UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

AGENT PRE PRIESKUM V REAL-TIME STRATEGICKÝCH HRÁCH

DIPLOMOVÁ PRÁCA

2016

Bc. Lucia Piváčková

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

AGENT PRE PRIESKUM V REAL-TIME STRATEGICKÝCH HRÁCH

DIPLOMOVÁ PRÁCA

Študijný program: Informatika
Študijný odbor: 2508 Informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Vedúci práce: prof. Ing. Igor Farkaš, Dr.
Konzultant: RNDr. Viliam Dillinger

2016

Bc. Lucia Piváčková

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Lucia Piváčková
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Agent pre prieskum v real-time strategických hrách
Scouting agent for real-time strategy games

Cieľ:
1. Naštudujte metódy navigácie agenta v prostredí s prekážkami a prístupy v učení posilňovaním.
2. Navrhnite a implementujte model agenta pre prieskum v real-time stratégiách.
3. Navrhnite a vykonajte experimenty, pomocou ktorých Vášho agenta otestujete.

Vedúci: prof. Ing. Igor Farkaš, Dr.
Konzultant: Mgr. Viliam Dillinger
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 10.12.2014

Dátum schválenia: 16.12.2014 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestné vyhlásenie

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

V Bratislave .....

Poďakovanie

Ďakujem svojmu vedúcemu práce prof. Ing. Igorovi Farkašovi a svojmu konzultantovi RNDr. Viliamovi Dillingerovi za rady, nápady a odbornú pomoc a rodine a priateľom za podporu.

Abstrakt

Cieľom práce bolo navrhnúť, implementovať a otestovať model agenta pre prieskum v real-time stategickej hre StarCraft: Brood War. Úlohou agenta bolo, čo najrýchlejšie a s najmenším poškodením prísť s jednotkou na dané miesto. Navrhli sme vytvorenie grafu, reprezentujúceho hernú mapu, kde cena hrany pozostávala z euklidovskej vzdialenosti medzi políčkami a nebezpečenstvom, ktoré hrozí agentovej jednotke od všetkých nepriateľov v okolí. V práci sme predstavili tri rôzne modely výpočtu tohto nebezpečenstva. Natrénovaných agentov sme testovali v dvoch experimentoch. V prvom experimente sme pozorovali správanie každého modelu na navrhnutej mape. Pri druhom experimente sme pre každý model porovnávali priemerný čas, potrebný na príchod do cieľového miesta a priemerné zranenie, ktoré agentova jednotka popritom dostala. V testoch dopadol najlepšie nami navrhnutý model, ktorý predikoval nebezpečenstvo v budúcnosti.

Kľúčové slová: *učenie posilňovaním, StarCraft, prieskum, mikromanažment jednotiek*

Abstract

The goal of this master thesis was to design, implement and test an agent for scouting in the real-time strategy game StarCraft: Brood War. Agent was tasked to navigate his unit to given place on the map on the shortest path with the least damage recieved. We proposed reprezentation of map as a graph where edge cost consists of euclidian distance and imminent danger from nearby enemy units. We presented three different models of danger in this paper. Trained agents were then tested in two experiments. In the first experiment we observed the behaviour of each model. In the second experiment we compared the amount of damage recieved by the agent's unit and how long it took to successfully arrive to its final destination. The model we proposed which could predict danger in future was overall the best of all tested models.

Keywords: *reinforcement learning, StarCraft, scouting, unit micromanagement*

Obsah

Úvod

Cieľom našej práce bolo vytvoriť agenta pre prieskum v real-time strategických hrách a otestovať jeho úspešnosť vo vybraných experimentoch. Vo všeobecnosti sa prieskum chápe ako objavovanie nepreskúmaných miest na mape, hľadanie nových zdrojov surovín alebo aj nájdenie a pozorovanie nepriateľa. V našej práci pod prieskumom myslíme výber miesta na mape a bezpečnú a rýchlu navigáciu k nemu. Práca sa zameriava hlavne na ovládanie jednotky pri prieskume. To znamená, že agentova jednotka dostane miesto na mape, na ktoré by mala ísť čo najkratšou cestou a popritom by sa mala vedieť vyhýbať nepriateľským jednotkám, ktoré ju môžu zničiť.

Túto úlohu sme riešili aj v predchádzajúcej práci, kde sme agenta učili robiť prieskum pomocou algoritmu CACLA (Piváčková, 2014). Agent sa naučil prísť na dané miesto bez ohľadu na vzdialenosť daného miesta, avšak nedokázal úspešne sa naučiť vyhýbať sa nepriateľom.

Prvá kapitola našej práce sa venuje rôznym metódam navigácie agenta v prostredí s prekážkami. Druhá kapitola pojednáva o učení posilňovaním, kde sme sa venovali hlavne TD-učeniu. Nasleduje kapitola, kde popisujeme dopredné neurónové siete s radiálnou bázickou funkciu na skrytej vrstve. Ďalšia kapitola približuje čitateľovi hlavné princípy real-time strategických hier a hru StarCraft: Brood War, v ktorej sme nášho agenta implementovali. Venuje sa taktiež opisu hernej mapy a jej prieskumu. V nasledujúcich dvoch kapitolách je popísaný navrhovaný model agenta a jeho implementácia. V poslednej kapitole ukážeme, ako sa agent trénoval a ako sa testovala jeho úspešnosť. Dosiahnuté výsledky sme zhodnotili v diskusii.

Kapitola 1

Metódy navigácie agenta

Problém navigácie agenta v prostredí s prekážkami je veľmi často riešený problém, hlavne v robotike. V tejto kapitole si popíšeme niektoré metódy navigácie agenta a spôsoby modelovania nebezpečných častí priestoru.

1.1 Algoritmus A*

Algoritmus A* je rozšírením Dijkstrovho algoritmu na hľadanie optimálnej cesty v grafe s kladne ohodnotenými hranami použitím heuristiky. Algoritmus hľadá cestu z počiatočného vrchola do cieľového postupným prehľadávaním ostatných vrcholov. Počas behu si algoritmus udržiava množinu už navštívených vrcholov a množinu vrcholov, ktoré treba navštíviť. Pre každý vrchol n si taktiež pamätá optimálnu dĺžku cesty $d[n]$ z počiatočného vrchola a predpokladanú vzdialenosť do cieľového vrchola, ktorú určuje ohodnocovacia funkcia $f(n)$. Na začiatku algoritmu má počiatočný vrchol $d[start] = 0$ a ostatné $d[n] = \infty$. Algoritmus vždy vyberá ďalší vrchol z množiny ešte nenavštívených vrcholov s najmenšou hodnotou funkcie $f(n)$. Algoritmus skončí vtedy, keď navštívi cieľový vrchol alebo už navštívil všetky dostupné vrcholy (Algoritmus 1).

Algoritmus A* počíta funkciu $f(n)$ pomocou heuristiky $h(n)$ ako $f(n) = g(n) + h(n)$, kde $g(n)$ je cena cesty z počiatočného vrchola do vrchola n a heuristika $h(n)$ je odhadovaná cena najkratšej cesty z vrchola n do cieľa.

Ak heuristika $h(n)$ spĺňa nasledovné podmienky, tak je algoritmus A* kompletný a aj optimálny (Russell and Norvig, 2003):

1. $h(n)$ je prípustná - nikdy nenadhodnocuje cenu cesty do cieľa
2. $h(n)$ je konzistentná (monotónna) - pre každý uzol n a jeho nasledovníka n' platí:

$$h(n) \leq c(n, a, n') + h(n')$$

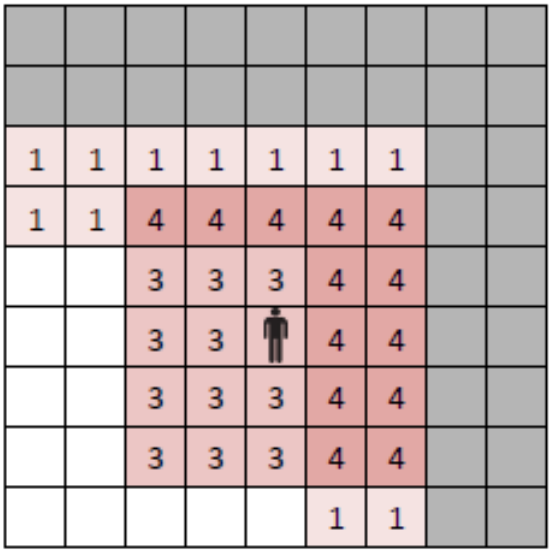
Algoritmus 1 Pseudokód pre algoritmus A*

```
1:  function A*(start, ciel)
2:      close := {}                          ▷ mnozina navstivenych vrcholov
3:      open := {start}                      ▷ mnozina vrcholov, ktore treba navstivit
4:      g[start] := 0                        ▷ cena optimalnej cesty
5:      f[start] := heuristika(start, ciel)  ▷ odhad ceny optimalnej cesty
6:      while open != ∅ do
7:          n := vrchol v open s najmensou hodnotou f(n)
8:          if n = ciel then
9:              return VYTVOR_CESTU(predchodca[n])
10:         Odstran n z open
11:         Pridaj n do closed
12:         for sused in SUSEDNE_VRCHOLY(n) do
13:             if sused ∈ closed then
14:                 continue
15:             nove_g := g[n] + cena_hrany(n, sused)
16:             if sused ∉ open then
17:                 Pridaj sused do open
18:             else
19:                 if nove_g ≥ g[sused] then
20:                     continue
21:             predchodca[sused] := n
22:             g[sused] := nove_g
23:             f[sused] := g[sused] + heuristika(sused, ciel)
24:     return (neexistuje cesta)
```

1.2 Mapa vplyvu

Mapa vplyvu (influence map) je technika, ktorá spočíva v rozdelení priestoru na menšie časti a každej tejto časti sa priradí hodnota. Hodnoty v mape predstavujú nejakú relevantnú informáciu o danej časti priestoru. Dôležitým rozhodnutím pri vytváraní mapy vplyvu je, ako sa priestor rozdelí. Nízke rozlíšenie nedokáže poskytnúť dostatočné informácie pre dobré taktické rozhodovanie. Príliš vysoké rozlíšenie je výpočtovo náročné, hlavne pri real-time rozhodovaní.

