre zjednodušenie výpočtu môžeme rozšíriť váhy o bias hodnotu $w_n = \theta$. Vstupný vektor potom rozšírime o hodnotu $x_n = -1$ (Russell and Norvig, 2003).

Aktivácia neurónu potom počítame podľa vzorca:

$$y = f(\sum_{i=0}^{n} x_i w_i)$$



Obr. 3.3: Diagram znázorňuje výpočet aktivácie neurónu s tromi vstupmi x_0, x_1 a x_2. Hodnoty w_0, w_1 a w_2 sú váhy a θ je bias hodnota. Vstupy prenásobíme s váhami a odčítame od nich bias hodnotu. Aktivácia neurónu y je funkčná hodnota tohto rozdielu, pre aktivačnú funkciu f

3.2.3 Učiaci algoritmus

Neurón trénujeme metódou učenia s učiteľom (Supervised Learning), v ktorej máme k dispozícii množinu dvojíc vstupov a k nim očakávaných výstupov na základe ktorých sa neurón má naučiť aproximovať funkciu (Engelbrecht, 2007).

Pre očakávaný výstup t a aktiváciu y počítame *celkovú chybu* ako polovica štvorcu rozdielu týchto dvoch hodnôt (Marsland, 2009):

$$E = \frac{(t-y)^2}{2}$$

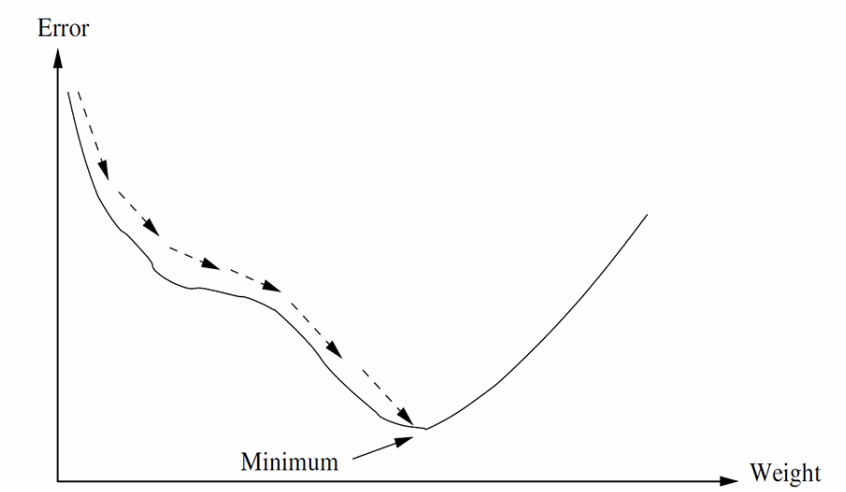
Učenie hľadá také hodnoty váh, ktoré minimalizujú celkovú chybu E. Podľa učiaceho pravidla Gradient descent, sa na úpravu váh využíva derivácia celkovej chyby E vzhľadom na váhy. V našom prípade ide o hodnotu $(t-y)$ (Engelbrecht, 2007).

Pre vstupný vektor $\mathbf{x}$, aktiváciu y a očakávanú výstupnú hodnotu t sa váhy upravujú podľa vzťahu:

$$w_i(t) = w_i(t-1) + \alpha(t-y)y'x_i \text{ pričom } y' = f'(\sum_{i=0}^{n} x_i w_i(t-1))$$

kde $w_i(t)$ je hodnota váh v čase t a f' je derivácia aktivačnej funkcie f.

Parameter $\alpha \in [0,1]$ nazývame rýchlosť učenia (learning rate). Čím vyššiu rýchlosť učenia zvolíme, tým väčší vplyv má chyba na určovaní novej hodnoty pre váhy (Engelbrecht, 2007).



Obr. 3.4: Graf znázorňuje chybu E pre pevne zvolený vstup $\mathbf{x}$ ako funkciu konfigurácie honôt pre váhy. Učiaci algoritmus sa snaží nájsť takú konfiguráciu váh, ktorá dosahuje minimum tejto funkcie pre každý vstup

3.3 Viacvrstvový perceptrón

Jednoduchý perceptrón obsahuje jeden alebo viac neurónov, ktoré prijímajú rovnaký vstup. *Sieť neurónov* je sústava neurónov, ktoré sú navzájom prepojené. Neuróny v neurónovej sieti sú na základe týchto prepojení rozdelené do jednej alebo viacerých *vrstiev*. Posledná vrstva neurónov sa nazýva *výstupná vrstva*, ostatné vrstvy neurónov sú *skryté vrstvy*. *Dopredná neurónová sieť* (Feed-forward neuron network - FFNN) je taká neurónova sieť, v ktorej prepojenia neurónov nevytvárajú cyklus. (Engelbrecht, 2007).

Viacvrstvový perceptrón (Multi Layered Perceptron - MLP) je viacvrstvová dopredná neurónová sieť, ktorá obsahuje jednu alebo viac skrytých vrstiev a výstupnú vrstvu (Russell and Norvig, 2003). MLP je schopný aproximovať akúkoľvek funkciu $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}^m$, kde vstup je vektor $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ a výstup je vektor $\mathbf{y} = (y_0, y_1, \ldots, y_{m-1})$. Toto platí aj pre MLP s jednou skrytou vrstvou, preto ďalej uvažujeme MLP s jednou skrytou vrstvou (Engelbrecht, 2007). Počet neurónov na skrytej vrstve s môžeme voliť nezávisle od dĺžky vstupu n a výstupu m.

