UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

TRÉNOVANIE AGENTA PRE PRIESKUM V REAL-TIME STRATEGICKÝCH HRÁCH POMOCOU UČENIA POSILŇOVANÍM

BAKALÁRSKA PRÁCA

2014

Lucia Piváčková

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

TRÉNOVANIE AGENTA PRE PRIESKUM 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

Lucia Piváčková

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Lucia Piváčková
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 prieskum v real-time strategických hrách pomocou učenia posilňovaním / *Agent training for scouting tasks in real-time strategy games using reinforcement learning*

Cieľ:
1. Naštudujte problematiku Continuous Actor-Critic Learning Automaton (CACLA) algoritmu a dopredných neurónových sietí
2. Navrhnite, implementujte a natrénujte agenta pre prieskum 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

Čestné vyhlásenie

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

V Bratislave .....

Poďakovanie

Ďakujem svojmu vedúcemu práce Mgr. Viliamovi Dillingerovi za rady, nápady a odbornú pomoc a rodine a priateľom za podporu.

Abstrakt

Cieľom práce bolo navrhnúť, implementovať a natrénovať agenta pre prieskum v real-time stategickej hre StarCraft: Brood War pomocou učenia posilňovaním. Pri implementácii sme využili algoritmus CACLA (Continuous Actor-Critic Learning Automaton), dopredné neurónové siete a C++ framework BWAPI, ktorý umožňuje vytvárať vlastnú umelú inteligenciu do hry StarCraft: Brood War. Trénovanie agenta malo dve fázy - učenie najkratšej cesty na dané miesto a vyhýbanie sa nebezpečenstvu. V prvej fáze sa agent naučil prísť na dané miesto, ale v druhej už nebol veľmi úspešný.

Kľúčové slová: *CACLA, učenie posilňovaním, neurónové siete, StarCraft, prieskum, mikromanažment*

Abstract

Our task in this paper was to design, implement and train an agent for scouting in the real-time strategy game StarCraft: Brood War using reinforcement learning. We used the CACLA algorithm, feedforward neural networks and the BWAPI framework , a C++ library that allows you to create an artificial intelligence for StarCraft. The agent's training was divided into two phases - learning of the shortest path to a designated place and avoiding danger. The agent has successfully accomplished his task in the first phase, but he hasn't performed very well in the second one.

Keywords: *CACLA, reinforcement learning, neural networks, StarCraft, scouting, micromanagement*

Obsah

Úvod

Teória optimálneho riadenia je dôležitou súčasťou umelej inteligencie a našla si svoje uplatnenie vo viacerých odboroch. Skúma situácie, kedy na ceste agenta k cieľu exisuje viacero alternatív, agent k cieľu postupuje v krokoch a po každom kroku sa opakovane rozhoduje, ktorým smerom sa ďalej vyberie. Jeho rozhodnutie ovplyvní nielen výsledok v nasledujúcom kroku, ale aj možnosti ďalšieho výberu. Real-time strategické hry ponúkajú v tejto oblasti veľa nových výziev. V prostredí existuje viacero agentov, veľké množstvo stavov, do ktorých sa agent môže dostať, akcií, ktoré vie vykonať a významný faktor tu hrá čas. Agent sa musí vedieť rýchlo rozhodovať vzhľadom na dané okolnosti. Výpočty nesmú trvať príliš dlho, inak udalosti, na ktoré reaguje, budú v tom čase neaktuálne.

Tému tejto práce sme si zvolili na riešenie problému, na ktorý narazil iný projekt, ktorého cieľom bolo naučiť agenta stratégiu „cannon rush“ v real-time strategickej hre StarCraft: Brood War. Táto stratégia spočíva v zaskočení nepriateľa tým, že na začiatku hry začneme stavať kanónové veže v blízkosti jeho základne. V tomto momente väčšina nepriateľov nemá ešte vybudovanú efektívnu obranu a nemôže sa nijako brániť. Kanóny potom zničia dôležité budovy alebo nepriateľských robotníkov, ktorí ťažia suroviny a tým narušia jeho ekonomiku. Projekt narazil na problém, kde robotník, ktorý staval kanóny nebol efektívny – ľahko sa nechával zničiť nepriateľom alebo po postavení kanónu ostal stáť na mieste.

Cieľom našej práce bolo naučiť robotníka, aby vedel bezpečne stavať kanóny a taktiež robiť prieskum, pričom by sa vyhýbal poškodeniu a následnému zničeniu. Pod prieskumom môžeme chápať objavovanie nepreskúmaných častí mapy, hľadanie nových zdrojov surovín, ale aj nájdenie a pozorovanie nepriateľa. Čím skôr vidíme nepriateľa, tým rýchlejšie dokážeme identifikovať jeho stratégiou a adekvátne na ňu reagovať. Dobré informácie o nepriateľovi sú jednou z kľúčových vecí pre víťazstvo v real-time strategických hrách. Naša práca sa dá taktiež využiť aj na iné problémy v hre, napríklad ovládanie jednotiek v boji.

Prvá kapitola práce sa venuje neurónovým sieťam a ich učeniu pomocou algoritmu spätného šírenia chyby. V druhej kapitole sme popísali učenie posilňovaním, Markovove rozhodovacie procesy a niektoré algoritmy učenia posilňovaním. Bližšie sme sa venovali metóde aktér-kritik a jej rozšíreniu v podobe algoritmu CACLA (Continuous Actor-Critic Learing Automaton), ktoré využívame na učenie nášho agenta. Ďalšia kapitola pojednáva o hre StarCraft: Brood War, kde sme priblížili základné princípy a priebeh hry a podrobnejšie opísali prieskum mapy. V kapitole s názvom „Implementácia“ sme opísali navrhnutý model agenta, ktorý sme aj implementovali do hry. V poslednej kapitole sme prezentovali výsledky učenia agenta a porovnania vykonaných pokusov. Dosiahnuté výsledky sme zhodnotili v diskusii.

Kapitola 1

Neurónové siete

Umelé neurónové siete (ďalej len neurónové siete) predstavujú výpočtový model inšpirovaný štruktúrou a funkcionalitou neurónov v ľudskom mozgu. Funkciou neurónu v mozgu je rýchle spracovanie informácie, ktorú prijme z prostredia a reakcia na ňu. A práve túto schopnosť sa snažia neurónové siete napodobniť. Taktiež pre ne existuje viacero učiacich algoritmov.

Neurónová sieť je realizácia nelineárneho mapovania z R^n do R^m,

$$f : R^n \rightarrow R^m$$

kde n a m sú veľkosti vstupného a cieľového priestoru (Engelbrecht, 2007).

Neurónové siete môžu reprezentovať akúkoľvek spočítateľnú funkciu. V praxi sa neurónové siete najviac používajú na klasifikáciu a funkčnú aproximáciu alebo mapovanie funkcií pri použití množstva trénovacích dát, kde systémy s jasne stanovenými pravidlami (ako sú expertné systémy) zlyhávajú. Neurónové siete sú pritom tolerantné k neurčitostiam v trénovacích dátach. (Kvasnička et al., 1997)

Existuje viacero typov neurónových sietí (rekurentné, Hopfieldove,...). V našej práci budeme využívať dopredné neurónové siete, ktoré sú popísané v nasledujúcich častiach. Ako prvé si zadefinujeme základný model neurónovej siete – perceptrón.

1.1 Perceptrón

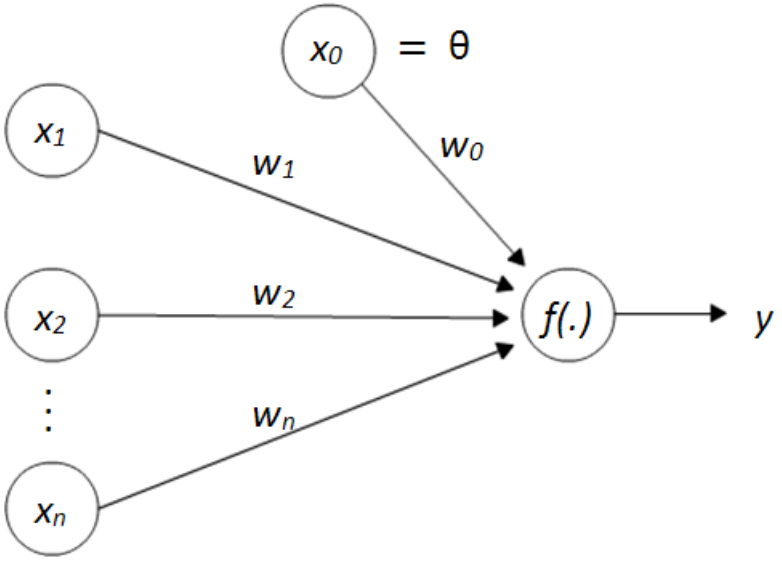
Základnou stavebnou jednotkou neurónovej siete je perceptrón. Perceptrón dostane na vstup vektor reálnych hodnôt $\boldsymbol{x} = (x_1, x_2, .., x_n)$. Ku každému vstupu x_i je priradená synoptická váha w_i, ktorá ovplyvňuje silu daného vstupného signálu. Perceptrón vypočíta váženú sumu všetkých vstupných signálov a aplikovaním aktivačnej funkcie f dostane výstup $\boldsymbol{y}$, ktorého silu ovplyvňuje prah extitácie θ. Matematicky môžeme jeho aktiváciu popísať v tvare:

$$y = f(\sum_{i=1}^{n} w_i x_i - \theta)$$

Prah θ môžeme považovať za špeciálny prípad váhy so vstupom $x_{n+1} = -1$ a $w_{n+1} = \theta$. Aktivácia sa potom zmení na:

$$y = f(\sum_{i=1}^{n+1} w_i x_i)$$

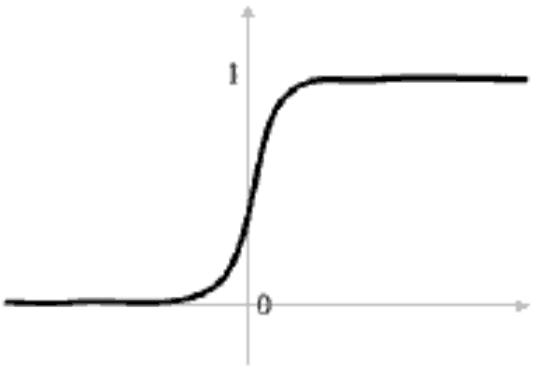
(Engelbrecht, 2007)



