Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

RoboCup – tretí rozmer
(Dokumentácia k projektu)

Tím č. 17: Naurotics
Členovia tímu: Aleš Katona, Gabriel Braniša, Peter Čimo,
Tomáš Labuda, Juraj Šimon, Ján Kolesár
Študijný program: IS/SI
Ročník prvý, inžinierske štúdium
Email: team17.robocup@gmail.com
Web: http://www2.dcs.elf.stuba.sk/TeamProject/2007/team17is-si/

Vedúci projektu: Ing. Marián Lekavý
Ak. rok: 2007/2008

Obsah

Zoznam obrázkov

1 Úvod

1.1 Účel dokumentu

Tento dokument bol vytvorený v rámci predmetu Tímový projekt v akademickom roku 2007-2008. Venuje sa virtuálnej simulácii 3D robotického futbalu pod názvom RoboCup 3D. Dokument je členený na jednotlivé kapitoly nasledovne. V prvej kapitole si určujeme cieľ nášho projektu, opisujeme zadanie a ponúkame stručný úvod do prostredia RoboCup-u. Druhou kapitolou je analýza. V nej sa najprv pozrieme na kategórie do ktorých sa delí simulačná liga RoboCup-u. Ďalej si povieme niečo o pravidlách a samotnom simulačnom prostredí. Nakoniec sa pozrieme na niekoľko tímov resp. hráčov robotického futbalu a podrobnejšie si ich opíšeme. Tretia kapitola predstavuje našu špecifikáciu požiadaviek. V štvrtej kapitole opisujeme náš hrubý návrh humanoidného hráča. Využijeme pri ňom znalosti z analýzy z kapitoly tri a prispievame svojou vlastnou ideou ako vylepšiť pohyby humanoida na základe evolučného učenia sa.

1.2 Cieľ projektu

Cieľom projektu je vytvoriť humanoidného hráča so základnými orientačnými a pohybovými schopnosťami (chôdza, státie, postavenie sa). Ako ideologický základ je vhodné využiť prácu a poznatky minuloročného týmu. Nový humanoidný hráč bude pracovať na novej verzií serveru vo verzií 0.5.6. Je potrebná vyhľadať iných humanoidných hráčov, jedného z nich si vybrať a na jeho základe zostaviť hráča vlastného. Implementovať mu základné schopnosti a poskytnúť tak určitý základ pre prácu tímov v budúcich rokoch. Hráč má byť modulárny a schopný rozšírenia v budúcich rokoch.

1.3 Zadanie

Téme RoboCup, presnejšie lige simulovaného robotického futbalu sa naši študenti venujú už osem rokov. Tímy študentov, či už v rámci umelej inteligencie alebo tímového projektu, sa snažia vytvárať a vylepšovať programy, ktoré simulujú správanie sa futbalového hráča. Každý tím sa v rámci obmedzení, určených pravidlami hry futbal a špecifikami simulačného prostredia, snaží vytvoriť čo najlepšieho hráča. Mužstvo, vytvorené z takýchto hráčov, by malo vyhrať nad mužstvom súpera. O súťaži a doterajšej činnosti je dosť popísané aj na stránke STU [11].

Simulácia futbalu pôvodne prebiehala iba v dvoch rozmeroch. Pre zvýšenie reálnosti simulácie bolo vytvorené 3D simulačné prostredie, ktoré rozširuje možnosti hry. 3D simulačné prostredie sa pomerne výrazne líši od doposiaľ používaného 2D prostredia, a to jednak spôsobom simulácie, ale hlavne možnosťami ktoré poskytuje hráčom.

Hlavným cieľom projektu bude vytvoriť hráča pre 3D simuláciu, ktorý dokáže plnohodnotne využívať možnosti poskytované simulačným prostredím. Úlohou teda bude prevziať jednoduchého hráča vytvoreného na našej fakulte v minulom roku a doplniť do neho komplexnejšie typy správania a rozhodovania. Keďže sa predpokladá ďalšie rozširovanie hráča v ďalších rokoch, dôležitými požiadavkami sú prehľadnosť a ďalšia rozširovateľnosť, a to na úrovni návrhu aj implementácie. Dôraz pri vytváraní hráča by mal byť kladený najmä na dobre prepracované a odladené nižšie schopnosti hráča, ktoré umožnia hráčovi spracovávať vnemy z prostredia a

efektívne konať v prostredí (pohybovať sa, pracovať s loptou). Pri návrhu a implementácii bude tiež možné (a žiadané) čerpať z veľkého množstva prístupov existujúcich v 2D simulácii.

Zimný semester je vyhradený na oboznámenie sa s celým simulačným prostredím a hráčom ktorý sa bude rozširovať, a takisto s existujúcimi hráčmi (2D aj 3D), ďalej návrhu a prototypovej realizácii hráča. Dôležitou súčasťou bude vytvorenie plánu implementácie a overovania prístupu v nasledovnom semestri. V letnom semestri nás čaká dokončenie realizácie návrhu a jeho overovanie. Nemenej podstatnou časťou projektu bude vytvorenie dokumentácie, ktorá poskytne tímom v ďalších rokoch odrazový mostík pri použití vytvoreného hráča.

1.4 RoboCup

RoboCup je medzinárodná súťaž autonómnych robotov softvérových agentov. Súťaž je určená na podporu výskumu a výučby umelej inteligencie, robotiky a ďalších príbuzných oblastí. *Robot World Cup Soccer Games and Conferences*, ide o neziskovú organizáciu, ktorá každoročne usporadúva turnaje v robotickom futbale. Snahou tejto spoločnosti je podporovať vývoj rôznych technologických oblastí pomocou riešenia štandardnou metódou, pri ktorej je možné využiť obrovské množstvo technológií. Na tento účel programátori zvolili futbal. Základnou ideou je vytvoriť čo najviac dokonalých fyzických, ale i syntetických, čiže programovo realizovaných agentov, ktorí dokážu hrať futbal na vysokej úrovni. Míľnikom, o ktorý RoboCup-u ide, je zostrojenie tímu humanoidných robotov, ktorí by hrali proti najlepšiemu "ľudskému" tímu podľa oficiálnych pravidiel FIFA. Predpovede organizácie RoboCup hovoria o uskutočnení tohto zápasu okolo polovice tohto storočia.

RoboCup Federation je medzinárodná nezisková organizácia zaregistrovaná v Ženeve za účelom podpory, propagácie a posilňovania výskumu RoboCup-u. Túto iniciatívu v súčasnosti nasleduje približne 1500 výskumníkov v 17 rôznych krajinách. Do súťaže sa môže zapojiť každý tím (resp. jednotlivec), bez ohľadu na to, či je alebo nie je zameraný na samotný výskum. Niektoré tímy sa zúčastňujú súťaže iba zo zábavy (najmä v simulačnej lige), iné sú zamerané na seriózny výskum, pričom svoje výsledky pravidelne publikujú.

Existuje liga simulovaného futbalu pre 2D, kde sú agenti v tvare kruhu v počte jedenásť na hracej ploche. V 3D existujú dva typy simulovaného futbalu a to reprezentácia agenta guličkou, čo je pokračovanie, nadstavba 2D simulovaného futbalu. Tento typ postupne zaniká a nahradila ho simulácie 3D humanoidov v počte troch hráčov na jeden tím. Simulačné prostredie disponuje realistickým simulovaním fyzikálnych zákonov pomocou ODE knižnice.

2 Analýza

V nasledujúcej kapitole si postupne predstavíme najprv základné kategórie súťaže RoboCup a ich pravidlá. Ďalej sa oboznámime so samotným simulačným prostredím a modelom nového humanoidného robota. Následne si predstavíme niekoľko tímov pôsobiacich v oblasti simulovaného robotického futbalu.

2.1 Pravidlá RoboCup-u

Bohužiaľ na oficiálnych stránkach RoboCup-u [1] sme nenašli žiadne užitočné a aktuálne informácie. Kategória 3D simulovaného futbalu sa od roku 2005 kedy boli tieto informácie naposledy aktualizované, dosť zmenila (predovšetkým zmena modelu hráča). Na stránke ligy sa však nachádzajú aktuálnejšie informácie, konkrétne pravidlá z turnaju RoboCup 2007 v Atlante [3]. V Atlante sa použil server rcssserver3d-0.5.6 spustený pod operačným systémom SuSE Linux 10.1. Server a monitor bežali na samostatných počítačoch, a každý tím mal k dispozícií jeden počítač pre svojich hráčov.

2.1.1 Kategória Soccerbot

Jedná sa o kategóriu v ktorej proti sebe súťažia dva tímy hráčov.
Tými môžu použiť jedného z dvoch typov hráča:

1. *soccerbot055.rsg* – Je to verzia bota, ktorá prišla so serverom rcssserver3d-0.5.5.
   - jednodielny trup
   - žiadne obmedzenia kĺbov – môžu sa otáčať aj do neprirodzených polôh
   - TCH senzor – poskytuje len informáciu o tom že sa niečo niečoho dotýka
   - nemá detekciu kolízií rúk
2. *soccerbot056.rsg* – Tento bot je súčasťou servera rcssserver3d-0.5.6.
   - dvojdielny trup
   - obmedzenia kĺbov – nemôžu sa otáčať do neprirodzených polôh
   - FRP senzor – poskytuje informáciu o sile, ktorá pôsobí pri dotyku a o mieste jej pôsobenia.
   - detekcia kolízií rúk

Pozn.: Obidvaja boti majú rovnakú hmotnosť.

Štruktúra zápasov je nasledovná:

- Jeden tým je zložený z 3 hráčov (v pravidlách nie je napísané či je presne určený brankár, ale pravdepodobne to bude hráč s číslom jeden).
- Hra je rozdelená na dva polčasy. Každý trvá 7,5 min.
- Počas súťaže tímy môžu zmeniť alebo nahradiť svoje zdrojové kódy. Tieto zmeny budú však vykonané na vlastné riziko. Zmena je možná len medzi jednotlivými zápasmi, nie počas hry.
- Ak hráč nebude fungovať organizátori ho nebudú rozbehávať. (Tímy totiž nemusia prísť na súťaž osobne, ale môžu svojich hráčov poslať.)

Na hru dohliada ľudský rozhodca, pričom niektoré fauly sú automaticky detekované simulátorom, no ten nemôže detekovať všetky. Rozhodca má právo udeliť voľný kop poškodenému tímu. Za faul sa pokladá napríklad (no nie len):

- Ak jeden z hráčov úmyselne chytí, alebo sa dotkne lopty rukou, okrem brankára.
- Ak niektorý hráč leží na lopte viac ako 10 sekúnd.
- Ak hráč úmyselne udrie alebo postrčí súperovho hráča.
- Ak jeden tým obkľúči loptu tak, že druhý tým nemá na loptu dosah.
- Ak hráči blokujú bránku tak že sa lopta nemôže dostať dnu (stena z hráčov v bránke).
- Ak tým úmyselne blokuje pohyb hráčov.
- Čokoľvek, čo sa javí ako porušenie fair play, môže byť tiež považované za faul. Záleží na posúdení rozhodcu.

Cieľom hry je hrať futbal podľa obvyklého vnímania fér hry a obmedzení daných virtuálnym simulovaným svetom 3D simulátora. Obchádzanie týchto pravidiel je považované za porušenie povinnosti hrať fér hru a počas turnajových zápasov je to prísne zakázané.

Za porušenie fair play sa považuje napríklad:

- Používanie hráčov iného tímu.
- Zahlcovanie servera posielaním neúmerného počtu správ od klienta.
- Priama komunikácia hráčov použitím iných komunikačných spôsobov, ako napríklad medzi procesová komunikácia.
- Práca so súťažnými počítačmi alebo ich úmyselné reštartovanie.

Čokoľvek z uvedeného je prísne zakázané. Iné stratégie môžu byť označené za porušenie fair play, po konzultácií s komisiou zaoberajúcou sa pravidlami. Obzvlášť deštruktívne narušenie operácií súperových agentov, alebo získanie výhody spôsobom iným, ako explicitne ponúkaným simulátorom, je považovaný za porušenie fair play. Ak si účastníci súťaže nie sú istý ohľadne použitia niektorej metódy, konzultujú to s komisiou zaoberajúcou sa pravidlami pred začiatkom turnaja. Ak sa počas turnaja zistí, že niektorý tým používa ne fér praktiky, bude tento tým okamžite vylúčený z turnaja.

V pravidlách sa nepíše nič o koučovi alebo trénerovi. Predpokladáme teda že ešte nie je implementovaný v servery a teda ani povolený (použiteľný).