3.3.1 Výpočet výstupu

Váhy (a bias hodnoty) neurónov na skrytej vrstve tvoria spolu maticu $\mathbf{W}$. Váha $w_{i,j}$ na skrytej vrstve, kde $i \in \{0, 1, \ldots, n\}$ a $j \in \{0, 1, \ldots, s-1\}$, prislúcha i-temu vstupu j-teho neurónu v skrytej vrstve. *Skrytá aktivácia* $\mathbf{a}$ je vektor výstupov neurónov na skrytej vrstve. Váhy (a bias hodnoty) neurónov na výstupnej vrstve tvoria spolu maticu $\mathbf{V}$. Váha $v_{j,k}$ na výstupnej vrstve, kde $j \in \{0, 1, \ldots, s\}$ a $k \in \{0, 1, \ldots, m-1\}$, prislúcha j-tej skrytej aktivácii, ktorá je vstup pre k-ty neurón skrytej vrstvy. Na skrytej vrstve majú neuróny aktivačnú funckiu $f : \mathbb{R} \to \mathbb{R}$ a na výstupnej vrstve majú neuróny aktivačnú funckiu $g : \mathbb{R} \to \mathbb{R}$ (Marsland, 2009).

Nasleduje postup pre výpočet výstupného vektora **y** pre vstupný vektor **x**:

1. vstupný vektor rozšírime o hodnotu $x_n = -1$
2. skrytú aktiváciu vypočítame zo vstupného vektora a váh na skrytej vrstve

$$a_j = f(\sum_{i=0}^{n} x_i w_{i,j}) \text{ kde } j \in \{0,1,\ldots,s-1\}$$

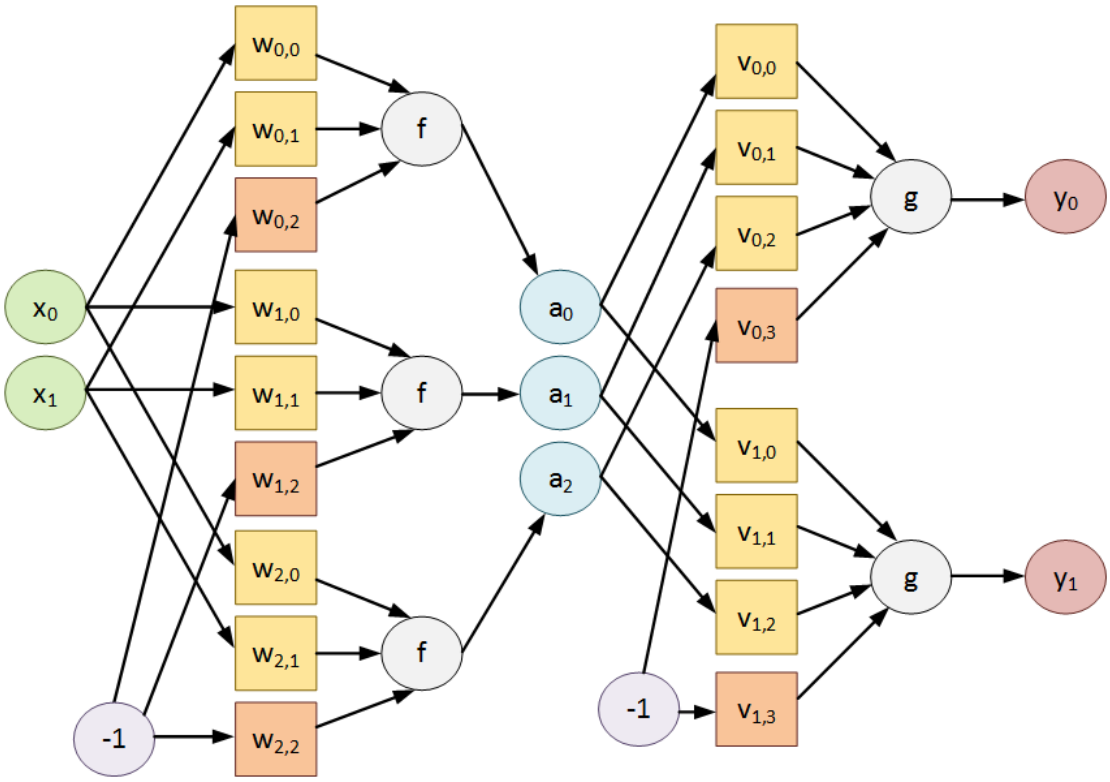
3. vektor skrytej aktivácie rozšírime o hodnotu $a_s = -1$
4. výstup MLP vypočítame zo skrytej aktivácie a váh na výstupnej vrstve

$$y_k = g(\sum_{j=0}^{s} a_j v_{j,k}) \text{ kde } k \in \{0,1,\ldots,m-1\}$$

(Marsland, 2009)

Výpočet môžme zjednodušiť použitím maticových operácií

$$\mathbf{y} = g(\mathbf{a} \times \mathbf{V}) \text{ kde } \mathbf{a} = f(\mathbf{x} \times \mathbf{W})$$



Obr. 3.5: Diagram znázorňuje postup výpočtu výstupného vektora **y** pre MLP s dvomi vstupmi, tromi neurónmi na skrytej vrstve a dvomi výstupmi

3.3.2 Učiaci algoritmus

Pre učenie MLP sa používa rovnaký algoritmus ako pre neurón. Ten však nepopisuje spôsob ako určiť chybu $\delta_{\mathbf{a}}$ na skrytej vrstve neurónu. Metóda *spätného šírenia* (Back propagation) počíta túto chybu spätným prenásobím chyby $\delta_{\mathbf{y}}$ na výstupnej vrstve váhami $\mathbf{V}$ na výstupnej vrstve. Pomocou vektora skrytých aktivácií $\mathbf{x}$ je potom možné upraviť váhy $\mathbf{W}$ na skrytej vrstve (Marsland, 2009).