Obr. 1.1: Model perceptrónu

Pre výber aktivačnej funkcie existuje viacero možností – napríklad prahová funkcia, Gaussovská funkcia, hyperbolický tangens alebo sigmoida. Najčastejšie sa ako aktivačná funkcia používa sigmoida pre jej dobré vlastnosti - monotónnosť, neohraničenosť a derivovateľnosť.

Sigmoida má tvar $f(x) = \frac{1}{1+e^{-x}}$ a $f(x) \in (0, 1)$ (Engelbrecht, 2007).



Obr. 1.2: Sigmoida (Marsland, 2009)

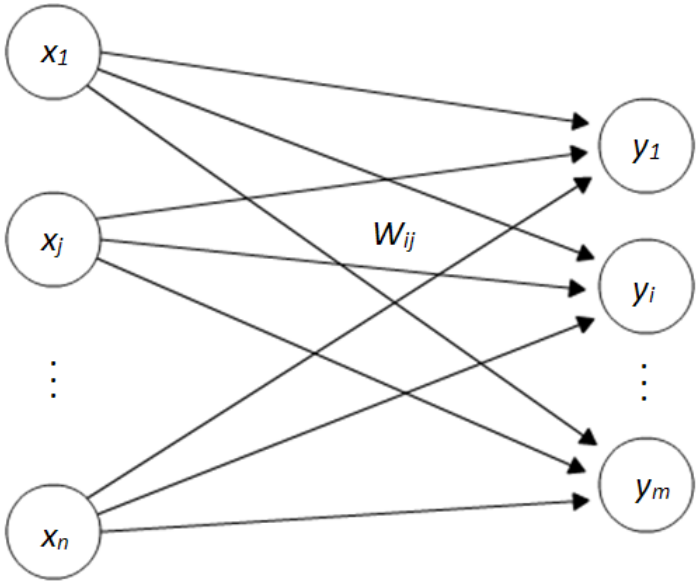
1.2 Jednovrstvová neurónová sieť

Prepojením viacerých perceptrónov získame jednovrstvovú neurónovú sieť. Každý neurón v sieti pracuje samostatne nad celou množinou vstupov.

Ak neurónová sieť pozostáva z m neurónov a n vstupných kanálov, potom váhy všetkých prepojení možno prehľadne usporiadať do váhovej matice $\boldsymbol{W}$ veľkosti [m $\times$ n]. Po priložení vektora $\boldsymbol{x} = (x_1, x_2, .., x_n)^T$ na vstup siete dostávame ako výstup vektor $\boldsymbol{y} = (y_1, y_2, .., y_m)^T$, kde $y_1, y_2, .., y_m$ sú aktivácie výstupných neurónov, pričom

$$\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{x})$$

(Kvasnička et al., 1997)

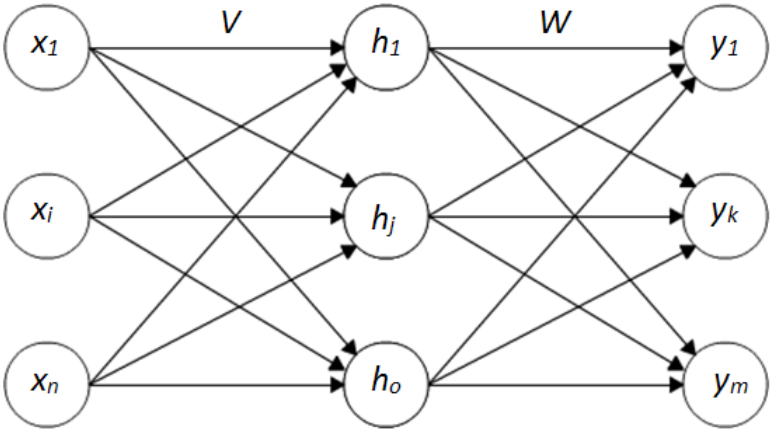


Obr. 1.3: Model jednovrstvovej siete

1.3 Viacvrstvová dopredná neurónová sieť

Viacvrstvová neurónová sieť vznikne pridaním ďalších (skrytých) vrstiev tak, že výstup nižšej vrstvy je vstupom pre vyššiu. Pre viacvrstvové dopredné siete platí, že pre výpočet vrstvy V_i ($i > 1$) použijeme len výstupy z nižších vrstiev $V_1, ..., V_{i-1}$.

Takýmto spojením získame väčšiu silu ako pri jednej vrstve – napríklad jednovrstvová sieť nedokáže riešiť XOR, ale viacvrstvová už áno (Marsland, 2009). Naviac bolo dokázané, že neurónové siete tohto typu sú univerzálnym aproximátorom, t.j. sú schopné aproximovať s požadovanou presnosťou ľubovoľnú spojitú funkciu, čiže môžu byť chápané ako univerzálny prostriedok pre regresnú analýzu, kde tvar modelovej funkcie je určený architektúrou neurónovej siete (Kvasnička et al., 1997).



Obr. 1.4: Dopredná neurónová sieť s jednou skrytou vrstvou

Na učenie viacvrstvových neurónových sietí sa využíva **algoritmus spätného šírenia chyby** (angl. back-propagation). Na začiatku sú váhy nastavené na náhodné malé pozitívne aj negatívne hodnoty. Učenie pozostáva z dvoch častí, ktoré opakujeme až kým sa sieť nenaučí na nami požadovanú úroveň, pričom vstupné dáta stále náhodne obmieňame:

1. *Dopredný prechod* - vypočíta sa aktivácia na každej vrstve siete
2. *Spätné šírenie* - vypočítaná chyba na výstupe sa šíri z výstupnej vrstvy naspäť k vstupu, podľa ktorej sa upravujú váhy (a s nimi aj prah exitácie)

Nech $\boldsymbol{x} = (x_1, x_2, .., x_n)^T$ je vstup siete, $\boldsymbol{h} = (h_1, h_2, .., h_o)^T$ je vektor hodnôt na skrytej vrstve a $\boldsymbol{y} = (y_1, y_2, .., y_m)^T$ sú hodnoty neurónov na výstupe. Váhy medzi vstupnou a skrytou vrstvou sú reprezentované maticou $\boldsymbol{V}$ a váhy medzi skrytou a výstupnou vrstvou maticou $\boldsymbol{W}$. Aktivačná funkcia na skrytej vstve musí byť nelineárna.

Potom aktiváciu neurónov na výstupnej vrstve vypočítame ako:

$$\boldsymbol{h} = f(\boldsymbol{V}\boldsymbol{x})$$

$$\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{h})$$

Nech $\boldsymbol{d} = (d_1, d_2, .., d_n)^T$ je požadovaná hodnota výstupu, $\boldsymbol{h'} = \boldsymbol{V}\boldsymbol{x}$ a $\boldsymbol{y'} = \boldsymbol{W}\boldsymbol{h}$. Potom chybu na výstupnej a skrytej vrstve vypočítame ako:

$$\boldsymbol{\delta_y} = f(\boldsymbol{y'})(\boldsymbol{d} - \boldsymbol{y})$$

$$\boldsymbol{\delta_h} = f(\boldsymbol{h'})(\boldsymbol{W^T}\boldsymbol{\delta_y})$$

Váhy pre výstupnú a skrytú vrstvu upravíme nasledovne:

$$\boldsymbol{W} = \boldsymbol{W} + \alpha\boldsymbol{\delta_y}\boldsymbol{h^T}$$

$$\boldsymbol{V} = \boldsymbol{V} + \alpha\boldsymbol{\delta_h}\boldsymbol{x^T}$$

kde $\alpha \in (0, 1)$ je konštanta (learning rate), ktorá určuje ako rýchlo sa má sieť učiť. Pri vysokej α sa váhy menia o veľké hodnoty, čo môže spôsobiť nestabilitu siete. Menšie hodnoty zasa spôsobia, že sieť sa učí pomalšie, čím sa na každý vstup musí pozrieť viackrát a tým je sieť viac stabilná a odolná voči nepresnostiam vo vstupných dátach.

