UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
Katedra aplikovanej informatiky

Simulácia šermu v 3D za pomoci metód umelej inteligencie

Diplomová práca

Bratislava, 2011 **Bc. Jaroslav Blanář**

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
Katedra aplikovanej informatiky

Simulácia šermu v 3D za pomoci metód umelej inteligencie

Diplomová práca

Bc. Jaroslav Blanár

Študijný program: Aplikovaná informatika

Študijný odbor: 9.2.9 Aplikovaná informatika – Umelá inteligencia

Vedúci diplomovej práce:

doc. Ing. Igor Farkaš PhD.

Bratislava, 2011

Zadanie školského diela

Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky

ZADANIE

Meno a priezvisko študenta: Jaroslav Blanář

Študijný program: Aplikovaná informatika

Študijný odbor: 9.2.9 Aplikovaná informatika – Umelá inteligencia

Typ záverečnej práce: Diplomová práca

Jazyk záverečnej práce: Slovenčina

Sekundárny jazyk: Angličtina

Názov: Cieľ práce: Simulácia šermu v 3D za pomoci metód umelej inteligencie

Vedúci: doc. Ing. Igor Farkaš PhD.

Školiteľ: doc. Ing. Igor Farkaš PhD.

Katedra: Aplikovaná informatika

Vedúci katedry: doc. Phdr. Ján Rybár, Phd.

Spôsob sprístupnenia elektronickej verzie práce:

Dátum schválenia: .....

vedúci katedry[1]

študent

vedúci, resp. školiteľ[2]

Dátum potvrdenia finálnej verzie práce, súhlas s jej odovzdaním (vrátane spôsobu sprístupnenia)

[1] Príslušná fakulta si určí, kto schvaľuje zadanie.
[2] Príslušná fakulta si určí, kto schvaľuje zadanie.

Čestne prehlasujem, že som tú to diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

. . . . . .

Poďakovanie

Svojmu školiteľovi doc. Ing. Igorovi Farkašovi, PhD. ďakujem za cenné rady a tiež profesionálne vedenie počas tvorby mojej diplomovej práce. Taktiež chcem poďakovať Mišovi Antoničovi za jeho podporu a tvorivé myslenie, ktorým spestril moju prácu.

Bibliografická identifikácia

BLANÁŘ, Jaroslav. Simulácia šermu v 3D za pomoci metód umelej inteligencie [Diplomová práca].
Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky; Katedra aplikovanej informatiky.
Školiteľ: doc. Ing. Igor Farkaš, PhD. Bratislava: FMFI UK, 2011. 56 s.

Abstrakt

Táto práca sa zaoberá simuláciou šermovania v 3D prostredí, kde hlavným problémom je schopnosť naučiť agenta obranné manévre, aby sa dokázal ubrániť trénerovým výpadom. Zložitosť problému je o to náročnejšia, že trénerov meč nie je statickou entitou, ale dynamickou. Prostredie, v ktorom sa agent nachádza, je spojité vzhľadom na stavy a akcie. Na tento účel využívame učenie s posilňovaním (reinforcement learning - RL) pomocou algoritmu CACLA (Continuous actor critic learning automaton). Pôvodná verzia algoritmu, použitá na trénovanie doprednej neurónovej siete, neviedla k efektívnemu riešeniu, a tak sme navrhli modifikáciu algoritmu CACLA, ktorá sa ukázala ako výborne riešenie pre tento problém. Táto modifikácia dosahovala až 100% úspešnosť pri dosť náročných a zložitých výpadoch, ktoré agent nepoznal.

Kľúčové slová

Neurónové siete, učenie s posilňovaním, šermovanie, CACLA

Bibliography identification

BLANÁŘ, Jaroslav. Fencing simulation in 3D using artificial intelligence methods [Master thesis].

Comenius University in Bratislava. Faculty of Mathematics, Physics and Informatics; Department of Applied Informatics.

Supervisor: doc. Ing. Igor Farkaš, PhD. Bratislava: FMFI UK, 2011. 56 p.

Abstract

This work deals with fencing simulation in 3D environment, where the main problem is agent's ability to learn defensive maneuvers to defend against trainer's lunge. Complexity of the problem is increased by the fact that coach's sword is not a static but a dynamic entity. The agent operates in an environment with continuous states and actions. For this purpose, we use reinforcement learning algorithm using CACLA (Continuous actor critic learning automaton). The original version of the algorithm used to train feed-forward neural networks did not lead to effective solutions. Therefore, we modified the CACLA algorithm. The modified algorithm proved to be an excellent solution for this problem. This modification was up to 100% successful in the task of defending against complex lunges that agent did not know.

Keywords

Neural networks, reinforcement learning, fencing, CACLA

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

1.1 Cieľ práce

Hlavným cieľom práce je navrhnúť a implementovať adaptívneho agenta s využitím umelej inteligencie pre simuláciu šermovania. Pre tento účel je potrebné implementovať simulátor šermovania v už existujúcom virtuálnom prostredí alebo navrhnúť a implementovať vlastné virtuálne prostredie pre simulátor v zjednodušenej podobe. Toto virtuálne prostredie bude trojdimenzionálne (3D). Cieľom je vytvoriť dva jednoduché modely virtuálnych humanoidných agentov, ktoré by dokázali vierohodne napodobňovať pohyby pri šermovaní. Prvý robot bude tréner, ktorý automaticky simuluje rôzne útočne výpady na agenta. Druhý robot bude agent, ktorý bude riadený umelou neurónovou sieťou. Jej úlohou bude naučiť sa obranné manévre a úspešne tak odrážať trénerove výpady.

1.2 Motivácia práce

Mojou hlavnou motiváciou tohto experimentu sú počítačové hry, ktoré často poskytujú hráčom nie veľmi sofistikovanú umelú inteligenciu. Keď hráč odhalí

spôsob (algoritmus), akým umelá inteligencia funguje v hre, stáva sa preňho hra nudnou. Umelá inteligencia v hre sa nedokáže adaptovať, prispôsobovať hráčovi, učiť sa od hráča novým stratégiám, či vytvárať si hybridné stratégie. Počítačové hry predstavujú bohatú oblasť pre experimenty s umelou inteligenciou, ktorá disponuje sofistikovanými virtuálnymi prostrediami, ktoré sú schopné poskytovať vstupné dáta pre neurónové siete v reálnom čase. Ako prvý krok som si preto zvolil vytvoriť umelú inteligenciu, ktorá by sa dokázala naučiť obranné manévre v spojitom prostredí akcií a stavov.

1.3 Štruktúra práce

Práca pozostáva z piatich kapitol. Prvá kapitola je úvod do práce, v ktorej sú opísané ciele a motivácia tejto práce.

Druhá kapitola je venovaná teoretickej časti. Zameriava sa hlavne na problematiku učenia s posilňovaním a prechodom z diskrétneho prostredia stavov a akcií do spojitého prostredia stavov a akcií.

Tretia kapitola sa zaoberá implementáciou konkrétneho riešenia simulátora šermovania a implementáciou konkrétneho algoritmu učenia sa pre danú problematiku v spojitých priestoroch.

Štvrtá kapitola je venovaná dosiahnutým výsledkom a experimentom simulácií s ohľadom na zadanie práce.

Posledná kapitola obsahuje záver práce, zhrnutie dosiahnutých výsledkov a či sa podarilo naplniť ciele práce.

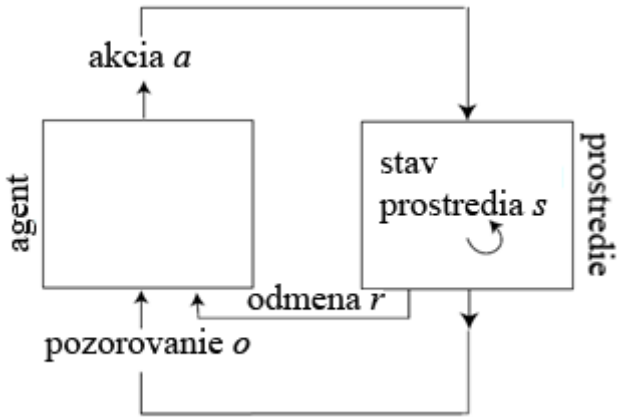
Kapitola 2

Učenie s posilňovaním

Táto kapitola poskytne prehľad o všeobecných pojmoch a metódach učenia s posilňovaním (*reinforcement learning*).

2.1 Úvod

Učenie s posilňovaním korešponduje so širokou skupinou tried strojového učenia, ktoré umožňujú agentovi naučiť sa správať v prostredí, založenom na základe skalárnej odmeny signálu. *Agent* interaguje s *prostredím* (obrázok 2.1). V teórii riadenia sa agent nazýva *regulátor* a prostredie *plán*. Agent vykonáva *akcie a*, ktoré môžu zmeniť *stav s* prostredia, napríklad: pozícia robota alebo orientácia robotického ramena, agent spracováva *poznatky o* z prostredia, ktoré mu poskytujú informácie o aktuálnom stave. Správanie agenta je dané jeho *stratégiou* π, na základe ktorej sa rozhoduje v rôznych situáciách. Cieľom učenia s posilňovaním je naučiť stratégiu založenú na špecifickej spätnej väzbe od prostredia k agentovi, nazývanú *odmena r*. Odmeny sú skalárne hodnoty, ktoré ukazujú, aká je dobrá konkrétna situácia. Potom cieľom je naučiť stratégiu, ktorá maximalizuje mieru krátkodobých alebo aj dlhodobých odmien (Sutton a Barto, 2004).



Obr. 2.1 Schematická reprezentácia učenia s posilňovaním.

2.1.1 Vzťah s učením ľudí a zvierat

Tento základný obraz o učení na základe odmien vyplynul z odboru psychológie, nazývanej *behaviorizmus.* Štúdie behavioristickej psychológie na základe odmien u zvierat a ľudí pochádzajú zo začiatku 20. storočia, zvyčajne pod názvom „Operatívne podmieňovanie“ (Skinner, 1938/ 1991). Vytváraním väzby medzi správaním zvieraťa a nasledovným uspokojením alebo nepohodlím zvieraťa, vedie k ovplyvneniu pravdepodobnosti opakovania jeho správania. Väčšia spokojnosť alebo nepohodlie vedie k väčšiemu posilneniu, alebo oslabeniu väzby správania zvieraťa. Operatívne podmieňovanie sa ukázalo ako veľmi dobrá a silná metóda spôsobu učenia, vďaka ktorej sa zvieratá dokázali naučiť prekvapivo zložité úlohy, napríklad holuby sa naučili hrať stolný tenis. Dôležitou súčasťou učenia s posilňovaním pri zvieratách je ich prirodzené prostredie, ktoré poskytuje prirodzenú odmenu, napríklad jedlo, sex, rovnako prirodzený trest, napríklad bolesť.

2.1.2 Učenie s učiteľom a učenie bez učiteľa

Pri učení s posilňovaním musíme rozlišovať dva základné prístupy k učeniu založené na kvalite trénovaných informácií, poskytovaných prostredím agenta (Hinton, 1987; Gullapalli, 1991).

Učenie s učiteľom (*supervised learning*) - prostredie povie agentovi presne, aký výstup by mal byť pre každý jeho vstup. Agent sa musí naučiť mapovať vstupy na výstupy na základe obmedzeného počtu vstupno-výstupných príkladov, a túto stratégiu správne zovšeobecniť na nových príkladoch.

V úlohách *učenia bez učiteľa* (*unsupervised learning*) prostredie predstavuje len vstupy do siete bez toho, aby poskytovalo akékoľvek informácie o výstupe. Učenie pozostáva zo samostatného zhlukovania vstupov na základe nejakej miery podobnosti medzi vstupmi.

2.1.3 Explorácia verzus exploatácia

Agent je odmeňovaný za „dobrú“ sekvenciu akcií. Avšak, nie je povedané, aké akcie sú správne; to sa musí agent naučiť metódou pokusov a omylov. Proces otestovania akcií s cieľom zistiť, aké sú dobré, sa nazýva explorácia (*exploration*). Keď agent nájde dobré akcie, môže to využiť namiesto naučenej stratégie. Agent čelí dileme „explorácia/exploatácia“ (*exploration/exploitation dilemma*). Ak pozná dobrú stratégiu, má sa snažiť hľadať ďalšie stratégie? Na jednej strane môže nájsť lepšie stratégie, ale na druhej strane viac prieskumu obyčajne znamená obetovanie súčasnej dobrej stratégie, bez záruky nájdenia lepšej.

2.1.4 On-line verzus off-line učenie

Je užitočné vedieť rozlišovať medzi on-line a off-line učením s posilňovaním, pretože tieto metódy zodpovedajú rozličným cieľom učenia s posilňovaním (Wyatt, 1995). Cieľom on-line učenia je vybudovať učenie robota, aby dokázal

vykonávať nejaké úlohy aj v neznámych prostrediach. Hodnotou tohto prístupu je možnosť použiť iba jeden typ robota v rôznych úlohách a v rôznych prostrediach a potom ho použiť v neznámom prostredí, napríklad na inej planéte. Na druhej strane v off-line učení s posilňovaním je cieľom vyvinúť dobrý riadiaci mechanizmus pre niektoré úlohy, no nie je cieľom učenie samotného robota. Chceme však vytvoriť robota, ktorý môže vykonávať určité úlohy a učenie je len prostriedok k dosiahnutiu tohto cieľa.