Postup úpravy váh pre vstupný vektor $\mathbf{x}$, výstupný vektor $\mathbf{y}$ a očakávaný výstupný vektor $\mathbf{t}$:

1. vypočítame vektor chyby $\delta_{\mathbf{y}}$ na výstupnej vrstve

$$\delta_{y_k} = (t_k - y_k)y'_k \text{ pričom } y'_k = g'(\textstyle\sum_{j=0}^{s} a_j v_{j,k}(t-1)) \text{ pre } k \in \{0,1,\ldots,m-1\}$$

2. použitím spätného šírenia vypočítame vektor chyby $\delta\mathbf{a}$ na skrytej vrstve

$$\delta_{a_j} = a'_j(\textstyle\sum_{k=0}^{m-1} \delta_{y_k} v_{j,k}) \text{ pričom } a'_j = f'(\textstyle\sum_{i=0}^{n} x_i w_{i,j}(t-1)) \text{ pre } j \in \{0,1,\ldots,s-1\}$$

3. vektor skrytej aktivácie rozšírime o hodnotu $a_s = -1$

4. upravíme váhy na výstupnej vrstve

$$v_{j,k}(t) = v_{j,k}(t-1) + \alpha a_j \delta_{y_k} \text{ pre } j \in \{0,1,\ldots,s\} \text{ a } k \in \{0,1,\ldots,m-1\}$$

5. vstupný vektor rozšírime o hodnotu $x_n = -1$

6. upravíme váhy na skrytej vrstve

$$w_{i,j}(t) = w_{i,j}(t-1) + \alpha x_i \delta_{a_j} \text{ pre } i \in \{0,1,\ldots,n\} \text{ a } j \in \{0,1,\ldots,s-1\}$$

(Marsland, 2009)

Tento postup je možné zjednodušiť použitím maticových operácií.

Kapitola 4

Starcraft: Broodwar

Starcraft: Broodwar je úspešná RTS hra vydaná v roku 1998, ktorá sa do dnes hráva na medzinárodných súťažiach a na AI turnajoch po celom svete. Hra je zasadená do budúcnosti, kde medzi sebou bojujú tri rasy: Terania, Zergovia a Protosi. Hrač môže hrať za akúkoľvek z týchto rás, pričom každá má vlastné budovy a jednotky a vyžaduje si odlišný spôsob hrania od ostatných.

Každá rasa má bojové jednotky a robotníkov. Robotníci ťažia suroviny a stavajú budovy. Jednotky sú buď pozemné alebo vzdušné, pričom pozemné jednotky musia pri pohybe obchádzať prekážky na hernom poli.

Budovy rozdeľujeme na:

- *hlavné*, kde robotníci odovzdávajú naťažené suroviny
- *produkčné*, ktoré vyrábajú nové jednotky
- *technologické*, ktoré umožňujú vyrábať a stavať pokročilejšie jednotky a budovy

Hra má dva typy surovín: minerály a plyn, ktoré sa dajú ťažiť na rôznych miestach na hernom poli. Tretia surovina je počet voľných miest pre jednotky (supply). Každá jednotka, ktorú ovládame zaberá istý počet miest a na to, aby sme vyrobili novú potrebujeme mať na ňu dostatok voľných miest. Každá rasa si vie počet miest zvyšovať tým že, postaví špeciálnu budovu (alebo jednotku). Stavanie každej budovy a výroba každej jednotky vždy trvá istý čas.

V štandardnom zápase hrajú proti sebe dvaja hráči, pričom víťazí hráč, ktorý zničí všetky súperove budovy.

4.1 Cannon rush

Stratégiu cannon rush môže hráč využiť jedine ak hrá za Protossov. Útok prebieha tak, že na začiatku hry vyšle probu (robotnícka jednotka Protossov) do súperovej základne. Na miesto kam súper nemá dohľad začne stavať kanóny (obranné veže Protossov) tak, aby sa postupne približoval k súperovej základni, ktorú nakoniec kanóny zničia.

Hráč môže stavať kanóny až keď má postavenú budovu forge. Kanóny sa dajú postaviť len v blízkosti budovy pylón, pričom samotné pylóny je možné stavať kdekoľvek. Kanóny a pylóny sa snažíme umiestniť tak, aby boli kryté aspoň jedným kanónom.

Hráč, ktorý využíva túto stratégiu ťaží len minerály, keďže na žiadnu z budov ktoré stavia nepotrebuje plyn.

Obr. 4.1: Ukážka protosskej základne. Na obrázku vidno hlavnú budovu nexus (ružový štvorec), forge (červený štvorec) a dva pylóny (zelené kruhy) a proby (žlté kruhy), ktoré ťažia minerály a plyn

Okrem toho hráč musí:

- nájst súperovu základňu
- udržiavať naživе probu v súperovej základni
- vhodne voliť miesta na stavanie kanónov a pylonov
- rozhodovať na čo míňať naťažené minerály

Súper sa dokáže proti cannon rushu ubrániť ak ho odhalí ešte v počiatočnej fázi. Útočník je vtedy v nevýhodnej situácii, keďže nemá ešte žiadne bojové jednotky ani produkčné budovy.