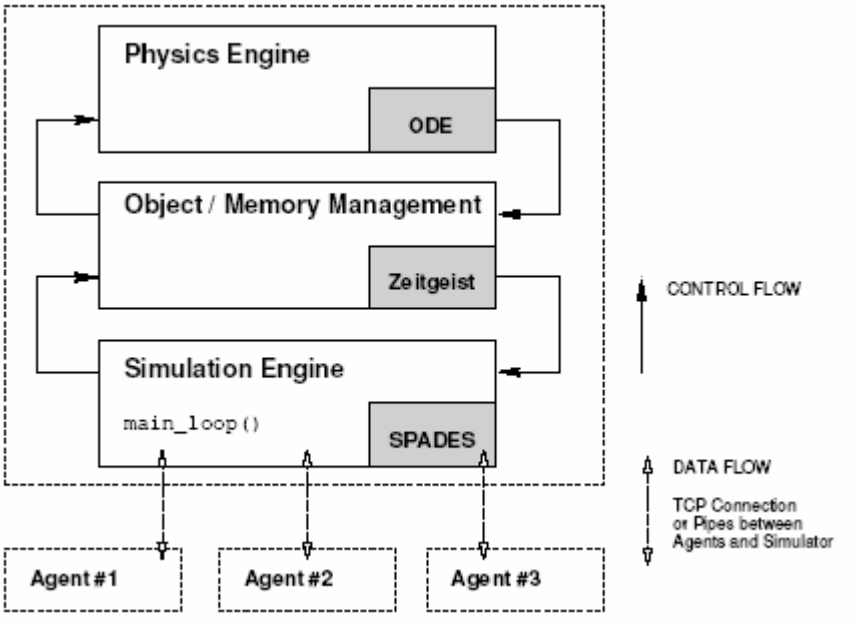
2.1.2 Kategória 3D development

Kategória 3D development je zameraná na urýchlenie vývoja nízko úrovňových schopností hráča, aby tými mohli čo najrýchlejšie prejsť na vývoj vyšších schopností hráčov. Presné pravidlá ešte nie sú definované v použiteľnom rozsahu, ale na oficiálnej stránke ligy [2] sú spomenuté niektoré súťaže, z tejto kategórie:
Technické súťaže - môžu obsahovať kráčanie, kopanie, vstávanie, penalty, ...
Futbalové zápasy 5-vs-5 - Keďže zápas 5-vs-5 je komplexná úloha, boli pravidlá pre rok 2007 upravené. Hráči mohli narábať s loptou akoukoľvek časťou tela (vrátane rúk). Tiež sa mohli pohybovať kráčaním, behaním, plazením, ...

2.2 Simulačné prostredie

Ako simulačné prostredie sa využíva SPARK [14], multi-agentový simulačný systém hmotných agentov v troj-dimenzionálnom prostredí. Nasledujúci obrázok (obr. 1.) znázorňuje hlavné časti simulačného systému:

- *Prostriedok pre fyzikálnu simuláciu (Physics Engine)* – implementovaný pomocou knižnice ODE. ODE slúži na fyzikálnu simuláciu komplexných objektov zložených z pevných častí prepojených kĺbmi.
- *Prostriedok na správu objektov (Object / Memory Management)* –implementovaný pomocou frameworku Zeitgeist. Prostriedok umožňuje flexibilný a jednotný prístup k jednotlivým komponentom simulátora:
  - objektom reprezentujúcim simulovanú scénu (sprístupnené ODE objekty),
  - objektom zabezpečujúcim základnú funkcionalitu simulátora.
- *Simulačný prostriedok (Simulation Engine)* – zabezpečuje hlavný cyklus vykonávania a interakciu medzi agentmi a simulátorom. Simulácie väčšieho počtu agentov rozmiestnených na viacerých počítačoch sú ovplyvňované faktormi ako aktuálne zaťaženie siete, rôzne typy latencií a rôzne rýchlosti CPU. Simulácie sú tak často nereprodukovateľné. Z tohto hľadiska je prostriedok implementovaný dvoma spôsobmi:
  - Implementácia pomocou SPADES – berie do úvahy vyššie spomenuté faktory a umožňuje reprodukovateľné simulovanie.
  - Jednoduchá implementácia – vyššie uvedené faktory jednoducho ignoruje, nezaručuje reprodukovateľné simulovanie, ale poskytuje vyšší výkon.

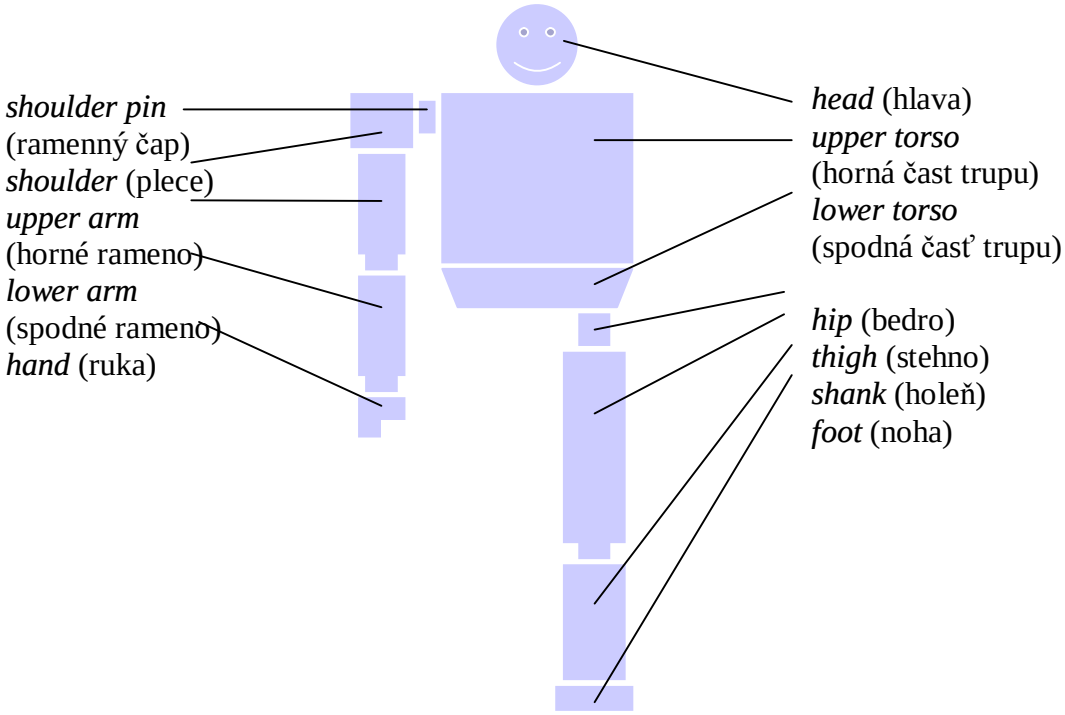


Obr. 1 Architektúra simulátora [14]

Aktuálny server zabezpečujúci simuláciu robotického futbalu je implementovaný práve systémom SPARK bez použitia SPADES. V súčasnosti je dostupný server vo verzii *0.5.6* a ďalší popis bude preto aktuálny pre túto verziu.

2.2.1 Štruktúra robota

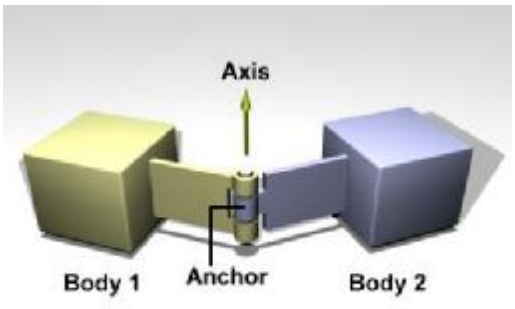
Ako už bolo spomenuté v predchádzajúcej kapitole, simulačný prostriedok SPARK je univerzálny a umožňuje simuláciu rôznych typov hráčov (s rôznymi tvarmi, kĺbmi, efektormi, perceptormi). V súčasnosti sa používa typ robota "soccerbot056" a práve jeho štruktúra bude ďalej popísaná. Nasledujúci obrázok (obr. 2.) znázorňuje jednotlivé časti robota.



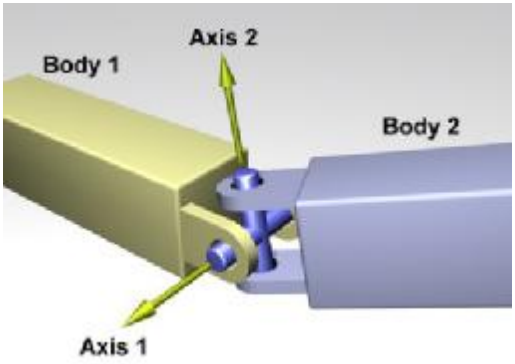
Obr. 2. Štruktúra robota "soccerbot056"

Robot je pre potreby pohybu vybavený niekoľkými kĺbmi. V popisovanom type robota sa využívajú dva typy pohyblivých kĺbov definovaných knižnicou ODE [15]:

- *Pántový kĺb (Hinge joint, HJ)* – poskytuje jeden stupeň voľnosti otáčania (obr. 3.),
- *Univerzálny kĺb (Universal joint, UJ)* – poskytuje dva stupne voľnosti otáčania (obr. 4.).



Obr. 3 Pántový kĺb (Hinge joint)



Obr. 4 Univerzálny kĺb (Universal joint)

Kĺby robota sú vybavené perceptormi, ktoré poskytujú informácie o stave kĺbov a efektormi, pomocou ktorých je možné stav kĺbov meniť. Prepojenie jednotlivých častí robota kĺbmi, názvy perceptorov a efektorov jednotlivých kĺbov popisuje nasledujúca tabuľka (tab. 1.).

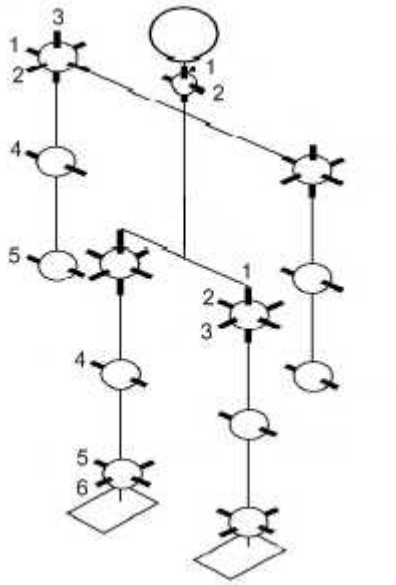
prepojené časti tela robota		**typ kĺbu**	**perceptor***	**efektor***
head	upper torso	FJ		
lower torso	upper torso	HJ		
shoulder	upper torso	UJ	laj1_2	lae1_2
upper arm	shoulder	HJ	laj3	lae3
lower arm	upper arm	HJ	laj4	lae4
hand	lower arm	FJ		
hip	lower torso	HJ	llj1	lle1
thigh	hip	UJ	llj2_3	lle2_3
shank	thigh	HJ	llj4	lle4
foot	shank	UJ	llj5_6	lle5_6

FJ – Fixed joint, nepohyblivý kĺb
HJ – Hinge joint
UJ – Universal joint
** – názvy perceptorov a efektorov sú uvedené uvažujúc ľavé končatiny (pre pravé by sa názvy začínali písmenom "r")*

Tab. 1. Prepojenie častí tela kĺbmi

Orientácia osí jednotlivých kĺbov je zrejmá z nasledujúceho obrázku (obr. 5.) (pohyblivé kĺby na krku a rukách sa v súčasnej verzii robota nevyskytujú).

.

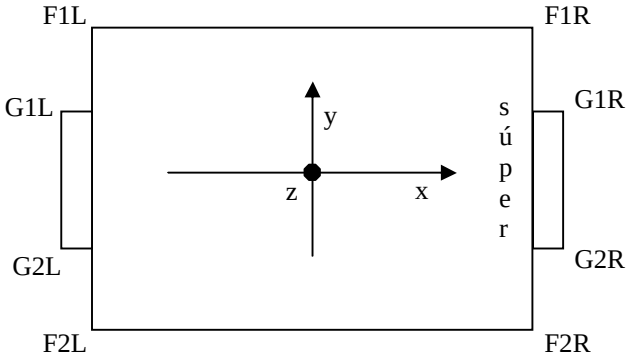


Obr. 5 Orientácia osí jednotlivých kĺbov robota [16]

2.2.2 Štruktúra prostredia robota

Prostredie robota pri simulácii pozostáva z:

- spoluhráčov,
- protihráčov,
- lopty,
- ihriska (obr. 6.):
  - 4 rohové zástavky (*F1L*, *F2L*, *F1R*, *F2R*),
  - 4 okraje bránok (*G1L*, *G2L*, *G1R*, *G2R*).