Mapa vplyvu bola úspešne použitá v práci (Uriarte and Ontañón, 2012) pri navigácii jednotiek v boji. Agent sa vždy rozhodol, či so svojou jednotkou dokáže prísť k nepriateľovi, zaútočiť na neho a následne ujsť z dostrelu bez toho, aby jeho jednotka dostala nejaké zranenie. Mapu vplyvu sa tu použila na určenie najbezpečnejšieho miesta v danej oblasti, kam mohla jednotka ustúpiť. Informácie ukladali do 2D matice, kde každé políčko matice prislúcha nejakej pozícii na mape - čím má väčšiu hodnotu, tým väčšie nebezpečenstvo na danom mieste hrozí. Ako nebezpečenstvo uvažovali okrem nepriateľských jednotiek aj steny, pretože pri stene sa zvyšuje pravdepodobnosť, že sa jednotka ocitne v pasci a nebude mať kam ďalej ujsť, keďže mu stena môže zablokovať únikové cesty. V mape vplyvu si pre každé políčko uchovávali 2 hodnoty - či je políčko pri stene a ako veľmi ohrozujú nepriatelia dané miesto (suma veľkostí útoku nepriateľov, ktorý na miesto dostrelia).

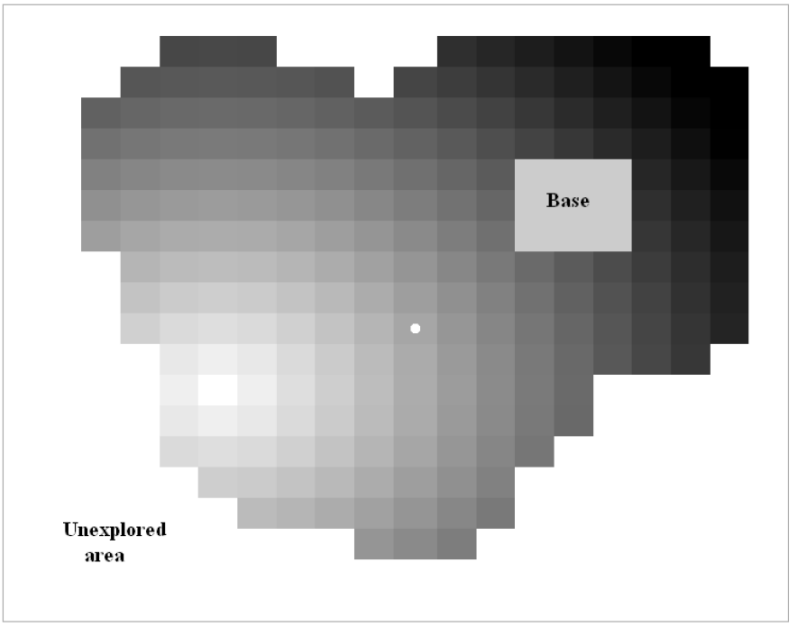


Obr. 1.1: Mapa vplyvu - Vyššie hodnoty v mape určujú väčšie nebezpečenstvo v danej oblasti (Uriarte and Ontañón, 2012)

Navigácia agenta v boji je podobná prieskumu mapy v tom, že pri oboch sa agent snaží, aby jeho jednotka dostala čo najmenej zranenia tým, že sa pohybuje po najbezpečnejších miestach. Avšak pri prieskume sa často stane, že cesta len po najbezpečnejších miestach agenta nedokáže doviesť na dané miesto alebo je veľmi zdĺhavá. Preto agent nemá inú možnosť, ako svoju jednotku čiastočne vystaviť nebezpečenstvu, čo s navigáciou len pomocou mapy vplyvu je komplikované.

1.3 Potenciálové polia

Táto metóda navigácie pochádza z robotiky a prvýkrát ju predstavil Oussama Khatib (Khatib, 1986). Principiálne sú potenciálové polia podobné mape vlyvu, len sa inak reprezentujú informácie, ktoré sú pre nás relevantné. Hlavnou myšlienkou tejto metódy je priradenie priťahujúcich alebo odpudzujúcich nábojov na významné miesta na mape. Každý náboj generuje pole určitej veľkosti. Priťahujúce náboje umiestňujeme k objektom, ku ktorým sa má agent dostať a naopak prekážkam, ktorým sa má vyhýbať, priradíme odpudzujúce náboje. Agent si počíta celkové pole ako váženú sumu všetkých polí. Následne sa agent pohybuje k najatraktívnejšej pozícii (miesto s najvyššou hodnotou) zo všetkých pozícií v určitej vzdialenosti od seba (zvyčajne je to maximálna vzdialenosť, ktorú vie prejsť v jednom časovom úseku).



Obr. 1.2: Prieskumník (biela bodka) sa postupne pohybuje smerom k najatraktívnejšiemu miestu na mape - nepreskúmanej oblasti (biela oblasť) (Hagelbäck and Johansson, 2009)

Priraďovanie typov a veľkostí nábojov objektom na mape sa môže meniť vzhľadom na úlohy jednotiek. Bojové jednotky budú nepriatelia priťahovať, robotníkov naopak odpudzovať, taktiež ložiská minerálov budú priťahovať jednotky, ktoré ich ťažia a ostatné jednotky budú natoľko odpudzovať, aby sa s nimi fyzicky nezrazili (Hagelbäck and Johansson, 2009). Nevýhodou potenciálových polí je nutnosť ladenia parametrov.

Pri tomto spôsobe navigácie sa však agent môže zaseknúť v lokálnom optime. Toto by bolo možné odstrániť tak, že sa agent bude hýbať v určitej vzdialenosti náhodným smerom, čo však môže spôsobiť návrat do lokálneho optima. Jedným z možných riešení je vytvoriť malé odpudzujúce polia na posledné pozície agenta, ktoré zabránia, aby sa agent vracal. Toto bolo úspešne použité v práci (Hagelbäck and Johansson, 2008) pri posledných 20 pozíciách.

Ďalší problém pri navigácii agenta pomocou potenciálových polí objavil Hägelback vo svojej práci (Hagelbäck, 2012). Ak mapa obsahuje veľa úzkych priechodov, tak veľký počet lokálnych optím spôsobí značné problémy pri navigácii agenta. Ako riešenie navrhuje kombináciu navigácie pomocou potenciálových polí a algoritmu A*. Agent používa algoritmus A*, ak nezaznamenal v okolí žiadne iné jednotky, čím minimalizuje problém lokálnych optím a taktiež odstráni nevýhodu A*, ktorý zle zvláda dynamické svety (ak sa cesta zablokuje iným objektom, tak sa musí znovu prepočítať). Ak agent zaznamená v okolí nepriateľa, naviguje pomocou potenciálových polí.

Táto metóda by sa na prieskum dala použiť nasledovne. Miestu, ktoré sa má objaviť, priradíme priťahujúci náboj a pole, ktoré daný náboj generuje, by malo pokrývať celú mapu. Odpudzujúce náboje priradíme ostatných jednotkám, budovám a iným prekážkam (minerály, útesy) tak, že generované pole je len natoľko veľké, aby do nich agent nenarazil. Nepriateľské jednotky, ktoré môžu ohroziť agenta, generujú väčšie odpudzujúce polia (napríklad ako veľkosť ich dostrelu). Avšak predstavme si situáciu, kedy najkratšia cesta k cieľovému miestu vedie cez úzky priechod, na konci ktorého je nepriateľská jednotka. Okrem toho je tu ešte dlhšia cesta, na ktorej agenta nič neohrozí. V takomto prípade by algoritmus A* vybral prvú cestu a na navigáciu pomocou potenciálových polí by sa preplo, až keď nepriateľská jednotka bude v okolí agenta, čo však nastane až na konci priechodu. Pri vhodne nastavených parametroch pre potenciálové polia bude agent pokračovať v ceste ďalej a dostane zranenie, ktorému sa však dalo vyhnúť, ak by hneď zvolil druhú cestu. Ďalšia možnosť je, že sa vráti naspäť, čo však spôsobí prepnutie navigácie na algoritmus A* a situácia sa opakuje.

1.4 Algoritmus CACLA

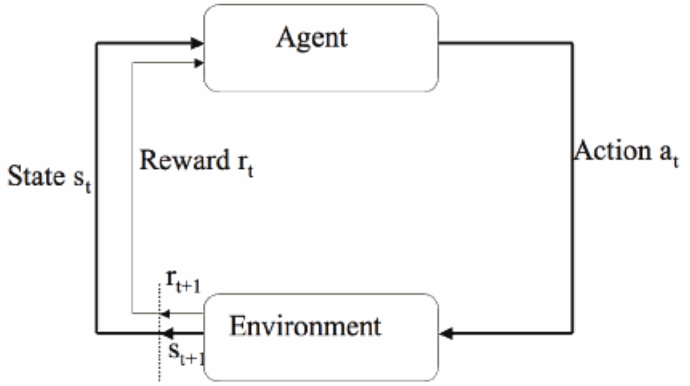
Algoritmus CACLA (van Hasselt and Wiering, 2007, van Hasselt and Wiering, 2009) bol použitý na navigáciu agenta v našej predchádzajúcej práci (Piváčková, 2014) s využitím viacvrstvovej doprednej siete ako funkčného aproximátora. Agentovi sa vygenerovalo miesto vo fixnej vzdialenosti od pozície jeho jednotky na mape. Stav agenta pozostával z vektora smerujúceho na začiatok najkratšej cesty vypočítanej pomocou algoritmu A*, veľkosti zostávajúcej cesty a z hrozieb v okolí agentovej jednotky v podobe mapy vplyvu. Akcia agenta bola reprezentovaná vektorom, ktorý mu hovoril, akým smerom má ísť. Odmena agenta závisela od jeho vzdialenosti k cieľu a zranenia, ktoré počas cesty dostal.

Trénovanie agenta bolo rozdelené do dvoch častí. V prvej časti sa agent úspešne naučil prísť s jednotkou na dané miesto na mape po najkratšej ceste. V druhej časti sa 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. Problémom mohol byť nedostatok času na trénovanie agenta, nevyvážená odmena a trest agenta, ako aj nevhodnosť použitého algoritmu na daný problém.

Kapitola 2

Učenie posilňovaním

Učenie posilňovaním je jedna z metód strojového učenia. Na rozdiel od učenia s učiteľom, kde sa agent učí kopírovať správanie učiteľa, pri učení posilňovaním sa agent učí podľa odmeny a trestu, ktoré dostane od prostredia po vykonaní akcie.



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

Na obrázku 2.1 je popísaný cyklus učenia posilňovaním. Agent sa nachádza v stave s_t. Podľa svojej stratégie vyberie a vykoná akciu a_t, ktorou sa dostane do stavu s_{t+1}. Od prostredia dostane odmenu r_{t+1}, pomocou ktorej upraví svoju stratégiu.

Odmena r_t, ktorú dostane agent od prostredia určuje, ako výhodný je aktuálny stav pre agenta. Formálne môžeme zapísať celkovú odmenu agenta ako:

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

Ak neexistuje konečný stav, čiže $T = \infty$, očakávanú odmenu možno zapísať s použitím konštanty $0 \leq \gamma \leq 1$ (discount factor), ktorá určuje váhu budúcich odmien.