Obr. 4.2: Na obrázok je ukážka úspešneho Cannon Rush-u. Útok postupuje ďalej do súperovej základne, kam proba akurát pokladá nový kanón

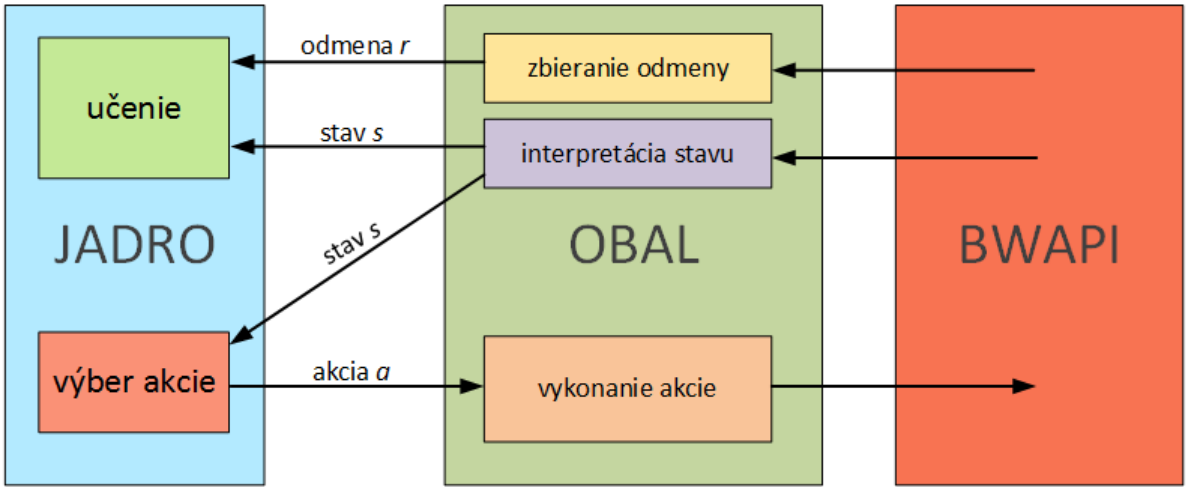
Kapitola 5

Implementácia

Na vytvorenie agenta sme použili knižnicu BWAPI, ktorá poskytuje API na priamu interakciu s hrou Starcraft: Broodwar. V agentovi navyše využívame knižnicu, vytvorenú Jakubom Trančíkom (Trančík, 2013), ktorá za nás vyberá najvhodnejšie umiestnenie pre pylóny a kanóny v súperovej základni. Celý kód agenta je napísaný v jazyku C++.

Náš agent sa skladá z dvoch častí:

- *Jadro*, ktoré vyberá akcie v rozhodovacích krokoch a učí sa optimálnu stratégiu
- *Obal*, ktorý realizuje zvolené akcie, zbiera odmeny medzi rozhodovacími krokmi a posiela ich jadru spolu s aktuálnym stavom



Obr. 5.1: Diagram ukazuje ako v agente medzi sebou komunikujú jadro, obal a BWAPI. Obal sleduje stav hry cez volania BWAPI, pričom v rozhodovacích krokoch pošle jadru odmenu a stavový vektor. Jadro sa na základe týchto informácii učí a vyberá akcie, ktoré obal za neho vykoná cez volania v BWAPI

5.1 Jadro

Prvá časť agenta, jadro, obsahuje implentáciu učiaceho algoritmu. Jeho úloha je voliť akcie v rozhodovacích krokoch. V impementácii používame *boltzmannovú exploráciu* a na učenie algoritmus *Advantage learning*. Pre každú akciu má jadro samostatnú neurónovú sieť, ktorá aproximuje jej advantage funkciu $A(s,a)$ pre každý stav $s \in S$. Jadro funguje nezávisle od prostredia, pre ktoré vyberá akcie.

Neurónovú sieť sme realizovali ako viacvrstvový perceptrón s jednou skrytou vrstvou. Na skrytej vrstve používame sigmoidu $f(x) = \frac{1}{1+e^{-x}}$ ako aktivačnú funckiu, pričom na výstupnej vrstve používame lineárnu funkciu $f(x) = x$.

V jadre je možné nastaviť nasledujúce parametre:

- počet neurónov na skrytej vrstve neurónových sietí
- rýchlosť učenia α pre neurónové siete
- faktor diskontácie γ pri ohodnocovaní stavov
- teplotu τ pre boltzmannovú exploráciu
- časový koeficient K pre určovanie advantage hodnôt

5.2 Obal

Obal tvorí zvyšok agenta a vytvára abstraktné prostredie pre jadro, ktoré priamo narába s funkciami BWAPI. Úlohou obalu je sledovať prostredie hry a posielať jadru stav, reprezentovaný ako vektor číselných hodnôt.

Stav v sebe nesie len také informácie, ktoré sme usúdili ako relevantné pre riešenie úlohy. Hodnoty stavu obal normalizuje do intervalu $[0,1]$.

Nasledujúca tabuľka popisuje jednotlivé hodnoty, ktoré sa v stavovom vektore **s** nachádzajú spolu so spôsobom akým ich obal normalizuje:

	Názov	**Normalizácia** $f(x)$
s_0	Počet voľných miest pre jednotky	$sig(0,4x-2,5)$
s_1	Počet jednotiek typu probe	$sig(0,375x-3,0)$
s_2	Počet budov typu Nexus	x
s_3	Počet budov typu Forge	x
s_4	Pomer počtu kanónov ku pylónom	$tanh(0,3x)$
s_5	Počet súperových robotníkov	$sig(0,375x-3,75)$
s_6	Počet slabých súperových jednotiek	$tanh(0.175x)$
s_7	Počet silných súperových jednotiek	$tanh(0.35x)$
s_8	Počet súperových obranných budov	$tanh(0.37x)$
s_9	Počet súperových produkčných budov pre slabé jednotky	$tanh(0.37x)$
s_{10}	Počet súperových produkčných budov pre silné jednotky	$tanh(0.37x)$
s_{11}	Počet súperových technologických budov	$tanh(0.275x)$

Obr. 5.2: Agent zbiera informácie o stave hry, ktoré následne normalizuje a posiela jadru v podobe stavového vektora **s**. Na normalizáciu používame sigmoidu $sig(x)$ alebo hyperbolický tangens $tanh(x)$

Hlavnou úlohou obalu je vykonať akciu, ktorú zvolilo jadro. Agent môže vykonávať nasledujúce akcie:

- **Train Probe**, pridá do produkcie probu
- **Build Base Pylon**, postaví v základni pylón
- **Build Forge**, postaví v základni budovu forge
- **Build Attack Pylon**, postaví pylon na vhodné miesto v súperovej základni
- **Build Attack Cannon**, postaví kanón na vhodné miesto v súperovej základni
- **Wait**, počká kým sa našetrí dostatok surovín na ďalšie akcie

Akciu Train Probe je možné vykonať len ak agent má dostatok minerálov a ak vlastní budovu Nexus, v ktorej práve neprebieha výroba inej jednotky.