Kapitola 2

Učenie posilňovaním

Poznáme viacero metód strojového učenia:

- *Učenie s učiteľom* – agent sa učí porovnávaním svojho výsledku so správnou hodnotou, ktorú mu povie učiteľ
- *Učenie bez učiteľa* – alebo samoorganizácia, agent sa učí rozoznávať, v čom sa líšia dané vstupy (napríklad klasterizácia)
- *Učenie posilňovaním*

Pri učení posilňovaním sa agent učí na základe získaných odmien, ktoré dostane od prostredia po vykonaní akcie. Odmena môže byť aj záporná, čo je ekvivalentné trestu. Stratégiou agenta nazývame funkciu, ktorá vyberá akciu, ktorú má agent v danom stave vykonať. Cieľom agenta je nájsť takú stratégiu, ktorá maximalizuje jeho celkovú odmenu - optimálna stratégia. Agent si postupne ohodnocuje navštívené stavy, podľa čoho si upravuje svoju stratégiu.

Hodnota stavu je celková odmena, ktorú môže agent očakávať, že dostane v budúcnosti začínajúc týmto stavom. Kým odmeny určujú okamžitú vhodnosť stavu, hodnoty určujú dlhodobú vhodnosť stavov, ktoré pravdepodobne budú nasledovať berúc do úvahy odmeny, ktoré sú dostupné v týchto stavoch. Môže sa napríklad stať, že stav s nízkou odmenou má vysokú hodnotu, pretože za ním nasledujú iné stavy s vysokými odmenami.

(Sutton and Barto, 1998)

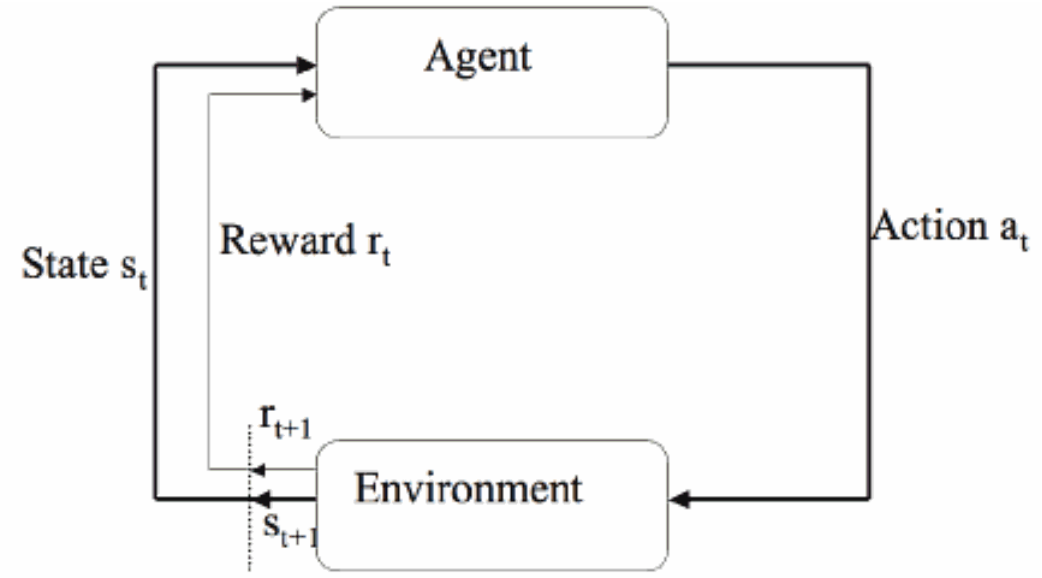
2.1 Markovove rozhodovacie procesy

Markovove rozhodovacie procesy (MDP) nám poskytujú matematický model pre rozhodovanie v situáciách, kde výsledky sú z časti náhodné a z časti pod kontrolou agenta. MDP sa zvyčajne využíva pri strojovom učení na formuláciu prostredia.

MDP je štvorica (S,A,R,P), kde:

- $\boldsymbol{S}$ je konečná množina všetkých stavov prostredia a $s_t \in S$ je stav agenta v čase t
- $\boldsymbol{A}$ je konečná množina všetkých dostupných akcií a $a_t \in A$ je akcia, ktorú agent vykoná v čase t
- $\boldsymbol{R} : S \times A \times S \to \mathbb{R}$ je funkcia, kde $R(s_t, a_t, s_{t+1})$ určuje odmenu pri prechode zo stavu s_t do s_{t+1} vykonaním akcie a_t
- $\boldsymbol{P} : S \times A \times S \to [0, 1]$ je prechodová funkcia, kde $P(s, a, s')$ je pravdepodobnosť, s akou sa agent dostane do stavu s', ak vykoná akciu a v stave s

(van Hasselt and Wiering, 2007)



Obr. 2.1: Cyklus učenia posilňovaním (Marsland, 2009)

Agent sa nachádza v stave s_t, kde vykoná akciu a_t, od prostredia dostane odmenu r_t a následne sa presunie do stavu s_{t+1}. Cieľom agenta je maximalizovať získanú odmenu tým, že si vytvorí stratégiu $\pi(s_t) = a_t$, ktorá mu hovorí, akú akciu a_t má vykonať v stave s_t.

2.2 Očakávaná odmena

Očakávaná odmena R_t je formálny zápis celkovej odmeny agenta, ktorú sa snaží maximalizovať. Podľa tejto predpovede sa agent rozhoduje, ako ďalej konať.

Nech $r_{t+1}, r_{t+2}, r_{t+3}, ...$ je sekvencia odmien, ktorú agent dostane po časovom úseku t, T je konečný časový úsek. Potom očakávanú odmenu možno vyjadriť ako:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T$$

(Sutton and Barto, 1998)

Nastanú však aj prípady, kedy neexistuje konečný stav a $T = \infty$. Predchádzajúcu rovnicu upravíme pridaním parametra $0 \leq \gamma \leq 1$ na:

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

(Marsland, 2009)

Konštanta γ (discount factor) určuje hodnotu budúcich odmien. Cena odmeny, ktorú agent dostane k časových úsekov v budúcnosti je γ^{k-1}-krát menšia ako by bola, ak by ju dostal okamžite. Ak $\gamma = 0$, agent sa sústredí len na maximalizovanie odmien, ktoré dostane hneď po vykonaní akcie. Čím viac sa γ blíži k 1, tým viac agent prihliada na budúce odmeny.

(Sutton and Barto, 1998)

2.3 Algoritmy učenia posilňovaním

V tejto časti si ukážeme niektoré najpoužívanejšie algoritmy učenia posilňovaním založené na metóde „temporal difference“ (TD).

TD-učenie kombinuje v sebe myšlienky Monte Carlo a dynamického programovania. Monte Carlo metódy, ako aj TD metódy sa vedia učiť priamo zo skúseností bez modelu prostredia. Ako dynamické programovanie, TD metódy upravujú svoje odhady založené z časti na iných naučených odhadoch, bez čakania na konečný výsledok. (Sutton and Barto, 1998)

2.3.1 Q-učenie (Q-learning)

Tento algoritmus sa zakladá na ohodnocovaní vykonaných akcií v danom stave. Toto ohodnotenie je reprezentované Q-funkciou. Výberom akcií s najvyšším ohodnotením Q-funkcie dostaneme optimálnu stratégiu.

Nech $Q(s_t, a_t)$ označuje ohodnotenie vykonanej akcie a_t v stave s_t, r je odmena, ktorú agent dostane za vykonanie akcie a_t, s_{t+1} je stav, do ktorého sa agent dostane po vykonaní akcie a_t a a_{t+1} je ďalšia vybraná akcia v stave s_{t+1}. Konštanta α určuje ako veľmi vplýva nová informácia na hodnoty Q-funkcie. Q-funkcia sa upravuje nasledovne:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

(Russell and Norvig, 2003)

2.3.2 SARSA (State-Action-Reward-State-Action)

Tento algoritmus funguje podobne ako Q-učenie, rozdiel je len v spôsobe úpravy Q-funkcie. Na jej úpravu sa nepoužíva najlepšia akcia ako v prípade Q-učenia, ale ďalšia vybraná. Pravidlo pre úpravu Q-funkcie môžeme zapísať ako:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

(Sutton and Barto, 1998)

2.3.3 Model aktér-kritik

Model aktér-kritik oddeľuje štruktúru pre výber akcie (aktér) a funkciu, ktorá ohodnocuje stavy (kritik). Počas behu algoritmu sa obe štruktúry učia a navzájom spolupracujú.