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

Stratégia π je funkcia, ktorá určuje akciu, ktorú má agent v danom stave vykonať. Agent sa učí postupným upravovaním svojej stratégie tak, aby maximalizoval odmenu R_t.

Ohodnotením stavu $V^\pi(s)$ nazveme funkciu, ktorá určuje hodnotu očakávanej odmeny R_t, ak bude agent začínať v stave s a ďalšie akcie bude vyberať podľa stratégie π.

Ohodnotením akcie $Q^\pi(s, a)$ nazveme funkciu, ktorá určuje hodnotu očakávanej odmeny R_t, ak bude agent začínať v stave s, vykoná akciu a a nasledovne sa bude riadiť podľa stratégie π.

Obe funkcie $V(s)$ a $Q(s, a)$ je potrebné vhodne reprezentovať. Pri menšom diskrétnom stavovom priestore nám postačí na ich reprezentáciu tabuľka. Avšak pri rozšírení algoritmov pre veľké diskrétne stavové priestory, kde majú stavy určitú geometrickú štruktúru, ako aj pri spojitých priestoroch potrebujeme už na ich reprezentáciu funkciu. Toto dosiahneme použitím funkčných aproximátorov, ako napríklad neurónová sieť.

Optimálna stratégia π^* je stratégia, ktorá maximalizuje odmenu, ktorú agent dostane vo všetkých stavoch. Takýchto stratégií môže byť aj viac, preto π^* označuje množinu všetkých takých stratégií. Optimálne ohodnocovacie funkcie môžeme zapísať ako:

$$\forall s \in S : V^*(s) = \max_\pi V^\pi(s)$$

$$\forall s \in S, \forall a \in A : Q^*(s, a) = \max_\pi Q^\pi(s, a)$$

2.1 TD-učenie

TD-učenie (temporal difference) je metóda, ktorá v sebe kombinuje myšlienky Monte Carlo a dynamického programovania. **Dynamické programovanie** je metóda výpočtu, ktorá daný problém rozbije na menšie podproblémy a pre každý si zapamätá riešenie, ktoré neskôr môže použiť pri ďalších výpočtoch. Pri výpočte optimálnej stratégie pomocou dynamického programovania je potrebné poznať úplný model prostredia, čo môže byť niekedy problém. **Monte Carlo** metódy na druhú stranu nepotrebujú úplný model prostredia, stačí im len interakcia s prostredím (stavy, akcie, odmeny).

TD-učenie si od Monte Carlo metód berie schopnosť učiť sa zo skúseností, bez modelu prostredia a svoje odhady upravuje ako dynamické programovanie (bez konečného výsledku) vzhľadom na už naučené odhady (Sutton and Barto, 1998).

Základný model $TD(0)$ zohľadňuje len jeden krok v budúcnosti. Funkciu na ohodnotenie stavu upravuje nasledovne:

$$V(s_t) = V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \tag{2.1}$$

(Engelbrecht, 2007)

Na TD-učení sú založené mnohé používané algoritmy ako napríklad Q-učenie (Q-learning), SARSA (State-Action-Reward-State-Action) a CACLA (Continuous Actor-Critic Learning Automaton).

Q-učenie zakladá na ohodnocovaní vykonaných akcií v danom stave pomocou Q-funkcie $Q(s, a)$. Optimálnu stratégiu dostaneme výberom akcií s najvyšším ohodnotením Q-funkcie. Pravidlo pre úpravu Q-funkcie:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2.2}$$

(Russell and Norvig, 2003)

SARSA na rozdiel od Q-učenia nepoužíva na úpravu Q-funkcie najlepšiu akciu, ale ďalšiu vybranú. Pravidlo pre úpravu Q-funkcie môžeme zapísať nasledovne:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2.3}$$

(Sutton and Barto, 1998)

CACLA je rozšírením modelu aktér-kritik pre spojitý priestor akcií a stavov. Tento model sa skladá z dvoch štruktúr, ktoré počas behu algoritmu spolupracujú a učia sa. Aktér vyberá akciu, ktorá sa má vykonať. Potom kritik ohodnotí stav, do ktorého sa dostal, ako TD-chybu a upraví svoje ohodnotenie podľa pravidla 2.1. Podobne sa upraví aj aktér.

Kapitola 3

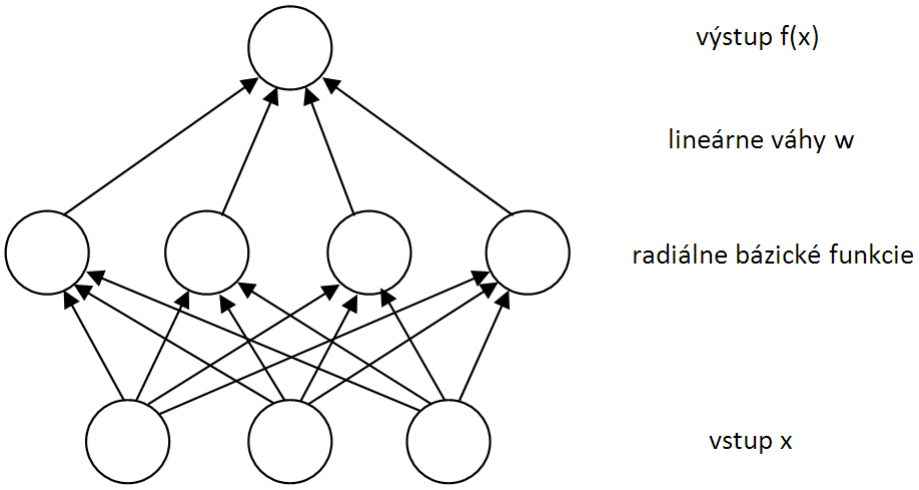
RBF-sieť

RBF-siete sú umelé neurónové siete, ktorých aktivačnou funkciou na skrytej vrstve je **radiálna bázická funkcia**. Tak ako dopredné viacvrstvové neurónové siete, sú aj RBF siete funkčnými aproximátormi. Ich výhodou oproti klasickým dopredným sieťam je rýchlejšie trénovanie a taktiež nie sú citlivé na poradie vstupných vzorov (Návrat et al., 2002).

Nech x je vstupný vektor siete, w sú lineárne váhy, ϕ je radiálna bázická funkcia i-teho neurónu s centrom c_i a m je počet RBF centier. Potom výstupom siete je:

$$f(x) = \sum_{i=1}^{m} w_j \phi(||x - c_i||)$$

(Orr, 1996)



Obr. 3.1: RBF-sieť

Radiálna bázická funkcia je matematická funkcia, ktorá monotónne klesá alebo stúpa so zväčšujúcou sa vzdialenosťou od centra. Často používané funkcie $\phi : R \to R$:

- Gaussovská

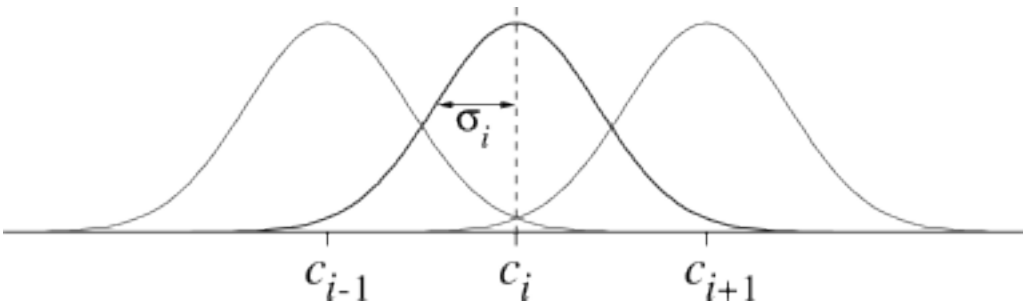
$$\phi(x) = exp(-\frac{x^2}{\sigma^2})$$

- Multikvadratická

$$\phi(x) = \sqrt{\sigma^2 + x^2}$$

- Inverzná multikvadratická

$$\phi(x) = \frac{1}{\sqrt{\sigma^2 + x^2}}$$



Obr. 3.2: Radiálne bázické funkcie (Sutton and Barto, 1998)

3.1 Učenie RBF-sietí

Na určenie pozícií RBF centier c_i možno využiť viacero metód. Najčastejšou metódou je k-priemerovancí klastrovací algoritmus (angl. k-means clustering algorithm). Algoritmus hľadá minimum celkovej sumy vzdialenosti medzi centrami c_i a bodmi x^p, ktoré patria do ich klastrov (Návrat et al., 2002). Ďalšou možnosťou je zvoliť za centrá náhodne vybraté body x^p (Marsland, 2009). Centrami je taktiež možné rovnomerne pokryť celý priestor.

Šírka RBF funkcií σ_i sa taktiež dá nastaviť rôznymi spôsobmi. Môžeme priradiť všetkým RBF funkciám rovnakú šírku a skúšať, aká veľkosť funguje najlepšie. Nastavením šírky σ_i ako maximálnu vzdialenosť medzi pozíciou skrytých neurónov a ich počtom zabezpečíme pokrytie celého priestoru (Marsland, 2009).

Učenie váh w výstupných neurónov

Lineárne váhy trénujeme až po nájdení parametrov c_i a σ_i výpočtom pomocou metódy najmenších štvorcov, napríklad pseudo-inverznej matice aktivácií RBF centier alebo pomocou gradientných metód. (Marsland, 2009).

V práci používame na trénovanie váh metódu **gradient descent** (Engelbrecht, 2007). Cieľom tejto metódy je nájdenie takých hodnôt váh w, aby sa minimalizovala celková chyba E. Nech P_T je celkový počet dvojíc vstupov x_p a k nim požadovaných výstupov d_p a y_p je výstup siete pre vstup x_p. Chybu E môžeme vyjadriť ako:

$$E = \sum_{p=1}^{P_T} (d_p - y_p)^2$$

Váhy upravujeme nasledovne:

$$w_i(t+1) = w_i(t) + \alpha(d_p - y_p)\phi(||x - c_i||$$

Konštanta $\alpha \in (0, 1)$ (learning rate) určuje, akou rýchlosťou sa sieť učí. Príliš vysoké hodnoty α spôsobujú, že sa váhy menia o veľké hodnoty, kvôli čomu môže byť sieť nestabilná. Pri menších hodnotách sa sieť učí pomalšie, ale tým, že dostane každý vstup viackrát, je zasa stabilnejšia a odolnejšia voči rôznym nepresnostiam vo vstupných dátach.