2.2 Formalizácia učenia s posilňovaním

Pomocou učenia s posilňovaním môžeme riešiť problémy, ktoré môžu byť modelované ako Markovovské rozhodovacie problémy (*Markov Decision Problems*) (Sutton a Barto, 1998). Kde prostredie má Markovovskú vlastnosť, že reakcie prostredia v čase $t + 1$ sú závislé len od udalostí v čase *t*. Markovovské rozhodovacie problémy sú usporiadaná štvorica $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$, kde:

- $\mathcal{S}$ je konečná množina stavov, $s_t \in S$ je stav agenta v čase *t*.
- $\mathcal{A}$ je konečná množina akcií, $a_t \in A$ je akcia agenta vykonaná v čase *t*.
- $\mathcal{R}: S \times A \times S \rightarrow \mathbb{R}$ je funkcia ohodnocujúca správanie agenta (funkcia odmeny a trestu), $\mathrm{R}(s_t, a_t, s_{t+1})$.
- $\mathcal{P}: S \times A \times S \rightarrow [0, 1]$ je funkcia prechodu, kde $\mathrm{P}(s_t, a_t, s_{t+1})$ určuje pravdepodobnosť prechodu do stavu s_{t+1} pri vykonaní akcie a_t v stave s_t.

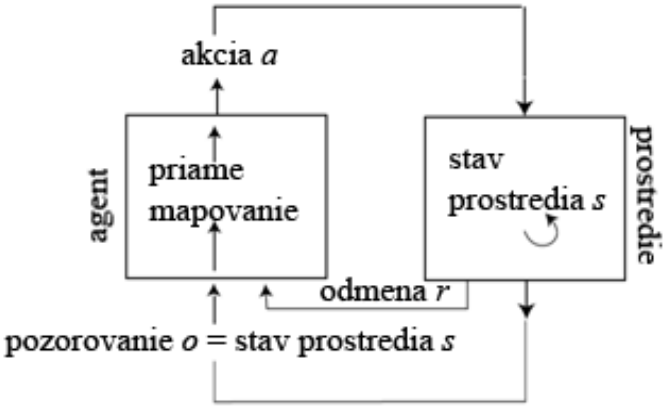
Agent sa v čase t nachádza v stave $s_t \in S$, na základe svojej aktuálnej naučenej stratégie $\pi : S \times A \rightarrow \langle 0, 1 \rangle$ sa snaží zvoliť najvhodnejšiu akciu $a_t \in A$, ktorou by maximalizoval svoje odmeny v ďalších krokoch. Odmeny agenta, ktoré sú ďalej v budúcnosti, sú menej hodnotné (diskontované), preto aby sa agent snažil čo najrýchlejšie maximalizovať svoje odmeny. Táto intuícia bola vynájdená v ekonomickej teórii; rovnaká odmena je hodnotnejšia viac, ak je k dispozícii okamžite, ako keď je k dispozícii až po dlhom čase. Odmena v budúcnosti je:

$$r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{1.1}$$

kde $\gamma \in \langle 0, 1 \rangle$ je diskontný faktor.

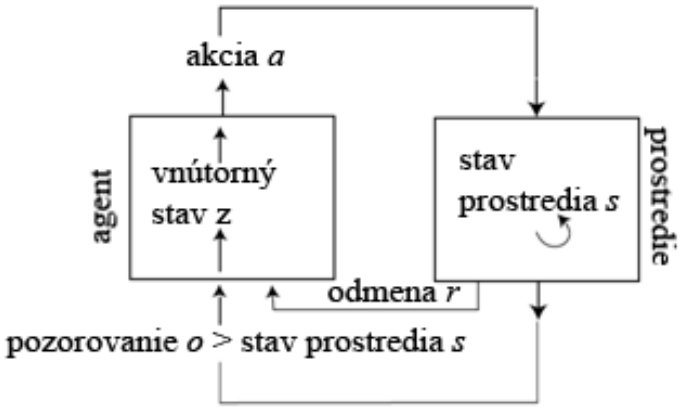
2.2.1 MDP verzus POMDP

Markovov rozhodovací proces (*Markov Decision Process* - MDP) a jeho schematická reprezentácia (Obrázok 2.2), v ktorej pozorovanie agenta je ekvivalentné stavu prostredia, čo môže viesť k jednoduchému učeniu agenta, priamym mapovaním pozorovaní (stavy prostredia) na akcie, zamerané na dosiahnutie optimálnej stratégie.



Obr. 2.2 Schematická reprezentácia MDP.

Čiastočne pozorovateľný Markovovský rozhodovací proces (*Partially Observable Markov Decision Process* - POMDP) a jeho schematická reprezentácia (Obrázok 2.3), v ktorej pozorovanie poskytuje len niektoré informácie o stave prostredia, nie úplnú informáciu ako pri MDP, čo indikuje symbol "$<$". Pre optimálny výkon musí agent používať nejakú formu vnútorného stavu.



Obr. 2.3 Schematická reprezentácia POMDP.

Problém pri POMDP je to, že podobné stavy majú podobné rozdelenie pravdepodobností na pozorovanie. Preto rôzne stavy môžu vyzerať rovnako pre agenta a rovnaké stavy môžu vyzerať inak v rôznych časoch. Z tohto dôvodu by POMDP mal mať skryté stavy alebo percepčný aliasing. POMDP môže byť nazývaný aj *Skrytý Markovov model* (*Hidden Markov Model* - HMM).

2.3 Metódy riešenia

2.3.1 Metódy učenia s modelom alebo bez modelu prostredia

Metódy riešenia pre MDP a POMDP sa odlišujú v tom, či používajú alebo nepoužívajú model prostredia. Model prostredia znamená explicitná reprezentácia stavov prostredia funkcie prechodu (prostredia funkcie prechodu stavov), pozorovacia funkcia (*observation function*) a funkcia odmeny (*reward function*). Nepriame učenie s posilňovaním (*model-based*) používa model ako základ pre hľadanie dobrej stratégie. Priame učenie s posilňovaním (*model-free*), naopak, nepoužíva model, ale namiesto toho učí priamo na základe on-line interakcie s prostredím. Výhodou nepriamych modelov je, že môžu byť použité priamo na riadenie. Nepriamemu modelu je daná „vysoká odmena“ ako vstup, na ktorú

reaguje akciu ako výstup. Týmdosiahne vysokú odmenu, pretože táto akcia je v tomto prostredí spojená s vysokou odmenou; toto funguje len v prípade priamej odmeny. Priame modely sú všeobecnejšie, môžu byť použité v prípade oneskorenej odmeny. V mnohých prípadoch je možné prejsť od učenia bez modelu prostredia k učeniu s modelom prostredia. Najprv sa pomocou učenia, bez modelu prostredia, učí model pomocou interakcie s prostredím. Keď je naučený správny model, môžu byť naň použité metódy učenia s modelom.

2.3.2 Priame hľadanie stratégie verzus ohodnocovacia funkcia

Ďalší rozmer, v ktorom môžeme metódy riešenia odlíšiť, je či používajú priame hľadanie stratégie (*direct policy search*) alebo ohodnocovaciu funkciu (*value function*). V jednoduchosti, priame hľadanie stratégie je hľadanie priamo v priestore stratégií, s cieľom nájsť takú, ktorá pracuje dobre. To znamená, že stratégia je hodnotená ako celok s nastavením parametrov, ktoré definujú stratégiu na základe týchto globálnych hodnotení. Odlišným prístupom je ohodnocovacia funkcia, ktorej základom je myšlienka spojiť jednotlivé stavy alebo stav a akciu s hodnotami. Tieto hodnoty sú obvykle aktualizované na základe priamej odmeny a hodnoty nasledujúceho stavu alebo stavu a akcie. Stratégie sú odvodené viac-menej nepriamo. Ohodnocovacia funkcia určuje, ktorá akcia má alebo vedie k najvyššej hodnote počas učenia.

2.3.3 Ohodnocovacia funkcia

Pri použití ohodnocovacej funkcie sa agent počas učenia optimálnej stratégie riadi hodnotami spojenými s jednotlivými stavmi alebo dvojicami (akcia, stav). Hodnoty spojené s jednotlivými stavmi alebo dvojicami (akcia, stav) predstavujú očakávaný diskontovaný výnos z tohto daného stavu alebo dvojice, za predpokladu, že agent nasleduje danú stratégiu π . $V_\pi(s)$ označujeme

ohodnotenie konkrétneho stavu s pri danej stratégii π. Podobne, pri ohodnocovaní akcií vzhľadom na stavy označujeme $Q_\pi(s,a)$ pre každú dvojicu. Označme π^* ako optimálnu stratégiu agenta, potom ohodnocovanie stavov je definované nasledovnou rovnicou optimality (Bellman, 1957):

$$V^*(s) = \max_a \sum_{s'} P(s,a,s')\big(R(s,a,s') + \gamma V^*(s')\big) \tag{2.2}$$

a pre ohodnotenie akcií vzhľadom na stavy platí:

$$Q^*(s,a) = \max_a \sum_{s'} P(s,a,s')\left(R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right) \tag{2.3}$$

2.4 MDP metódy riešenia

2.4.1 Q-učenie a Sarsa

Pravdepodobne najpoužívanejší algoritmus, učiaci sa pomocou ohodnocovacej funkcie bez použitia modelu, je Q-učenie (*Q-learning*). Základná idea je postupný odhad Q-hodnôt párov stav-akcia založený na odmene z prostredia a agentovom vlastnom odhade Q-hodnoty. Aktualizačné pravidlo pre Q-učenie bude:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q_t(s_{t+1},a_t) - Q_t(s_t,a_t)\right] \tag{2.4}$$

kde α je rýchlosť učenia. Q-hodnota dvojice stav-akcia, ktorú agent práve „navštívil“ v prostredí, je aktualizovaná na základe priamej odmeny a maximálnej dosiahnuteľnej Q-hodnoty v novom stave. Rozdiel medzi hranatými zátvorkami je počítaný na základe dvoch po sebe idúcich časových krokoch, preto sa nazýva časný rozdiel chyby (*temporal difference error* – TD-chyba) (Sutton, 1988). Q-učenie sa snaží naučiť optimálnu stratégiu π^* nasledovaním stratégie, ktorá zatiaľ nie je optimálna. To sa nazýva off-line stratégia učenia. On-line stratégia učenia je, že stratégia, ktorú sa agent snaží naučiť, je rovnaká ako tá, ktorú nasleduje. Príkladom on-line stratégie učenia je Sarsa algoritmus (Rummery a Niranjan, 1994), ktorého aktualizačné pravidlo je (Sutton a Barto, 1998):

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) \; + \; \alpha[r_{t+1} + \; \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)] \qquad (2.5)$$

Zvyčajne sa zdá, že má väčší zmysel učenie sa hodnoty stavu založeného na základe odhadovaní najlepšej akcie, ktorú agent môže vykonať v ďalšom stave (Q-učenie), namiesto akcie, ktorú agent aktuálne vykonáva, čo môže byť dobrý prieskum akcie, ale ďaleko od najlepšej akcie (Sarsa).

2.4.2 Učiace metódy aktér-kritik

Táto architektúra má oddelenú štruktúru (Obrázok 2.4): Jedna časť systému – *kritik* – reprezentuje hodnoty stavov ohodnocovacej funkcie $V(s)$. Ďalšou časťou je *aktér*, ktorý sa učí stratégiu $\pi(s)$ na základe signálov z *kritika.* Obaja *kritik* a *aktér* sa učia na základe časového rozdielu chýb (TD-chyba) (Sutton, 1988). TD pravidlo učenia *kritika*:

$$V_{t+1}(s_t) = V_t(s_t) \; + \; \alpha[r_t + \; \gamma V_t(s_{t+1}) - V_t(s_t)] \qquad (2.6)$$

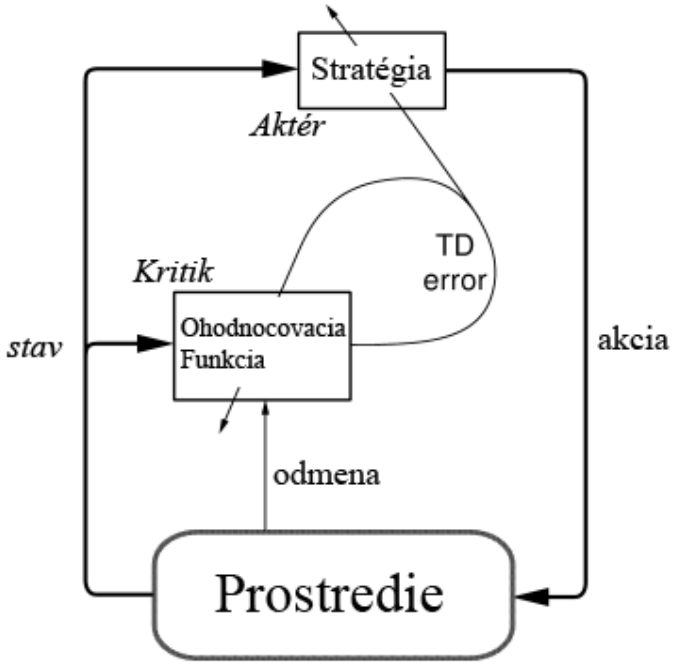
Po každej vykonanej akcii *kritik* zhodnotí nový stav a určí, aká vhodná bola daná akcia, výpočet TD chyby:

$$\delta = \; r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \qquad (2.7)$$

kde V je aktuálna ohodnocovacia funkcia vykonaná *kritikom.* TD chyba je použitá na ohodnotenie zvolenej akcie a_t v stave s_t. Ak je chyba pozitívna, potom pravdepodobnosť výberu tejto akcie by mala byť posilnená pre budúcnosť, naopak, ak je negatívna, potom pravdepodobnosť výberu by mala byť oslabená pre budúcnosť. TD pravdepodobnostná funkcia *aktéra*:

$$P_{t+1}(s_t, a_t) = P_t(s_t, a_t) \; + \; \beta\delta_t \qquad (2.8)$$

kde α, β sú parametre rýchlosti učenia.



Obr. 2.4 Typická aktér-kritik architektúra (Sutton a Barto, 2004).