Po vykonanej akcii sa kritik pozrie na stav, do ktorého sa agent dostal a zhodnotí, či je lepší alebo horší ako predpokladal. Toto ohodnotenie sa dá vyjadriť ako TD-chyba v tvare:

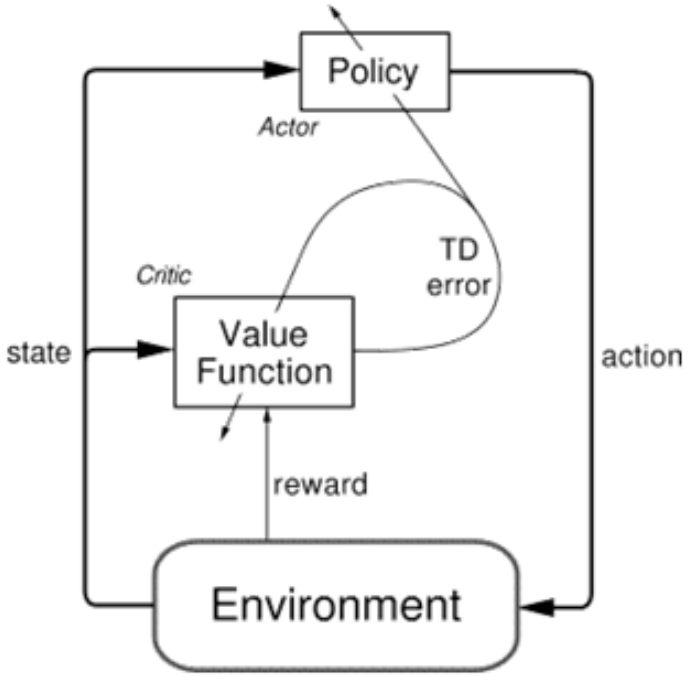
$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

kde V je ohodnotenie kritika pre daný stav a r_{t+1} je odmena, ktorú dostal agent od prostredia. Pomocou tejto chyby sa dá ohodnotiť akcia a_t vykonaná v stave s_t. Ak $\delta_t > 0$, značí to, že by sa mala posilniť pravdepodobnosť výberu akcie a_t v budúcnosti. Naopak, ak je δ_t negatívne, tak by sa v budúcnosti mala táto akcia voliť menej. (Sutton and Barto, 1998)

Chyba δ_t sa následne použije na úpravu pravdepodobností výberu pre každú akciu a ohodnotenia stavu:

$$P_{t+1}(s_t, a_t) = P_t(s_t, a_t) + \alpha\delta_t \qquad \text{(van Hasselt, 2012)}$$

$$V_{t+1}(s_t) = V_t(s_t) + \alpha\delta_t \qquad \text{(van Hasselt and Wiering, 2009)}$$



Obr. 2.2: Architektúra modelu aktér-kritik (Sutton and Barto, 1998)

2.3.4 CACLA (Continuous Actor-Critic Learning Automaton)

Pri predchádzajúcich algoritmoch sa počítalo s konečnou množinou stavov a akcií. V problémoch reálneho sveta však nastanú aj prípady, kedy je potrebné pracovať so spojitým priestorom akcií a stavov. Algoritmus CACLA rozširuje model aktér-kritik a tento problém rieši.

Ako pri modeli aktér-kritik aj v algoritme CACLA kritik aproximuje funkciu, ktorá ohodnocuje stavy a aktér vyberá práve jednu akciu pre daný stav podľa stratégie, ktorú sleduje. Vybraná akcia sa ďalej musí explorovať (vykoná sa trochu zmenená akcia), pretože len takýmto prehľadávaním priestoru dokážeme nachádzať lepšie stratégie pri výbere akcií.

Na rozdiel od modelu aktér-kritik CACLA využíva len znamienko TD-chyby a nie jej veľkosť. Ak je chyba negatívna, tak sa stratégia neupraví, pretože nie je isté, že vzdialením od akcie, ktorá skončila negatívnou chybou, sa priblíži k akcii, ktorá dá pozitívnu chybu. (van Hasselt and Wiering, 2009). Ak je chyba pozitívna, tak explorovaná akcia bola lepšia ako pôvodná a preto budúci výber aktéra priblížime k explorovanej akcii. Tu vidíme, prečo je explorácia dôležitá - bez nej by aktérom vybraná akcia bola rovnaká ako vykonaná akcia a aktér by nemohol svoj výber k nej viac priblížiť (van Hasselt, 2012).

Algoritmus Cacla:

Inicializácia Ac_0 (aktér), V_0 (krtik), s_0
for $i \in \{0, 1, 2, ...\}$ **do**
  Výber akcie $a_t = Ac(s_t)$ a jej explorácia na a'_t, $a_t \neq a'_t$
  Vykonanie akcie a'_t, odmena r_{t+1} a presun do stavu s_{t+1}
  $\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$
  $V_{t+1}(s_t) = V_t(s_t) + \alpha\delta_t$
  if $\delta_t > 0$ **then**
    $Ac_{t+1}(s_t) = Ac_t(s_t) + \alpha(a'_t - a_t)$
  end if
  if s_{t+1} je konečný stav **then**
    Reinicializácia s_{t+1}
  end if
end for

2.4 Funkčné aproximátory

Pri diskrétnom stavovom priestore nám na reprezentáciu Q-funkcie stačí tabuľka, avšak pri spojitých priestoroch ju treba reprezentovať funkciou. To isté platí, aj keď máme príliš veľký diskrétny stavový priestor a stavy v ňom majú istú geometrickú štruktúru a tak ich môže byť výhodné pretransformovať do spojitého priestoru. Pri rozšírení algoritmov pre spojité stavy potrebujeme nahradiť tabuľkovú reprezentáciu funkčnými aproximátormi.

Ako funkčný aproximátor sa využíva napríklad neurónová sieť, ktorá rozhoduje, aká akcia sa má vykonať v nejakom stave. Postupným trénovaním siete sa budeme stále viac približovať k optimálnemu výsledku.

Kapitola 3

StarCraft: Brood War

StarCraft: Brood war je real-time strategická hra (RTS), ktorú vydal Blizzard Entertaiment v roku 1998. Ako pri všetkých RTS aj tu všetky akcie prebiehajú v reálnom čase a hráč sleduje hru z vtáčej perspektívy. Snaží sa získať suroviny, ktoré potom využíva na stavanie bojových jednotiek a budov. Každá má iné vlastnosti a schopnosti a je len na hráčovi, akú stratégiu zvolí. Cieľom hry je zničiť všetky budovy nepriateľa.

Hra je zasadená do sci-fi prostredia, v ktorom hráč môže ovládať jednu z troch rás:

- **Protoss** - mimozemská rasa s vyspelou technikou a silnými, odolnými, ale aj drahými jednotkami
- **Zerg** - hmyzu podobná rasa, ktorá sa sústredí na rýchle vyrábanie veľkého množstva lacných, ale slabých jednotiek
- **Terran** - ľudia, ktorí svojou úrovňou techniky a sily jednotiek stoja medzi dvomi predchádzajúcimi (menej vyspelá technika a slabšie jednotky ako Protoss, ale silnejšie a do počtu menšie ako Zerg)

	Terran	**Zerag**	**Protoss**
Sila jednotiek	Stredná	Slabá	Silná
Cena jednotiek	Stredná	Malá	Veľká
Rýchlosť výroby	Pomalá	Rýchla	Stredná

Tabuľka 3.1: Porovnanie rás

3.1 Priebeh hry

Každý hráč začína s hlavnou budovou a niekoľkými robotníkmi, v ktorej blízkosti má k dispozícii oba druhy surovín – minerály a plyn, ktorých množstvo je obmedzené. To hráča v priebehu hry núti hľadať nové zdroje a stavať okolo nich ďalšie základne, aby ich mohol efektívne a bezpečne ťažiť.

Obr. 3.1: Screenshot z hry Starcraft: BroodWar

Robotníci sú základnou jednotkou každej rasy. Ako jediná jednotka v hre vedia stavať budovy a ťažiť suroviny. Každý hráč môže mať len obmedzený počet jednotiek, po ktorom už nevie vyrábať ďalšie. Tento limit si vie zvýšiť postavením špeciálnych budov(jednotiek). Jednotky sa vyrábajú v špecializovaných budovách. Ďalšie budovy zase slúžia na zakúpenie vylepšení alebo na obranu. Každá jednotka alebo budova má na začiatku určitý počet životov a po ich stratení zomiera (vymaže sa z hry). Niektoré jednotky taktiež vedia využívať špeciálne schopnosti, ktoré však použijú len na príkaz hráča.

3.2 Mikromanažment a makromanažment

Hru môžeme rozdeliť na dve úrovne, ktoré sa sústreďujú na rôzne aspekty hry – mikromanažment (mikro) a makromanažment (makro).

Makro sa sústreďuje na celkovú ekonomiku, získavanie a využitie surovín. Hráč musí vedieť, kedy má stavať akú budovu alebo jednotku, objavovať mapu a získavať nové zdroje surovín. Dobré makro vedie k získaniu surovín a tým aj k veľkej armáde. Napríklad ak robotníci ťažia málo minerálov alebo plynu, tak nebude dosť surovín na výrobu bojových jednotiek.

Mikro je individuálne ovládanie jednotiek, tj. hráč vydáva rozkazy každej jednotke zvlášť. Týmto vie docieliť výhodnejšie pozície v bitke pre každú jednotku, než keby ovládal všetky naraz. Typický príklad mikra je ak hráč bojuje proti inej skupine jednotiek, ktorá nedokáže strieľať, kým jeho jednotky áno. Hráč začne ustupovať len s jednotkou, ktorá je momentálne napadnutá a tým docieli, že súperove jednotky bežia za ňou, kým jeho zvyšné jednotky pokračujú v paľbe a sú mimo nebezpečenstva. Týmto si zabezpečí oveľa menšie straty, než keby nechal svoje jednotky len stáť a strieľať.