Kapitola 4

Strategické hry

Real-time strategické hry ako prostredie obsahujú veľké množstvo stavov, do ktorých sa agent môže dostať a akcií, ktoré vie vykonať. Prostredie je multiagentové a akcie sa vykonávajú v reálnom čase, preto je potrebné, aby sa agenti rozhodovali rýchlo vzhľadnom na neustále sa meniacu situáciu v hre. Ak by rozhodovanie trvalo príliš dlho, okolnosti, na ktoré agent reaguje, môžu byť v tom čase zastaralé a tým aj jeho neskorá reakcia zbytočná.

4.1 Priebeh hry

Priebeh hry pri real-time strategických hrách sa zvyčajne sleduje z ”vtáčej perspektívy”. Mapa obsahuje rôzne prekážky, ktoré sťažujú pohyb a sú po nej rozložené rôzne zdroje surovín, ktoré musí hráč počas hry získavať, aby vedel stavať budovy a vyrábať jednotky. Jednotky majú rôzne schopnosti ako napríklad ťaženie surovín, stavanie budov, útočenie alebo preprava iných jednotiek. Budovy sú potrebné na ťažbu surovín, stavanie nových jednotiek a na rôzne vylepšenia jednotiek. Cieľ každého hráča závisí od konkrétnej hry. Najčastejšie ide o kompletné zničenie nepriateľa (všetky jeho budovy a jednotky). V niektorých hrách sa zasa hráč musí určitý čas brániť pred zničením alebo vykonať špecifickú úlohu (zničiť konkrétny objekt, dostať sa na špecifické miesto...).

4.2 StarCraft: Brood War

Jednou z najznámejších real-time strategických hier je StarCraft: Brood War od spoločnosti Blizzard Entertaiment z roku 1998. Hra je zasadená do sci-fi prostredia s tromi rasami - Protoss, Zerg a Terran. Každá rasa má vlastné jednotky a budovy. **Protoss** sú mimozemšťania s vyspelou technikou, ich jednotky sú veľmi silné a odolné, ale aj drahé. **Zerg** je rasa podobná hmyzu, ktorej sila spočíva v ich veľkom počte. Jednotky majú relatívne slabé, avšak sú lacné a rýchlo sa vyrábajú. **Terran** sú ľudia, ktorí silou, cenou a počtom jednotiek stoja niekde medzi dvomi predchádzajúcimi rasami.

Na začiatku má hráč k dispozícii základňu - hlavnú budovu a niekoľkých robotníkov. V jej blízkosti sa nachádza obmedzené množstvo surovín - minerály a plyn. Ďalšie suroviny musí hľadať v priebehu hry a na ich efektívnu ťažbu je nútený okolo nich stavať ďalšie základne a brániť ich pred nepriateľom.

Obr. 4.1: Základňa v hre StarCraft: Brood War

4.3 Prieskum mapy

Prieskum mapy je jedna z veľmi dôležitých súčastí real-time strategických hier. Vo väčšine hier hráč nevidí celú mapu a musí ju objavovať, aby získal výhodu nad nepriateľom. Môže ísť o nachádzanie nových zdrojov surovín, dobrých miest pre postavenie základne a v neposlednom rade aj získavanie informácií o nepriateľovi.

Mapa v StarCrafte je reprezentovaná ako štvorcová mriežka, kde veľkosť mapy je udávaná v štvorcoch veľkosti 32×32 pixelov (build tiles). Rozmer mapy sa typicky pohybuje v rozmedzí od 64 × 64 do 256 × 256 build tiles. Rozlíšenie prechodných oblastí býva vyjadrené v štvorcoch veľkosti 8×8 pixelov (walk tiles).

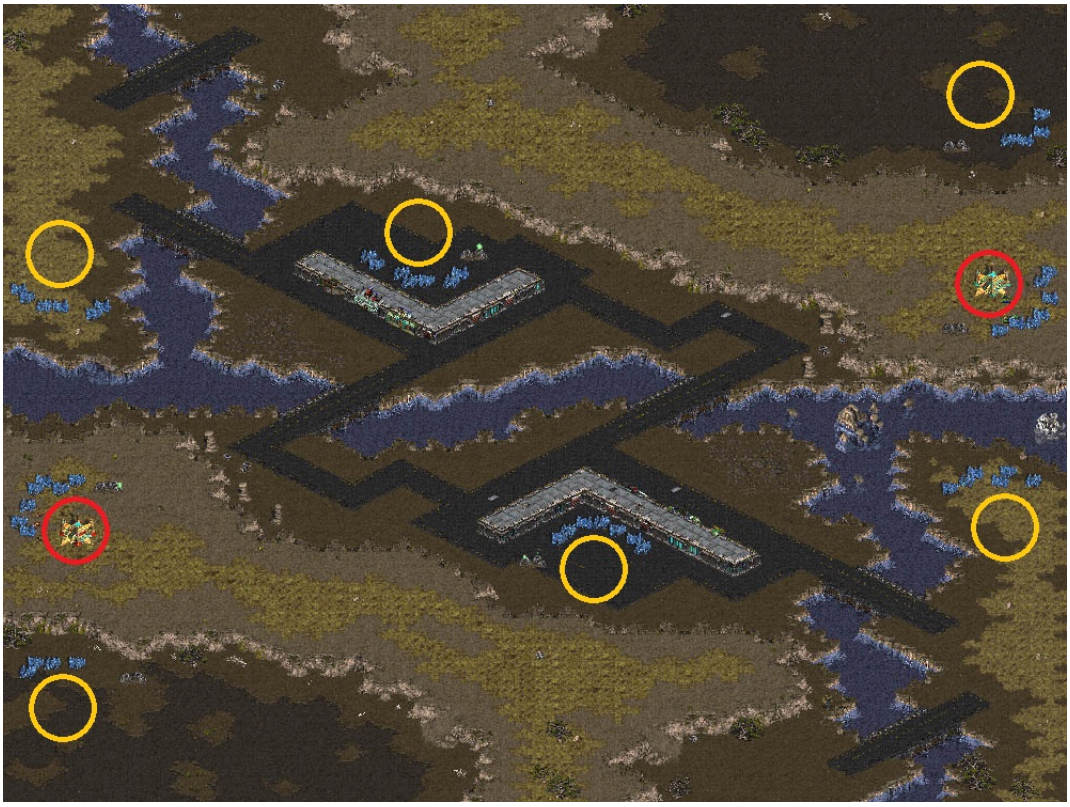
Na začiatku hry hráč vidí len časti mapy, kde sa nachádzajú jeho budovy a jednotky. Ostatné časti mapy sú skryté. Časti mapy, kde sa niekedy v priebehu hry nachádzala hráčova jednotka alebo budova, avšak teraz tam nie je, sú zahalené v hmle (fog of war). Hráč v nej vidí, ako naposledy vyzerala oblasť, aké tam boli prekážky a budovy. Nezachytí však nepriateľské jednotky a taktiež udalosti, čo sa odohrali po opustení danej časti mapy.

Obr. 4.2: Fog of war (hmla), viditeľné (okolie jednotky) a nepreskúmané (čierne) časti mapy

Vo všeobecnosti môžeme prieskum mapy rozdeliť na dve úlohy:

1. výber miesta na mape, ktoré treba preskúmať
2. rýchla a bezpečná cesta na dané miesto

V StarCrafte sú veľmi výhodnými miestami na preskúmanie oblasti na mape, kde môžu mať hráči na začiatku hry základňu (start locations) a oblasti pri väčšom zdroji surovín, ktoré sú výhodné pre stavanie nových základní (base locations). To, že mapy sa negenerujú náhodne a terén je vo väčšine prípadov známy, umožňuje efektívnejšie hľadanie týchto miest. Ich lokácia pre hráča nie je dostupná priamo v hre a musí si ju zistiť mimo hry a zapamätať pre každú mapu. Agent si ju však vie zistiť v rámci informácií, ktoré dostane o mape. Priradenie začiatočných lokácií každému hráčovi je náhodné. Podrobnejší návrh a implementácia tejto úlohy je však nad rámec našej práce, preto výber miesta na preskúmanie bude pre naše potreby napevno daný.



Obr. 4.3: Mapa pre dvoch hráčov (červená = start locations, oranžová = base locations)

V našej práci sa zameriavame na nájdenie rýchlej a bezpečnej cesty na dané miesto na mape. Hráčovu jednotku môžu pri tejto ceste ohroziť rôzne nepriateľské jednotky a spôsobiť jej zranenie. Každá jednotka má určitý počet životov a po ich stratení zomrie - vymaže sa z hry. V ďalšej kapitole predstavíme model agenta, ktorý sa bude snažiť tomuto scenáru zabrániť.

Kapitola 5

Model agenta

V tejto kapitole predstavíme náš navrhovaný model agenta. Prostredie hry, v ktorom sa agentova jednotka nachádza, sme reprezentovali ako neorientovaný hranovo-ohodnotený graf. Agent hľadá najkratšiu cestu v grafe k cieľovému miestu pomocou algoritmu A* s euklidovskou vzdialenosťou ako heuristikou.

5.1 Vytvorenie grafu

Hernú mapu sme reprezentovali ako graf v tvare mriežky. **Vrcholy** grafu predstavujú miesta na mape, kam vie agentova jednotka prísť. **Hrany** reprezentujú prechod medzi susednými miestami na mape. Pre dve susedné políčka i a j vypočítame **cenu hrany** $e(i, j)$ ako súčet euklidovskej vzdialenosti políčok i a j a váhovaného nebezpečenstva na políčku:

$$e(i,j) = ||p_i - p_j|| + \delta \sum_{k=1}^{n} f_{type(k)}(||p_j - p_k||)$$

- p_i, p_j - pozície políčok i a j na mape
- n - počet nepriateľov v okolí políčka j
- $f_{type(k)}$ - funkcia nebezpečenstva pre k-tu nepriateľskú jednotku
- p_k - pozícia k-tej nepriateľskej jednotky
- δ - konštanta určujúca váhu prikladanú nebezpečenstvu na políčku j

Funkcia nebezpečenstva

Funkcia nebezpečenstva popisuje predpokladané zranenie, ktoré dostane agentova jednotka od danej nepriateľskej jednotky na zvolenom políčku. Vstupom je vzdialenosť políčka od nepriateľskej jednotky (v pixeloch). Výstupom funkcie je predpokladané zranenie za frame. Pre rôzne typy jednotiek (dvojica agent - nepriateľ) uvažujeme inú funkciu nebezpečenstva.

Celkovo v práci predstavíme a otestujeme tri rôzne modely výpočtu funkcie nebezpečenstva, kde každý v nejakom smere vylepšuje predchádzajúci model. To znamená, že posledný model by podľa nás mal danú úlohu zvládnuť najlepšie zo všetkých.