2.5 Spojitý priestor a generalizácia

Vo väčšine realistických úloh existuje veľa stavov, a tiež veľa možných akcií. Stavy a akcie môžu byť kontinuálne; agent v takýchto prípadoch nemôže získať dostatočnú skúsenosť o každom stave a akcii v obmedzenom čase. Chceme, aby agent zovšeobecňoval, využíval skúsenosti z obmedzeného počtu skúseností stavov a akcií v nových stavoch a akciách. Vo väčšine sa generalizácia scvrkáva na predpokladanie, že stavy, ktoré vyzerajú podobne z hľadiska pozorovaní, máju podobné hodnoty. Rovnako aj generalizácia nad akciami; podobné akcie majú podobný výsledný efekt.

Stavové agregačné metódy rozkladajú stavový priestor do *diskrétnych* oblastí, obvykle s hranicami kolmými na pozorované rozmery. Každá diskrétna oblasť korešponduje so separátnym diskrétnym stavom. Z tohto dôvodu stavové agregačné metódy nemôžu naučiť hladkú funkciu cez stavový priestor, zatiaľ čo v mnohých prípadoch je hladká funkcia vhodnejšia. Prístup, ktorý môže naučiť spojité funkcie v stavovom priestore, používa nejaký druh parametrizovaného

aproximátoru funkcie k tomu, aby aproximoval stratégiu alebo ohodnocovaciu funkciu. Zvyčajne sú parametre aproximátora funkcie w_i učené pomocou gradientovej metódy. Vstupom aproximátora funkcie je vektor pozorovaní a na výstupe akcie danej stratégie. Aproximátor funkcie bude zovšeobecňovať do nového stavu, ale neexistuje žiadna záruka, že tento predpoklad je správny, že podobné stavy majú podobné hodnoty alebo by mali viesť k rovnakým akciám. Okrem toho, aproximátor funkcie môže niekedy zovšeobecňovať nepredvídateľné cesty a silnejšie a komplexnejšie môže uviaznuť v lokálnom optime s učením gradientovou meódov. Z týchto dôvodov je malá záruka nájdenia optimálnej stratégie alebo optimálnej ohodnocovacej funkcie cez celý stavový priestor. Napriek tomu, satisfakcia výkonu môže byť dosiahnutá v prípadoch, kedy neprichádza exaktné riešenie do úvahy.

V prípade Q-učenia s aproximátorom funkcie a trénovania pomocou gradientového skoku, aktualizačné pravidlo pre každý parameter w_i aproximátora funkcie je:

$$w_{i,t+1} = w_{i,t} + \alpha \left[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right] \frac{\partial Q_t(s_t, a_t)}{\partial w_{i,t}} \qquad (2.9)$$

kde α je rýchlosť učenia. Ak aproximátor funkcie má naučené dostatočne blízke hodnoty k optimálnym hodnotám $Q^*(s, a)$, môžeme pomocou neho jednoducho vypočítať hodnotu $Q(s, a)$ pre každú z možných akcií v aktuálnom stave a vybrať akciu, ktorá má najvyššiu $Q(s, a)$ hodnotu.

Boli použité jednoduché lineárne aproximátory funkcie, kde hodnota funkcie je váhový súčet nezávislých pozorovaných vlastností, z ktorých všetky môžu byť buď „on“ alebo „off“ (Sutton a Barto, 1998). V závislosti na zložitosti funkcie spojenej s algoritmom učenia s posilňovaním, lineárna funkcia prináša určité limity, ktoré funkcia dokáže zastúpiť. Z tohto dôvodu sa mnohí autori rozhodli použiť zložitejšie aproximátory funkcie, ako napríklad viacvrstvové dopredné neurónové siete použité na reprezentáciu Q-učenia (Lin, 1992; Anderson, 1993; Crites a Barto, 1996; Humphrys, 1997; Abul, Polat, a Alhaji, 2000; ten Hagen a Krose, 1998), SARSA (Rummery a Niranjan, 1994) a ďalšie.

2.6 Algoritmus CACLA

Algoritmus CACLA (*Continuous actor critic learning automaton*) (van Hasselt a Wiering, 2007) je rozšírením algoritmu ACLA (*Actor Critic Learning Automaton*) učenia s posilňovaním a zavedením do spojitého prostredia stavov a akcií. CACLA nepoužíva model prostredia, ale učí agenta priamo na základe on-line interakcie s prostredím.

Keď je stavový priestor spojitý, používame aproximátory funkcie (*function approximators* - FA), ktoré sa používajú na ukladanie hodnôt stavov prostredia a akcií. *Kritika* a *aktéra* nahradíme spojitými funkciami. *Kritika* nahradíme funkciou $V: s \rightarrow R$, ktorá pre aktuálny stav s_t vypočíta ohodnotenie $V_t(s_t)$. *Aktéra* nahradíme tiež spojitou funkciou $A: s \rightarrow R^n$, ktorá pre aktuálny stav vygeneruje najvhodnejšiu akciu $A_t(s_t)$. Ako parametre aproximátorov funkcie A berieme vektor parametrov θ^A a pre funkciu V vektor parametrov θ^V.

2.6.1 Učiace pravidlo

Aktualizácia sa vykonáva na parametroch FA. Nech θ^V je vektor parametrov kritika, aktualizačné pravidlo pri učení kritika na jeho váhach neurónovej siete je nasledovné:

$$\theta^V_{i,t+1} = \theta^V_{i,t} + \alpha\delta_t \frac{\partial V_t(s_t)}{\partial \theta^V_{i,t}} \tag{2.10}$$

kde α je rýchlosť učenia a $\theta^V_{i,t}$ je *i*-tá zložka vektora θ^V v čase *t* a $V_t(s_t)$ je výstup z FA v čase *t* so vstupom s_t.

Aktualizačné pravidlo pri učení aktéra na jeho váhach neurónovej siete je nasledovné:

$$\text{if } \delta_t > 0: \qquad \theta^A_{i,t+1} = \theta^A_{i,t} + \alpha\big(a_t - A_t(s_t)\big)\frac{\partial A_t(s_t)}{\partial \theta^A_{i,t}} \tag{2.11}$$

kde $\theta^A_{i,t}$ je *i*-tá zložka vektora θ^A v čase *t* a $A_t(s_t)$ je výstup z FA v čase *t* so vstupom s_t. Akcia a_t je požadovaný výstup neurónovej siete, podľa ktorej

spätným šírením chyby upravujeme jej váhy. Zmena parametrov *aktéra* θ^A sa vykoná iba vtedy, ak je $\delta_t > 0$.

2.6.2 Explorácia

Prieskum okolia je rozhodujúci pri učení s posilňovaním, pretože je to jediný spôsob, ako objavovať nové a lepšie stratégie. Keď sa agent nachádza v stave $s_{t,,}$ musí vybrať najvhodnejšiu akciu a_t pre tento stav, za pomoci spojitej funkcie $A(s_t)$. Jeho voľba je modifikovaná z dôvodu lepšieho prieskumu okolia. Explorácia umožňuje agentovi objaviť nové a často aj vhodnejšie stratégie. Voľba pravidla explorácie je závislá od konkrétneho problému. Poznáme dve základné explorácie:

- Gaussovká explorácia – pravdepodobnosť výberu akcie a_t je:
  $\pi_t(s_t, a) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(a - (A_t(s_t))^2/2\sigma^2)}$
- ϵ – greedy explorácia – s pravdepodobnosťou ϵ je vybratá úplne náhodná akcia, ktorú akceptujeme ako pôvodnú. S pravdepodobnosťou (1 - ϵ) je ponechaná pôvodná akcia.

2.6.3 Algoritmus

Na začiatku sa zvolí počiatočná konfigurácia prostredia, napríklad: pozícia agenta a pozícia cieľa. V každom kroku získame nový stav prostredia s_t, potom aktér vygeneruje najvhodnejšiu akciu a_t, na ktorú sa aplikuje explorácia a dostaneme zmenenú akciu $a_t{}'$. Aplikovaním akcie $a_t{}'$ dostaneme nový stav prostredia s_{t+1}. Funkcia REWARD nám ohodnotí kvalitu prechodu medzi stavmi s_t a s_{t+1}. Kritikom ohodnotíme obidva stavy a vypočítame δ, ktorá nám určuje kladnú alebo zápornú spätnú väzbu, na základe ktorej je potom trénovaný aktér. Kritik je trénovaný v každom kroku, ale aktér je trénovaný len pri kladnej δ. Hodnoty δ posilňujú alebo oslabujú väzby kritika so stavmi.

Algoritmus 1: Trénovanie modelu CACLA (van Hasselt a Wiering, 2007)

s0 – počiatočný stav;
for t ← 0 **to** #krokov **do**
    a_t ← EXPLORATION(AC(s_t));
    s_{t+1} ← APPLY(s_t, a_t);
    r_t ← REWARD(a_t, s_t , s_{t+1});
    $\delta_t \leftarrow r_t + \gamma V_t(s_{t+1}) - V_t(s_t)$;
    $\theta^V_{i,t+1} = \theta^V_{i,t} + \alpha\delta_t \frac{\partial V_t(s_t)}{\partial \theta^V_{i,t}}$; // Back-Propagation Critic
    if $\delta_t > 0$ **then**
        $\theta^A_{i,t+1} = \theta^A_{i,t} + \alpha\big(a_t - A_t(s_t)\big) \frac{\partial A_t(s_t)}{\partial \theta^A_{i,t}}$; // Back-Propagation Actor
    end if
end for

Kapitola 3

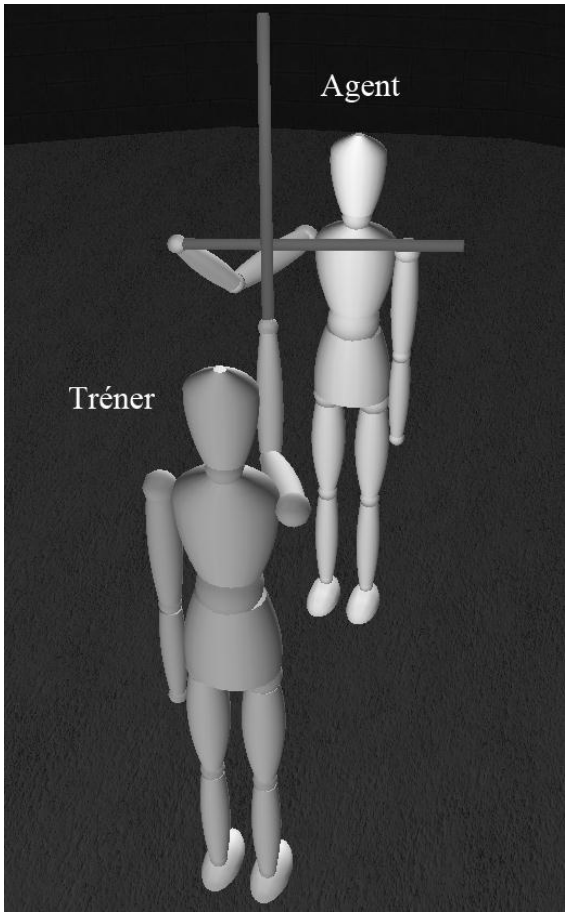
Implementácia

3.1 Voľba programovacieho jazyka

Pre svoju aplikáciu som si zvolil programovací jazyk C++, pretože podporuje objektovo orientovaný prístup. Rýchlosť jazyka C++ je značnou výhodou pre moju aplikáciu, ktorá obsahuje veľa matematických výpočtov, ako napríklad kolízie rozličných objektov v 3D scéne a hlavne výpočty s neurónovými sieťami. Aj napriek veľkej rýchlosti jazyka C++ sa snažím dostatočne optimalizovať svoju aplikáciu. Veľkou nevýhodou tohto jazyka je správa pamäte, keďže využívam smerníkové štruktúry, ktoré môžu spôsobovať veľké ťažkosti pri nesprávnej manipulácií s nimi, či ich následnom uvoľňovaní z pamäte. Na vykresľovanie 3D scény a objektov v scéne som mal možnosť zvoliť si medzi dvoma knižnicami DirectX alebo OpenGL s obidvoma mám skúsenosti, ale priklonil som sa k voľbe OpenGL, keďže som v nedávnej minulosti, v spolupráci s jedným kolegom, vytvoril 3D engine, ktorý mi jednoducho umožní implementovať môj vlastný simulátor šermovania. 3D engine obsahuje základné 3D operácie s objektmi v scéne, a taktiež plynulé pohybovanie sa v scéne.