Obr. 6 Štruktúra ihriska

2.2.3 Komunikačný protokol

Každý hráč (robot) reprezentuje v architektúre klient-server jedného klienta. Komunikácia so serverom prebieha prostredníctvom protokolov TCP a UDP (prednastavený je TCP protokol). Správy vymieňané medzi klientom a serverom majú tvar tzv. S-výyrazu. Ide o konvenciu na reprezentáciu semištruktúrovaných dát[1]. Nasleduje popis komunikačné protokolu medzi klientom a serverom [17][18].

Server->klient
Klient dostáva od servera informácie z perceptorov robota. Správy sú posielané v diskrétnych časových intervaloch 0.02s. Nasleduje popis správ prijímaných z jednotlivých perceptorov z hľadiska syntaxe a sémantiky.

Príklad kompletnej správy prijatej klientom sa nachádza v prílohe A.

2.2.4 TimePerceptor

Perceptor poskytuje informácie o aktuálnom simulačnom čase (čas od začiatku simulácie, nie hry).
Vzor: `(time (now <simulačný_čas>))`
Príklad: (príloha A)

2.2.5 GameState perceptor

Perceptor poskytuje informácie o aktuálnom stave hry (hrací čas a mód hry).
Vzor:
```
(GS
      (time <hrací_čas>)
      (pm <mód_hry>))
```

- <mód_hry> – môže nadobúdať jednu z hodnôt: *BeforeKickOff, KickOff_Left, KickOff_Right, PlayOn, KickIn_Left, KickIn_Right, corner_kick_left, corner_kick_right, goal_kick_left, goal_kick_right, offside_left, offside_right, GameOver, Goal_Left, Goal_Right, free_kick_left, free_kick_right, unknown*.

Príklad: (príloha A)

2.2.6 Vision preceptor

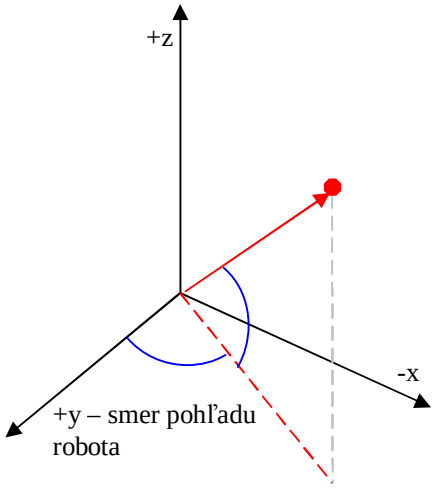
Perceptor poskytuje informácie o robotom viditeľných objektoch. V súčasnosti má robot 360 stupňové videnie (vidí všetky objekty) a objekty majú podobu hmotných bodov. Pre každý objekt poskytuje perceptor polohu objektu relatívne od robota vo sférických súradniciach. Stredom sférickej sústavy je ťažisko horného trupu robota (upper torso). Pre hráčov okrem polohy poskytuje informáciu o tíme a čísle dresu hráča.
Vzor:
```
(See
      (<typ_objektu>
            (pol <vzdialenosť> <uhol_1> <uhol_2>))
      ...
      (P
```

[1]Viac v: S-expression, Wikipedia, http://en.wikipedia.org/wiki/S-expression

```
(team <názov_tímu>)
(id <číslo_hráča>)
(pol <vzdialenosť> <uhol_1> <uhol_2>)))
```

- *<typ_objektu>*
  - 4 rohové zástavky (*F1L*, *F2L*, *F1R*, *F2R*) (obr. 6.),
  - 4 okraje bránok (*G1L*, *G2L*, *G1R*, *G2R*) (obr. 6.),
  - 1 lopta (*B*),
  - spoluhráči a súperi (*P*),
- *<vzdialenosť>* – vzdialenosť objektu od ťažiska hornej časti trupu (stred sférickej sústavy),
- *<uhol_1>* – uhol v rovine *xy* v stupňoch (obr. 7),
- *<uhol_2>* – uhol s rovinou *xy* v stupňoch (obr. 7).

Príklad: (príloha A)



Obr. 7 Znázornenie uhlov poskytovaných perceptorom pre polohu červenej bodky

2.2.7 GyroRate perceptor

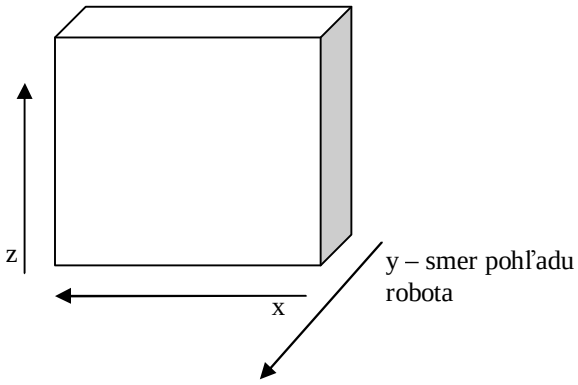
Perceptor je umiestnený v hornom trupe robota (upper torso) a poskytuje informácie o jeho uhlovom zrýchlení okolo jednotlivých osí.

Vzor:

```
(GYR
      (name torso)
      (rt <x> <y> <z>))
```

- *<x> <y> <z>* – uhlové zrýchlenia horného trupu robota okolo osí *x*, *y* a *z* v rad.s^{-2} (obr. 8.).

Príklad: (príloha A)



Obr. 8 Orientácia súradnicových osí robota

2.2.8 Touch perceptor

(Súčasná verzia robota nemá tento perceptor aktívny) Perceptor poskytuje binárnu informáciu, či bola určitá z častí tela účastníkom kolízie.

Vzor: `(TCH (name <názov_časti_tela>) (val <hodnota>))`

- <*hodnota*> – *1* v prípade kolízie, inak *0*.

Príklad: `(TCH (name footleft) (val 1))`

2.2.9 ForceResistance perceptor

Perceptor poskytuje informáciu o kolíznej sile a priemerný bod pôsobenia kolíznej sily. V prípade, že sa dva povrchy kompletne dotýkajú a existuje množstvo kontaktných bodov, informácia obsahuje len priemerný bod a celkovú silu ako súčet síl pôsobiacu vo všetkých bodoch dotyku.

Vzor: `(FRP (n <názov_časti_tela>) (c <px> <py> <pz>) (f <fx> <fy> <fz>))`

- <*názov_časti_tela*> – v súčasnosti poskytuje perceptor informácie z ľavej (*ls*) a pravej (*rf*) nohy,
- <*px*> <*py*> <*pz*> – lokálne súradnice bodu pôsobenia kolíznej sily,
- <*fx*> <*fy*> <*fz*> – kolízny vektor.

Príklad: (príloha A)

2.2.10 Joint perceptors (kĺbové perceptory)

Perceptory poskytujú informácie o stave jednotlivých kĺbov robota (aktuálnom otočení podľa osí otáčania).

Hinge joint

Vzor:
```
(HJ
      (n <názov>)
      (ax <uhol>))
```

- <*názov*> – názov kĺbového perceptora (tab. 1.),
- <*uhol*> – uhol otočenia okolo osi otáčania v stupňoch (obr. 5).

Príklad: (príloha A)

2.2.11 Universal joint

Vzor:
```
(HJ
      (n <názov>)
      (ax <uhol_1>)
      (ax <uhol_2>))
```

- <*názov*> – názov kĺbového perceptora (tab. 1.),
- <*uhol_x*> – uhol otočenia okolo osi *x* otáčania v stupňoch (obr. 5).

Príklad: (príloha A)

2.2.12 Create effector

Efektor slúži na vytvorenie hráča. Simulačný prostriedok je univerzálny a umožňuje simuláciu rôznych hráčov (s rôznymi tvarmi, kĺbmi, efektormi, perceptormi). Aktuálne sa používa hráč typu "soccerbot056".

Vzor: `(scene <cesta_k_definícii_typu_hráča>)`
Príklad: `(scene rsg/agent/soccerbot056.rsg)`

2.2.13 Init effector

Efektor slúži na nastavenie čísla dresu a názvu tímu hráča.

Vzor: `(init (unum <číslo_dresu>) (teamname <názov_tímu>))`
Príklad: `(init (unum 7) (teamname RoboLog))`

2.2.14 Beam effector

Efektor slúži na nastavenie pozície hráča na ihrisku pred začatím zápasu.

Vzor: `(beam <x> <y> <uhol>)`

- <*x,y*> – pozícia na ihrisku (obr. 6.),
- <*uhol*> – natočenie hráča v stupňoch (0 stupňov = x+, 90 stupňov = y+).

Príklad: `(beam <10.0> <-10.0> <0.0>)`

2.2.15 Joint effectors (kĺbové efektory)

Efektory slúžia na obsluhu motorčekov v kĺboch robota. Pre každý kĺb je možné určiť rýchlosť a orientáciu (určuje znamienko rýchlosti) otáčania podľa osí v kĺbe. Zmysel otáčania podľa osí na obr. 5. je pre kladné hodnoty rýchlosti možné určiť pravidlom pravej ruky (palec ukazuje v kladnom smere príslušnej osi (obr. 8.) a zahnuté prsty v smere otáčania).

Hinge joint
Vzor: `(<názov> <rýchlosť_otáčania>)`
- <*názov*> – názov kĺbového efektora (tab. 1.),
- <*rýchlosť_otáčania*> – rýchlosť otáčania okolo osi otáčania v rad.s^{-1} (obr. 5).

Príklad: `(lae3 1.0)`

Universal joint
Vzor: `(<názov> <rýchlosť_otáčania_1> <rýchlosť_otáčania_2>)`
- <*názov*> – názov kĺbového efektora (tab. 1.),
- <*rýchlosť_otáčania_1*> – rýchlosť otáčania okolo osi otáčania 1 v rad.s^{-1} (obr. 5).
- <*rýchlosť_otáčania_2*> – rýchlosť otáčania okolo osi otáčania 2 v rad.s^{-1} (obr. 5).

Príklad: `(lae1_2 1.0 0.0)`

2.2.16 Zhodnotenie

Kapitola poskytla prehľad simulačného prostredia, štruktúry robota a komunikačného protokolu. Vzhľadom na relatívne nedávne uvedenie humanoidného typu robota a príslušného servera, bolo (je) najväčším problémom hľadanie relevantných zdrojov. Do dnešného dňa totiž nie je dostupná kompletná dokumentácia k serveru pre humanoidný typ robota. Ako zdroj informácií pre túto kapitolu bol okrem uvedenej literatúry použitý aj triviálny prototyp na testovanie komunikačného protokolu zasielaním jednoduchých správ. Empirický pôvod majú najmä informácie o funkčnosti *Vision perceptora* a *GyroRate perceptora*. Funkčnosť niektorých perceptorov nebola vôbec overená (*Touch perceptor*, *ForceResistance perceptor*). Jednou z úlohou prototypu bude preto overiť aj ich funkcionalitu.

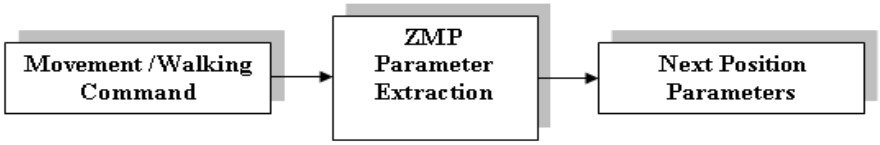
2.3 Analýza rôznych hráčov a tímov

Pred vytvorením našej špecifikácie sme trochu podrobnejšie preštudovali viacero iných tímov a ich hráčov najmä tímov účastniacich sa turnaja v roku 2007 v Atlante.