Hráč by nemal zanedbávať ani jednu úroveň, pretože to môže znížiť jeho šancu na výhru. Napríklad, ak sa sústredíme len na makro a vyprodukujeme veľa silných jednotiek a tie len pošleme na nepriateľa, nemáme zaručené, že tým získame nad ním výhodu. Nepriateľ síce nemá takú silnú armádu ako my, ale stará sa o svoje jednotky ako bolo popísané, čím dokáže vyvážiť svoju nevýhodu menšej armády a spôsobí nám značné straty. A naopak, ak zanedbáme makro, tak máme len pár slabších jednotiek, ktoré aj keď dobre premiestňujeme po bojisku, padnú, lebo nepriateľ má lepšie makro a tým aj väčšiu armádu a môže si dovoliť stratiť viac jednotiek.

Efektívne prepojenie medzi mikrom a makrom je veľmi ťažké, pretože to vyžaduje od hráča vynikajúcu znalosť hry, dobré reflexy, rýchle rozhodovanie a schopnosť multitaskingu. Hráč musí mať dobrý prehľad o rozohratej hre a zvládnuť vykonávanie potrebných akcií za veľmi krátky čas. Začiatočníci majú zvyčajne APM pod 50 (počet akcií za minútu), naopak profesionálni hráči dosahujú APM 300, čo sa mierne zvýši počas intenzívnejších bojov. V tomto majú umelé inteligencie v porovnaní s človekom značnú výhodu, keďže vedia vykonávať akcie takmer okamžite.

3.3 Prieskum mapy

Mapa v StarCrafte je definovaná ako štvorcová mriežka, kde šírka × výška mapy je vyjadrená v štvorcoch veľkosti 32 × 32 pixelov, taktiež nazývaných „build tiles“ (dlaždice na stavanie). Avšak rozlíšenie prechodných oblastí je v štvorcoch veľkosti 8 × 8 pixelov nazývaných „walk tiles“ (dlaždice na chodenie). Typický rozmer mapy je od 64 × 64 do 256 × 256 build tiles. (Ontañón et al., 2013)

Na začiatku hry hráč vidí len časť mapy, na ktorej sa nachádzajú jeho jednotky. Časti mapy, ktoré už hráč preskúmal a v ich okolí sa nenachádza žiadna jeho jednotka, sa zahalia do šedej hmly (fog of war), ktorá ukazuje ako naposledy oblasť vyzerala, ale nezachytí zmeny, ktoré nastanú neskôr. Zvyšné nepreskúmané oblasti sú čierne.

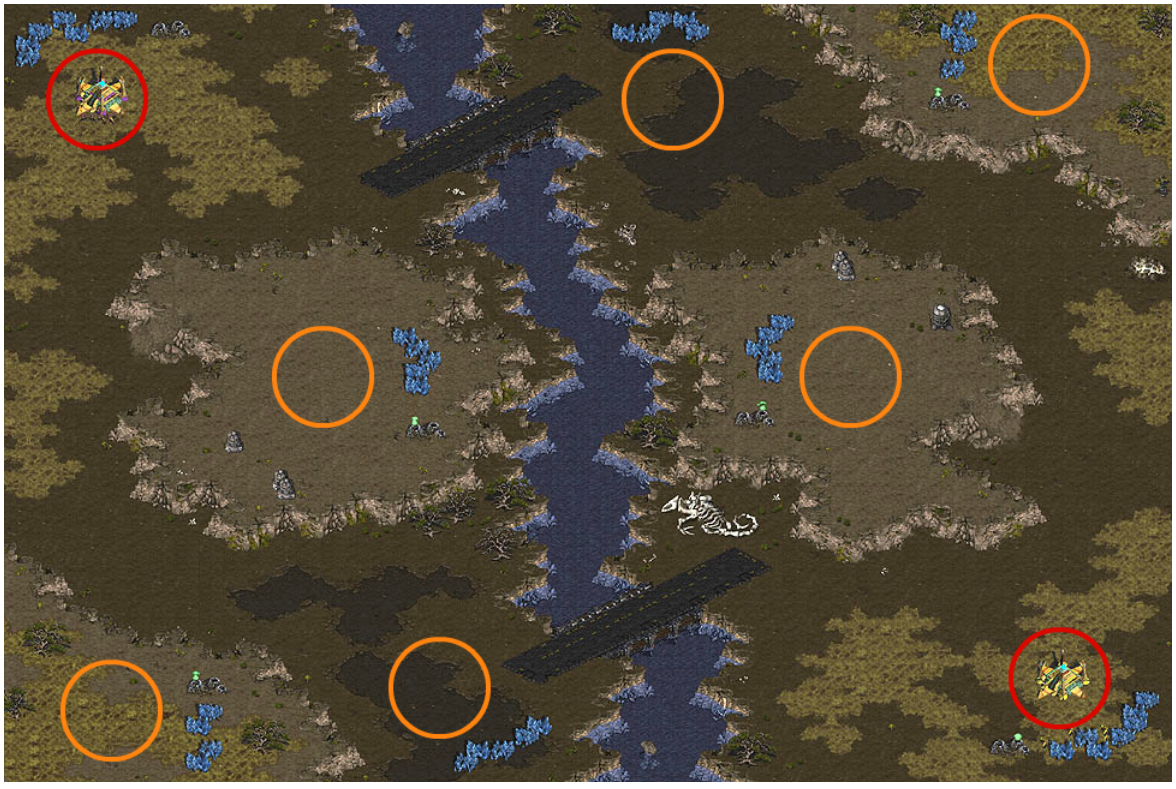
Obr. 3.2: Fog of war (šedá hmla), viditeľné (okolie robotníka) a nepreskúmané (čierne) časti mapy

Prieskum mapy je veľmi dôležitá časť hry. Aby sme si mohli vytvoriť efektívnu výhernú stratégiu, musíme vedieť o našom nepriateľovi čo najviac informácií. Nestačí však len poznať silné a slabé stránky každej rasy, ale je potrebné aj zistiť, čo nepriateľ plánuje. Ako prvé musíme poznať polohu nepriateľskej základne. Čím skôr ju nájdeme, tým rýchlejšie vieme reagovať na prípadné hrozby, keďže existuje veľa známych stratégií, ktoré spoliehajú na moment prekvapenia. Včasným zistením plánov nepriateľa (aké jednotky vyrába, pohyb armády...) získame čas na prípravu obrany alebo prípadného protiútoku. Taktiež objavením jeho nových základní a ich následným napadnutím dokážeme oslabiť jeho ekonomiku.

Prieskum mapy môžeme rozdeliť na dve časti:

1. výber miesta, ktoré sa má preskúmať
2. bezpečná cesta na dané miesto

Každá mapa v StarCrafte obsahuje miesta („start locations“), na ktorých môže mať hráč na začiatku hry danú základňu (hlavná budova a robotníci). Ak je hráčov menej ako týchto miest, tak sa pre každého náhodne vyberie jedno z nich. Práve preskúmaním týchto miest vieme efektívne nájsť nepriateľa. Ďalšími zaujímavými miestami pre prieskum, sú miesta vedľa zdrojov surovín, pri ktorých sa zvyknú stavať ďalšie základne („base locations“). Nestačí sledovať len priestor okolo hlavnej základne nepriateľa, pretože si môže stavať základňu aj na druhej strane mapy a prekvapiť nás. Implementácia učenia tejto časti je nad rámec našej práce, takže pre naše potreby bude výber miest napevno daný.



Obr. 3.3: Mapa pre dvoch hráčov (červená = start locations, oranžová = base locations)

V druhej časti robotník dostane dané miesto a jeho úlohou je dostať sa naň, čo najbezpečnejšie, tj. s čo najmenším poškodením, v ideálnom prípade so žiadnym. Ekvivalentom tejto úlohy je aj bezpečné stavanie kanónov – robotník dostane miesto, kde má postaviť kanón a potom sa tam snaží prísť najbezpečnejšou cestou. Práve touto časťou sa zaoberá naša práca.

Kapitola 4

Implementácia

4.1 Prostredie

Na implementáciu agenta využívame open-source C++ framework BWAPI (Brood War Application Programing Interface), ktorý je určený na vytváranie agentov do real-time strategickej hry StarCraft: Brood War. Poskytuje rozhranie, pomocou ktorého je možné ovládať hru presne tak, ako by to vedel hráč. BWAPI nepodvádza – neposkytuje informácie, ktoré by nemal mať ľudský hráč k dispozícii (poskytuje ich dokonca menej). Kód sa skompiluje do DLL (Dynamic-link library) a tá je pomocou aplikácie Chaoslauncher vložená do StarCraftu.

Každá jednotka a budova v hre má priradené svoje ID a rôzne metódy a vlastnosti, ktoré môžeme použiť. Vieme tiež pristupovať k statickým informáciám o hre (veľkosť mapy, počet framov, hráčov...) alebo reagovať na rôzne udalosti (zmena framu, zničenie jednotky, koniec hry...). Pomocou BWAPI sa dá taktiež vykresľovať do hry jednoduchá grafika (body, čiary, text...), ktorá je však viditeľná len na počítači, kde beží daný kód.