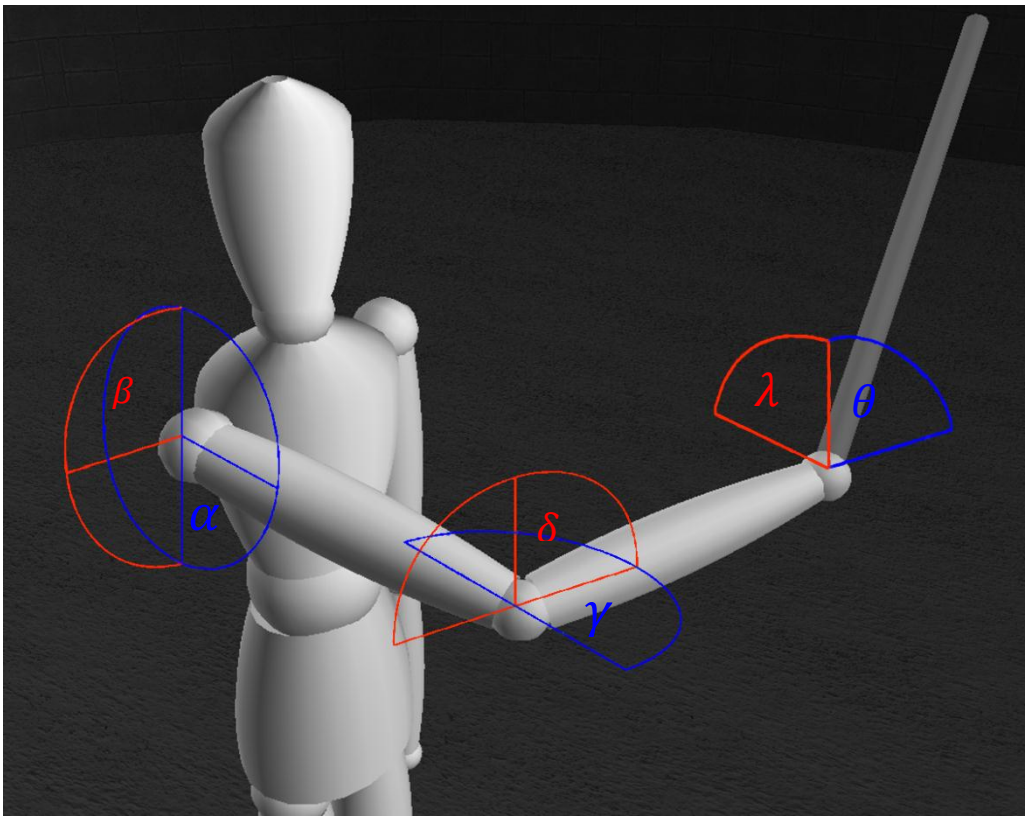
3.2 Simulátor šermovania

Jedným z cieľov tejto práce bolo navrhnúť a implementovať simulátor šermovania. Čo je simulátor šermovania? Simulátor šermovania je prostredie, v ktorom sa nachádzajú dva modely robotov, z ktorých jeden je tréner a druhý je agent (Obrázok 3.1). Tréner má predprogramované rôzne akcie v podobe výpadov. Základom simulátora sú modely robotov. Sú reprezentované základnými 3D geometrickými objektmi (valec, guľa), ktoré sú pospájane dokopy tak, aby čo najviac pripomínali jednoduchý tvar postavy robota, dve ruky, dve nohy, telo, hlava a meč. Jednotlivé geometrické objekty sú vytvárané pomocou metódy zametania (sweeping), čo nám umožňovalo manipulovať so šírkou objektov, vďaka čomu sme mohli lepšie modelovať jednotlivé krivky postavy.



Obr. 3.1 Simulátor šermovania.

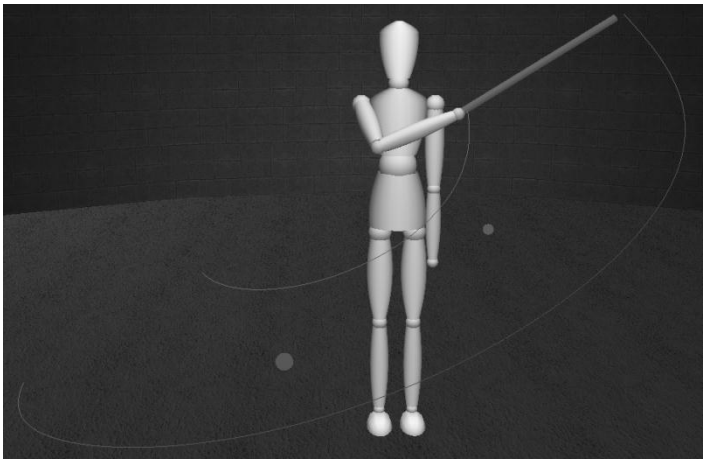
Dôležitou a zložitou časťou modelov sú kĺby (ramenný kĺb, lakťový kĺb, zápästný kĺb), vďaka ktorým dokážeme zabezpečiť lepšie a prirodzenejšie reprezentovanie pohybov rúk. Modely neobsahujú ďalšie kĺby, keďže ide o hlavne simuláciu pohybov pravej ruky, ktorá drží meč. Jednotlivé kĺby obsahujú limity, ktoré určujú, ako sa dané časti vedia ohnúť alebo vyrovnať. Ruka modelov obsahuje šesť stupňov voľnosti (Obrázok 3.2). $\alpha\langle -180°, 180°\rangle, \beta\langle -90°, 90°\rangle, \gamma\langle 0°, 145°\rangle, \delta\langle 0°, 120°\rangle, \theta\langle 0°, 60°\rangle, \lambda\langle 0°, 90°\rangle$



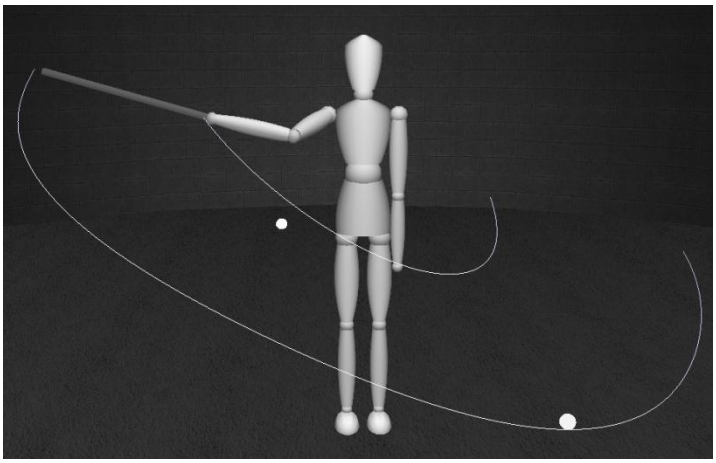
Obr. 3.2 Stupne voľnosti ruky agenta.

3.2.1 Typy výpadov

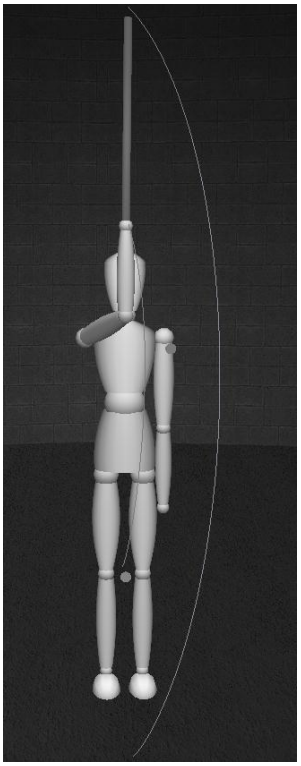
Na trénovanie používame tri základné typy výpadov zľava (Obrázok 3.3), sprava (Obrázok 3.4) a stredový (Obrázok 3.5). Agent je trénovaný celkovo na deviatich výpadoch tri ľavé, tri pravé a tri stredové, tieto sú počas testovania upravované generátorom výpadov, ktorý mení sklon a natočenie jednotlivých výpadov. Takto vzniká veľké množstvo nových a zložitejších výpadov, než ktoré pozná náš agent.



Obr. 3.3 Výpad zľava



Obr. 3.4 Výpad sprava

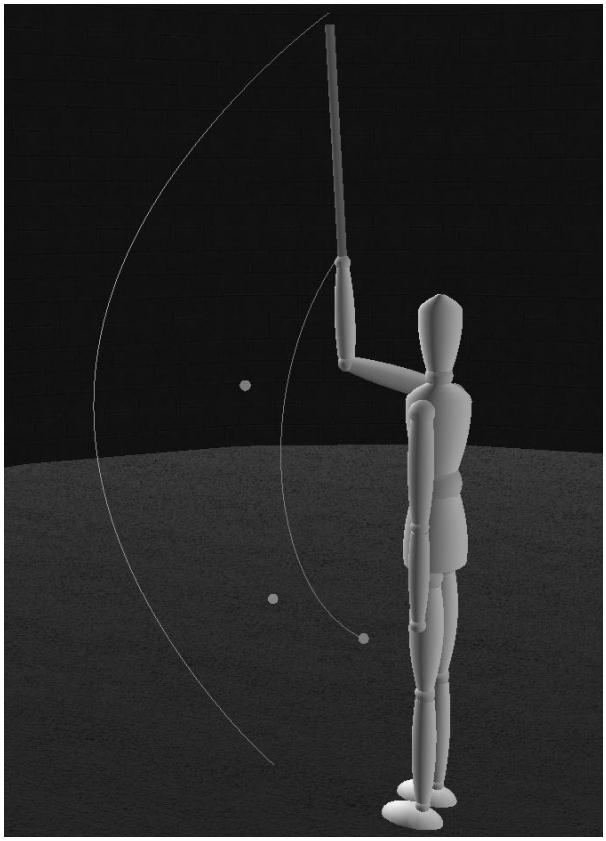


Obr. 3.5

Výpad zo stredu

3.2.2 Tréner

Na reprezentáciu výpadov trénera som sa rozhodol použiť Bézierové krivky (Obrázok 3.6) z dôvodu, že sa nimi dajú jednoducho reprezentovať rôzne typy výpadov, čo dodáva tomuto systému dostatočnú flexibilitu a možnosť neskoršieho pridania nových a komplexnejších druhov výpadov. Výpady sú určené štvoricou bodov; dva sú body krivky (počiatočný a koncový bod krivky) a dva sú jej kontrolné, ktoré riadia tvar a smer krivky – inak povedané Bézierová kubika, bližšie popísaná v ďalšej podkapitole 3.2.3. Repertoár tréner obsahuje zoznam výpadov,



Obr. 3.6 Reprezentácia výpadu Bézierovou krivkou.

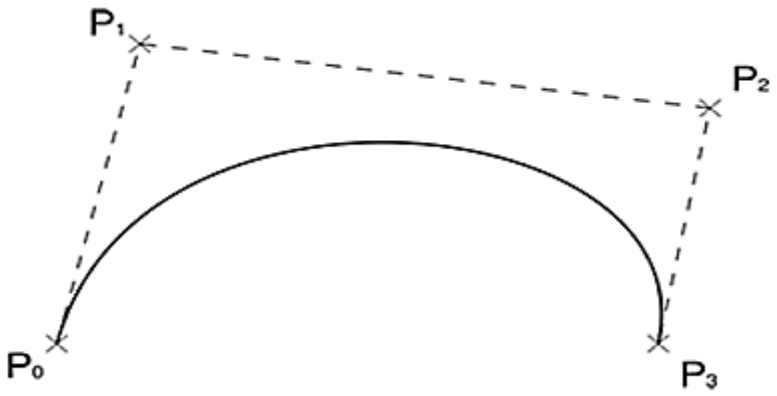
z ktorých sa počas učenia agenta vygeneruje náhodná postupnosť, ktoré sú potom simulované v danej postupnosti za sebou.

Na obrázku 3.6 môžeme vidieť tri biele body a štvrtý nie je vidno, lebo je umiestnený v pozícii zápästného kĺbu. Tieto body reprezentujú jeden konkrétny výpad.

3.2.3 Bézierová krivka

Bézierová kubika (Farin, 1997) (Obrázok 3.7) je Bézierová krivka tretieho rádu a je zadaná štyrmi bodmi P0, P1, P2 a P3. Vychádza z prvého bodu P0 a končí v poslednom bode P3. Body P1 a P2 určujú zakrivenie celej krivky. Je určená vzťahom:

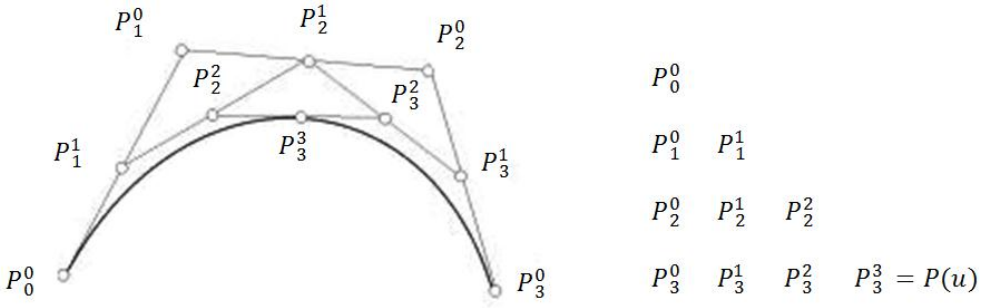
$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)^2 t^2 P_2 + t^3 P_3, t \in [0,1] \qquad (1.5)$$



Obr. 3.7 Bézierová kubika.

3.2.4 Konštrukcia Bézierovej krivky (algoritmus)

Algoritmus de Casteljau (Farin a Hansford, 2000) sa používa na výpočet bodov na Bézierovej krivke (Obrázok 3.8). Tento algoritmus vychádza z vlastnosti Bernsteinových polynómov $B(t) = \sum_{i=0}^{n} \beta_i b_{i,n}(t)$, ktorý môžeme graficky znázorniť takto:



Obr. 3.8 Vizualizácia de Casteljauho algoritmu.

V skutočnosti nejde o nič iné, ako o postupné delenie úsečiek riadiaceho polynómu v zadanom pomere (u). Počet novovzniknutých bodov na úsečkách sa v každom kroku zmenší o jeden. Posledné delenie nám zanechá jeden bod, ktorý leží na bézierovej krivke.