2.3.1 Sama3D [4]

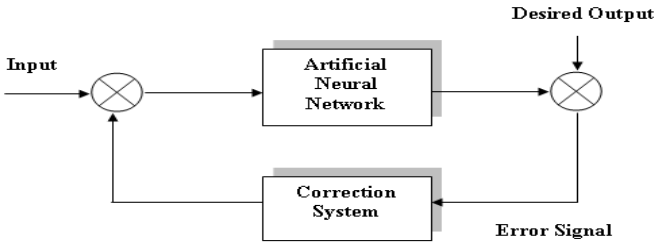
Jedná sa o humanoidného robota ktorý dokáže vykonávať jednoduché pohybové úkony. Na chôdzu využíva neurónovú sieť. Pôvodne robot využíval tzv. ZMP (Zero Moment Point) metódu. Pri ZMP metóde je cieľom udržanie bodu v ktorom je súčet všetkých momentov síl rovné nule, vo vnútri konvexného n-uholníka všetkých dotykových bodov robota s podlahou počas pohybu. Metóda vychádza z matematického modelu kinematiky každej časti schopnej pohybu. Pre každý pohyb je odvodená 3 dimenzionálna rotačná matica a pre každý kĺb je určený translačný vektor. Využitie tejto metódy však vedie k obrovskému zaťaženiu procesora pre každý krok resp. pohyb. Toto zaťaženie autori robota však dokázali znížiť učiacou procedúrou. Na tento účel bola vytvorená neurónová sieť. Sieť bola trénovaná na hodnotách extrahovaných pomocou metódy ZMP. Naučená sieť bola potom schopná určiť ďalší krok (súradnice a ohyby kĺbov) bez potreby zložitých kalkulácií za pomoci ZMP metódy. Robot bol zostrojený a otestovaný v prostredí Webots.

Implementácia robota prebiehala v troch krokoch. V prvom kroku boli určené základné možné pohyby robota v hernom prostredí. Následne boli prepočítané metódou ZMP (v prostredí Matlabu). Z týchto výpočtov boli extrahované údaje pre pohyby kĺbov pre zvolený typ pohybu ako je znázornené na obrázku č. 9.



Obr. 9 Extrahovanie ZMP parametrov pre určenie výslednej pozície

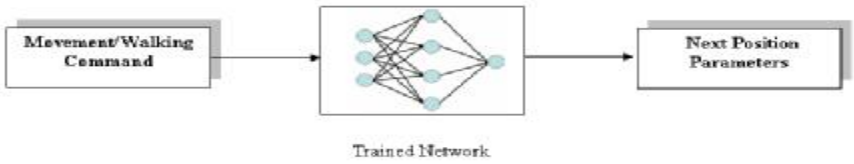
Ďalším krokom bolo vytvorenie a učenie neurónovej siete so vstupmi, ktorými boli dvojice príkaz pre pohyb a k nemu údaje získané ZMP metódou (polohy a rýchlosti kĺbov). Schéma učenia sa je znázornená na obrázku č. 10.



Obr. 10 Učenie neurálnej siete

Sieti je poskytnutý vstup (príkaz pohybu a ZMP údaje). Z výstupu je vypočítaná chyba ktorá je zavedená do korekčného systému. Aplikuje sa korekcia vstupu a sieť sa trénuje ďalej až dokým chyba nie je zanedbateľná.

Po tom čo sa sieť naučí je možné jej priamo vložiť príkaz pohybu a na jej výstupoch dostaneme priamo hodnoty ktoré by poskytla ZMP metóda avšak bez nutnosti toľkých výpočtov, ako ukazuje obrázok.



Obr. 11 Určovanie parametrov za pomoci neurónovej siete

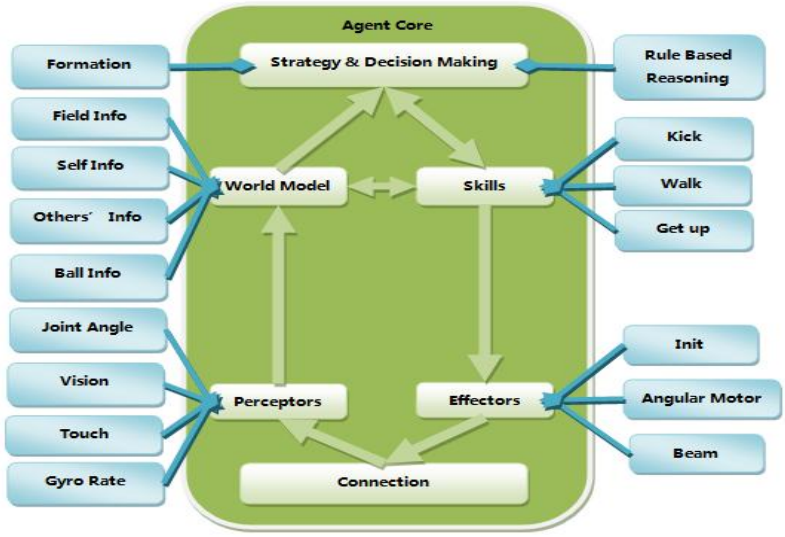
Robot tohto tímu disponoval len základnými pohybovými funkciami na najnižšej úrovni. Robot nebol schopný žiadnych vyšších funkcií ako je práca s loptou či plánovanie a pod.

2.3.2 SEU-3D [5]

Tím vznikol v roku 2005 a úspešne sa zúčastnil viacerých súťaží s robotmi v podobe gule čo bola

staršia verzia serveru pre RoboCup 3d. Po prechode na nov3iu verziu serveru a humanoidného hráča sa tím sústredil na vývoj základných schopností, vlastného vývojového prostredia a robustného kódu. V roku 2007 sa umiestnili na prvom mieste na RoboCup Iran Open a na treťom mieste v svetovom robocup3d.

Architektúra agenta je modulárna, formou zásuvných modulov a tzv. singletonov. Teda napríklad pri zmene verzie servera, nie je potrebné meniť celý kód stačí len zmeniť príslušný modul. Na obrázku je znázornená schéma agenta.



Obr. 12 Vnútorné usporiadanie agenta

Zelený box predstavuje jadro robota. Moduly vo vnútri štvorca sú implementované ako singletony pre jednoduchšiu komunikáciu. Moduly mimo štvorca sú implementované ako zásuvné moduly.

World model

Slúži na uchovanie si stavu sveta. Z tohto stavu môže robot odvodiť svoju pozíciu P_v a Rotáciu R_v na hracej ploche. Pozícia je odvodzovaná z troch pevných bodov ktoré predstavujú okraje ihriska. Napríklad z horného a dolného ľavého rohu a vrchného pravého rohu p1l, p2l a p1r. Môžeme zostaviť nasledujúce rovnice.

$$\begin{cases} |P_v - P_{1l}| = \rho_{1l_d} \\ |P_v - P_{2l}| = \rho_{2l_d} \\ |P_v - P_{1r}| = \rho_{1r_d} \\ -P_{1l_x} = -P_{2l_x} = P_{1r_x} = 0.5 \cdot field\_length \\ P_{1l_y} = -P_{2l_y} = -P_{1r_y} = 0.5 \cdot field\_width \\ P_{1_z} = P_{2_z} = P_{3_z} = 0 \end{cases} \quad (1)$$

Z ktorých pozícia robota P_v môže byť vypočítaná takto:

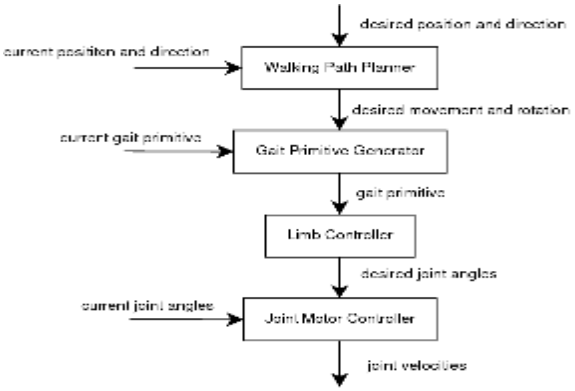
$$\boldsymbol{P_v} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} \frac{\rho_{1l_d}{}^2 - \rho_{1r_d}{}^2}{2 \cdot field\_length} \\ \frac{\rho_{2l_d}{}^2 - \rho_{1l_d}{}^2}{2 \cdot field\_width} \\ \sqrt{\rho_{1l_d}{}^2 - (P_x - P_{1l_x})^2 - (P_y - P_{1l_y})^2} \end{bmatrix} \quad (2)$$

Určenie ťažiska je taktiež dôležité z hľadiska udržania stability robota. Ťažisko môže byť vypočítané vzorcom

$$P_{CoM} = \sum P_i \cdot m_i$$

kde P_i je pozícia časti tela a mi je jeho objem. Ťažisko sa počíta v reálnom čase. Pohľad robota je posunutý na jeho torzo (a nie v hlave).

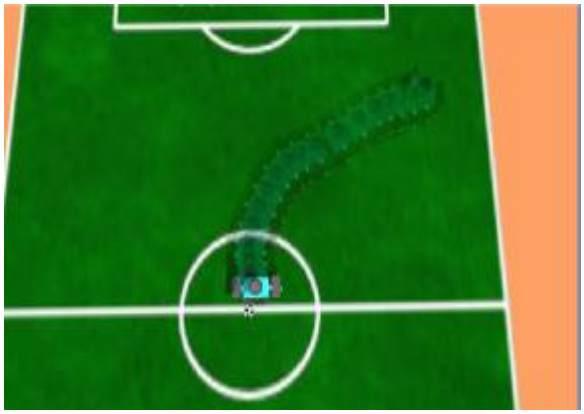
Ovládač vše smerového chodenia agenta má nasledujúcu štruktúru.



Obr. 13 Model ovládača chôdze

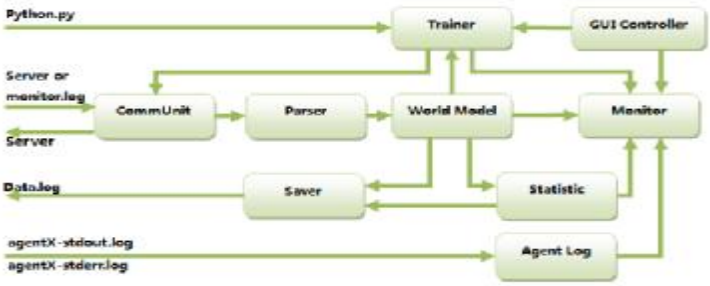
Model ovládača dvojnohej chôdze - Obsahuje viacero vrstiev. *Walking Path Planner* obdrží súradnice a smer chôdze resp. cieľu chôdze. Naplánuje pohyb a odošle ho do *Gait Primitive Generator*. Tento vygeneruje ďalší primitívny pohyb. Tento sa odošle do *Limb Controller* ktorý určí potrebné ohyby kĺbov a potrebné akcie odošle do *Joint Motor Controler* ktorý pohyb kĺbov vykoná.

Výsledkom je hladký neprerušovaný pohyb do všetkých smerov. Robot sa tak nepotrebuje zastaviť keď chce zmeniť smer.



Obr. 14 Kontinuálny pohyb robota

Ako vývojový nástroj bolo použité prostredie *seu-3d-toolkit*. Slúži ako server i nástroj na simuláciu a trénovanie robotov. Z popisu robota nebolo zrejmé či si toto prostredie navrhli sami členovia tímu alebo či sa jedná o určité všeobecné testovacie a vývojové prostredie.



Obr. 15 štruktúra seu-3d-toolkit.

Hlavné výhody tohto prostredia:

- záznam zápasu v priamom aj spätnom smere
- prezeranie výstupov z agenta (logy) v danom čase
- kontrola kamery: pevná alebo napojená na objekt
- „motion blur“ pre indikáciu pohybu robota
- podpora tvorby a analýzy modelu robota
- všetky objekty sveta sú dostupné prostredníctvom stromu objektov na obrazovke
- nezávislosť na platforme

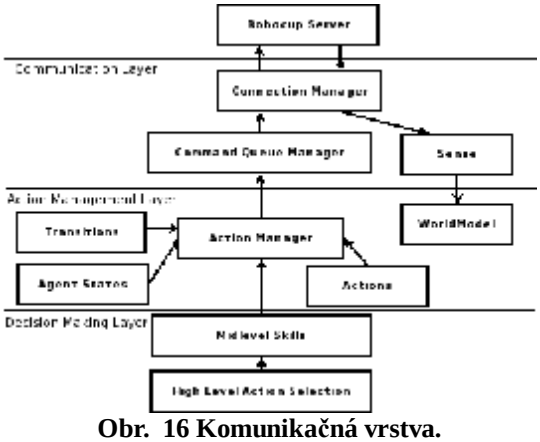
Ako vidno zo schémy agenta tento už pravdepodobne okrem základných pohybov dokázal vykonávať aj vyššie funkcie, o čom nasvedčuje aj fakt že sa tento tým umiestnil v Atlante na treťom mieste.

2.3.3 UI-AI [10]

Pohybový model robota a jeho základné schopností boli definované deterministickým konečným automatom. Jeden stav automatu tvorili polohy jednotlivých častí tela a uhly kĺbov. Tento jednoduchý dizajn v sebe však priniesol niekoľko úskalí:

- akcie robota nebolo možné rozdeliť na menšie podakcie ktoré by boli znovupoužiteľné.
- Robot nebol schopný opustiť vykonávanú akciu uprostred a začať vykonávať inú.
- Agent sa nebol schopný okamžite prispôsobiť aktuálnym údajom zo senzorov.