5.2 Model 1: Predpočítané nebezpečenstvo

Prvý model funkcie nebezpečenstva je v princípe mapa vplyvu. Nebezpečenstvo vypočítame podľa niektorých vlastností nepriateľskej jednotky a agenta.

Nech dmg je veľkosť a cd je rýchlosť útoku nepriateľskej jednotky. $minR$ ($maxR$) je minimálny(maximálny) dostreľ nepriateľskej jednotky. Konštanta $fact \in \{0.25, 0.5, 0.75, 1\}$ závisí od typu zbrane nepriateľa a veľkosti agentovej jednotky.

$$f(x) = \begin{cases} \frac{dmg.fact}{cd} & \text{ak } x \in [minR, maxR] \\ 0 & \text{inak} \end{cases}$$

Nevýhodou tohto prístupu je, že počíta len s obmedzením počtom vlastností, ktoré ovplyvňujú, koľko zranenia môže agent dostať. Pri výpočte by sa dali brať do úvahy aj rýchlosti a rôzne špeciálne schopnosti daných jednotiek a mnohé iné. Efekt týchto špeciálnych schopností je pre každú jednotku iný a ťažko sa odhaduje, preto s nimi tento model nepočíta. Taktiež každá jednotka sa rôzne správa, čo sťažuje odhad nebezpečenstva. Tento model neberie do úvahy schopnosť súpera ovládať dané jednotky. Toto spôsobuje, že výpočet je často nepresný. Podobný model výpočtu nebezpečenstva bol použitý vo viacerých prácach (Uriarte and Ontañón, 2012), (Tozour, 2001).

5.3 Model 2: Aktuálne nebezpečenstvo

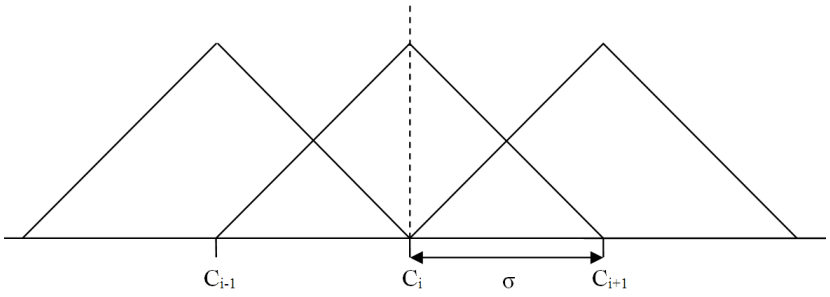
V tomto modeli funkciu nebezpečenstva aproximujeme pomocou funkčného aproximátora. Ako funkčný aproximátor sme zvolili RBF-sieť, ktorú postupne učíme na aktuálne zranenie r, ktoré agent dostal v danej vzdialenosti od nepriateľskej jednotky.

Počet RBF centier sme zvolili ako $m = 20$ a napevno ich rozmiestili vo vzdialenosti σ. $maxR = 320$ je maximálna vzdialenosť (v pixeloch), v ktorej môže agenta ohroziť nepriateľ.

$$\sigma = (maxR/m)/maxR$$

Ako radiálnu bázickú funkciu ϕ sme zvolili funkciu zobrazenú na obrázku 5.1. Dôvod pre výber tejto funkcie bol ten, že je možné túto funkciu veľmi rýchlo počítať a pri aktivácii RBF-siete sú aktívne maximálne dve centrá.

$$\phi(x) = \begin{cases} \sigma - x & \text{ak } x \leq \sigma \\ 0 & \text{inak} \end{cases}$$



Obr. 5.1: Použitá radiálna bázická funkcia

Na začiatku učenia RBF-siete nastavíme váhy siete na nulu a sieť sa trénuje tým, že agent dostáva zranenie, ak vykoná akciu, ktorá ho ohrozila. Veľkosť zranenia, ktoré agent dostal po vykonanej akcii zistíme ako rozdiel počtu životov agenta v stave s_t a s_{t+1}.

Ak daná nepriateľská jednotka vo väčšine prípadov neútočí, agent sa naučí, že mu od nej nehrozí veľké nebezpečenstvo. Toto môžu byť napríklad robotníci alebo jednotky, ktoré majú len špeciálne schopnosti a musia dostať explicitný príkaz od hráča, aby zaútočili na nejakú jednotku (použili svoju útočnú schopnosť).

Nevýhodou tohto modelu je, že nepočíta s možným nebezpečenstvom v budúcnosti. Často sa stáva, že nepriateľská jednotka je rýchlejšia a ak začne agenta naháňať, tak ho postupne dobehne a zničí, čomu sa chceme vyhnúť.

5.4 Model 3: Nebezpečenstvo v budúcnosti

Tretí model zlepšuje predchádzajúci tým, že uvažuje aj možne zranenie, ktoré agent dostane v budúcnosti. Práve absencia tohto predpokladu zranenia v budúcnosti viedla pri predchádzajúcom modeli k situáciám, kedy sa agentova jednotka priblížila k nepriateľskej jednotke do takej vzdialenosti, kde ešte nedostala zranenie, avšak nepriateľ ju zaregistroval a začal naháňať. Ak bol rýchlejší, tak ju vedel neskôr dobehnúť a spôsobiť zranenie. Toto zranenie však spôsobí zväčšenie odhadu nebezpečenstva len pri danej vzdialenosti, keď zranenie dostal a ktorá je menšia ako vzdialenosť, kedy nepriateľ začal agentovu jednotku naháňať.

Tento model výpočtu funkcie nebezpečenstva, tak ako predchádzajúci využíva RBF-sieť ako funkčný aproximátor, ale funkciu nebezpečenstva aproximuje pomocou učenia posilňovaním (TD-učenie):

$$d_t = r + \gamma f(x_{t+1})$$

Odhad nebezpečenstva v budúcnosti si ukážeme na predchádzajúcej situácii. Nech p_t a p_{t+1} sú políčka vo vzdialenosti x_t a x_{t+1} od nepriateľa. Agentova jednotka sa nachádza na políčku p_t a pohne sa na políčko p_{t+1}. Ak jednotka dostane zranenie ($r > 0$), tak si podobne ako predchádzajúci model zvýši nebezpečenstvo na políčku p_t vzhľadom na zranenie r a navyše ešte vzhľadom na odhad nebezpečenstva pre políčko p_{x+1}. Ak jednotka nedostane žiadne zranenie ($r = 0$), ale na políčku p_{t+1} hrozí nejaké nebezpečenstvo ($f(x_{t+1} > 0)$ predchádzajúci model si nijako neupravoval odhad nebezpečenstva. Toto je napríklad vtedy, keď nás jednotka naháňa a zranenie dostaneme až pri ďalšom pohybe. Tento model si však svoj odhad nebezpečenstva upraví a tým sa neskôr dokáže takejto situácii vyhnúť. Hlavnou myšlienkou tohto modelu teda bolo modelovať situácie, kedy na políčku v aktuálnom momente agentovi nehrozí nebezpečenstvo, ale neskôr sa dostane do stavu, kedy bude ohrozený.

Kapitola 6

Implementácia

Agenta sme implementovali v C++ s využítím open-source rozhrania BWAPI (Brood War Application Programming Interface) (BWA, 2015) a BWTA2 (Broodwar Terrain Analyzer) (BWT, 2016). Agent obsahuje aj vlastnú implementáciu algoritmu A* a RBF neurónovej siete.

6.1 BWAPI a BWTA

BWAPI sa využíva na vytvorenie vlastného agenta do hry StarCraft: Brood War. Agentom poskytuje len informácie, ktoré má aj ľudský hráč, nič viac. Toto umožňuje vytvorenie agenta, ktorý nepodvádza a pri vytváraní svojej stratégie sa musí opierať len o čiastočné informácie.

Rozhranie poskytuje rôzne funkcie na ovládanie jednotiek a prístup k ich vlastnostiam. Taktiež je tu možnosť rôznych testovacích výstupov ako vykresľovanie na mapu alebo výpis údajov na obrazovku.

BWTA je doplnok BWAPI na analýzu mapy. Používa sa na výpočet miest vhodných pre základne (base locations), regiónov a úzkých miest na mape (chokepoints). V práci využívame hlavne dáta o prekážkach a priechodnosti terénu (nepriechodné polygóny).

6.2 Agent

Mapu sme rozdelili na políčka veľkosti 8x8px (walk tiles). Tieto políčka predstavujú vrcholy grafu. Nepriechodné miesta s prekážkami, na ktoré sa agent nemôže dostať, sme z grafu odstránili na začiatku programu. Miesta, ktoré len dočasne blokovali jednotky a budovy, sme dynamicky odstraňovali a pridávali do grafu pred každým spustením algoritmu A*. Odstraňovanie a pridávanie vrcholov sme riešili prepínaním premennej, ktorá určovala existenciu vrcholu v grafe. Tú musíme kontrolovať aj pri práci s hranami, keďže tie z grafu explicitne neodstraňujeme.

Každá hrana si pamätá euklidovskú vzdialenosť medzi jej vrcholmi, ku ktorej sa vždy pričíta nebezpečenstvo na koncovom vrchole, ktoré udáva váhovaná funkcia nebezpečenstva. Tá sa vždy prepočíta pre každý vrchol v okolí nepriateľskej jednotky, pri každom novom spustení algoritmu A* a jej hodnota sa uloží.

Podľa vlastností agentovej jednotky sa prispôsobí aj vytváranie grafu, ako aj výpočet prepočítaného nebezpečenstva. Napríklad lietajúce jednotky môžu lietať nad rôznymi prekážkami, čiže nemusíme odstraňovať dané políčka z grafu. Zároveň ich neohrozujú jednotky, ktoré vedia útočiť len na pozemné jednotky.

RBF-sieť sme implementovali sami. Keďže v našom modeli je veľkosť vstupného aj výstupného vektora rovná jednej, výpočet sa dal urýchliť tak, že implementácia siete vždy na vstupe čaká len jednu hodnotu a vracia taktiež len jednu hodnotu. Toto odstráni zbytočnú prácu s poliami pri výpočte.