3.3 Hierarchia tried implementujúcich UI

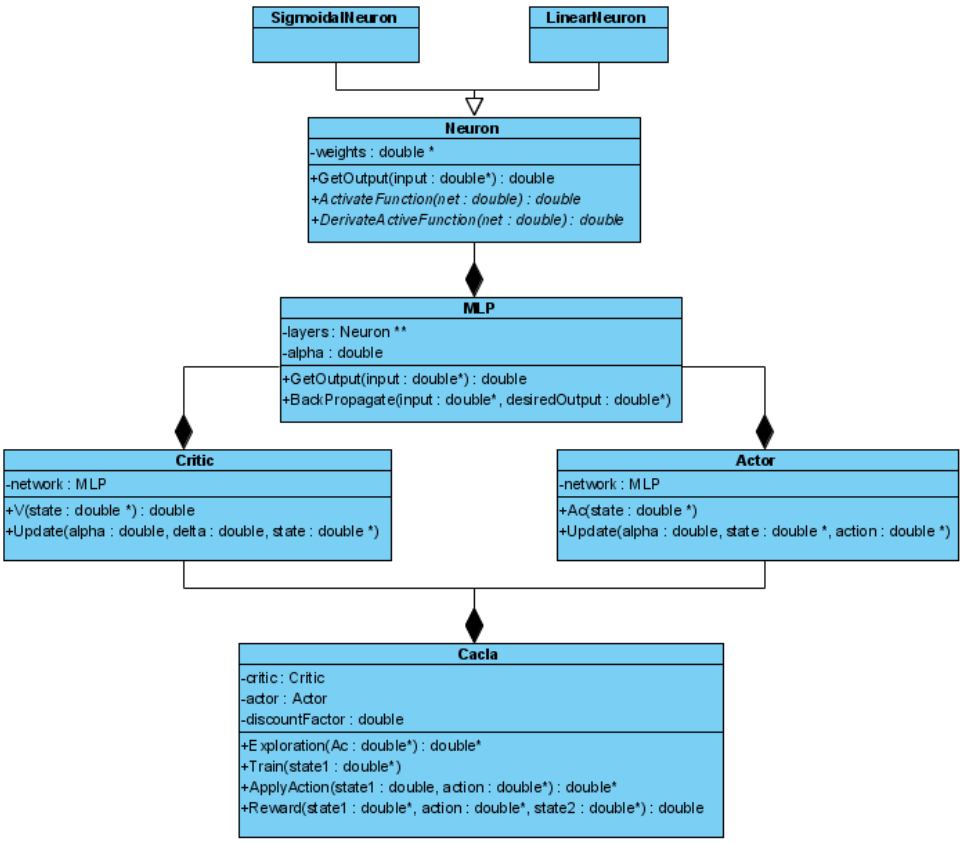
Na obrázku 3.9 je UML (Unified Modeling Language) diagram hierarchie základných tried UI z implementačnej stránky.

Trieda *Neuron* reprezentuje jeden perceptrón a jeho správanie v neurónovej sieti, od nej sú odvodené ďalšie triedy *LinearNeuron* a *SigmoidalNeuron,* ktoré reprezentujú lineárny alebo sigmoidálny typ neurónu s aktivačnou funkciou daného typu.

Trieda *MLP* reprezentuje viacvrstvovú neurónovú sieť (Multilayer perceptron), ktorá obsahuje dynamické dvojrozmerné pole neurónov a metódy ako výstup siete a metódu spätného šírenia chyby.

Triedy *Actor* a *Critic* reprezentujú správanie *aktéra* a *kritika* v algoritme CACLA, obsahujú viacvrstvovú neurónovú sieť.

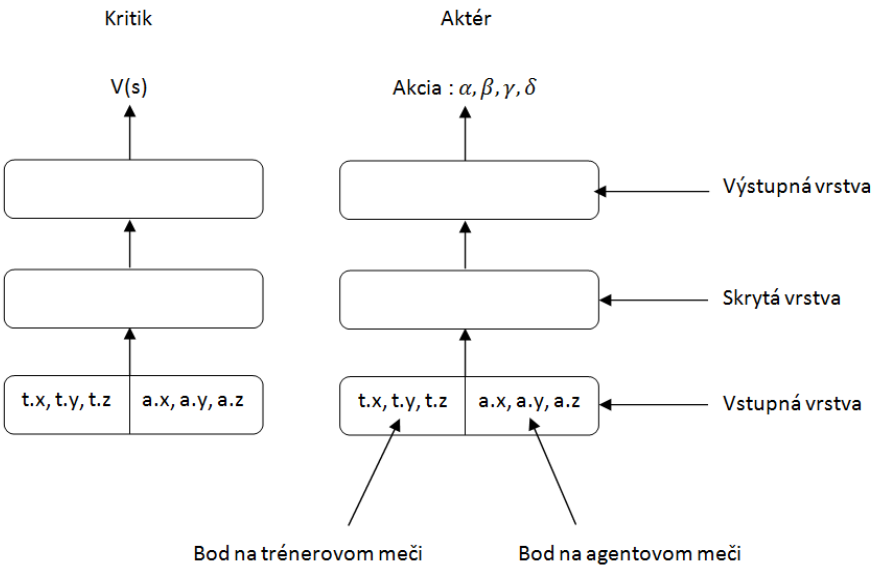
Trieda *Cacla* je implementáciou daného algoritmu CACLA, spomínaného v kapitole 2.6.



Obr. 3.9 Hierarchia základných tried UI.

3.4 Agent

Agent je v 3D virtuálnom priestore reprezentovaný jednoduchým modelom robota, ktorého pozícia v priestore je statická a pohybuje sa len jeho pravá ruka a jeho meč. Na začiatku trénovania, pred každým výpadom trénera, sa agentova ruka a jeho meč nastavia do počiatočnej pozície, ktorá bola zvolená tak, aby pozícia vyhovovala každému druhu výpadu. Následne sa náhodne zvolí jeden z možných výpadov pre trénera, na ktorý sa agent snaží vhodne voliť akcie pomocou RL algoritmu. Prostredie, v ktorom sa agent nachádza, je spojité vzhľadom na stavy a akcie, preto ako jednu z možností algoritmov učenia s posilňovaním sme zvolili algoritmus CACLA. Algoritmus CACLA je aplikovaný nad dvoma dvojvrstvovými doprednými neurónovými sieťami (aktér, kritik) (Obrázok 3.10) a ich učenie je realizované algoritmom spätného šírenia chyby.



Obr. 3.10 Model siete Agenta.

Aktér a kritik dostávajú na vstupnej vrstve siete šesť reálnych čísel. Prvé tri reálne čísla určujú pozíciu (x, y, z) bodu na trénerovom meči a druhé tri určujú pozíciu bodu na agentovom meči. Výstupom neurónovej siete aktéra $A(s)$ sú štyri reálne čísla, ktoré určujú štyri stupne voľnosti ruky agenta. Tieto hodnoty sú z intervalu $\langle 0, 1 \rangle$, pretože používame neuróny so sigmoidálnou aktivačnou funkciou. Aktér nevolí akciu Δa čiastočnej zmeny posunutia jeho ruky, ale

výstupné hodnoty sú preškálované na celé uhly v radiánoch, napríklad $\alpha\langle -\pi,\ \pi\rangle$, z ktorých sa vypočíta Δa čiastočná zmena posunutia ruky $(\texttt{nováAlpha - aktuálnaAlpha}) \rightarrow zmena$.

```
double cCacla::AngleChange(double angle){                // O aký uhol sa zmenia jednotlivé uhly.
        if (abs(angle) > M_PI_10){
                return angle > 0.0 ? M_PI_10 : - M_PI_10;   // M_PI_10 je 10° v radiánoch.
        }
        return angle;
}

double * cCacla::ModifyAction(double * state, double * action) {

        double * a = new double[sizeOutput];

 // SCALE angles

        // alpha
                a[0] = ScaleAlpha(action[0]);

        // beta
                a[1] = ScaleBeta(action[1]);

        // gamma
                a[2] = ScaleGamma(action[2]);

        // delta
                a[3] = ScaleDelta(action[3]);

        a[0] -= actualAlpha;
        a[1] -= actualBeta;
        a[2] -= actualGamma;
        a[3] -= actualDelta;

        a[0] = AngleChange(a[0]);
        a[1] = AngleChange(a[1]);
        a[2] = AngleChange(a[2]);
        a[3] = AngleChange(a[3]);

        if (print)
                MePrintf("ACTION CHANGE", a, sizeOutput);

        return  a;
}
```

Obr. 3.11 Modifikácia akcie.

Na každú akciu, ktorú v danom stave zvolil aktér, je najprv aplikovaná gaussovská explorácia, potom je následne modifikovaná, preškálovaná a nakoniec je z nej vypočítaná čiastočná zmena akcie $\Delta \mathrm{a} = [\Delta\alpha, \Delta\beta, \Delta\gamma, \Delta\delta]$. Akcia Δa je aplikovaná (Obrázok 3.12), čím dôjde k posunu pravej ruky agenta.

```
void cCacla::ApplyAction(double* action) {
        actualAlpha += action [0];                      // Uloženie si nových uhlov ramena.
        actualBeta  += action [1];
        actualGamma += action [2];
        actualDelta += action [3];

        agent->rightArm.SetAngles(actualAlpha, actualBeta, actualGamma, actualDelta);
                                                        // Nastavenie nových uhlov ramena.
        agent->Sword.A = agent->P.C;
}
```

Obr. 3.12 Aplikovanie akcie.

3.5 Funkcia odmeny

Veľmi dôležitou metódou učenia s posilňovaním je *Reward()- funkcia odmeny* (Obrázok 3.13). Táto metóda je riadená prostredím, v ktorom sa agent nachádza a vracia kladnú alebo zápornú odmenu na stav, v ktorom sa agent nachádza, inak povedané, prostredie ohodnocuje kvalitu stavov, v ktorých sa agent nachádza vzhľadom na cieľ učenia.

V našom prípade je cieľom učenia naučiť agenta správne reagovať na pohyb trénerovho meča tak, aby sa snažil agent približovať svoj meč k trénerovmu meču, a tiež čiastočne odhadovať trajektóriu trénerovho útočného výpadu.

Výpočet odmeny stavu - odmena stavu závisí od vzdialenosti mečov od seba. Ak meč v jednoduchosti reprezentujeme ako úsečku začínajúcu v bode A a končiacu v bode B jej stred leží v bode $S = \frac{A+B}{2}$, potom funkcia *DistanceSword(newPositionPoint)* zoberie stredné body oboch mečov a vypočíta ich vzdialenosť od seba *distance.* Táto vzdialenosť je predelená maximálnou vzdialenosťou mečov *maxDistance*, ktorá sa vypočíta vždy nazačiatku každého kola, kedy sú meče umiestnené do počiatočných pozícii. Hodnota *reward1* je škálovaná do intervalu $\langle 1, -1 \rangle$, kde hodnota 0 určuje počiatočný stav, kedy *maxDistance* sa rovná hodnote *distance.* Kladné hodnoty znamenajú priblíženie a záporné hodnoty oddialenie mečov. Hodnota *reward1* by nám nezabezpečila, aby sa agent snažil čo najviac posúvať svoj meč od svojho tela, preto hodnota *reward2* pôsobí na danú odmenu ako diskontovaný faktor. Čím sa viac trénerov meč vzďaľuje od stredovej osi trénera, tým zápornejšia je hodnota *reward2*, čím diskontuje výslednú odmenu daného stavu. Preto agent môže získať väčšiu odmenu práve vtedy, keď sa snaží viac explorovať do priestoru. Posledným prírastkom k celkovej odmene *miDistance* je bonus, keď vzdialenosť mečov je menšia ako 0.8 , 0.5 alebo 0.2 .

Ako vidieť z kódu, uvažujeme tri možné kombinácie výslednej odmeny, a to len *reward1* alebo *reward1 + reward2*, alebo *reward1 + reward2 + minDistance*.

```
double cCacla::Reward() {

        double minDistance = 0.0;

        minDistance = DistanceSwordsFromCenter();       // Vzdialenosť meča od stredovej osi
                                                        // trénera.
        double distance = DistanceSwords(newPositionPoint); // Vzdialenosť stredov mečov od seba.
        double reward1 = (1 - distance/ maxDistance);   // maxDistance je počiatočná vzdialenosť
                                                        // mečov.
        double reward2 = -(1 - minDistance/(maxDistanceFromCenter));

        if (distance < 0.2){                            // Bonus keď je bližšie k trénerovmu meču.
                minDistance = 0.25;
        } else
        if (distance < 0.5){
                minDistance = 0.2;
        } else
        if (distance < 0.8){
                minDistance = 0.1;
        }

        if (print){
                printf("REWARD = %f + %f + %f \n", reward1, reward2, minDistance);
        }
                                                        // Tri typy ohodnotení, s ktorými som
        //return reward1;                               // experimentoval.
          return reward1 + reward2;
        //return reward1 + reward2 + minDistance;
}
```

Obr. 3.13 Odmena prostredia.

Kapitola 4

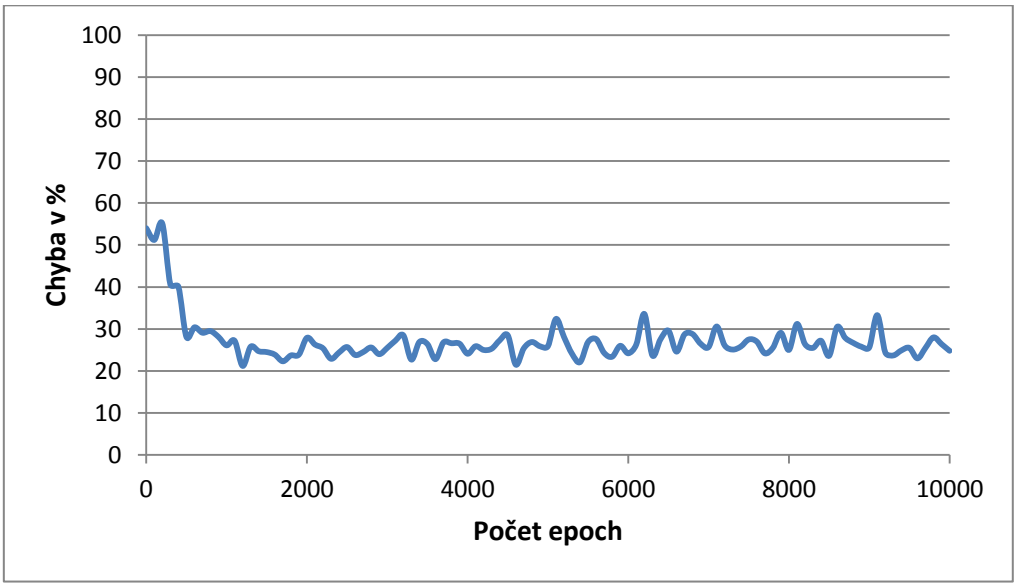
Výsledky a experimenty

4.1 Výsledky algoritmu CACLA

Pri experimentoch s algoritmom CACLA sme sa snažili natrénovať agenta na rozličných modeloch sieťach. Vyskúšali sme rôzne kombinácie modelov sieti pre aktéra a kritika s jednou skrytou vrstvou, ale aj s viacerými skrytými vrstvami. Taktiež sme vyskúšali rozličné kombinácie rýchlosti učenia pre aktéra a kritika. Podarilo sa nám pomocou tohto algoritmu natrénovať agenta, tak aby dokázal rozlišovať jednotlivé typy výpadov od seba, a tiež sa snažil voliť čo najvhodnejšie akcie, ktoré by viedli k jeho úspešnej obrane. Dlhú dobu sme si mysleli, že agent nie je schopný naučiť sa rozlišovať jednotlivé typy výpadov pomocou algoritmu CACLA, ale ukázalo sa, že je to možné, aj keď len do určitej miery. Agent sa síce naučil rozpoznávať rôzne typy výpadov, ale na jeden druh výpadu volil stále približne rovnaký obranný manéver. Ako keby sa naučil len nejaké tri postupnosti akcií, ktoré zodpovedali trom základným typom výpadov trénera.