4.2 Agent

Náš agent ovláda robotníka rasy Protoss nazývaného Probe. Úlohou tejto jednotky je ťažiť suroviny a stavať budovy. Ako všetky jednotky Protossov má aj robotník energetický štít, ktorý sa po poškodení časom obnovuje (životy si obnoviť nevie). Dokáže aj útočiť, ale jeho sila je veľmi malá. Agent dostane miesto na mape a jeho úlohou je bezpečne s týmto robotníkom prísť na dané miesto.

4.3 Akcia

Akcia je reprezentovaná vektorom, ktorý určuje smer, ktorým sa má robotník pohnúť. Podľa polohy robotníka a danej akcie vypočíta miesto na mape (vždy konštantne ďaleko od robotníka), na ktoré sa robotník pošle. Agent volí novú akciu každých 6 framov.

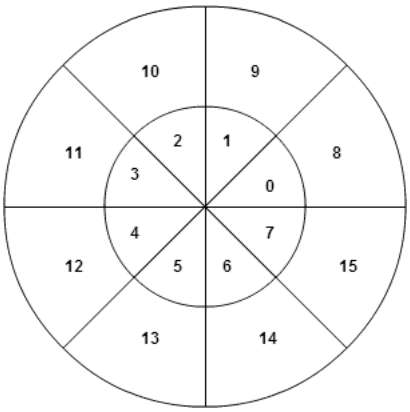
4.4 Stav

Stav agenta pozostáva z nasledujúcich častí:

- vektor smerujúci na začiatok najkratšej cesty z pozície robotníka k danému miestu vyrátanej pomocou algoritmu A*
- veľkosť najkratšej cesty v pixeloch
- zostávajúce životy a štít robotníka
- „threat mapa“ okolia robotníka

Informácie o najkratšej ceste sú potrebné, aby sa robotník naučil prísť po nej na dané miesto na mape. Vektor ho navádza, ktorým smerom sa tam najrýchlejšie dostane a veľkosť cesty mu hovorí, či sa už blíži k cieľu. Podľa zostávajúcich životov a štítu agent vie, ako veľmi si môže dovoliť riskovať. Napríklad, ak má plný štít, tak môže zvoliť rýchlejšiu, ale nebezpečnejšiu cestu, pri ktorej dostane pár zásahov, lebo štít sa mu obnovuje. Toto by si nemohol dovoliť, keď mu zostáva málo životov a nemá žiadny štít. Threat mapa hovorí agentovi, ktoré miesto v okolí je najbezpečnejšie a naopak, kde mu hrozí najväčšie nebezpečenstvo.

Threat mapa je implementovaná ako vektor, kde každý jeho prvok označuje určitú časť okolia robotníka (Obr. 4.1). Ak sa v okolí robotníka nenachádza žiadny nepriateľ, vektor obsahuje len nulové hodnoty. Pre každú nepriateľskú jednotku v dohľade robotníka sa jej útočná sila pripočíta k prvku vektora prislúchajúcemu miestu, ktoré daná jednotka ohrozuje (kam môže zaútočiť).



Obr. 4.1: Threat mapa - každá časť vektora udáva nebezpečenstvo v danej oblasti v okolí robotníka

Všetky časti stavu je potrebné upraviť na malé hodnoty, aby sme zabránili prepáleniu siete. To znamená, že ak by na vstup prišli príliš veľké hodnoty, tak by vždy po aplikovaní sigmoidy na skrytej vrstve vznikali čísla veľmi blízke jednej a sieť by potom nevedela rozlišovať medzi rôznymi stavmi. Veľkosť vektora k najkratšej časti zmenšíme na 1 a ostatné hodnoty upravíme nasledovne:

$$dist = \frac{1}{1 + (dist)0.001},\ hp = \frac{hp}{maxHp},\ shield = \frac{shield}{maxShield},\ map[i] = \frac{1}{map[i]0.01},$$

kde $dist$ je dĺžka cesty, hp a $maxHp$ sú momentálne a počiatočné životy robotníka, $shield$ a $maxShield$ je momentálny a počiatočný štít robotníka a $map[i]$ je i-ty prvok v threat mape.

4.5 Odmena

Veľkosť odmeny, ktorú agent dostane ovplyvňujú 2 časti - dĺžky zostávajúcej cesty a zranenia, ktoré robotník obdržal. Prvú časť odmeny vypočítame ako:

$$(1 - 2\frac{dist}{maxDist})^2 sgn(1 - 2\frac{dist}{maxDist}),$$

kde $maxDist$ je najväčšia možná dĺžka cesty. Táto časť zabezpečí, že agent vie, či ho jeho akcia priblížila k cieľu. O veľkosti nebezpečenstva akcie mu hovorí druhá časť odmeny, ktorej model hľadáme pri trénovaní agenta v ďalšej kapitole.

Obe časti by mali byť vyvážené natoľko, aby sa agent naučil optimálnu cestu. Mohlo by sa stať, že prvá časť ďaleko preváži druhú, čím sa agent naučí len bežať po najkratšej ceste a nedbá na nebezpečenstvo. Naopak by sa mohlo stať, že agent len uteká pred nebezpečenstvom a nevie nájsť cestu k miestu, do ktorého sa má dostať.

4.6 Neurónové siete

Aktér aj kritik boli implementovaní ako neurónové siete s jednou skrytou vrstvou. Ako akvivačné funkcie boli zvolené sigmoida na skrytej vrstve a lineárna funkcia na výstupe.

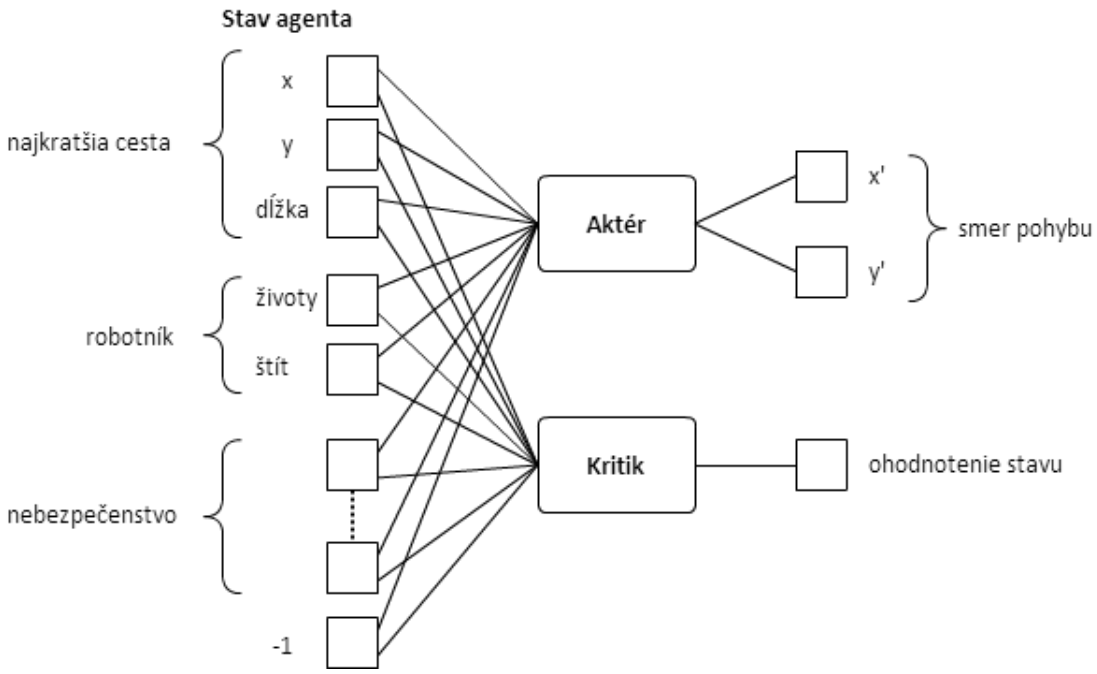
	Počet neurónov	α
Aktér	5	0.09
Kritik	7	0.009

Tabuľka 4.1: Nastavenie neurónových sietí

Neurónová sieť bola implementovaná ako samostatná trieda, ktorá si pamätá obe matice svojich váh. Pri inicializácii ich hodnoty načíta zo súboru a ak súbor nenájde, tak matice vyplní malými náhodnými hodnotami. Sieť sa učí algoritmom spätného šírenia chyby.

4.7 CACLA

Algoritmus CACLA bol tiež implementovaný ako samostatná trieda. Na začiatku aktér vyberie novú akciu – vektor, ktorý sa následne normalizuje a exploruje (otočenie o náhodný uhol). Vypočíta sa chyba pre aktéra a akcia sa vykoná. Po 6 framoch kritik ohodnotí danú akciu a aktér aj kritik sa učia. Po každej iterácii sa taktiež zmenšuje explorácia.