Architektúra agenta bola rozdelená na 3 vrstvy



Obr. 16 Komunikačná vrstva.

Communication layer - Vrstva sa stará o komunikáciu so serverom. Beží paralelne spolu s ostatným

kódom. S každou prichádzajúcou správou aktualizuje svet robota. Ak akčná vrstva vyžaduje vykonanie príkazov táto vrstva zabezpečí ich vhodné usporiadanie v rade príkazov a následné postupné odosielanie serveru na spracovanie.
Akčná vrstva. Táto vrstva zabezpečuje základné zručnosti agenta ako chodenie, otáčanie sa, kopanie a pod. Je rozdelená na štyri časti:

- *Stavy agenta*: Tu sú popísané stavy agenta. Každý stav môže byť opísaný buď veľmi konkrétne ako napr. polohami jednotlivých častí tela, alebo môže byť opísaný aj abstraktne formou podmienok ohraničujúcich určité hodnoty.
- *Prechody*: definujú prechody medzi stavmi. Jeden prechod medzi stavmi jeden a dva znamená že robot sa vie dostať zo stavu jeden do stavu dva.
- *Akcie*: Obsahujú základné schopnosti robota ako chodenie , otáčanie sa, postavenie sa a kopanie. Každá akcia je definovaná ako určitá cesta po množine stavov. Tieto stavy musia medzi sebou mať definované prechody. Pre opakujúce sa pohyby sa využíva opakujúci sa cyklus.
- *Manažér akcií*: Stará sa o prepínanie akcií medzi sebou. V súčasnej verzií nie je možné okamžite sa prepnúť na inú akciu ak sa už vykonáva iná akcia.

Ďalšou vrstvou je *Vrstva rozhodnutí*. Implementuje stredné schopnosti agenta a stará sa o radenie akcií za sebou tak aby sa dosiahol vyšší celok. Príkladom sú funkcie kopni-loptu-smeron, choď-na-miesto, chyť-loptu a pod.

Agent disponoval nasledujúcimi schopnosťami:

- *Postavenie sa*. Agent využíva ľudský spôsob postavenia sa. Táto metóda fungovala primerane pre všetky náhodne vygenerované polohy.
- *Chôdza.* Na základe metódy určenia a udržovania ťažiska tím určil tri základné kroky algoritmu pre chôdzu:
    - *nakláňanie sa do strán* – agent prenesie ťažisko na podpornú nohu, odrazením sa od zeme chodidlom druhej nohy.
    - *posunutie nohy dopredu* – druhá noha sa posunie vpred o určenú dĺžku.
    - *posunutie tela* – ťažisko agenta sa po minulom kroku posunie a agent tak musí pohnúť celou vrchnou časťou tela na vyváženie

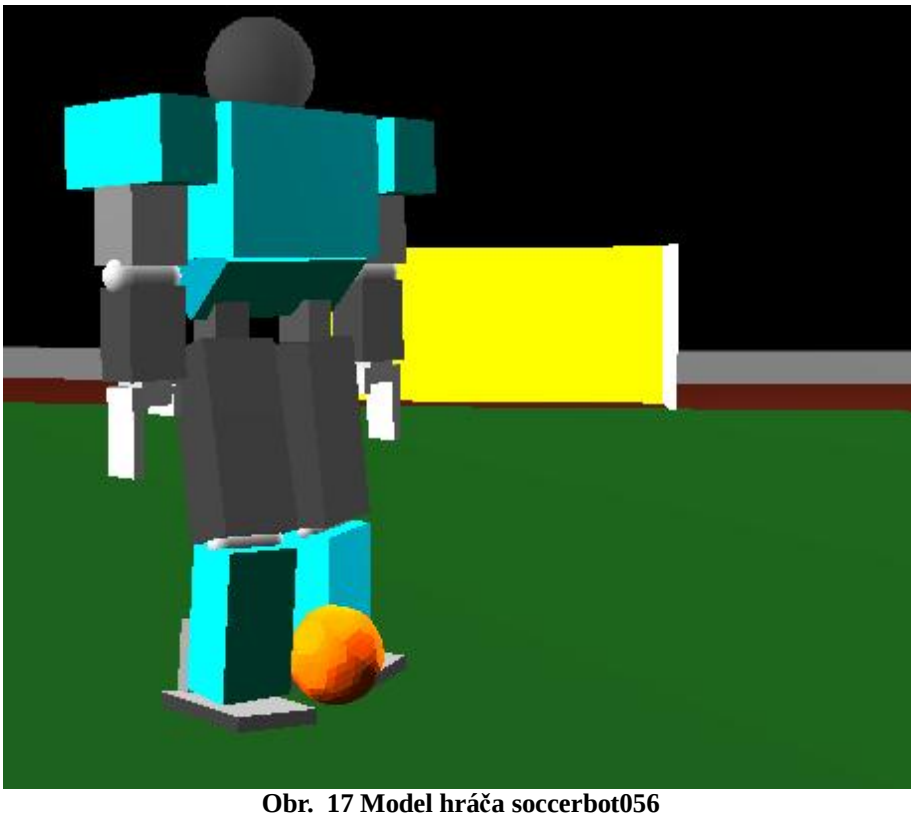
Táto metóda napriek svojej stabilite je veľmi pomalá.

Vývojový nástroj
Tím si vyvinul vlastný nástroj na ovládanie robota za pomoci klávesnice. Bolo možné jednoducho nastaviť ohyby všetkých kĺbov a rýchlosť pohybu. Pomocou tohto nástroja potom vytvorili stavy ktoré boli obsiahnuté v množine stavov v akčnej vrstve agenta ako aj prechodov medzi nimi.

2.3.4 Agent Zigorat [6]

Skupina Zigorat je výsledkom spolupráce viacerých univerzít. Cieľom celého projektu je kooperácia výskumu robotiky medzi univerzitami. Ideou je vytvoriť multi-agentový systém pre výskum širokého spektra algoritmov umelej inteligencií.

Hráč Zigorat je modelom „soccerbot056.rsg“.



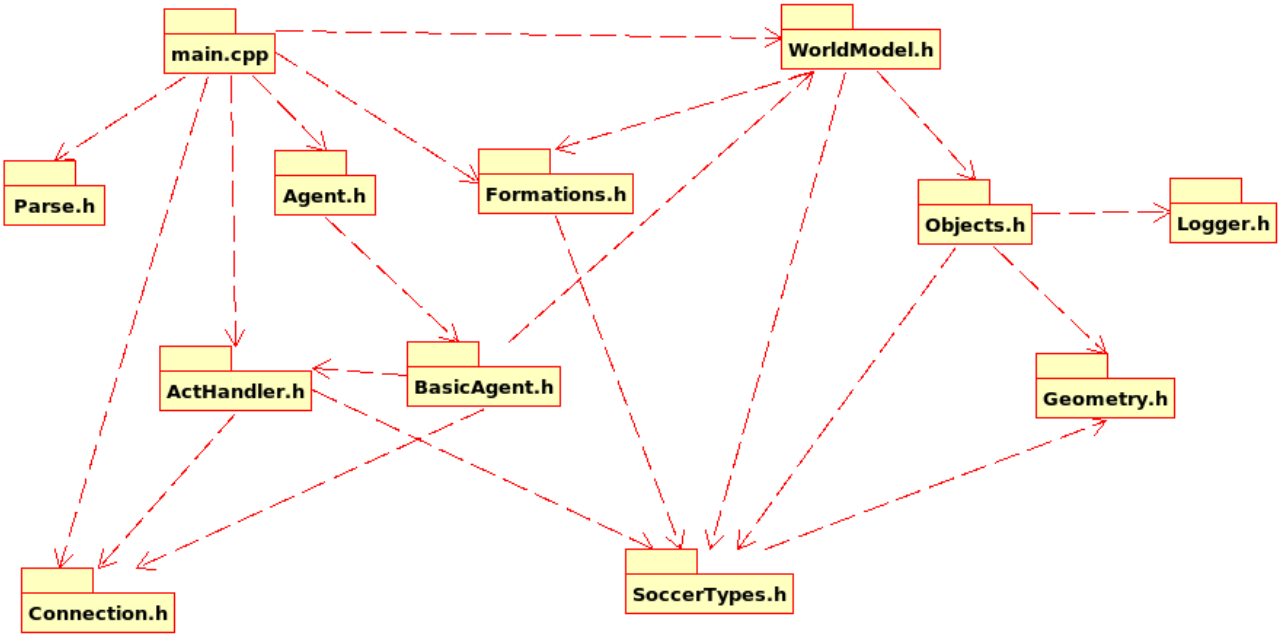
Obr. 17 Model hráča soccerbot056

Na Obr. 18 je znázornené poprepájanie zdrojového kódu na úrovni súborov. Prezeranie začína v súbore main.cpp, ktorý obsahuje main funkciu aplikácie. Tu sa deklarujú všetky potrebné globálne objekty tried:

- *TroboCupConnection [Connection.h]* – poskytuje nadviazanie klienta na server pomocou soketov cez TCP/IP protokol. Prednastavená IP servera je localhost a 3100 port.
- *Formations [Formations.h]* – slúži na určenie rozostavenia hráčov v poli. Rôzne formácie ako inicializačná, ofenzívna, defenzívna apod. sú zaznamenané v súbore *formations.conf*. Každý hráč zastupuje určitú rolu v rámci formácie, ktorá je určená v konfiguračnom súbore formácií a označená číslom (brankár, obranca, útočník).
- *WorldModel [WorldModel.h]* – objekt tejto triedy poskytuje agentovi všetky potrebné informácie o svojom okolí, rozmery ihriska, bránky, aký je aktuálny čas simulácie, pozície rohových zastávok, informácie o kĺbových spojoch agenta, jeho energiu a pod.

- *ActHandler [ActHandler.h]* – slúži na zasielanie úkonov / príkazov na server. Úkon je určitá abstrakcia, zovšeobecnenie. Trieda dané úkony pred zaslaním na server prepíše do syntaxe príkazu. Obsahuje zásobník úkonov, ktorý ich akumuluje a jedným príkazom sa odošlú všetky na server. Je možné zasielať príkazy aj úkony na server osobitne.

Agent [Agent.h] – objekt triedy *Agent* informuje server, aký model agenta využíva, pracuje s telom agenta ovládaním jeho spojov, t.j. vkladá jednotlivé úkony do zásobníka, ktoré sa odošlú naraz na server. Tu sa aj nachádzajú metódy rôznych akcií, jedna z nich je aj chôdza riešená Zigorat agentom. Podstatnou metódou je mainLoop, ktorá vytvára cyklus ukončovaný správou zo servera pri ukončení simulácie. V cykle prebieha rozhodovanie sa agenta na základe prijatých nových informácií prostredia zo servera (ktoré rozparsuje a uloží do svojich lokálnych štruktúr), naplánovanie úkonov do zásobníka a ich odoslanie na server.



Obr. 18 Poprepájanie zdrojového kódu na úrovni súborov

Chôdza Zigorat agenta

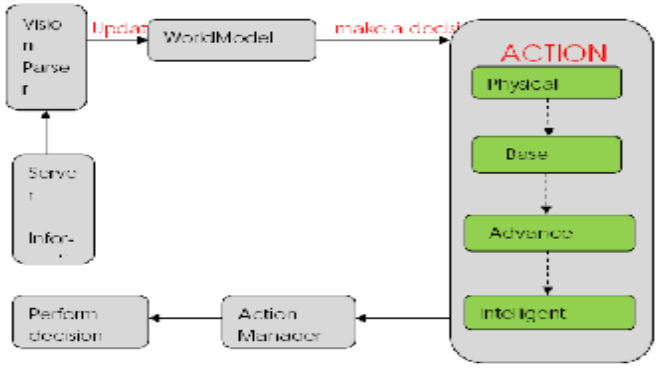
Je definovaná staticky, agent slepo vykonáva úkony bez akejkoľvek dynamickej korekcie. Úkony tela, ako zdvihnutie kolena, posunutie ruky apod., sú plánované podľa relatívneho času simulácie, čo je problém ak príkazy neprijal server v danom potrebnom simulačnom čase. Postupnosť úkonov sa tak naruší a agent padá. Ďalej ak agent narazí na nečakanú prekážku, chôdza sa neprispôsobí na vyrovnanie rovnováhy. Takto určená chôdza je vhodná a úspešná pri bezchybnom spojení a priamej ceste bez prekážok, kde tím agenta Zigorat našli vhodnú postupnosť správne určených úkonov tela pre priamočiary pohyb agenta.