Natrénovali sme 10 nezávislých modelov sieti s jednou skrytou vrstvou, ktorá obsahovala 10 perceptrónov so sigmoidálnou aktivačnou funkciou. Na obrázku 4.1 je znázornený priebeh priemernej chyby v percentách cez týchto 10 modelov, počas 10 tisíc epoch. Jedna epocha znamená jeden trénerov výpad. Sieť sme trénovali postupne a po každých 100 natrénovaných epochách (výpadoch) sme ju

následne otestovali na ďalších 100 náhodných epochách (výpadoch) a zaznamenali sme si počet neúspešných obranných manévrov.

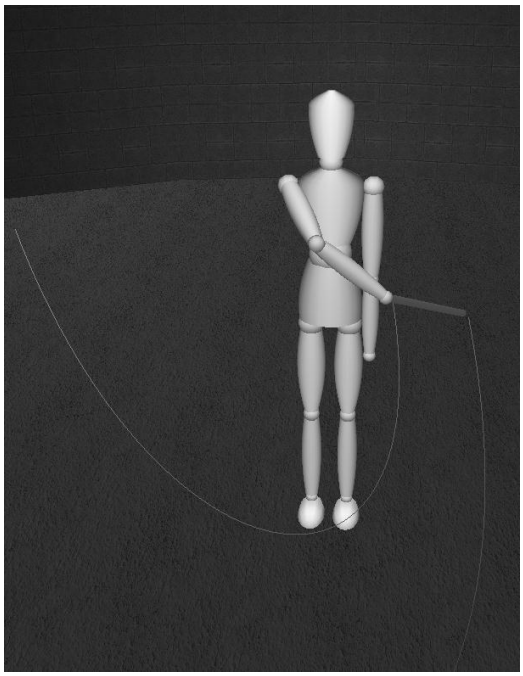


Obr. 4.1: Priebeh priemernej chyba agenta pri trénovaní algoritmom CACLA.

Aby agent dokázal zlepšiť svoju obrannú schopnosť, a tak znížil celkovú chybu pri testovaní, nutne si k tomu potrebuje osvojiť schopnosť rozlišovania jednotlivých typov výpadov. Ako vidieť z obrázku 4.1 agent sa celkom rýchlo dokázal naučiť túto schopnosť, už počas prvých 800 trénovaných epoch. Potom, ako sa agent dokázal naučiť rozlišovať jednotlivé typy výpadov od seba, začal sa zlepšovať v obrane, ale len po určitú hranicu. Agent mal tendencie zhoršovať a zlepšovať svoju obrannú stratégiu, ani počas 10 tisíc epoch sa nedokázal razantne zlepšiť. Testovacia krivka oscilovala. Testovacia chyba pri použití algoritmu CACLA nedokázala klesnúť pod hranicu 20%.

Snažili sme sa zistiť, či natrénovaný agent pomocou algoritmu CACLA dokáže schopne reagovať na nový typ výpadu - výpad zospodu (obrázok 4.2). Agent preukázal dobrú schopnosť reagovať aj na úplne nové typy výpadov. Tento výpad je dosť neštandardný pre agenta, lebo sa nepodobá na žiaden zo základných výpadov nami zvolenými (zľava, sprava, na stred). Bohužiaľ agent nedokázal vôbec schopne reagovať na výpad zospodu. Jeho reakcia na tento druh výpadu

bola dosť chaotická a obranný manéver skôr pripomínal nejakú medzikombináciu pravého a stredového obranného manévru.



Obr. 4.2: Výpad zospodu.

Keďže agent trénovaný pomocou algoritmu CACLA nenaplnil naše očakávania, rozhodli sme sa pre vlastnú modifikáciu algoritmu CACLA, ktorá by dosahovala vyššiu úspešnosť ako algoritmus CACLA, a tiež by dokázal schopne reagovať aj na výpad zospodu.

4.2 Návrh modifikácie algoritmu CACLA

Navrhli sme nasledovnú modifikáciu algoritmu CACLA (Algoritmus 2 MCACLA). Prvou modifikáciou je, že pri trénovaní neberieme do úvahy len akciu a_t, ktorú dostávame exploráciou z akcie a_{actor}, ale berieme aj samotnú akciu a_{actor}, ktorú zvolil *aktér* ako najhodnejšiu akciu v stave s_t podľa aktuálnej stratégie. Pri trénovaní využívame bonus v podobe toho, že pracujeme s dvomi stavmi $s1_{t+1}$ a $s2_{t+1}$, do ktorých sa dostávame práve pomocou týchto dvoch akcii a_t a a_{actor} prechodom zo stavu s_t. Ďalšou dôležitou modifikáciou je, že sa snažíme *kritika* učiť pomocou odmeny a trestu z prostredia r_t a nie pomocou

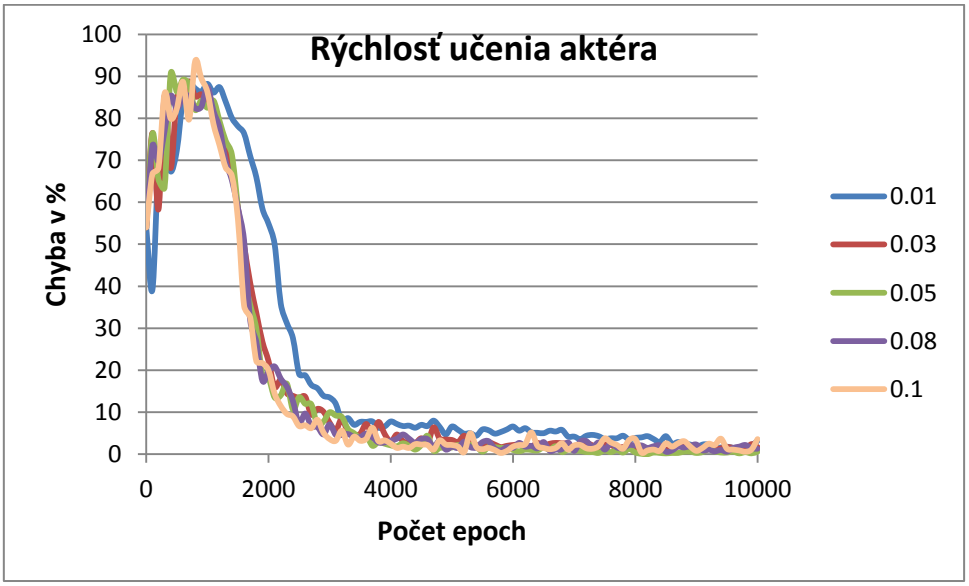
časového rozdielu chyby (TD-chyby). Chceme, aby sa *kritik* naučil čo najlepšie reprezentovať hodnoty r_t, pretože ním odohnocujeme už vypočítané nové stavy $s1_{t+1}$ a $s2_{t+1}$. Hodnoty v_1a v_2 dostaneme ohodnotením týchto stavov. Dôležitou modifikáciou prešla aj podmienka $\delta > 0$, ktorá určovala, aké akcie sa má *aktér* učiť. Pri tejto podmienke sa *aktér* učil akcie s príliš širokého pásma (viď. Kapitola 4.6). Domnievame sa preto, že práve kvôli tomuto sa *aktér* nedokázal lepšie špecializovať na rôzne typy výpadov. Túto podmienku sme upravili tak, aby sa *aktér* učil len vtedy, keď jeho akcia a_{actor} je horšia ako akcia a_t získaná exploráciou, čo vyjadruje táto podmienka $v_1 > v_2$. Touto podmienkou zaručujeme, že sa *aktér* bude snažiť zlepšovať, a tiež bude mať dostatočnú voľnosť pri explorácii. Poslednou modifikáciou tohto algoritmu je, že táto istá podmienka $v_1 > v_2$ tiež rozhoduje o tom, aká akcia bude v konečnom kroku aplikovaná v prostredí. Aplikuje sa akcia, ktorá bola vyhodnotená ako lepšia.

Algoritmus 2: Trénovanie modelu MCACLA (Jaroslav Blanář, 2010)

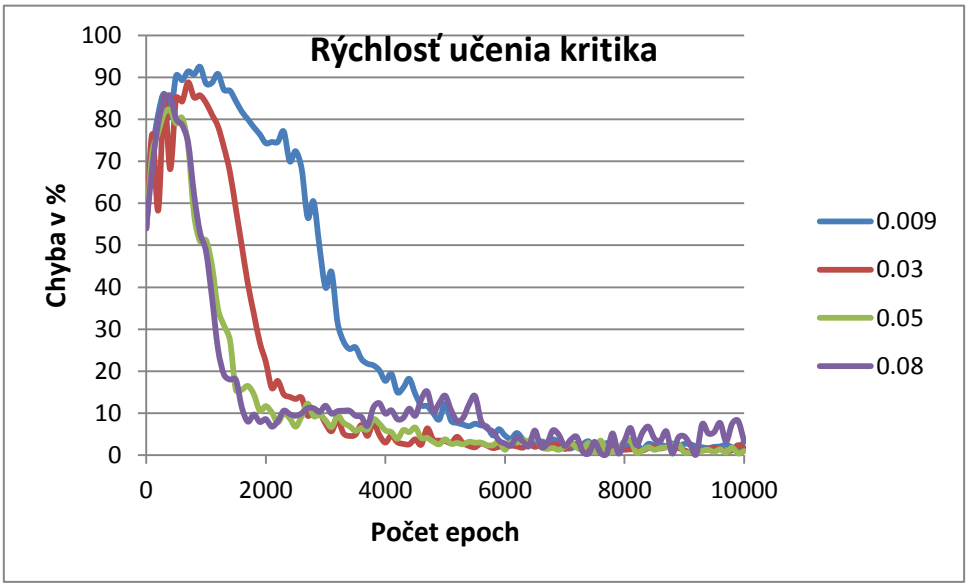
1: s0 – počiatočný stav;
2: **for** t ← 0 **to** #krokov **do**
3: $a_{actor} \leftarrow$ A(s_t);
4: $a_t \leftarrow$ EXPLORATION(a_{actor});
5: $s1_{t+1} \leftarrow$ GetState(s_t, a_t);
6: $s2_{t+1} \leftarrow$ GatState(s_t, a_{actor});
7: $r_t \leftarrow$ REWARD(a_t, s_t , $s1_{t+1}$);
8: $\delta_t \leftarrow r_t$;
9: $\theta^V_{i,t+1} = \theta^V_{i,t} + \alpha\delta_t \frac{\partial V_t(s_t)}{\partial \theta^V_{i,t}}$; // Back-Propagation Critic
10: $v_1 = V_t(s1_{t+1})$;
11: $v_2 = V_t(s2_{t+1})$;
12: **if** $v_1 > v_2$ **then**
13: $\theta^A_{i,t+1} = \theta^A_{i,t} + \alpha\big(a_t - A_t(s_t)\big)\frac{\partial A_t(s_t)}{\partial \theta^A_{i,t}}$; // Back-Propagation Actor
14: APPLY(s_t, a_t);
15: **else**
16: APPLY(s_t, a_{actor});
17: **end if**
11: **end for**

4.3 Výsledky modifikácie algoritmu CACLA

Tak ako pri algoritme CACLA sme natrénovali 10 nezávislých modelov s náhodne inicializovanými váhami z intervalu $\langle -0.1, 0.1 \rangle$. Týchto 10 modelov sme trénovali pri rôznych rýchlostiach učenia *aktéra* α, ktorú sme postupne zvyšovali $\{0.01, 0.03, 0.05, 0.08, 0.1\}$ (obrázok 4.3). Rýchlosť učenia kritika sme nemenili $\beta = 0.02$. Na obrázku 4.3 sú znázornené priebehy priemernej chyby modelov pri rôznych rýchlostiach učenia *aktéra*.