Akcie, ktoré stavajú budovy je možné vykonať ak má agent dostatok minerálov a ak existuje miesto, kde sa dá príslušná budova postaviť. Akcia prebieha kým sa budova nezačne stavať a teda nie je možné dovtedy vykonávať žiadnu inú akciu.

Po zvolení akcie Wait agent čaká istý pevne stanovený počet framov, pričom čakanie je prerušené akonáhle sa agentovi sprístupní nejaká nová akcia.

Obal čaká na frame (časový krok hry), v ktorom je potrebné zvoliť nejakú akciu. Takýto frame (rozhodovací krok) nastane ak práve neprebieha žiadna akcia a ak je možné vykonať aspoň jednu akciu.

Obr. 5.3: Ukážka testovacej verzie agenta vytvára lepšiu predstavu o fungovaní nášho agenta. Na ľavom hornom okraji sú zeleným vypísané akcie, ktoré agent môže vykonať. Zvyšný textový výstup je výpis stavového vektora **s**

Kapitola 6

Experimenty

Náš agent sa v experimentoch snaží poraziť natívne AI použitím stratégie Cannon Rush. Správanie natívneho AI nie je možné presne predpovať, keďže v každom zápase náhodne volí stratégiu a snaží sa do istej miery prispôsobiť sa súperovi. To nám dáva možnosť overiť schopnosť agenta prispôsobovať svoje rozhodnutia rôznym situáciám.

Všetky experimenty sa odohrávajú na jednej mape, a súper hrá len za rasu Teranov. Chceli sme, aby sa agent rozhodoval na základe neskreslených informácií o súperovi, preto sme mu poskytli plnú viditeľnosť celej hracej plochy. V štandardnom zápase musí agent posielať jednotky na prieskum, aby získal akúkoľvek informáciu o súperovi, to je však nad rámec našej práce.

Aby sme zaručili správnosť výsledkov, pre každý experiment sme nechali zbehnúť 5 inštancií, pričom v každej sme nechali agenta odohrať aspoň 5000 zápasov, aby sa stihol natrénovať.

Najdôležitejší aspekt, ktorý skúmame je konečný výsledok každého zápasu, teda či agent vyhral alebo prehral. Zápasy sme nechali bežať maximálne 25000 herných frame-ov (približne 20 minút v turnajovej rýchlosti), po ktorých sme hru ukončili a na základe zostávajúcich jednotiek každého hráča vyhodnotili zápas buď ako výhru, prehru alebo ako remízu.

Pre každý experiment sme vytvorili graf, ktorý porovnáva celkový počet výhier, prehier a remíz v priebehu trénovania.

Uviedli sme si, že Cannon Rush väčšinou zlyhá, keď je útočník priskoro odhalený. Sledujeme preto aj čas postavenia prvého pylónu a kanónu, nakoľko jeho nižšie hodnoty zvyšujú šancu agenta na úspech.

Tabuľka spoločných hodnôt parametrov agenta

Názov	**Hodnota**
Počet skrytých neurónov	15
Faktor diskontovania	0,9999
Časový koeficient	0,35
Rýchlosť učenia	0,05