Kapitola 7

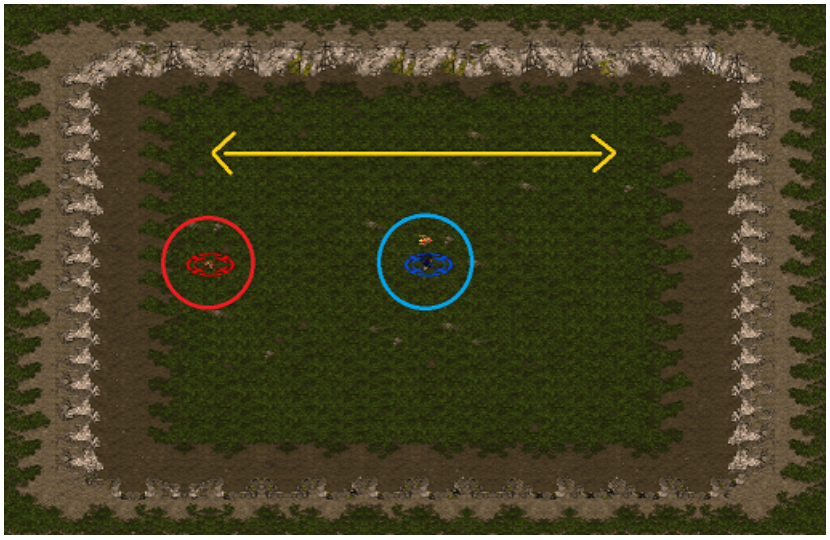
Experimenty

V rámci experimentov sme agenta ako prvé natrénovali a potom sme ho testovali v dvoch experimentoch. Výsledky sme zhodnotili neskôr v diskusii.

7.1 Trénovacia fáza

Inicializácia

Trénovanie agenta spočívalo v aproximácii funkcie nebezpečenstva pre danú jednotku agenta a nepriateľa. Mapa, na ktorej sa agent trénoval, mala tvar uzavretej arény s jednou nepriateľskou jednotkou v strede a jednotkou agenta na ľavej strane. Veľkosť arény bola dostatočná na to, aby sa agent vedel svojou jednotkou bezpečne vyhnúť nepriateľovi. Agentovej jednotke bolo umelo nastavené maximálne možné množstvo životov (1000) a bol vypnutý štít, aby jeho priebežné obnovovanie nemalo vplyv na učenie.



Obr. 7.1: Trénovacia mapa - agentova jednotka (červená), nepriateľ (modrá)

Priebeh

Agent dostáva náhodne vygenerované miesta na opačnej strane nepriateľskej jednotky vzhľadom na svoju jednotku. Agent naviguje svoju jednotku na dané miesto a priebežne si upravuje funkciu nebezpečenstva. Ak agentovi jednotka zomrie, hra sa reštartuje a pokračuje sa v učení ďalej. Učenie skočí, ak agent 4000 krát úspešne príde s jednotkou na dané miesto.

Výber jednotiek a ich učenie

Ako agentovu jednotku sme vybrali robotníka **Probe**, ktorá sa bežne používa na prieskum, pretože je to relatívne rýchla jednotka vzhľadom na ostatné jednotky a je k dispozícii od začiatku hry. Nepriateľské jednotky sme vyberali podľa ich vlastností tak, aby reprezentovali určitú skupinu jednotiek v hre, s ktorými sa agent vie stretnúť. Tabuľka 7.1 ukazuje porovnanie vybraných jednotiek. Natrénované funkcie nebezpečenstva pre vybrané jednotky môžeme vidieť v grafoch 7.2 (a) - (f).

Hydralisk - Jednotka s priemernou silou, dostrelom a rýchlosťou.

Goliath - Relatívne rýchla a silná jednotka s veľkým dostrelom.

Vulture - Veľmi rýchla jednotka, ktorá vie väčšinu ostatných ľahko dobehnúť a zničiť.

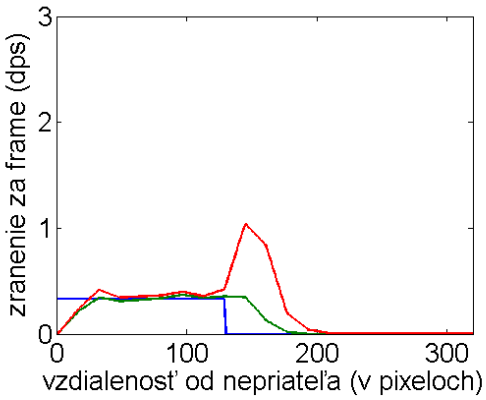
Siege Tank (Tank mode) - Jednotka so silným útokom a veľkým dostrelom.

Ultralisk - Veľká silná a relatívne rýchla jednotka, ktorá nedokáže strieľať.

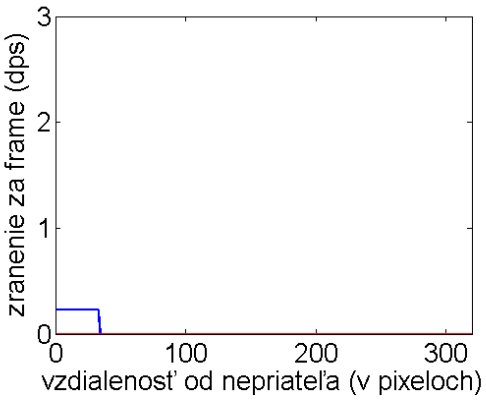
Drone - Jednotka, ktorá ako jediná vie ťažiť suroviny a stavať budovy (robotník). Dokáže aj útočiť, avšak bez explicitného príkazu od hráča to bežne nerobí.

Type jednotky	Veľkosť útoku	Rýchlosť útoku	Dostrel (px)	Pohyb (px za frame)
Probe	5	22	32	4.92
Drone	5	22	32	4.92
Hydralisk	5	15	128	3.66
Goliath	12	22	192	4.57
Vulture	20	30	160	6.40
Siege Tank	15	37	224	4.00
Ultralisk	20	15	25	5.12

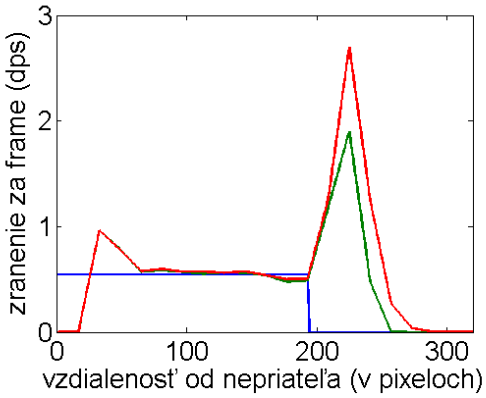
Tabuľka 7.1: Porovnanie jednotiek. Veľkosť útoku je počítaná vzhľadom na agentovu jednotku (prenásobené konštantou podľa typu útoku a veľkosti agentovej jednotky).



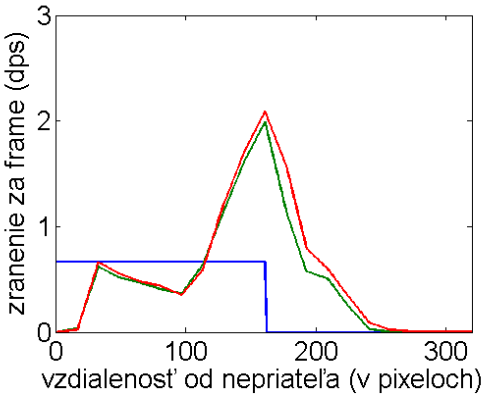
(a) Hydralisk



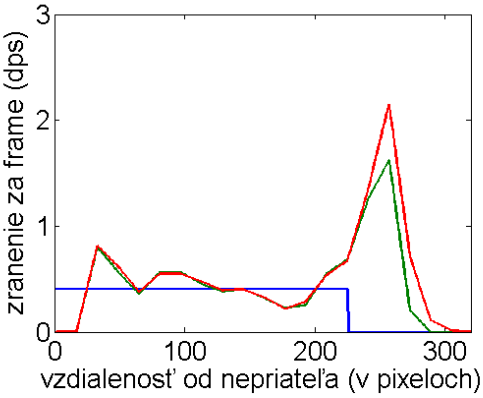
(b) Drone



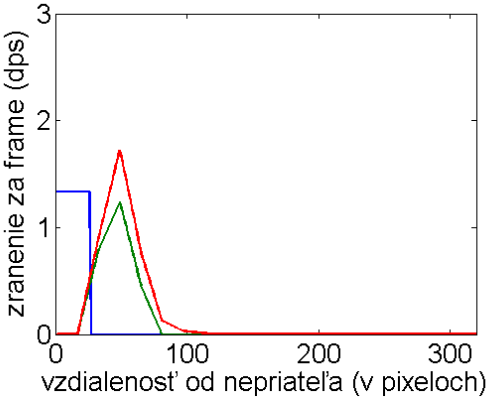
(c) Goliath



(d) Vulture



(e) Siege Tank



(f) Ultralisk

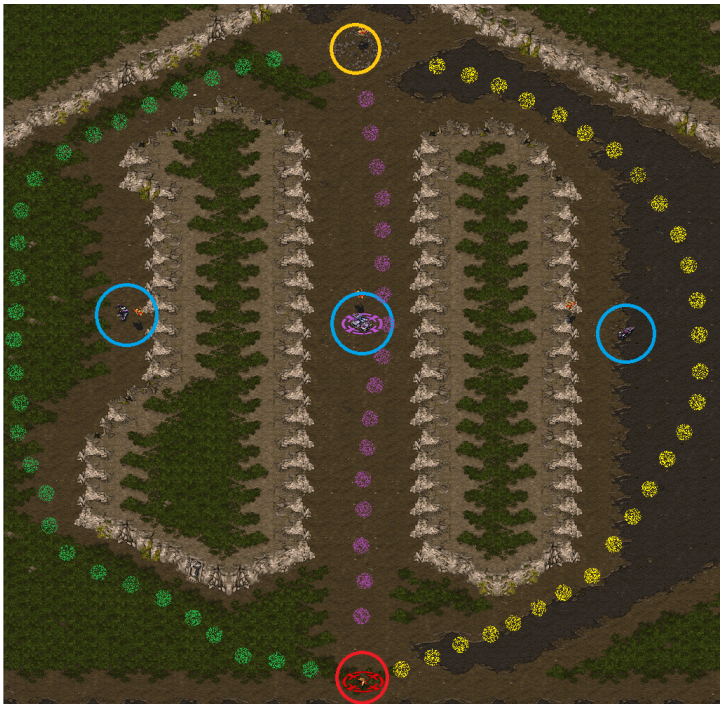
Obr. 7.2: Porovnanie modelov výpočtu funkcie nebezpečenstva: 1. Predpočítané nebezpečenstvo (modrá), 2. Aktuálne nebezpečenstvo (zelená), 3. Nebezpečenstvo v budúcnosti (červená)

7.2 Testovacia fáza

7.2.1 Experimet 1: Tri cesty

Cieľom tohto experimentu bolo ukázať správanie a problémy jednotlivých funkcií nebezpečenstva. Úlohou agenta je dostať jednotku zo dolnej na hornú časť mapy, pričom si musí vybrať jednu z troch ciest. Každá cesta je rôzne dlhá a nebezpečná (obrázok 7.3):