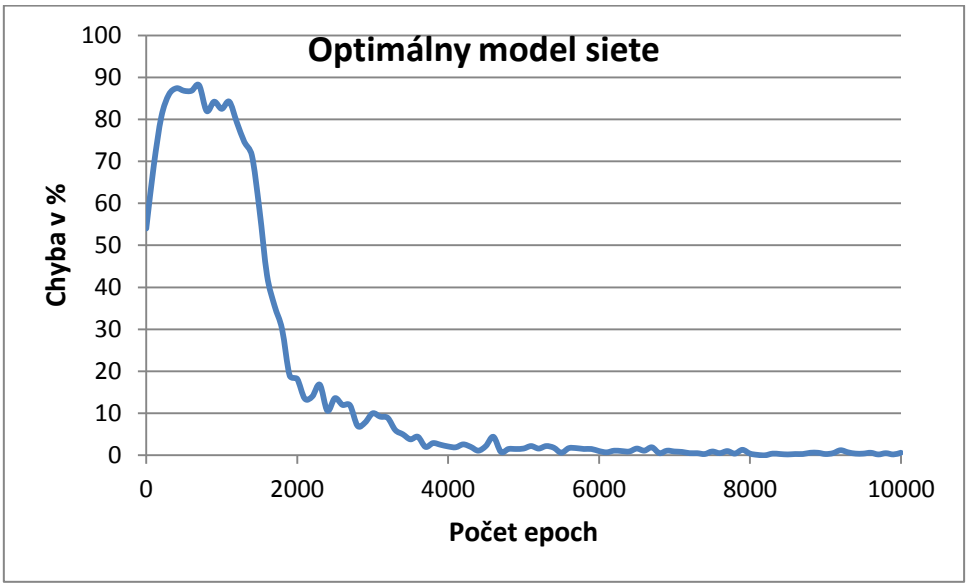
Obr. 4.3: Rýchlosť učenia aktéra.



Obr. 4.4: Rýchlosť učenia kritika.

Rovnakým spôsobom sme sa tiež snažili nájsť optimálnu rýchlosť učenia pre *kritika*. Keď rýchlosť učenia bola príliš malá, *aktér* sa učil príliš pomaly (obrázok 4.4), napríklad pri $\beta = 0.009$. Naopak, ak rýchlosť učenia bola príliš vysoká, testovacia krivka mal tendenciu oscilovať, napríklad $\beta = 0.08$.

Po detailnom preskúmaní grafov 4.3 a 4.4 sme zvolili nasledovnú výslednú konfiguráciu modelu siete, dve dopredné neurónové siete s jednou skrytou vrstvou, ktorá obsahovala 10 sigmoidálnych perceptrónov s počiatočnou rýchlosťou učenia *aktéra* $\alpha = 0.03$, ktorú znižujeme na hodnotu 0.01, keď celková chyba klesne pod 20%. Týmto znížením dosahuje, že *aktér* jemnejšie vylaďuje svoju aktuálnu stratégiu (obrázok 4.5). Rýchlosť učenia *kritika* sme zvolili $\beta = 0.02$.



Obr. 4.5: Optimálny model siete.

Agent potreboval približne 1000 epoch na osvojenie si schopnosti rozlišovania jednotlivých typov výpadov. Na obrázku 4.5 môžeme pozorovať, že potom, ako si agent osvojil túto schopnosť, jeho celková chyba exponenciálne klesala. Agent, trénovaný pomocou modifikácie algoritmu CACLA, potreboval približne 3000 epoch na to, aby dosiahol viac než 90% obrannú úspešnosť.

Snažili sme sa zistiť, či natrénovaný agent pomocou modifikácie algoritmu CACLA dokáže schopne reagovať na nový typ výpadu - výpad zospodu (obrázok 4.2). Agent preukázal dobrú schopnosť reagovať aj na úplne nové typy výpadov.

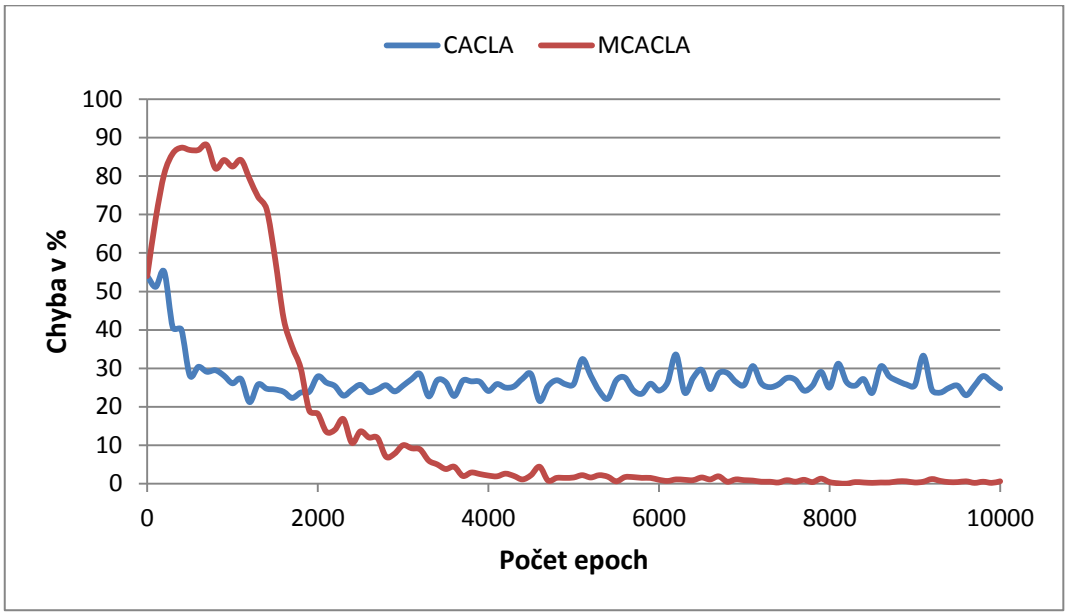
Porovnali sme obranný manéver *agenta1*, ktorý nebol trénovaný na tomto type výpadu s obranným manévrom *agenta2*, ktorého sme zvlášť natrénovali na tomto type výpadu. Obranný manéver *agenta1* sa podobal obrannému manévru *agenta2*, aj preto *agentovi1* stačil len krátky čas, aby tento svoj manéver dostatočne vylepšil.

4.4 Porovnanie výsledkov algoritmu CACLA s algoritmom MCACLA

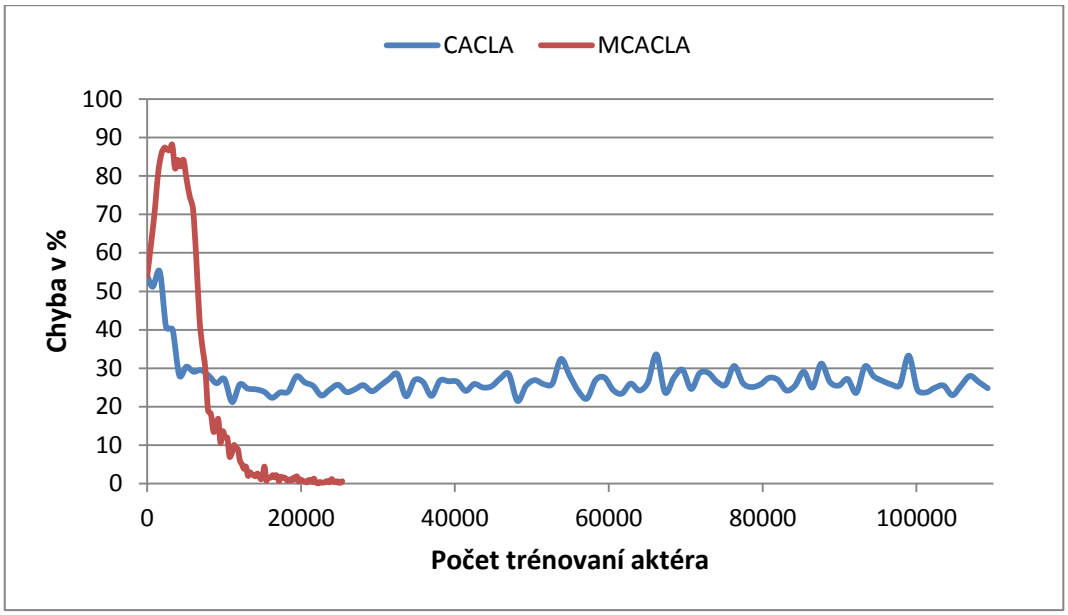
Z výsledkov jednoznačne vyplýva, že dominantou je algoritmus MCACLA. Testovacia chyba algoritmu CACLA nedokázala klesnúť pod hranicu 20%, zatiaľ čo algoritmus MCACLA sa s týmto problémom dokázal úspešne popasovať, keďže jeho testovacia chyba dokázala klesnúť až na 0%. Takže dosahuje až 100% úspešnosť. Druhým zásadným rozdielom je, že algoritmus MCACLA preukázal schopnosť úspešne reagovať aj na dosť neštandardný výpad zospodu (obrázok 4.2). Nielenže agent trénovaný pomocou algoritmu CACLA nedokázal schopne reagovať na výpad zospodu, ale ani sa pomocou tohto algoritmu nedokázal natrénovať nejaký obranný manéver, ktorý by bol schopný úspešne naň reagovať. Preto algoritmus MCACLA považujeme za úspešnú modifikáciu.

Z obrázku 4.6 sa zdá, že algoritmus CACLA si dokáže oveľa rýchlejšie osvojiť schopnosť rozlišovania jednotlivých typov výpadov, a tak aj rýchlejšie zlepšiť svoju obrannú schopnosť. Tento poznatok je bohužiaľ mylný, ak vezmeme tieto dva isté priebehy testovacích chýb na týchto dvoch algoritmoch a nedáme ich do vzťahu vzhľadom na počet trénovaných epoch, ale vzhľadom na počet trénovaní *aktéra* (obrázok 4.7). Čo je počet vyjadrujúci koľkokrát sa skutočne *aktér* učil nejakú akciu počas trénovania. Ako môžeme vidieť z grafu 4.7, obidva algoritmy si osvojili schopnosť rozlišovania jednotlivých typov výpadov za približne rovnaký počet trénovaní *aktéra.* Tento graf nám prináša nový rozmer, nielenže algoritmus MCACLA je efektívnejší, ale aj z výpočtového hľadiska oveľa rýchlejší a menej náročný. Zatiaľ čo algoritmus CACLA potreboval na 10

tisíc výpadoch 110 tisíc trénovaní *aktéra*, algoritmus MCACLA potreboval len približne 30 tisíc trénovaní *aktéra*.



Obr. 4.6: Porovnanie CACLA s MCACLA vzhľadom na počet epoch.

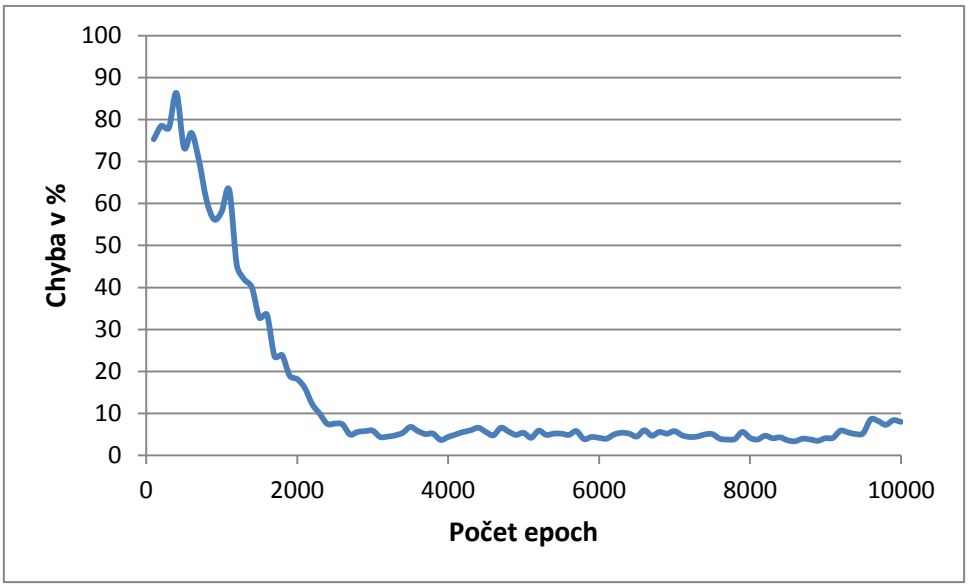


Obr. 4.7: Porovnanie CACLA s MCACLA vzhľadom na počet trénovaní aktéra.

4.5 Experiment s vektorovým vstupom

Doteraz obidve siete *aktér* aj *kritik* dostavali ako stav prostredia dva 3D body umiestnené v stredových bodoch na mečoch. Zaujímalo nás, ako by sa agent učil, keby dostával namiesto týchto dvoch bodov smerový vektor určený týmito bodmi. Takže siete, ako vstup dostávajú tri hodnoty určujúce smerový vektor s počiatkom v stredovom bode agentovho meča, smerujúceho do stredového bodu trénerovho meča.