Zigorat sa umiestnil na druhom mieste v robocup3d na turnaji v Atlante a v Teheráne na RoboCup IranOpen 2007 sa dostal až do finále.

2.3.5 DNU Explorer [7]

Hráč sa skladá z niekoľkých častí:

1. *Vision Parser*: Pre transformovanie informácií získaných zo serveru na informácie použiteľné pre "WorldModel".
2. *WorldModel*: Používa informácie z "Vision Parser" pre zistenie aktuálnych informácii o agentovi a hracom poli, a na podľa nich robí rozhodnutia.
3. *Action Manager*: Dáva konečné rozhodnutia podľa priority jednotlivých akcií, a či sa majú vykonať alebo nie.
4. *Action Perform*: Predá konečné rozhodnutia do vykonávacej sekvencie na vykonanie.



Obr. 19 Schéma agenta DNU Explorer

Action Manager:
3 1. vytvorí sekvenciu akcií - izoluje každú akciu na detailné kroky a vytvorí z nich sekvenciu
4 2. filtruje akcie
   a. posúdi, či sa má akcia vykonať alebo nie. Ak áno vyprázdni sa *vykonávacia sekvencia* a vloží sa do nej daná akcia. Rozhodnutie sa vykonáva podľa priority akcie.
   b. ak je výsledok predošlého kroku neúspešný, skontroluje sa *vykonávacia sekvencia*, či je prázdna alebo nie. Ak je prázdna vloží sa do nej daná akcia, v opačnom prípade sa vykonajú všetky akcie z *vykonávacia sekvencia* a potom sa do nej vloží daná akcia.
3. vykoná všetky akcie z *vykonávacej sekvencie*

4.1.1 Fantasia [8]

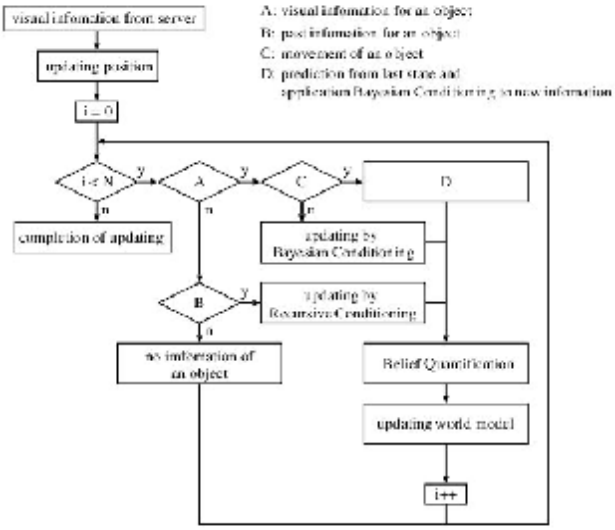
Hráč Fantasia sa zúčastnil RoboCup China Open 2006, kde vyhral prvé miesto v kategórií RoboCup simulation 3D. Využíva koncept *Zero Moment Point* (ZMP) – bod nulového pohybu pre zabezpečenie dynamickej stability dvojnohého robota. ZMP je definovaný ako bod na podlahe, v ktorom je suma všetkých momentov síl je rovná nule. Ak je ZMP vo vnútri konvexného obalu všetkých dotykových bodov medzi chodidlami a podlahou, je možné aby dvojnohý robot chodil. Ďalej sa tento konvexný obal všetkých kontaktných bodov volaný *stabilná oblasť*.

Robot je ovládaný zmenou uhlovej rýchlosti každého kĺbu. Ovládame ho pomocou kľúčových krokov (key frames):

Kďe ™ znamená čas trvania m-tého kroku, T_{mn} znamená uhol n-tého kĺbu v m-tého kroku. Kroky medzi týmito krokmi sú približne určené pojitou funkciou. Uhly kĺbov a dobu trvania korok môžu byť ďalej optimalizované pomocou EDA algoritmu (Estimation of Distribution Algorithm).

4.1.2 Naito Striker [9]

Pri tvorbe tohto hráča sa jeho tvorcovia zamerali na vybudovanie akéhosi druhu databázy tak, aby autonómny agent mohol správne vyhodnotiť informácie o ostatných hráčoch a lopte. Používajú vylepšený M. Saxena a T. Guputaov model:

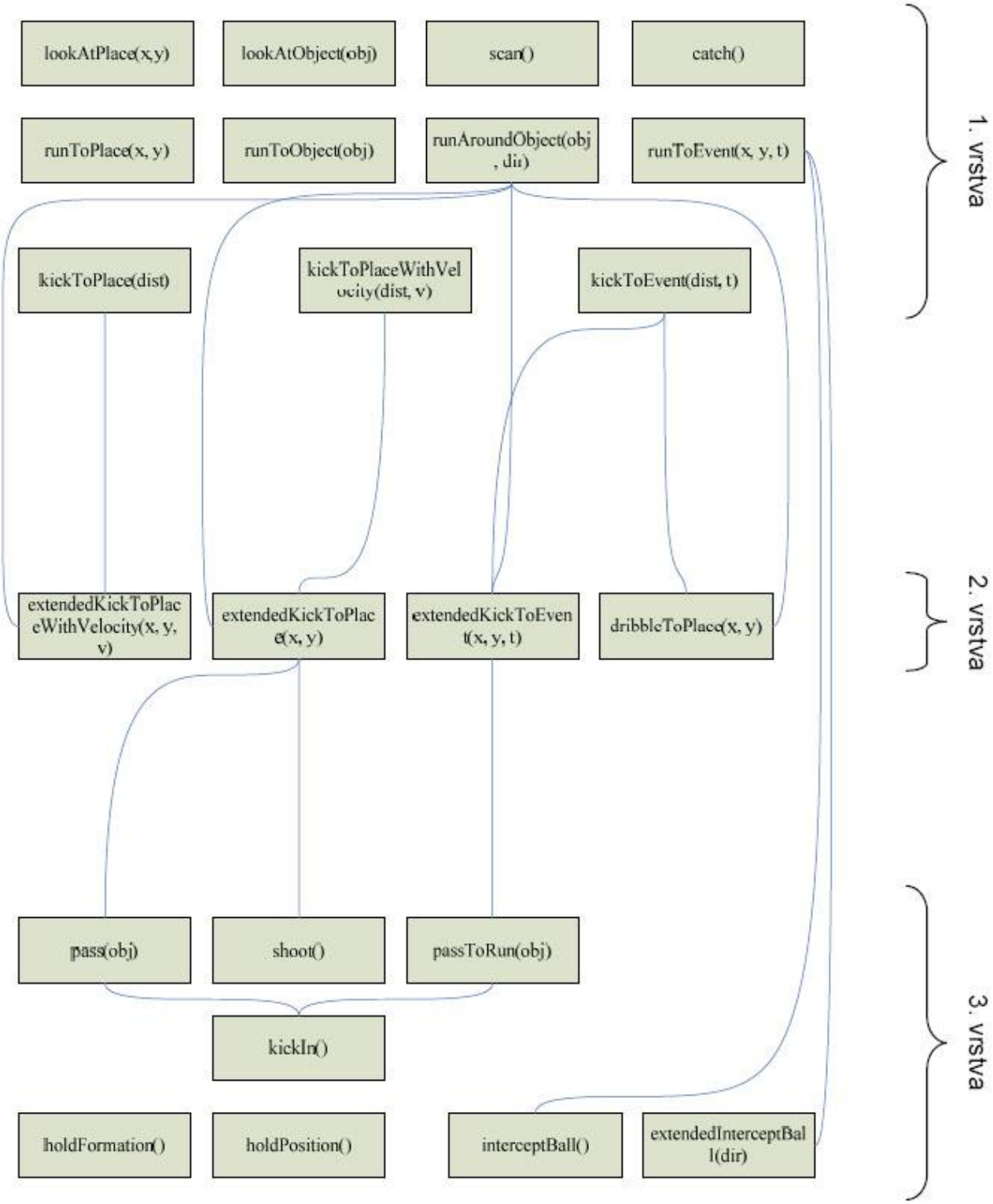


Obr. 20 Schéma agenta Naito Striker

4.1.3 Agent Hazard

Jedná sa o hráča z roku 2006 vytvorenom na predmete Tvorba softvérového systému v tíme. Cieľom bolo vytvoriť hráča na základe práce tímu z roku 2005 s pokročilejšími schopnosťami a vyspelejším rozhodovaním.

Moduly správania
Správanie sa hráča je robené prostredníctvom modulov a rozdelené do troch vrstiev na obr. 21.



Obr. 21 Návrh hierarchie modulov správania

Pre moduly správania vrstvy *n* platí, že využívajú iba moduly správania nižších vrstiev (t.j.*1...n-1*). Takéto rozdelenie je výhodné z hľadiska vyjadrenia závislostí.

Popis jednotlivých modulov:

Moduly správania prvej vrstvy	
Medzi základné moduly správania patria: • moduly pre beh hráča • moduly pre kopanie • moduly pre pozorovanie prostredia	
runToPlace(x, y)	Beh na miesto (x,y)
runToObject(obj)	Beh k objektu obj.
runAroundObject(obj, dir)	Beh k objektu obj. Pri dobehnutí k objektu má mať spojnica ťažísk objektu a hráča smerový uhol dir.
runToEvent(x, y, t)	Dobehnutie na miesto (x,y) v case t.
kickToPlace(dist)	Kopnutie na vzdialenosť danú parametrom dist. Kope sa smerom v ktorom je hráč natočený.
- kickToEvent(dist, t)	Kopnutie na vzdialenosť danú parametrom dist. Kope sa smerom v ktorom je hráč natočený. Lopta má doraziť na cieľové miesto v čase t.
kickToPlaceWithVelocity(dist, v)	Kopnutie na vzdialenosť danú parametrom dist. Kope sa smerom v ktorom je hráč natočený. Lopta má doraziť na cieľové miesto s rýchlosťou v.
lookAtPlace(x,y)	Hráč natočí kameru tak, aby zadané miesto bolo v strede jeho zorného pola.
lookAtObject(obj)	Hráč natočí kameru tak, aby zadaný objekt bol v strede jeho zorného pola.
- scan()	Hráč otáča kameru tak, aby postupne zaznamenal celé svoje okolie.

Tab. 2 Moduly správania prvej vrstvy

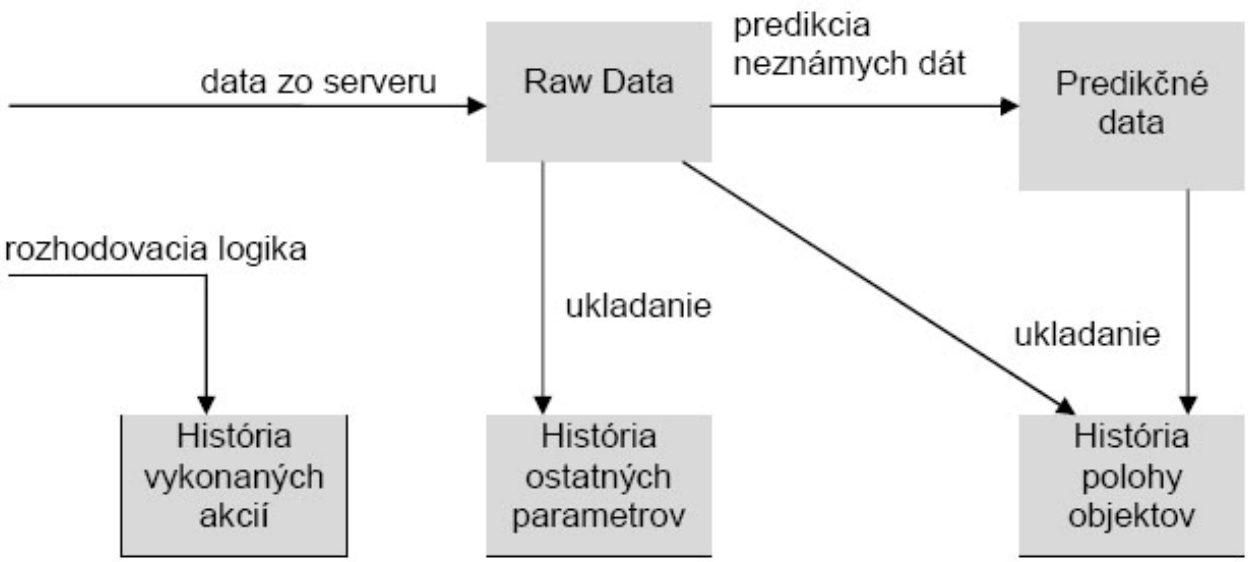
Moduly správania druhej vrstvy	
Medzi rozšírené moduly správania patria rozšírené moduly kopania a modul driblovania s loptou.	
extendedKickToPlace(x, y)	runAroundObject + kickToPlace
extendedKickToEvent(x, y, t)	runAroundObject + kickToEvent
extendedKickToPlaceWithVelocity(x, y, v)	runAroundObject + kickToPlace
dribbleToPlace(x, y)	Beh s loptou na dané miesto. Hrác musí mat pred zaciatkom tohto správania loptu vo svojej blízkosti.