1. najkratšia cesta (fialová) so silnou jednotkou v strede cesty (Siege Tank)
2. stredne dlhá cesta (žltá) s rýchlou jednotkou, ktorá vie agenta dobehnúť (Vulture)
3. najdlhšia cesta (zelená) s jednotkou, ktorá sa dá bezpečne obísť (Goliath)



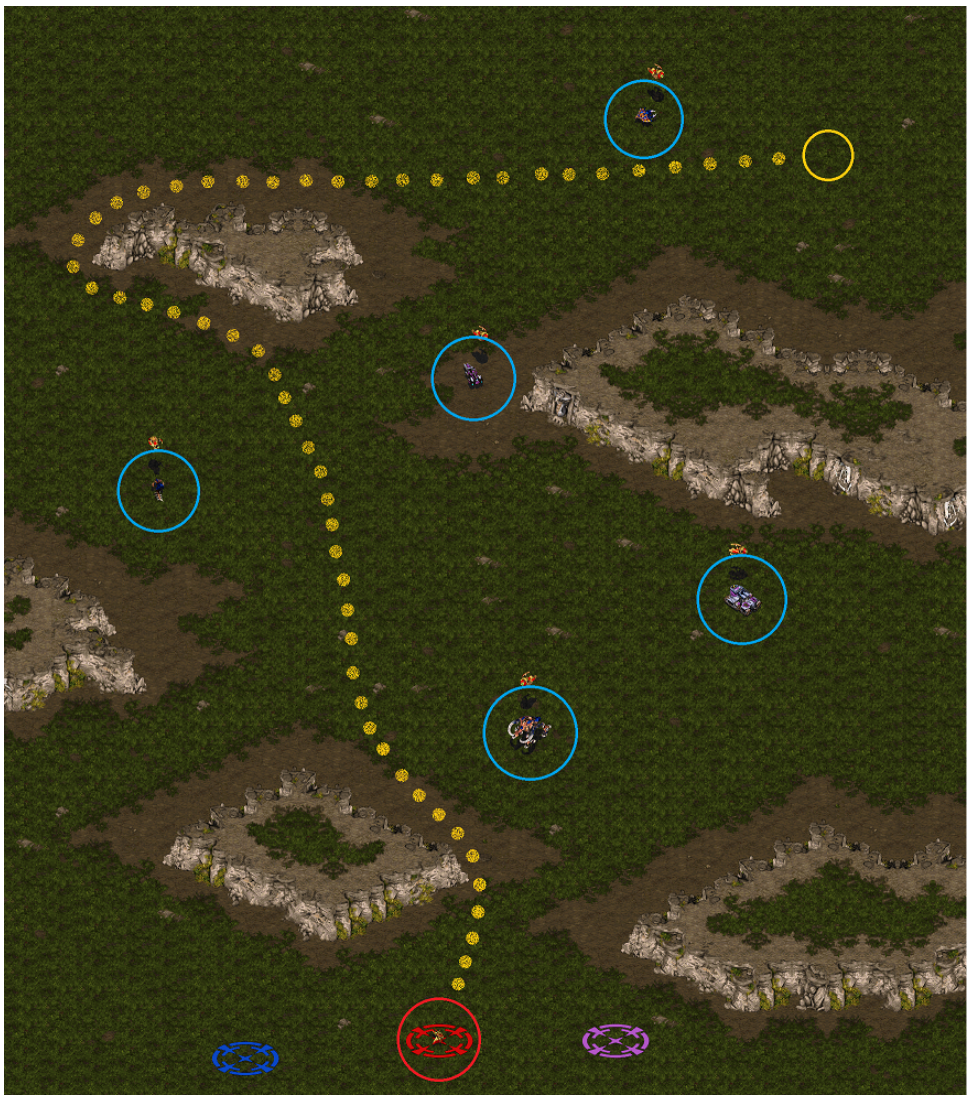
Obr. 7.3: Mapa s tromi cestami (1. fialová, 2. žltá, 3. zelená) - agentova jednotka (červená), nepriatelia (modrá), cielové miesto (oranžová)

Vyskúšali sme všetky tri modely funkcie nebezpečenstva pre $\delta = 50$ a agenta používajúceho na navigáciu len algoritmus A*. Tento agent si zvolil najkratšiu cestu, kde nepriateľská jednotka agentovu jednotku zničila. Agenti s predpočítaným modelom nebezpečenstva a modelom aktuálneho nebezpečenstva išli s jednotkou po stredne dlhej ceste, kde ju zakaždým nepriateľská jednotka dobehla a zničila. Agent s modelom nebezpečenstva v budúcnosti si vybral najdlhšiu cestu a úspešne prišiel s jednotkou na dané miesto bez toho, aby dostala nejaké zranenie.

7.2.2 Experimet 2: Váhovanie nebezpečenstva

V tomto experimente sme testovali všetky modely funkcie nebezpečenstva pri rovnakej situácii a porovnávali čas, ktorý agent potreboval na príchod s jednotkou na dané miesto a aké zranenie popritom jednotka dostala.

Na obrázku 7.4 je zobrazená použitá testovacia mapa. Agent sa musí odnavigovať jednotku z dolnej časti mapy do pravého horného rohu. Ako pri testovaní, tak aj tu mala agentova jednotka zvýšené životy na 1000 a vypnutý štít, pre lepšie zaznamenanie zranenia. Nepriateľské jednotky sme vyberali a rozmiestňovali tak, aby bránili agentovi v priamej ceste do cieľa. Na obrázku je vyznačená najbezpečnejšia cesta, ktorú použil najlepší agent.



Obr. 7.4: Testovacia mapa - agentova jednotka (červená), nepriatelia (modrá), najbezpečnejšia cesta k cielovému miestu (oranžová)

Výsledky experimentu

Agenta sme pre každý model funkcie nebezpečenstva, ako aj pre agenta bez funkcie nebezpečenstva (len algorimuts A*) pustili 10 krát. V tabuľkách 7.2 a 7.3 sú uvedené priemerné zranenia, ktoré agentova jednotka dostala a počet framov, potrebný na dosiahnutie cieľa pre rôzne nastavenia $\delta = \{1, 10, 50, 100\}$. Pri agentovi bez funkcie nebezpečenstva nemalo zmysel testovať iné hodnoty δ. Oranžovou farbou sú vyznačené hodnoty, kedy by za normálnych okolností jednotka prežila. Všetky hodnoty namerané počas 10 testov sú zobrazené v prílohe A.

δ	**1**	**10**	**50**	**100**
Bez modelu nebezpečenstva	104	-	-	-
Predpočítané nebezpečenstvo	100	118	126	114
Aktuálne nebezpečenstvo	102	44	26	27
Nebezpečenstvo v budúcnosti	92	22	12	0

Tabuľka 7.2: Priemerné zranenie jednotky

δ	**1**	**10**	**50**	**100**
Bez modelu nebezpečenstva	492	-	-	-
Predpočítané nebezpečenstvo	492	536	536	537
Aktuálne nebezpečenstvo	498	655	803	773
Nebezpečenstvo v budúcnosti	501	757	751	758

Tabuľka 7.3: Priemerný čas potrebný na dosiahnutie cieľa (počet framov)

Kapitola 8

Diskusia

8.1 Trénovanie

V trénovacej fáze sme aproximovali model aktuálneho nebezpečenstva a model nebezpečenstva v budúcnosti pre niektoré vybrané jednotky. Výsledky trénovania a porovnanie s predpočítaným modelom nebezpečenstva sú zobrazené na obrázku 7.2. Modrý graf znázorňuje predpočítaný model nebezpečenstva. Skok na nulu vo funkcii predstavuje koniec dostrelu danej jednotky.

Pri všetkých jednotkách môžeme vidieť, že nebezpečenstvo pri modeli aktuálneho nebezpečenstva je nebezpečenstvo zvýšené aj za dostrelom danej jednotky. Toto bolo zapríčinené tým, že projektil vystrelený nepriateľskou jednotkou, má pre každý typ jednotky rôznu rýchlosť letu, čo spôsobí, že agentova jednotka dostane zranenie až keď je mimo dostrelu. V prípade jednotky Ultralisk (7.2 f), ktorá útočí na blízko (bez projektilu) to bolo spôsobené tým, že ak jednotka príde dostatočne blízko na to, aby začala útok, tak druhá jednotka dostane po dokončení animácie útoku zranenie, aj keď sa už vzdialila od útočiacej jednotky. Model nebezpečenstva v budúcnosti je tvarom podobný modelu aktuálneho nebezpečenstva s tým rozdielom, že nebezpečenstvo je väčšie a ďalej od nepriateľa.

Graf pri jednotke Drone (7.2 b) ukazuje, že oba trénované modely sa naučili na nulové nebezpečenstvo pre všetky vzdialenosti, keďže táto jednotka nikdy nezaútočila. Pri všetkých jednotkách môžeme vidieť, že vo veľmi malej vzdialenosti od nepriateľa sa taktiež oba trénované modely naučili na hodnoty blízke nule. Toto vyplýva z vlastnosti RBF-siete, ktorá sa učí veľmi lokálne a do týchto vzdialeností sa agentova jednotka nikdy nedostala. Dobre viditeľné je to pri veľkých jednotkách - Goliath (7.2 c), Siege Tank (7.2 e) a Ultralisk (7.2 f).

Táto vlastnosť RBF-sietí spôsobuje taktiež veľký nárast nebezpečenstva na konci dostrelu, lebo v tých častiach sa agentova jednotka pohybovala častejšie ako vo väčšej blízkosti nepriateľa, pretože ju nepriateľ naháňal.

Výsledky trénovania z väčšej časti potvrdili naše očakávania. Tvary funkcií podľa nás dobre modelujú situácie, kedy agentova jednotka dostávala zranenie. Avšak očakávali sme väčší rozdiel medzi modelom aktuálneho nebezpečenstva a model nebezpečenstva v budúcnosti, hlavne pri konci dostrelu nepriateľa.

8.2 Testovanie

V prvom experimente sme chceli ukázať rozdiely v správaní modelov funkcie nebezpečenstva a prečo je náš tretí model, ktorý uvažuje aj zranenie v budúcnosti, lepší ako ostatné. Experiment potvrdil, že len použitie algoritmu A* bez uvažovania nebezpečenstva agentovu jednotku najviac ohrozilo, čo spôsobilo jej okamžité zničenie. Pri predpočítanom modeli nebezpečenstva a modeli aktuálneho nebezpečenstva, ktoré zvolili dlhšiu, ale stále nebezpečnú cestu, kde jednotka bola taktiež zničená, sa ukázala potreba zvažovať nielen aké aktuálne zranenie môže jednotka dostať v danom momente, ale aj jej možné ohrozenie v budúcnosti. Toto agent s modelom nebezpečenstva v budúcnosti úspešne zvládol tým, že dokázal navigovať svoju jednotku do cieľa nezranenú.