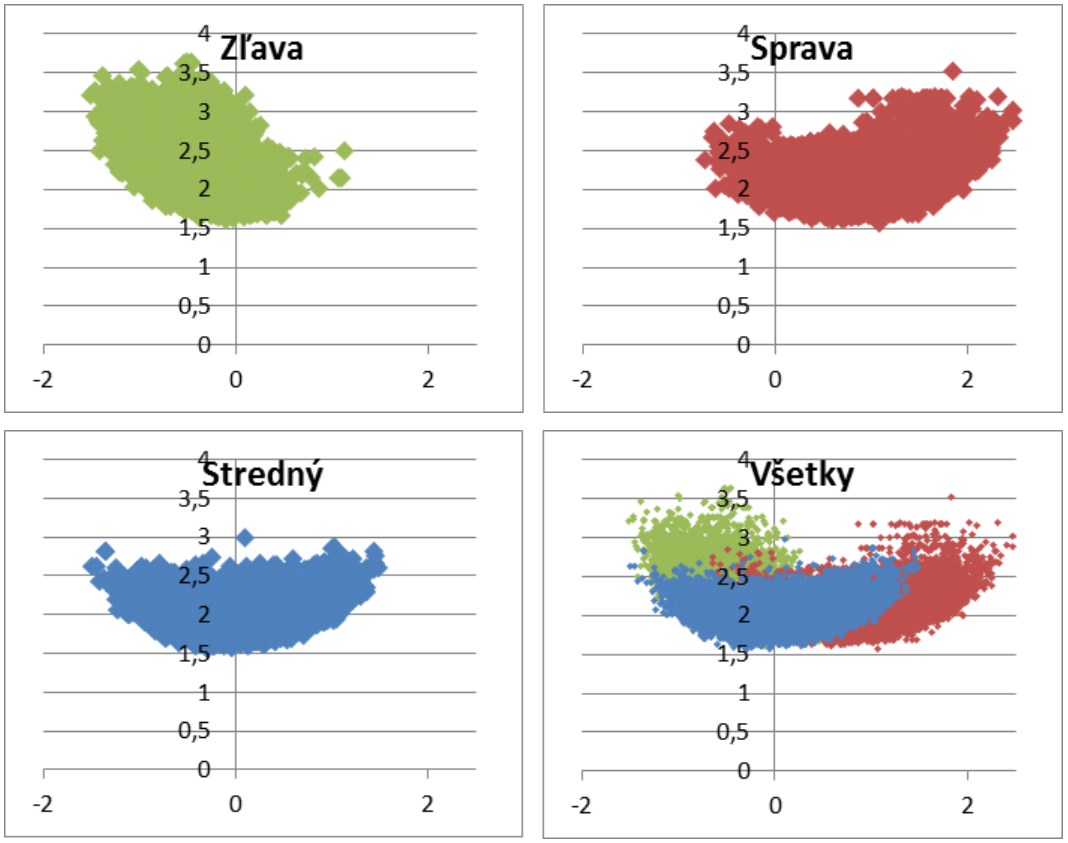
Na obrázku 4.8 môžeme pozorovať priebeh učenia sa agenta s takýmto vstupom a s počiatočnou rýchlosťou učenia *aktéra* $\alpha = 0.1$, ktorú znižujeme na hodnotu 0.01, keď celková chyba klesne pod 10%. Počiatočnú rýchlosť *kritika* sme tiež zvolil $\beta = 0.1$ a neskôr ju znižujeme na hodnotu 0.02. Agent, trénovaný pomocou algoritmu MCACLA, potreboval približne 2200 epoch na to, aby dosiahol viac než 90% obrannú úspešnosť. Agent bol pomerne citlivý na voľbu parametrov učenia, kvôli čomu sme z časového hľadiska nedokázali rýchlo nájsť optimálny model pre túto zmenenú reprezentáciu vstupného stavu prostredia. Agent tiež dokázal veľmi schopne reagovať na už spomínaný neštandardný výpad zospodu, z tohto dôvodu tento experiment považujeme za úspešný.



Obr. 4.8: Algoritmus MCACLA s vektorovým vstupom pre agenta.

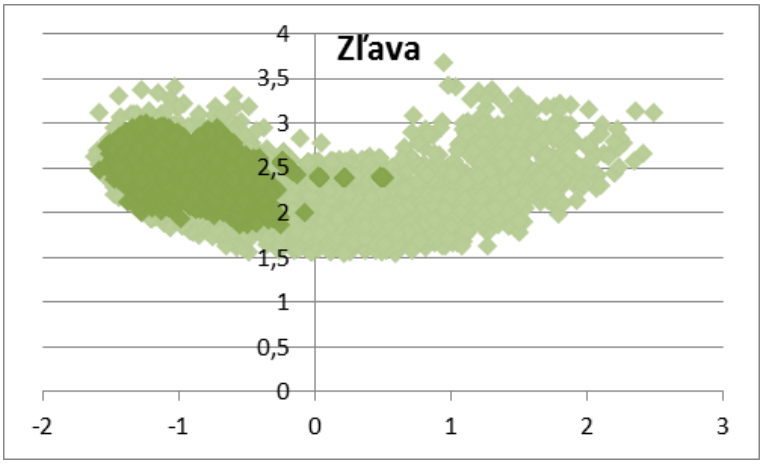
4.6 Skúmanie šírky pásma trénovaných akcií.

Skúmali sme šírky pásma trénovaných akcií, aby sme zistili, v čom sa líšia tieto dva algoritmy, ale hlavne, prečo je algoritmus CACLA tak oveľa horší. Počas trénovania jednotlivými algoritmami sme si zaznamenávali 3D body stredových bodov na meči vždy, keď bola splnená podmienka učenia *aktéra*. Tieto body sme potom premietli do 2D plochy tak, že sme im odobrali z-tovú súradnicu.

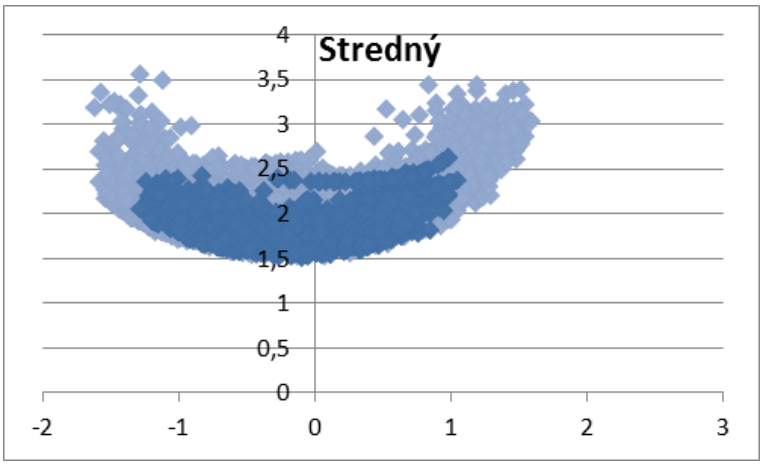


Obr. 4.9: CACLA šírka pásma trénovaných akcií.

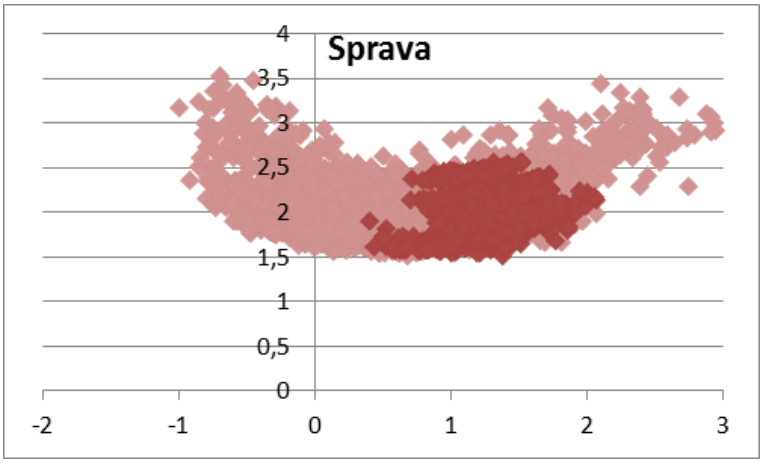
Môžeme pozorovať značný rozdiel medzi grafmi algoritmu CACLA a grafmi algoritmu MCACLA. Pri algoritme CACLA sa tieto oblasti veľmi nelíšili od oblasti (obrázok 4.9), ktoré boli zaznamenané už počas prvých epoch, kedy sa snažil *kritik* čo najviac aproximovať hodnoty časového rozdielu chyby, čiže δ-ty. Na rozdiel od grafov (4.10 – 12) algoritmu MCACLA, na ktorých môžeme pozorovať svetlejšie oblasti, ktoré boli zaznamenané počas prvých epoch, a tmavšie oblasti, ktoré boli zaznamenané, keď už agent dosahoval dobré výsledky.



Obr. 4.10: MCACLA šírka pásma trénovaných akcií - zľava.



Obr. 4.11: MCACLA šírka pásma trénovaných akcií - stredný.



Obr. 4.12: MCACLA šírka pásma trénovaných akcií - sprava.

Z tohto pozorovania sa preto snažíme vyvodiť dôsledky, že práve kvôli tomuto sa agent trénovaný pomocou algoritmu CACLA nedokázal zlepšovať. Učiaca podmienka *aktéra* pri algoritme CACLA prepúšťala akcie s príliš veľkých oblasti, zatiaľ čo pri algoritme MCACLA tieto oblasti boli oveľa menšie a presnejšie, vďaka čomu si agent mohol vyvinúť presnejšie správanie. Preto jeho obranné manévre boli oveľa presnejšie, na základe čoho mohol dosiahnuť až 100% úspešnosť.

Kapitola 5

Záver

Ciele diplomovej práce sa nám podarilo úspešne splniť. Navrhli sme a zimplementovali adaptívneho agenta s využitím umelej inteligencie pre simuláciu šermovania. Pre riešenie tohto problému sme sa rozhodli použiť učiaci algoritmu CACLA - *Continuous Actor Critic learning automaton*, ktorý pracuje nad spojitým priestorom stavov a akcií. Algoritmus CACLA je z kategórie učenia s posilňovaním. Rozhodli sme sa ho použiť, pretože bol úspešne implementovaný v minuloročnej diplomovej práci 2010 Richarda Korenčiaka na ovládanie robotického ramena v 3D priestore. S týmto učiacim modelom siete sa nám podarilo dosiahnuť len čiastočne natrénovanie agenta na danom probléme. Agent, natrénovaný pomocou algoritmu CACLA dosiahol maximálne 80% obrannú úspešnosť a nebol schopný reagovať na nové typy výpadov. Preto sme sa rozhodli pre vlastnú modifikáciu algoritmu CACLA, ktorá by bola schopná dosiahnuť vyššiu úspešnosť ako algoritmus CACLA, a tiež by dokázala schopne reagovať aj na nové typy výpadov.

Navrhli sme modifikáciu algoritmu CACLA, pomocou ktorej sme dokázali úspešne natrénovať agenta, ktorého obranné manévre boli oveľa presnejšie, na základe čoho agent mohol dosiahnuť až 100% úspešnosť. Agent tiež preukázal schopnosť úspešne reagovať aj na nové typy výpadov. Modifikácia algoritmu

CACLA sa ukázala, že je nielenže efektívnejšia pri trénovaní agenta, ale aj z výpočtového hľadiska je oveľa rýchlejšia a menej náročná.

Z implementačného hľadiska táto práca bola pomerne rozsiahla. Jedným z najdôležitejších systémov je fyzikálny engine (physical engine), ktorý vypočítava kolízie rôznych objektov v priestore. V tejto práci boli použite Bézierové krivky, inverzná kinematika, fyzikálny engine, dopredné neurónové siete a 3D vizualizácia v OpenGL. Celý projekt je implementovaný v C++ a implementácia obsahuje 5100 riadkov kódu.

Literatúra

Abul, A., Polat, F., & Alhaji, R. (2000), Multiagent reinforcement learning using function approximation. IEEE *Transactions on Systems, Man, and Cybernetics. Part C: Applications and reviews*, 30, 485-497.

Anderson, C. W. (1993), Q-learning with hidden-unit restarting. In S. J. Hanson, J. D. Cowan, & C. L. Giles (Eds.), *Advances in neural information processing systems* (Vol. 5, pp. 81-88). Morgan Kaufmann, San Mateo, CA.

Bellman , R. E., (1957), *Dynamic Programming.* Princeton University Press..

Crites, R. H., & Barto, A. G. (1996), Improving elevator performance using reinforcement learning. In D. S. T. amd M. C. Mozer & M. E. Hasselmo (Eds.), *Advances in neural information processing systems: Proceedings of the 1995 conference* (pp. 1017-1023). Cambridge, MA: MIT Press..

Farin, G. & Hansford, D. (2000), *The Essentials of CAGD*.

Farin, G. (1997), *Curves and surfaces for computer-aided geometric design* (4 ed.), Elsevier Science & Technology Books.

Humphrys, M. (1997). *Action selection methods using reinforcement learning*. Unpublished doctoral dissertation, University of Cambridge, Computer Laboratory.

Korenčiak, R. (2010), *Aproximácia motorického priestoru ramena simulovaného robota.* Diplomová práca, Univerzita Komenského v Bratislave, fakulta matematiky, fyziky a informatiky, Katedra aplikovanej informatiky.

Lin, L.-J. (1992), Self-improving reactive agents based on reinforcement learning, planning, and teaching. *Machine Learning*, 8, 293-321.

Rummery, G. A., & Niranjan, M. (1994), *On-line Q-learning using connectionist systems* (Technical report No. CUED/F-INFENG/TR 166). Engineering Department, Cambridge University.

Sutton, R. S. & Barto, A. G. (1998), *Reinforcement Learning: An Introduction*. The MIT press, Cambridge MA, A Bradford Book.

Sutton, R. S. (1988), Learning to predict by the methods of temporal differences. Machine Learning, 3, 9-44.

Skinner, B. F. (1938, 1991), The behavior of organisms: An experimental analysis. Acton, MA: Copley.

ten Hagen, S. H. G., & Krose, B. J. A. (1998), Pseudo-parametric Q-learning using feedforward neural networks. In L. Niklasson, M. Boden, & T. Ziemke (Eds.), *ICANN'98, Proceedings of the International Conference on Artificial Neural Networks* (pp. 449-454). Springer-Verlag.

van Hasselt. H, & Wiering, M. A. (2007), *Reinforcement learning in continuous action spaces.* In Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning, p. 272 - 279 .

Wyatt, J. (1995), Issues in putting reinforcement learning onto robots. In *Mobile robotics workshop, 10th biennial conference of the AISB*. Sheffield.

Dodatok

Prílohy diplomovej práce sa nachádzajú na priloženom médiu. Médium obsahuje:

- zdrojové kódy
- diplomovú prácu v elektronickom formáte
- prezentáciu k obhajobe
- videa