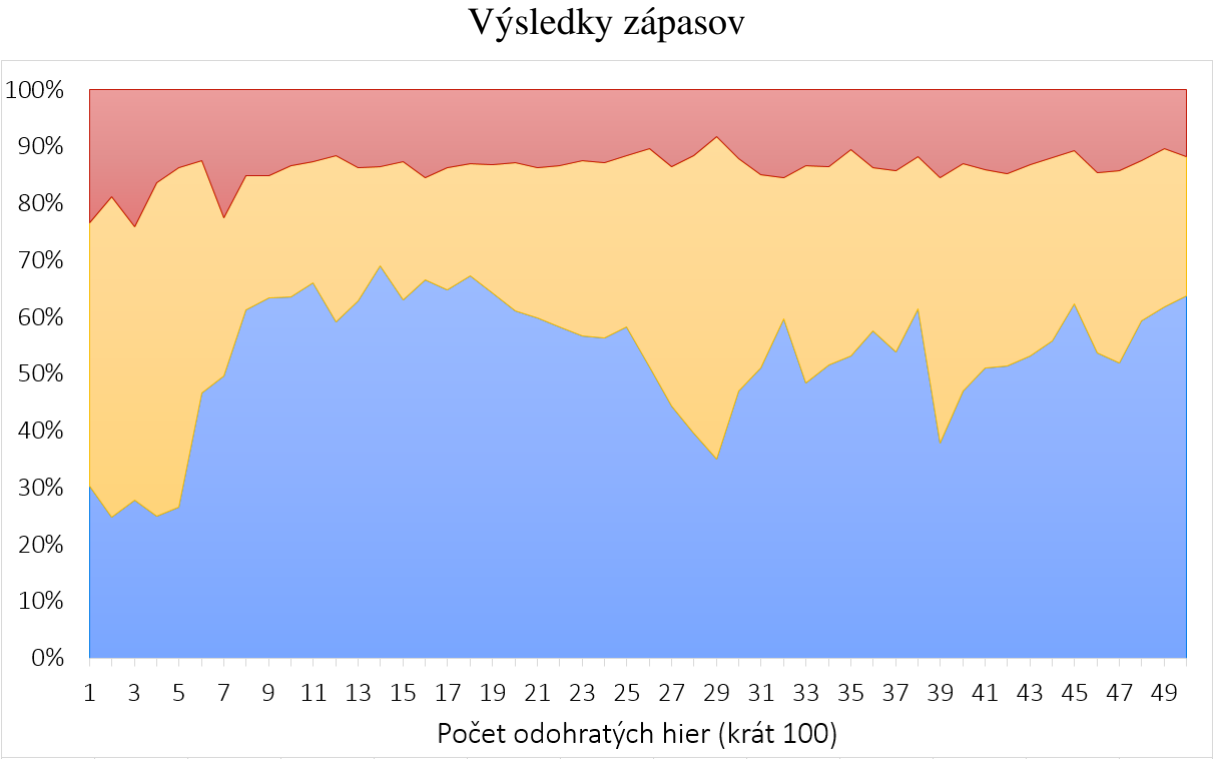
6.1 Prvý experiment: počiatočný pokus

Tento experiment, pre nás slúžil ako odrazový bod, návrh nasledujúcich expirementov závisel od výledkov práve tohto experimentu. Zvolili sme v ňom nízku hodnotu teploty pre boltznamovú exploráciu $\tau = 0,075$. Odmeny sme dávali len na konci hry. Za výhru dostal agent odmenu $+10$, za remízu trest -5 a za prehru trest -10.

Časy postavenia prvých budov v súperovej základni

	Pre výhry	**Pre remízy**	**Pre prehry**
Prvý pylón	2835	2789	2889
Prvý kanón	3602	3604	4359

Obr. 6.1: Tabuľka uvádza priemerný frame, v ktorom agent postavil prvý pylón a kanón v súperovej základni. Priemer sme počítali samostatné pre zápasy v ktorých agent vyhral, remizoval a prehral



Obr. 6.2: Graf ukazuje pomer počtu výhier (modré), prehier (žlté) a remíz (červené) v závislosti od počtu odohratých hier

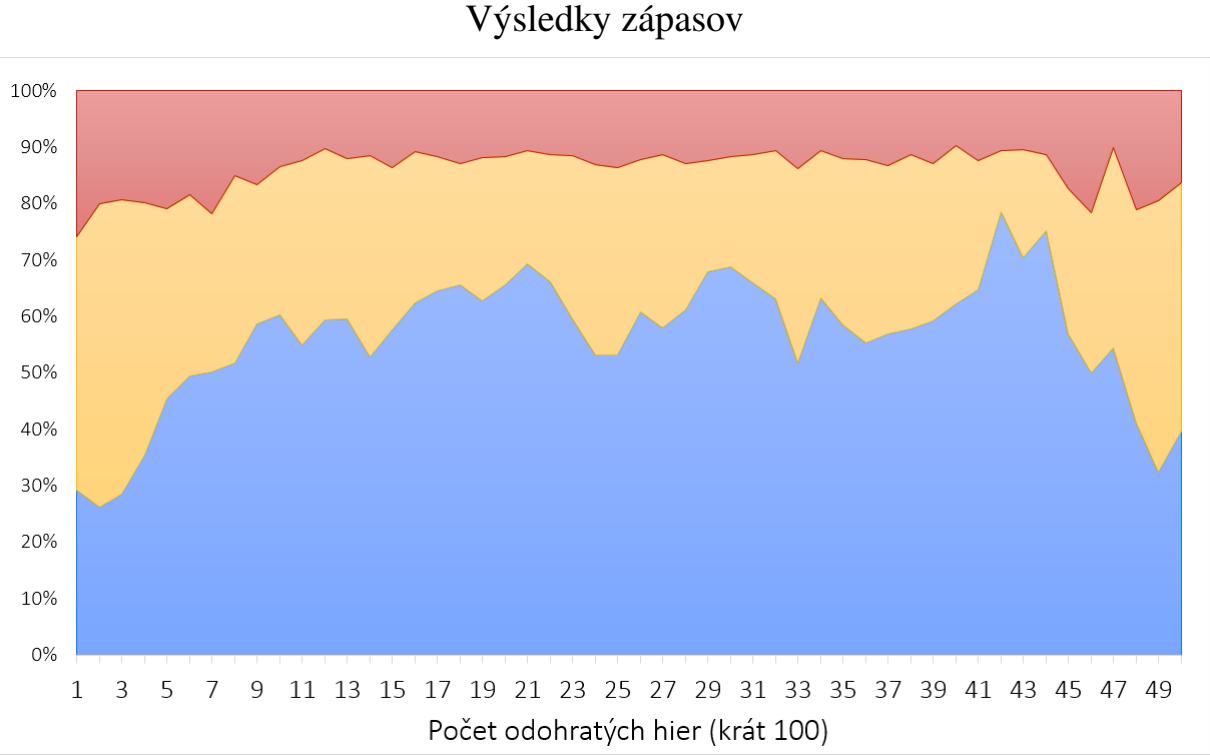
6.2 Druhý experiment: zníženie explorácie

Agent si dokázal v prvom experimente udržať pomerne nízky počet prehier, počet výhier a remíz sa však nedokázal ustáliť. Usúdili sme, že to bol dôsledok vysokej miery explorácie. V tomto experimente overujeme vplyv explorácie na úspešnosť agenta, teplotu sme preto znížili na polovicu oproti prvému experimentu $\tau = 0,0375$. Pravidlá odmeňovania sme použili rovnaké ako v prvom experimente.