Obr. 4.2: Model aktéra a kritika

Kapitola 5

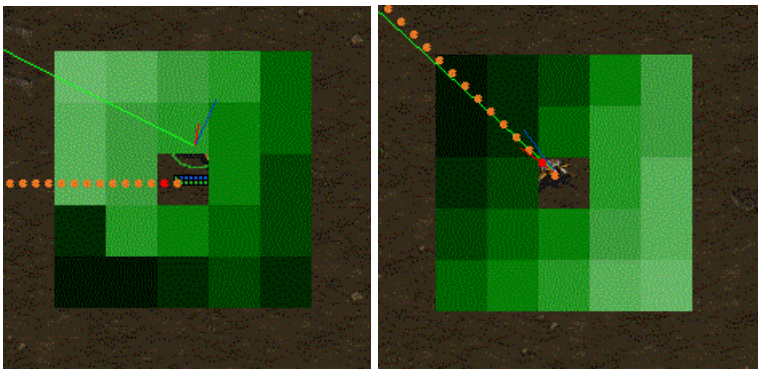
Trénovanie agenta

Trénovanie agenta sme rozdelili na dve časti - učenie najkratšej cesty a vyhýbanie sa nebezpečenstvu.

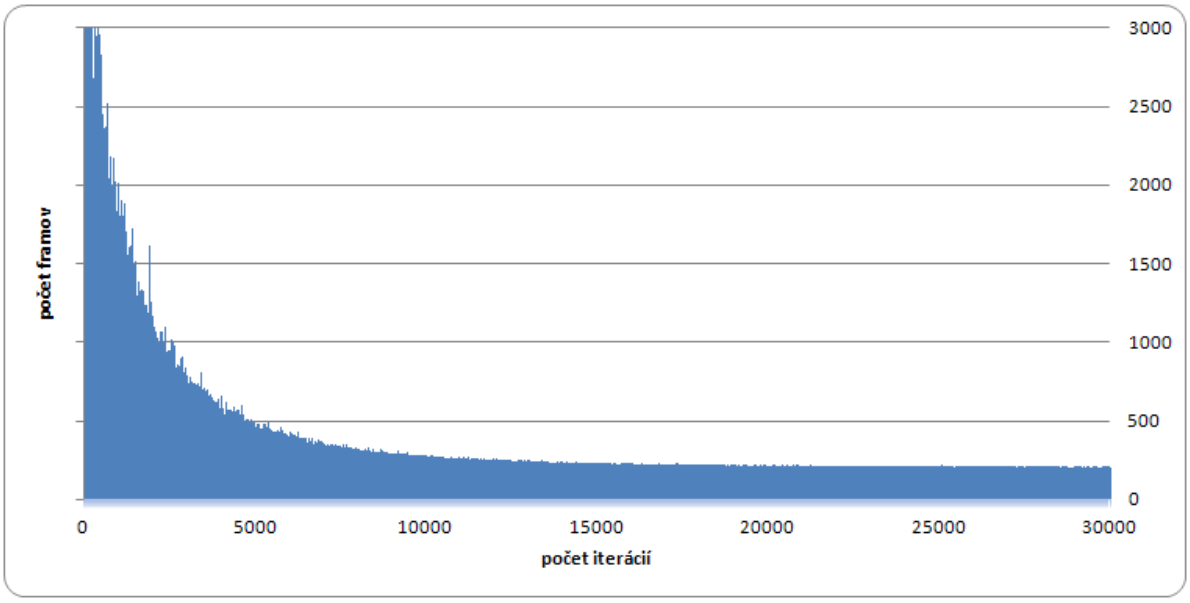
5.1 Prvá časť: Najkratšia cesta

V tejto časti sme agentovi vygenerovali miesto na mape a učili sme ho naň prísť. Hra sa reštartovala vždy, keď agent prišiel na okraj mapy. Agentovi sme v každom teste generovali body v rovnakej vzdialenosti a porovnávali za aký počet framov sa k nim dostal. Testované vzdialenosti boli 500 a 1000 pixelov a γ bola nastavená na 0.01. Priebeh trénovania agenta pri oboch vzdialenostiach je zobrazený v grafoch (Obr. 5.2, Obr. 5.3).

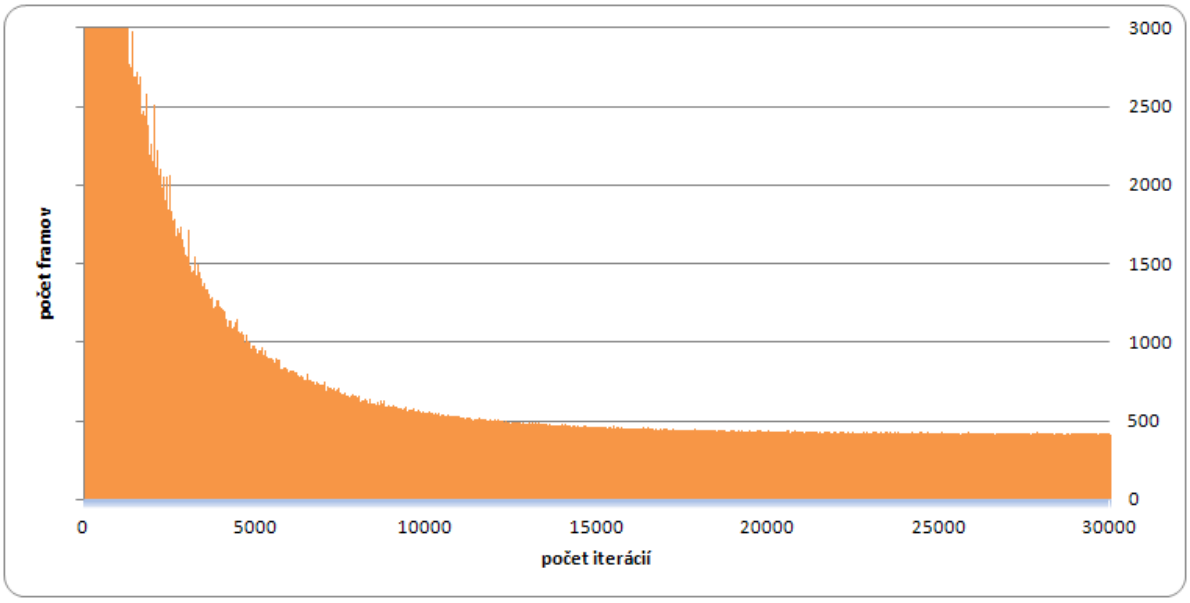
Prvých 30 kôl sme porovnali odmenu, ktorú by agent dostal, ak by vykonal explorovanú akciu s odmenou za akciu, ktorú vybral bez explorácie. Ak bola neexplorovaná akcia lepšia, tak sa agent učil približovať sa k vektoru zo stavu ako pri učení s učiteľom.



Obr. 5.1: Ohodnotenie kritika pre okolité stavy na začiatku (vľavo) a na konci učenia (vpravo). Tmavé štvorce predstavujú stavy s najvyššou predikovanou odmenou a najkratšia cesta je znázornená postupnosťou bodiek.



Obr. 5.2: Porovnanie pri vzdialenosti 500 pixelov



Obr. 5.3: Porovnanie pri vzdialenosti 1000 pixelov

5.2 Druhá časť: Nebezpečenstvo

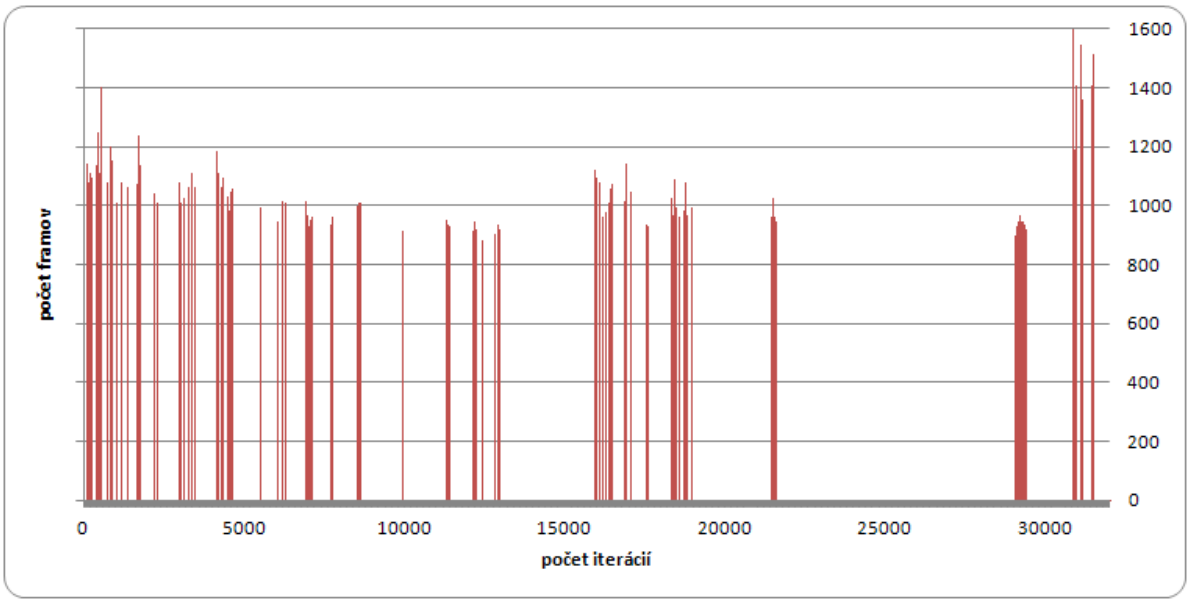
Požiadavkou pre túto časť bolo zvládnutie predošlej časti (agent sa naučil prísť na dané miesto po najkratšej ceste). Cieľom druhej časti trénovania bolo nájsť model odmien, pomocou ktorých sa agent popritom naučí vyhýbať sa nebezpečenstvu.