Tab. 3 Moduly správania druhej vrstvy

Moduly správania tretej vrstvy	
pass(obj)	Prihrávka objektu obj. Hráč musí mať pred začiatkom tohto správania loptu vo svojej blízkosti
passToRun(obj)	Prihrávka do pohybu objektu obj. Hráč musí mat pred začiatkom tohto správania loptu vo svojej blízkosti.
- shoot()	Strela na bránu. Hráč musí mat pred začiatkom tohto správania loptu vo svojej blízkosti. holdPosition() - Hráč zostáva stať na mieste
holdFormation()	Hráč sa presunie na svoje miesto vo formácii.
kickIn()	Vkopnutie lopty do hry
interceptBall()	Prerušenie pohybu lopty.
extendedInterceptBall(dir)	- Prerušenie pohybu lopty a nastavenie sa do polohy, pri ktorej má spojnica lopty a hráča uhol dir.

Tab. 4 Moduly správania tretej vrstvy

Agentov model sveta je reprezentovaný tzv. *information storage* znázornenom na obrázku



Obr. 22 Information Storage

Raw data – je vrstva, ktorá obsahuje a pracuje s informáciami, ktoré prichádzajú zo servera a sú (odhliadnuc od možného šumu) presné.
Predikčné dáta – sú generované prediktorom, sú nevyhnutné na tvorbu stratégie.
História polohy objektov – v tejto časti sú uložené všetky súradnice objektov od začiatku simulácie, sú to informácie o polohe v polárnych súradniciach ako prichádzajú zo servera a tiež prepočítané pravouhlé súradnice. Taktiež je tu história predikovaných súradníc.
História vykonaných akcií – všetky akcie vykonané daným hráčom v chronologickom poradí.
História ostatných parametrov – stav batérie, teplota, uhol pohľadu, mód hry.

Hráč ako taký je schopný samostatnej hry. Implementácia z časových dôvodov však nebola kompletná. Niektoré moduly sú vytvorené len na úrovni prototypu.

4.2 Zhodnotenie analýzy

Podrobne sme si rozobrali simulačné prostredie v ktorom prebiehajú zápasy RoboCup-u. Zistili sme si aktuálny model humanoidného robota. Je to pomerne veľký skok od predošlej verzie servera s ktorou pracoval napríklad tím v minulom roku kedy model robota tvorila len guľa. Aktuálny model má humanoidnú formu a disponuje niekoľkými kĺbmi po celom tele, pričom kĺby môžu byť dvoch rôznych druhov. Zistili sme že komunikácia medzi robotmi a serverom prebieha na protokoloch TCP alebo UDP je teda možné (a na súťaži nutné) spustiť svoj tím na osobitnom počítači a po sieti sa pripojiť na počítač kde beží serverová aplikácia. Analýza rôznych tímov ukázala aké rôznorodé riešenia je možné použiť pri tvorba agentov. Spomedzi analyzovaných agentov sme si pre ďalšiu prácu vybrali dvoch a to konkréte agenta Hazarda z minulého roku od tímu 6th sense, a agenta Zigorata. Agent hazard disponuje výhodným modelom správania sa organizovaným do vrstiev z hľadiska úrovne správania sa a ďalej do modulov. Táto architektúra sa javí ako výhodná, flexibilná a ľahko rozšíriteľná. Bohužiaľ agent hazard je postavený na staršom type servera a preto nemôžeme jeho kód využiť celý. Preto sme hľadali ďalej a ako už bolo spomenuté ako druhého kandidáta sme si vybrali agenta Zigorata. Tento agent je stavaný na nový typ serveru a poskytuje vynikajúci model komunikácie so serverom. Agent taktiež disponuje základnými pohybovými vlastnosťami ako je otáčanie sa a chôdza tie sa však pri našich testoch javili ako pomerne nestabilné. Rozhodli sme sa použiť oboch agentov a z každého zobrať to najlepšie.

5 Špecifikácia požiadaviek

Cieľom projektu bude navrhnúť prototyp hráča resp. časti hráča. Tento prototyp by mal byť schopný fungovať na novom type servera a mal by mať formu humanoidného robota. Opierať sa budeme o idei tímu z minulého roka v predmete TSST, ako aj našej analýzy iných svetových tímov.
Budeme sa sústrediť na základné schopnosti hráča. Nový typ serveru ako aj humanoidný hráč majú iné charakteristiky ako ich staršie verzie. Úlohou bude základná orientácia hráča v priestore, jednoduchý pohyb (chôdza) a schopnosť postaviť sa.

6 Hrubý návrh

Z analýzy predošlých agentov sme usúdili, že použitie statických metód na riadenie jednotlivých funkcií agentov nie je vhodné. Z tohto dôvodu sme si zvolili použitie evolučného algoritmu. Keďže táto metóda ešte nebola použitá rozhodli sme sa špecializovať len na jednu oblasť súťaže development a tou je vstávanie agenta z ľubovoľnej polohy.

Z analýzy štruktúr agentov vyplynulo, že najvhodnejšie bude skombinovať komunikačnú vrstvu ľubovoľného agenta a vytvoriť, respektíve prevziať architektúru modelu správania sa z agenta Hazarda. Tento postup sme zvolili z dôvodu veľmi kvalitnej štruktúry modelov správania sa v agentovi Hazardovi. Aj keď je agent vytvorený na starú generáciu RoboCup serverov (guličky), všeobecná štruktúra jeho modelov správania sa dá aplikovať aj na novú generáciu agentov.

Základnou myšlienkou nášho agenta bude využitie evolučného algoritmu na dosiahnutie cieľa. Naším cieľom je postavenie sa do polohy, ktorá bude vhodná na ďalší pohyb. Treba povedať, že táto poloha nebude určená staticky, ale bude sa odvíjať od celkovej situácie a potrieb, ktoré budú určené vyššou plánovacou vrstvou.

Pre príklad: agent stratí rovnováhu a spadne. Pred spadnutím bolo jeho cieľom priblížiť sa k lopte. Agent zahájí model správania ktorý zabezpečí postavenie, pričom však výsledným cieľom tohto kroku nebude statické rozloženie častí tela, ale taká poloha, z ktorej sa dá najefektívnejšie pokračovať v nadradenom pláne, t.j. poloha z ktorej agent najlepšie zahájí pohyb k lopte.

6.1 Štruktúra agenta

Agent neurotik sa bude skladať zo štyroch hlavných častí:

1. Hlavný vykonávací cyklus
2. Komunikačný modul
3. Pohľad na svet (worldview modul)
4. Modely správania sa

Hlavný vykonávací cyklus bude samotné jadro agenta. V tejto časti agenta sa bude vykonávať volanie jednotlivých podmodulov ktoré zaručia danú funkčnosť.

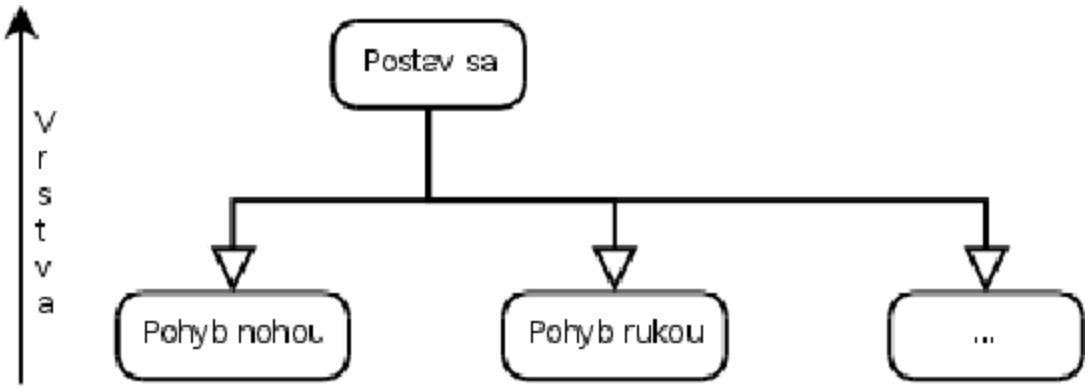
Komunikačný modul agenta bude prevzatý z agenta Zigorata. Modul bude upravený pre naše potreby, hlavne plánujeme zmeny v rozhraní, funkčnosť ostatne nezmenená. Modul sa bude starať o posielanie a prijímanie správ zo servera.

Pohľad na svet sa prevezme z agenta Zigorata. Pre naše potreby budú zmeny v tejto časti minimálne.

Agent Neurotik bude využívať štruktúru modelov správania sa (behaviors) z agenta Hazarda. Štruktúra je založená na modularite a jednoduchosti jednotlivých modelov.

Každý model je reprezentovaný triedou, ktorá je zdedená z abstraktného rozhrania BaseBehaviorModule. Toto rozhranie obsahuje metódy, ktoré sa v konkrétnych modeloch implementujú. Pomocou makier REGISTER_MODULE(meno) sa jednotlivé moduly zaregistrujú do systému. Makrom USE_MODULE(meno, cesta) sa modul prihlási ako aktívny na použitie.

Samotné moduly sú rozdelené do viacerých vrstiev. Agent Hazard mal 3 vrstvy, pričom platí, že každý model vrstvy *x*, môžu používať len moduly vrstvy *x – 1*.



Obr. 23 Vrstvy modelov udalostí

V našom agentovi tento princíp zachováme, avšak počet vrstiev modelov sme obmedzili na dve. Spodná vrstva bude obsahovať základné modely pre pohyb jednotlivých častí tela, a vrchná vrstva bude obsahovať primitívne modely typu „postav sa“. Systém bude navrhnutý tak, aby bol do budúcnosti rozšíriteľný.

6.2 Model vstávania

Hlavnou úlohou agenta Neurotika v tomto projekte je postavenie sa do vhodnej polohy vzhľadom na celkový plán akcie na vyšších úrovniach. Z analýz sme usúdili, že statický prístup k problémom rovnováhy nie je vhodný, zvlášť pre situáciu postavenia kde nie je možné spoliehať na predpoklady určitej začiatočnej polohy.

Z tohto dôvodu sme si vybrali ako spôsob riešenia použitie evolučného algoritmu.

6.3 Využitie evolučných algoritmov

Evolučné algoritmy využívajú, na riešenie optimalizačných problémov, princípy evolúcie živej hmoty. Simulujú darwinov evolučný proces. Ten obsahuje tieto tri zložky [12]:

1. **Prirodzený výber** – Silnejší jedinci majú väčšiu šancu reprodukcie ako slabší jedinci.
2. **Náhodný genetický drift** – Náhodné udalosti v živote jedincov. Napríklad náhodná smrť silného jedinca pred tým, než dostal šancu na reprodukciu.
3. **Reprodukcia** – Genetická informácia potomkov vzniká kombináciou génov rodičov, pričom môže dochádzať k jej poškodeniu (mutácii).

Simulovaná evolúcia prebieha nad určitou množinou jedincov, v ktorej sa cyklicky opakuje výber rodičov a reprodukcia. Po reprodukcii vzniká nová množina jedincov a evolúcia ďalej prebieha nad touto novou množinou.

Takouto simulovanou evolúciou môžeme napríklad naučiť hráča pohybovať sa. Pohyb hráča je v podstate zložený z množstva pohybov jeho motorčekov. Celý pohyb sa teda dá definovať ako chronologická postupnosť rýchlostí otáčania všetkých hráčových motorčekov. Na začiatok by bolo vhodné otestovať možnosti aplikácie niektorého z evolučných algoritmov na jednoduchšom pohybe. Napríklad na postavení sa.