Časy postavenia prvých budov v súperovej základni

	Pre výhry	**Pre remízy**	**Pre prehry**
Prvý pylón	2839	2860	3090
Prvý kanón	3605	3694	4569

Obr. 6.3: Tabuľka uvádza priemerný frame, v ktorom agent postavil prvý pylón a kanón v súperovej základni. Priemer sme počítali samostatné pre zápasy v ktorých agent vyhral, remizoval a prehral



Obr. 6.4: Graf ukazuje pomer počtu výhier (modré), prehier (žlté) a remíz (červené) v závislosti od počtu odohratých hier

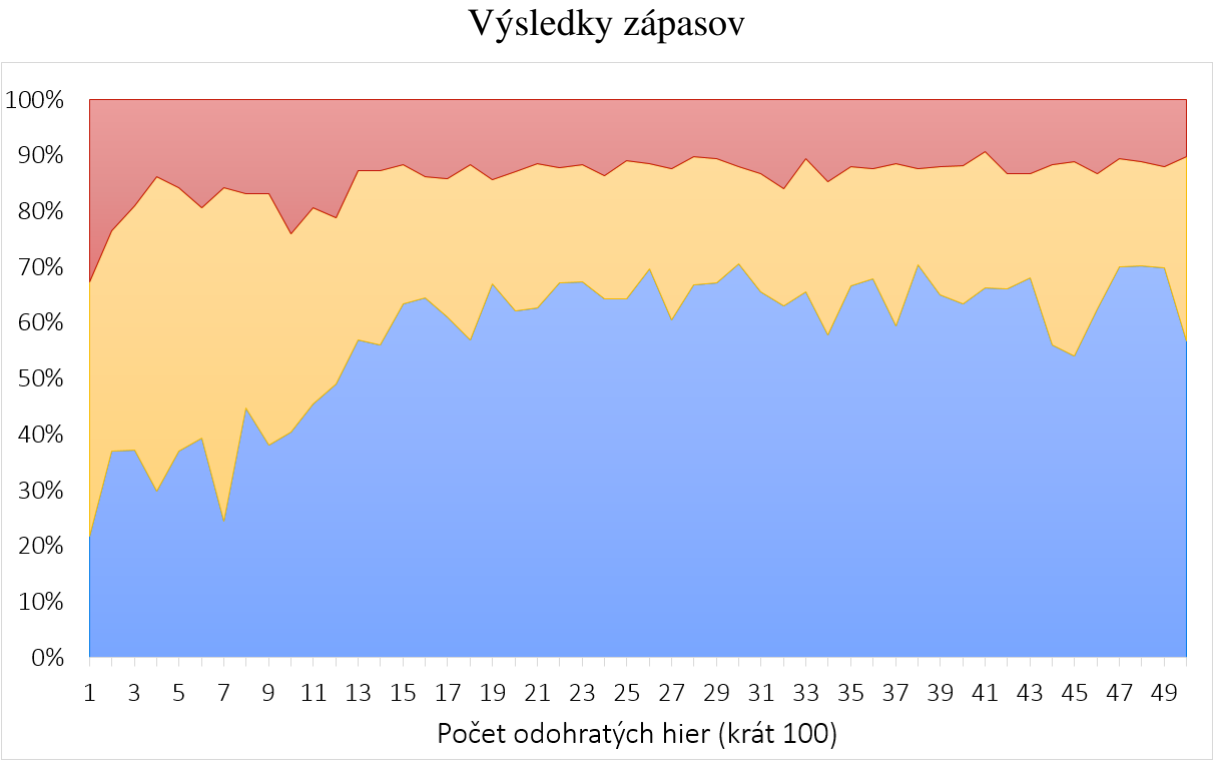
6.3 Tretí experiment: zmena spôsobu odmeňovania

V treťom experimente meníme pravidlá odmeňovania, tak že za výhru agent dostáva odmenu $+2$, za remízu trest -2 a za prehru trest -4. Navyše agent počas zápasu dostáva odmenu $+0.1$ za každú súperovu budovu ktorú zničí. Cieľom týchto zmien je usmerňovať agenta počas hry a relatívne prísnejšie trestať prehry a remízy. Samotné hodnoty odmien a trestov sme sa rozhodli znížiť, aby v boltzmannovej explorácii nevznikali príliš vysoké hodnoty. Mieru explorácie sme použili rovnakú ako v druhom experimente.

Časy postavenia prvých budov v súperovej základni

	Pre výhry	**Pre remízy**	**Pre prehry**
Prvý pylón	2833	2749	2836
Prvý kanón	3601	3585	4289

Obr. 6.5: Tabuľka uvádza priemerný frame, v ktorom agent postavil prvý pylón a kanón v súperovej základni. Priemer sme počítali samostatné pre zápasy v ktorých agent vyhral, remizoval a prehral



Obr. 6.6: Graf ukazuje pomer počtu výhier (modré), prehier (žlté) a remíz (červené) v závislosti od počtu odohratých hier

Kapitola 7

Diskusia

V každom experimente sa nám podarilo o málo zvýšiť celkový počet výhier oproti predošlému a keďže, parametre agenta nám vytvárajú veľký priestor možností predpokladáme, že je ešte možné nájsť konfigurácie s lepšími výsledkami.

Agentovi vo všetkých pokusoch trvá približne 1000 zápasov kým dosiahne aspoň nadpolovičný počet výhier. Občas sa však vyskytne krátky úsek, v ktorom počet remíz prudko narastie na úkor počtu výhier. Tento problém sme čiastočne vyriešili v druhom experimente znížením miery explorácie, ktorá nám niekedy vyberá aj nevýhodné akcie. Alternatívne riešenie je v priebehu učenia postupne znižovať mieru explorácie alebo meniť ju podľa úspešnosti agenta, čo znamená, že necháme agenta veľa explorovať kým nenájde dostatočne dobré riešenie a vtedy mu mieru explorácie znížime (Russell and Norvig, 2003).