V druhom experimente sa porovnávali všetky modely s tým, že sa vypočítanému nebezpečenstvu prikladala rôzna váha. Hoci algoritmus na navigáciu je deterministický, rôzne natočenie nepriateľských jednotiek spôsobuje rozdiely pri každom spustení testu. Ďalšie pozorovania sme robili len z oranžových hodnôt v tabuľkách 7.2 a 7.3, ktoré predstavujú situácie, kedy by v normálnej hre jednotka prežila. Toto sa podarilo len modelu aktuálneho nebezpečenstva a modelu nebezpečenstva v budúcnosti. Z hľadiska času, ktorý jednotka potrebovala, bol najlepší agent s modelom nebezpečenstva v budúcnosti s $\delta = 50$ a najmenšie zranenie dostal taktiež agent s modelom nebezpečenstva v budúcnosti a $\delta = 100$. Môžeme si všimnúť, že pre akékoľvek nastavenie δ agent s modelom nebezpečenstva v budúcnosti navigoval jednotku rýchlejšie a s menším zranením ako agent s modelom aktuálneho nebezpečenstva.

Veľký počet framov pri modeli aktuálneho nebezpečenstva s $\delta = 50$ bol zapríčinený tým, že sa agent s jednotkou snažil obísť nepriateľa, ktorý ju začal naháňať, viac ako mal pôvodne v pláne. To však neznížilo zranenie, lebo nepriateľ jednotku aj tak dobehol a zároveň zvýšilo čas oproti pôvodnej trase. Rozdiel v zranení agenta s predpočítaným modelom a modelom aktuálneho nebezpečenstva s väčšou váhou δ bol zapríčinení tým, že agent s aktuálnym modelom nebezpečenstva obchádzal nepriateľov vo väčšej diaľke a tak mali nepriatelia menej času mu spôsobiť zranenie.

Experiment potvrdil, že model nebezpečenstva v budúcnosti sa ukázal ako najlepší z testovaných modelov, či už z hľadiska zranenia jednotky, ako aj času. Zníženie váhy nebezpečenstva δ nemalo výrazný vplyv na čas cesty, len zväčšovalo zranenie.

Záver

V práci sme navrhli agenta pre prieskum a jeho implementáciu a testovanie v real-time strategickej hre StarCraft: Brood War. Jeho úlohou bolo navigovať svoju jednotku na dané miesto čo najrýchlejšie a hlavne s čo najmenším zranením.

Navrhli sme vytvorenie grafu, reprezentujúceho hernú mapu, kde vrcholy sú políčka tvoriace mapu a hrany reprezentujú prechod medzi susednými políčkami. Cena hrany pozostávala z euklidovskej vzdialenosti medzi políčkami a nebezpečenstvom od všetkých nepriateľov v okolí políčka. V práci sme predstavili tri rôzne modely výpočtu tejto funkcie nebezpečenstva pre daný typ nepriateľskej jednotky. Prvý model (predpočítané nebezpečenstvo) bol v princípe mapa vplyvu, ktorá už bola použitá v mnohých prácach. Druhý model (aktuálne nebezpečenstvo) aproximuje funkciu nebezpečenstva tým, že učí RBF-sieť na aktuálne zranenie, ktoré agent dostane. Jeho nájdené nedostatky sme chceli odstrániť v treťom modeli (nebezpečenstvo v budúcnosti), kde sme navrhli učiť RBF-sieť pomocou učenia posilňovaním.

Trénovanie agenta prebiehalo v uzatvorenej aréne s jednou nepriateľskou jednotkou v strede. Agent chodil so svojou jednotkou z jednej strany na druhú a popritom si upravoval svoju funkciu nebezpečenstva. Agenta sme natrénovali pre rôzne typy nepriateľských jednotiek.

Jednotlivé modely funkcie nebezpečenstva sme testovali na dvoch experimentoch. Prvý experiment spočíval v tom, že k cieľovému miestu existovali tri rôzne dlhé a nebezpečné cesty. Pozorovali sme, ktorú cestu si agent zvolí a či sa mu podarí dostať do cieľa. Úspešne to zvládol len agent s modelom nebezpečenstva v budúcnosti. Pri ostatných modeloch agentovi jednotka zomrela.

V druhom experimente mal agent prejsť do cieľa a v ceste mu stáli rôzne prekážky (terén aj nepriatelia). Každý model funkcie nebezpečenstva sme pre rôzne váhy nebezpečenstva δ testovali 10 krát a porovnávali priemerný čas, za ktorý agent zvládol cestu a koľko zranenia v priemere dostala jeho jednotka. Zaujímali nás hlavne prípady, kedy jednotka prežila (pri testovaní mala zvýšený počet životov). Toto zvládli len agenti s modelom aktuálneho nebezpečenstva a modelom nebezpečenstva v budúcnosti a pri vyššej váhe nebezpečenstva. Taktiež sme ukázali, že z tých prípadov, kedy jednotka prežila, to agent s modelom nebezpečenstva v budúcnosti dokázal zvládnuť najrýchlejšie a k tomu s najmenším zranením. Pri najväčšej testovanej váhe nebezpečenstva vedel danú cestu prejsť s nulovým zranením, čo považujeme za úspech.

V budúcnosti by sme chceli optimalizovať operácie súvisiace s upravovaním grafu. Taktiež by sme chceli naučiť agenta, ako má vyberať miesto na mape, ktoré sa má preskúmať. Práca by sa taktiež dala rozšíriť pridaním možnosti útoku na nepriateľa, pretože niekedy je výhodnejšie nepriateľskú jednotku hneď zničiť, ako pred ňou dlho utekať.

Príloha A

Namerané časy a obdržané zranenia agentov počas druhého experimentu

δ	**1**	
1	494	120
2	495	100
3	486	100
4	486	100
5	501	100
6	483	80
7	491	100
8	493	120
9	493	100
10	493	120

Tabuľka A.1: A*: Namerané časy a zranenia počas 10 testov

δ	**1**		**10**		**50**		**100**	
1	488	100	536	120	534	100	536	120
2	489	80	531	100	535	120	536	100
3	483	100	542	120	540	140	543	140
4	500	100	534	100	534	140	535	100
5	482	100	536	120	534	140	534	100
6	492	100	536	120	536	120	533	100
7	481	100	535	120	540	140	540	120
8	507	100	536	120	532	140	534	120
9	495	100	541	120	535	100	539	120
10	504	120	535	140	535	120	536	120

Tabuľka A.2: Model 1 - Predpočítané nebezpečenstvo: Namerané časy a zranenia počas 10 testov

δ	**1**		**10**		**50**		**100**	
1	495	120	767	20	784	20	771	20
2	512	100	773	60	789	20	770	45
3	498	100	576	20	785	40	769	20
4	505	100	766	40	989	60	771	20
5	496	120	589	60	789	20	784	45
6	499	100	576	40	787	20	771	20
7	496	100	579	40	777	20	780	25
8	483	100	774	80	776	20	777	40
9	496	100	575	60	765	20	767	5
10	495	80	578	20	787	20	767	30

Tabuľka A.3: Model 2 - Aktuálne nebezpečenstvo: Namerané časy a zranenia počas 10 testov

δ	**1**		**10**		**50**		**100**	
1	495	100	821	20	806	0	763	0
2	501	80	805	20	806	0	750	0
3	513	120	800	0	859	20	756	0
4	500	80	789	20	809	0	767	0
5	481	80	592	40	806	0	750	0
6	510	100	789	20	809	0	763	0
7	499	100	797	20	643	40	755	0
8	502	80	794	20	666	20	763	0
9	512	100	794	40	636	20	754	0
10	497	80	590	20	665	20	756	0

Tabuľka A.4: Model 3 - Nebezpečenstvo v budúcnosti: Namerané časy a zranenia počas 10 testov

Príloha B

CD médium

Priložené CD obsahuje zdrojový kód implementovaného agenta a použité mapy.

Literatúra

[BWA, 2015] (2015). Bwapi dokumentácia a git repozitár. `http://bwapi.github.io/` a `https://github.com/bwapi/bwapi`. [Online; accessed 30-April-2016].

[BWT, 2016] (2016). Bwta2 git repozitár. `https://bitbucket.org/auriarte/bwta2/overview`. [Online; accessed 30-April-2016].

[Engelbrecht, 2007] Engelbrecht, A. P. (2007). *Computational Intelligence: An Introduction, 2nd Edition*. Wiley.

[Hagelbäck, 2012] Hagelbäck, J. (2012). Potential-field based navigation in starcraft. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 388–393. IEEE.

[Hagelbäck and Johansson, 2008] Hagelbäck, J. and Johansson, S. J. (2008). Using multi-agent potential fields in real-time strategy games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 631–638. International Foundation for Autonomous Agents and Multiagent Systems.

[Hagelbäck and Johansson, 2009] Hagelbäck, J. and Johansson, S. J. (2009). A multi-agent potential field-based bot for a full rts game scenario. In *AIIDE*.

[Khatib, 1986] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98.

[Marsland, 2009] Marsland, S. (2009). *Machine Learning: An Algorithmic Introduction*. CRC Press, New Jersey, USA.

[Návrat et al., 2002] Návrat, P., Bieliková, M., Beňušková, `[U+FFFD]`, Kapustík, I., and Unger, M. (2002). *Umelá inteligencia*. Vydavateľstvo STU.

[Orr, 1996] Orr, M. J. L. (1996). *Introduction to Radial Basis Function Networks*. Centre for Cognitive Science, University of Edinburgh.

[Piváčková, 2014] Piváčková, L. (2014). Trénovanie agenta v real-time strategických hrách pomocou učenia posilňovaním.

[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*, volume 28. MIT press.

[Tozour, 2001] Tozour, P. (2001). Influence mapping. In *M. Deloura (Ed.), Game Programming Gems 2*, pages 287–297. Charles River Media, Inc.

[Uriarte and Ontañón, 2012] Uriarte, A. and Ontañón, S. (2012). Kiting in rts games using influence maps. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*.

[van Hasselt and Wiering, 2007] van Hasselt, H. and Wiering, M. A. (2007). Reinforcement learning in continuous action spaces. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*, pages 272–279. IEEE.

[van Hasselt and Wiering, 2009] van Hasselt, H. and Wiering, M. A. (2009). Using continuous action spaces to solve discrete problems. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 1149–1156. IEEE.