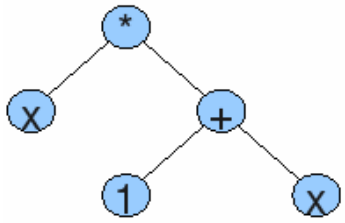
Postavenie sa pomocou evolučného algoritmu
Pre realizáciu simulovanej evolúcie je potrebné zadefinovať nasledovné:

- jedinca a jeho reprezentáciu
- spôsob ohodnocovania schopnosti jedincov prežiť (fitness)
- spôsob výberu rodičov
- operácie kríženia a mutácie
- spôsob výberu jedincov do novej generácie

6.3.1 Reprezentácia jedinca

Jedinec je definovaný genetickou informáciou. Tá musí (pre potreby evolúcie pohybu) definovať pohyb robota. Bude ju teda tvoriť chronologická postupnosť základných pohybov robota (pohybov jednotlivých kĺbov), indexovaných časom, kedy sa tento pohyb začne vykonávať. Dĺžka chromozómu nie je dopredu známa. Preto je vhodné použiť tzv. „Messy"-chromozómy [13] (m-chromozómy). Tie majú premenlivú dĺžku. Samotný chromozóm je tvorený postupnosťou usporiadaných dvojíc, zložených z indexu a hodnoty génu. Ak sa v chromozóme nachádza viac génov s rovnakým indexom, použije sa len jeden z nich, a to prvý v poradí. V tomto konkrétnom prípade by index tvoril čas a hodnotu génu zoznam vstupov pre metódy zabezpečujúce pohyb jednotlivých kĺbov.

Robot má senzory, ktoré mu poskytujú nejaké informácie o okolitom priestore. Bolo by teda vhodné, aby robot riadil svoj pohyb na základe týchto informácií. Pohyby by teda mali byť relatívne vzhľadom na tieto informácie (aspoň na niektoré z nich). Ale ako? Dopredu určiť túto závislosť je nemožné. Mohla by teda tiež byť predmetom evolúcie. Elementy zoznamu hodnôt jednotlivých génov by nemali byť len obyčajné čísla. Mohli by to byť funkcie, ktoré umožnia ich výpočet na základe robotových informácií o okolí. Problémom tohto typu sa zaoberá metóda *Genetického* programovania [13]. V nej je gén tvorený koreňovým stromom, kde sú vrcholy stromu ohodnotené funkciou a listy hodnotami.



Obr. 24 Príklad koreňového stromu. Funkcia F(x)=x*(1+x)

Jedinec teda bude reprezentovaný m-chromozómom, v ktorého génoch bude index reprezentovať čas a hodnotu zoznam koreňových stromov.

6.3.2 Fitness

Fitness označuje schopnosť prežitia a reprodukcie jedincov. Spôsob ohodnocovania fitness teda určuje kam bude evolúcia smerovať. V tomto prípade chceme aby sa robot čo najrýchlejšie postavil tak, aby mohol čo najskôr vykonať inú akciu (musí stáť na zemi) a aby pri tom minul čo najmenej

energie. Do výpočtu fitness teda vstupujú nasledovné parametre:

1. Rozdiel maximálnej dosiahnutej výšky hlavy a výšky hlavy vo vzpriamenej polohe
2. Čas za ktorý sa do tejto výšky dostal ako dlho v nej zotrval (či potom nespadol)
3. Množstvo energie potrebnej na postavenie sa.

Parameter číslo jedna je rozdiel z dôvodu zamedzenia vyskakovania do čo najvyššej výšky.

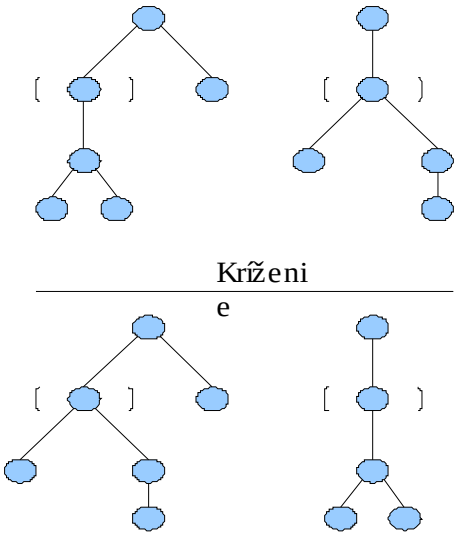
Presné závislosti a podiely jednotlivých parametrov sa nedajú dopredu presne určiť a budú určené pri testovaní a ladení.

6.3.3 Spôsob výberu rodičov

Po ohodnotení jedincov hodnotou fitness, je nutné vybrať niekoľko jedincov, ktorý pristúpia k reprodukcii. Stratégií ne tento výber je niekoľko. Napríklad výber najlepších jedincov, turnajový výber, pseudonáhodný výber, … No pseudonáhodný výber imituje aj náhodne udalosti v živote jedincov. Preto budeme používať túto stratégiu, konkrétne implementovanú pomocou ruletového výberu.

6.3.4 Operácie kríženia a mutácie

Operácie kríženia musia byť definované tak, aby zasahovali chromozóm aj jednotlivé gény v chromozóme. Začneme teda tieto operácie definovať od najnižších vrstiev.
Kríženie dvoch koreňových stromov spočíva vo výmene podstromov. V obidvoch stromoch sa nezávislé od seba náhodne zvolí bod kríženia. A podstromy začínajúce v týchto bodoch kríženia sa jednoducho vymania.



Obr. 25 Príklad kríženia dvoch stromov. Zátvorky označujú náhodne zvolené body kríženia.

Mutáciou stromu budeme rozumieť nahradenie náhodného podstromu iným, náhodne vygenerovaným, stromom. Kríženie dvoch génov bude spočívať v postupnom krížení všetkých ich stromov na zhodných indexoch (prvý s prvým, druhý s druhým, …) a výberu indexu z jedného z nich. Mutácia génu je náhodná zmena indexu a mutácia stromov. Kríženie dvoch m-chromozómov potom bude prebiehať nasledovne. Náhodne sa zvolí bod kríženia a začnú sa krížiť jednotlivé gény. Až po bod kríženia chromozómov sa budú brať indexy z prvého rodiča a za bodom kríženia sa budú brať indexy z druhého rodiča. Mutácia m-chromozómu je náhodné pridanie alebo odobratie génu.

6.3.5 Výber jedincov do novej generácie

V novej populácii budú určite všetci potomkovia, ktorí vynikli pomocou kríženia alebo mutácie z predchádzajúcej populácie. No treba sa rozhodnúť či do nej zahrnieme aj niekoľko jedincov z predchádzajúcej populácie. Napríklad najlepších jedincov. Evolučné algoritmy imitujú evolúciu v prírode. Dalo by sa teda napríklad uvažovať o nejakom veku jedincov.

Ak do novej generácie zahrniem niekoľko najlepších jedincov, ich gény nebudú zabudnuté pri nevhodnej voľbe rodičovských párov a zlom výsledku reprodukcie. No jedince by sa mohli vyberať aj ruletovým systémom.

Presný systém výberu spresníme pri testovaní.

7 Prototyp

Ako prototyp sme zvolili vytvorenie jednoduchého agenta, ktorý bude vedieť komunikovať zo serverom a ukázať funkčnosť modelov udalostí. Jeho modely nebudú ešte spĺňať užitočnú funkciu.

Ciele:

1. Prvým cieľom je preukázať, že nami zvolená kombinácia častí jednotlivých modulov a ich architektúra je implementovateľná.
2. Ďalším cieľom je overenie správnosti komunikačnej vrstvy prevzatej z agenta Zigorata. Treba overiť, či je vrstva kompletná, teda či obsahuje spracovanie všetkých možných príkazov a správ.
3. Posledným cieľom je otestovanie systému modelov udalostí prevzatého z agenta Hazarda, či je dostatočný pre nové prostredie a jeho požiadavky, alebo potrebuje rozšírenia.

8 Literatúra

[1] RoboCup home page
http://www.robocup.org/02.html – oficialna stranka robocuou, (20.10.2007).

[2] Socer Simulation
http://wiki.cc.gatech.edu/robocup/index.php/Soccer_Simulation, (21.10.2007).

[3] Rules for 3D Soccer Simulation Competition during RoboCup 2007 Atlanta,
http://www.uni-koblenz.de/~murray/robocup/rc07/rules/3d_rules.pdf,
1. júl 2007.

[4].. Hamid Reza Mohseni Nejad, Roya Helmmat Pour, Mohamad Mayanani, Amir Kharmandar, Hamid Bakhtiyari, Hojjat Nikan, Hamed Tirdad: Simulation of a Humanoid Soccer Robot, Team Description Proposal for RoboCup 2007, Sama 3D,
http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/Sama3D.pdf

[5] Xu Yuan, Tan Yingzi: SEU-3D 2007 Soccer Simulation Team Description,
http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/SEU-3D-TDP07.pdf

[6] Zigorat RoboCup teams,
http://zigorat3d.googlepages.com/home2

[7] Xichao Xia, Haonam Ma: DNU_Explorer 3D Team Description 2007,
http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/DNU_Explorer_3D_TDP_2007.pdf

[8] Ling Gao, Guangbin Cui, Pu Li, Hongxia Chai, Xiaomin Tang.: Fantasia 2007 Team Description,
http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/fantasia2007tdp.pdf

[9] Yoichi Setoguchi, Nobuhiro Ito: The World Model for Autonomous Soccer Agents, NAITO-Strikers,
http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/NAITO-StrikerS2007.pdf

[10] Saeid Akhavan, Mohammad Babaeizadeh, Hamid Reza Hasani, Arefeh Kazemi, Hoda Safaeipour : UI-AI3D Team Description,
http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/uiai2007TDP.pdf

[11] S-expresion,
http://en.wikipedia.org/wiki/S-expression, (20.10.2007)

[12] Vladimír Kvasnička: Genetický algoritmus,
http://www2.fiit.stuba.sk/~kvasnicka/NeuralNetworks/5.prednaska/GA_background.pdf

[13] V. Kvasnička, J. Pospíchal, P. Tiňo: Evolučné algorikmi, Slovanská technická univerzita v Bratislave, 2000, ISBN 80-227-1377-5

[14] Obst, O., Rollmann - M., Spark, A Generic Simulator for Physical Multi-agent Simulations, 2005.

[15] Smith, R.: Open Dynamics Engine v0.5 User Guide, 2006, http://ode.org/ode-latest-userguide.html#sec_3_5_0, (14.11.2007).

[16] Writing your own agent, Simspark Wiki,http://simspark.sourceforge.net/wiki/index.php/Getting_Started#Linux (14.11.2007).

[17] The RoboCup Soccer Simulator, rcssserver3D 0.5.6, http://sourceforge.net/project/showfiles.php?group_id=24184 (14.11.2007).

[18] The RoboCup Soccer Simulator, Email Archive: sserver-three-d, http://sourceforge.net/mailarchive/forum.php?forum_name=sserver-three-d (14.11.2007).

9 Prílohy

9.1 Príloha A (Príklad dát prijatých zo servera)

```
(time
      (now 2884.39))
(GS
      (t 0.00)
      (pm BeforeKickOff))
(GYR
      (n torso)
      (rt 0.00 0.01 0.04))
(See
      (F1L
            (pol 35.14 126.87 -5.19))
      (F2L
            (pol 21.61 -169.21 -8.45))
      (F1R
            (pol 40.44 44.00 -4.51))
      (F2R
            (pol 29.45 -7.85 -6.19))
      (G1L
            (pol 26.94 141.85 -7.58))
      (G2L
            (pol 22.58 160.35 -9.05))
      (G1R
            (pol 33.55 29.64 -6.08))
      (G2R
            (pol 30.16 14.50 -6.76))
      (B
            (pol 13.04 71.57 -14.00))
      (P
            (team RoboLog)
            (id 10)
            (pol 10.50 90.01 0.00)))
(UJ
      (n laj1_2)
      (ax1 0.00)
      (ax2 -0.00))
(UJ
      (n raj1_2)
      (ax1 -0.00)
      (ax2 0.00))
(HJ
      (n laj3)
      (ax 0.00))
(HJ
      (n raj3)
      (ax 0.00))
(HJ
      (n laj4)
      (ax 0.00))
(HJ
      (n raj4)
      (ax -0.00))
(HJ
      (n llj1)
      (ax -0.00))
(HJ
      (n rlj1)
      (ax 0.00))
(UJ
      (n llj2_3)
```

```
      (ax1 0.00)
      (ax2 -0.00))
(UJ
      (n rlj2_3)
      (ax1 -0.00)
      (ax2 0.00))
(HJ
      (n llj4)
      (ax 0.00))
(HJ
      (n rlj4)
      (ax -0.00))
(FRP
      (n lf)
      (c 0.10 0.08 -0.05)
      (f -0.42 -0.30 12.22))
(UJ
      (n llj5_6)
      (ax1 0.00)
      (ax2 0.00))
(FRP
      (n rf)
      (c -0.10 0.08 -0.05)
      (f 0.41 -0.28 12.30))
(UJ
      (n rlj5_6)
      (ax1 -0.00)
      (ax2 -0.00))
```