7.1 Analýza remíz

Natívne AI má takmer nulovú šancu vyhrať zápas, v ktorom sa agentovi podarí postaviť 2 až 4 kanóny pri jeho základni. To či takýto zápas pre agenta skončí víťazstvom alebo remízou závisí už len od akcií, ktoré ďalej zvolí.

Zápas končí remízou, keď agent príliš pomaly postupuje kanónmi do základne súpera. Zvyčajne sa to stáva keď agent stavia príliš veľa pylónov alebo prób v základni. Vzhľadom na konečný priestor, kde agent môže stavať budovy by takmer všetky remízy dopadli ako víťazstvo ak by sme ich neukončili predčasne.

Remízy sme trestali, aby sa agent naučil rýchlo porážať súpera. Agent však nevie rozlíšiť remízu, v ktorej chýba len krátko na zničenie poslednej súperovej budovy od remízy, v ktorej sa kanóny nedostali súperovi do základne a len mu bránili zaútočiť. Toto do istej miery riešia odmeny za ničenie budov v treťom experimente, kde celkovo pozorujeme pokles počtu remíz oproti výhram.

7.2 Analýza prehier

Ak súper dostatočne skoro odhalí budovy, ktoré agent stavia pri jeho základni a podarí sa mu ich všetky zničiť tak to znamená skoro istú prehru pre nášho agenta. Toto z časti závisí od stratégie, ktorú si natívne AI náhodne zvolilo ale z väčšej časti práve od rozhodnutí agenta.

V tabuľkách priemerných časov postavenia prvého kanónu a pylónu vidíme, že prvý pylón agent postaví vždy v takmer rovnaký čas, pričom v prípade prehry prvý kanón postaví v priemere neskôr, než v ostatných prípadoch. To znamená, že výsledok hry rozhodujú tie akcie, ktoré agent zvolí tesne po postavení prvého pylónu. Napriek tomu, že v ani jednom prípade sa agentovi nepodarilo naučiť sa vyberať tieto akcie spoľahlivo dobre, je v treťom experimente tento čas nižší než vo zvyšných dvoch experimentoch. Domnievame sa, že to je dôsledok zvýšenia trestu za prehru.

Záver

Naštudovali sme si učenie posilňovaním a dopredné neurónové siete a pomocou nich navrhli agenta, ktorý sa učí na základe odmien a trestov efetkivne stavať budovy a vyrábať jednotky v RTS hrách. Na základe tohto návrhu a pomocou knižnice BWAPI sme implementovali agenta pre hru Starcraft: Brood War, ktorý využíva stratégiu cannon rush na porazenie súpera.

Vo vybraných pokusoch sme skúšali trénovať agenta pri rôznej miere explorácie a odlišných spôsobov odmeňovania. Agent sa vždy dokázal naučiť hrať tak, aby prehrával maximálne 10% odohraných zápasov. Napriek tomu, že pre Cannon Rush vieme vopred zvoliť postupnosť akcií, ktorá dosahuje takmer stopercentnú úspešnosť (Trančík, 2013), pre samo učiaci algoritmus považujeme takýto výsledok za úspech.

Jadro agenta sme implementovali ako samostatnú knižnicu, ktorá sa nezávisle od úlohy učí vyberať akcie na základe stavu a odmien z prostredia. Stačí preto pozmeniť implementáciu agenta, pridaním nových akcií a zmenením interpretácie stavov, a vieme tak nielen zmeniť stratégiu, ktorú agent využíva ale dokonca vytvoriť agenta pre inú RTS hru podobnú Starcraftu.

Do návrhu agenta chceme pridať možnosť výroby bojových jednotiek a zakomponovať doňho moduly, ktoré by riešili úlohy ako je prieskum súperovej základne, či ovládanie jednotiek v boji. Našim cieľom je nakoniec vytvoriť agenta, ktorý je schopný hrať v turnajovom prostredí a zapojiť sa s ním do študentskej súťaže SSCAI pre umelé inteligencie v hre Starcraft.

Literatúra

Leemon C. Baird. *Reinforcement Learning Through Gradient Descent*. PhD thesis, School of Computer Science - Carnegie Mellon University, Pittsburgh, 1999.

Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators (Automation and Control Engineering)*. CRC Press, 2010. ISBN 978-1-439-82108-4.

Andries P. Engelbrecht. *Computational Inteligence (2nd Edition)*. Wiley, 2007. ISBN 978-0-470-03561-0.

Mance E. Harmon and Stephanie S. Harmon. Reinforcement learning: A tutorial. `http://www.nbu.bg/cogs/events/2000/readings/petrov/rltutorial.pdf`, 2000.

Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. CRC Press, 2009. ISBN 978-1-420-06718-7.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 2005. ISBN 978-0-471-72782-8.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2003. ISBN 978-0-137-90395-5.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. A Bradford Book, 1998. ISBN 978-0-262-19398-6.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2): 181–211, August 1999. ISSN 0004-3702. doi: 10.1016/S0004-3702(99)00052-1. URL `http://dx.doi.org/10.1016/S0004-3702(99)00052-1`.

Henk C. Tijms. *A First Course in Stochastic Models*. Wiley, 2003. ISBN 978-0-471-49880-3.

Jakub Trančík. Implementácia stratégie cannon rush v hre starcraft: Brood war (rozšírený abstrakt). Študentská vedecká konferencia FMFI UK, Bratislava, 2013.