Na začiatku testu bol agent naučený ísť po najkratšej ceste k cielu a do nej sme mu postavili nepriateľské jednotky. Ako prvú prekážku sme zvolili Protosskú budovu ”Photon Cannon”, ktorá strieľa po nepriateľoch v jej blízkosti. Druhá prekážka bola jednotka rasy Zerg ”Hydralisk”, ktorá útočí na diaľku. Testovali sme rôzne modely odmien - aký veľký trest musel agent dostať, aby sa naučil obísť hrozbu, ale popritom dokázal prísť do ciela. Model pozostával z trestov za poškodenie štítu, stratu životov a zničenie robotníka (Tabuľka 5.1). HP označuje pomer počtu životov, ktoré robotník stratil od posledného stavu k hodnote, ktorú mal na začiatku hry (SH analogicky pre štít). Všetky tresty sa odpočítajú od odmeny, ktorú by agent dostal v prvej časti.

Model	Strata životov	Poškodenie štítu	Smrť
1	$0.001 \cdot HP$	$0.001 \cdot SH$	1
2	$0.05 \cdot HP$	$0.005 \cdot SH$	$0.05 \cdot HP + 0.005 \cdot SH$
3	$0.01 \cdot HP$	$0.001 \cdot SH$	$0.01 \cdot HP + 0.001 \cdot SH$

Tabuľka 5.1: Testované tresty agenta

Žiadny z testovaných modelov nepriniesol výrazný úspech. Pri modeli č.3 agent dokázal počas učenia niekoľko kôl po sebe prísť do cieľa, ale následne to už veľký počet kôl nevedel. Toto sa párkrát zopakovalo. Pri modeli č.1 sa agent naučil ísť čo najďalej od cieľa, až vyšiel z mapy.



Obr. 5.4: Trénovanie agenta s modelom odmien č.3 (len iterácie, kedy agent prišiel do cieľa)

Kapitola 6

Diskusia

V prvej časti trénovania agenta sme ukázali, že bol schopný naučiť sa ísť po najkratšej ceste bez ohľadu na vzdialenosť trénovacích bodov. Výsledky trénovania (Obr. 5.2 a 5.3) ukázali výrazné zlepšenie času agenta po takmer 10 000 iteráciách, čím sme ukázali, že pridaním učenia s učiteľom prvých 30 kôl sme nevyriešili celkový problém. Toto vylepšenie slúžilo len na zrýchlenie učenia. Na začiatku je kritik nenatrénovaný, čo vedie k tomu, že aktér sa učí na náhodný smer a potom dlho trvá, kým sa preučí podľa už natrénovaného kritika. Učenie s učiteľom zabezpečí, že aktér má na začiatku nejakú predstavu, kam by mal ísť a neskôr sa doučí na správny smer podľa kritika.

V druhej časti sme sa snažili nájsť vhodný model odmien pre agenta, aby sa naučil vyhýbať sa po ceste prípadným hrozbám. Ako prvú hrozbu sme si vybrali obrannú vežu, pretože sa nepohybuje a jej sila nie je natoľko veľká, aby zabila robotníka jednou ranou, čiže dostane možnosť od nej ujsť. Druhá hrozba bola jednotka rasy Zerg Hydralisk, pretože ako v prvom prípade, jej sila bola relatívne malá a navyše jej dostrel bol o dosť menší, ako dovidel robotník, takže ju zaregistroval skôr ako po ňom vystrelila. Skúšali sme do trénovania zaradiť aj jednotky útočiace na blízko, avšak rozhodli sme sa ich nezaradiť, pretože sa stávalo, že prišli k robotníkovi a nezaútočili a robotník odišiel, čím stratil spätnú väzbu v podobe trestu a nenaučil sa, že sa nachádzal blízko nebezpečenstva. Neúspech druhej časti trénovania bol pravdepodobne zpôsobený nevyváženosťou odmeny za blíženie sa k cielu a trestom za poškodenie robotníka. Taktiež mohol byť náš odhad o počte iterácií potrebných na učenie agenta nedostačujúci.

Záver

V práci sme sa venovali učeniu agenta pre prieskum v real-time strategickej hre Starcraft: Brood War pomocou učenia posilňovaním. Úlohou agenta bolo doviesť robotníka, ktorého ovládal, na dané miesto čo najrýchlejšie bez toho, aby sa počas cesty dostal do veľkého nebezpečenstva.

V prvej kapitole sme definovali rôzne druhy neurónových sietí, popísali teóriu učenia posilňovaním a niektoré jeho známe algoritmy. Pri implementáci agenta sme použili algoritmus CACLA (Continuous Actor-Critic Learning Automaton) a viacvrstvové dopredné neurónové siete.

Ďalšia kapitola sa venovala popisu hry StarCraft: Brood War. V skratke sme ukázali základný princíp hry a potom sme sa venovali opisu mapy a jej prieskumu. Nášho agenta sme implementovali pomocou C++ frameworku BWAPI, ktorý umožňuje vytvoriť a vložit vlastnú umelú inteligenciu do hry.

Návrh agenta spočíval v definovaní jeho stavov, akcií a odmien, ktoré počas učenia dostáva. Stav sa skladal zo smeru a dĺžky najkratšej cesty k danému bodu (vypočítanej pomocou algoritmu A*), počtu zostávajúcich životov a štítu a nebezpečenstva v okolí v podobe threat mapy. Ako akciu sme si zvolili vektor, ktorý určuje, kam sa má robotník pohnúť. Za vykonanú akciu agent dostal odmenu v závislosti od dĺžky zostávajúcej cesty a od počtu životov a štítu, ktoré stratil. Taktiež sme museli riešiť aj správne zvolenie parametrov pre neurónové siete a CACLU ako počet neurónov na skrytej vrstve a konštanty, ktoré ovplyvňovali rýchlosť učenia sietí a relevantnosť odmien v budúcnosti.

Trénovanie agenta sme rozdelili do dvoch častí. V prvej časti sme agenta učili prísť na dané miesto na mape po najkratšej ceste. Generovali sme mu náhodné body na mape, ktoré však boli od neho rovnako vzdialené a pozorovali sme priebeh učenia. Agent sa dokázal naučiť ísť po najkratšej ceste bez ohľadu na vzdialenosť generovaných bodov. V druhej časti sme do cesty už naučeného agenta z prvej časti trénovania postavili nepriateľské jednotky a agent sa učil vyhnúť sa im a zároveň sa čo najrýchlejšie dostať na dané miesto. Táto časť však nepriniesla výrazný úspech.

V budúcnosti by sme chceli vylepšiť učenie časti týkajúcej sa vyhýbaniu nebezpečenstva a doplniť implementáciu o rôzne špeciálne prípady hrozieb, ktoré priamo nesúvisia s priamym útokom na robotníka, ale so špeciálnymi schopnosťami niektorých jednotiek v hre ako napríklad ovládnutie jednotky nepriateľom alebo privolanie búrky. Práca by sa tiež dala rozšíriť aj učením agenta vyberať miesta na mape, kam má posielať robotníka na prieskum.

Dodatok A

CD médium

Práca obsahuje CD so zdrojovým kódom implementovaného agenta.

Literatúra

[Engelbrecht, 2007] Engelbrecht, A. P. (2007). *Computational Inteligence: An Introduction*, volume 2. John Wiley and Sons.

[Kvasnička et al., 1997] Kvasnička, V., Ľubica Beňušková, Pospíchal, J., Farkaš, I., Tiňo, P., and Kráľ, A. (1997). *Úvod do teórie neurónových sietí*. Iris.

[Marsland, 2009] Marsland, S. (2009). *Machine Learning: An Algorithmic Perspective*. CRC Press.

[Ontañón et al., 2013] Ontañón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., and Preuss, M. (2013). A survey of real-time strategy game ai research and competition in starcraft. In *IEEE Transactions on Computational Intelligence and AI in games*.

[Russell and Norvig, 2003] Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*, volume 2. Prentice Hall.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT press.

[van Hasselt, 2012] van Hasselt, H. (2012). Reinforcement learning in continuous state and action spaces. In *Reinforcement Learning: State-of-the-Art*, pages 207–251. Springer Berlin Heidelberg.

[van Hasselt and Wiering, 2007] van Hasselt, H. and Wiering, M. A. (2007). Reinforcement learning in continuous action spaces. In *IEEE International Symposium on proceeding of: Approximate Dynamic Programming and Reinforcement Learning*.

[van Hasselt and Wiering, 2009] van Hasselt, H. and Wiering, M. A. (2009). Using continuous action spaces to solve discrete problems. In *International Joint Conference on Neural Networks*.