Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný odbor: INFORMATIKA

# RoboCup 3D
## Critical Error

Vedúci projektu: Ing. Ivan Kapustík

Členovia tímu:
Bc. Juraj Drahoš
Bc. Ľuboš Gelányi
Bc. Ivan Hujsi
Bc. Jaroslav Chnúrik
Bc. Ján Janík
Bc. Michal Jantošovič
Bc. Maroš Urbanec

Kontakt: critical_error@googlegroups.com


# Obsah


# 1 Úvod

## 1.1 Účel dokumentu

Tento dokument slúži ako projektová dokumentácia pre projekt tímu Critical Error, vytvorený na základe predmetu Tvorba softvérového projektu v tíme v akademickom roku 2009/2010. Čitateľ v nej môže nájsť podrobnosti týkajúce sa východiskových analýz, podklady pre návrhové a implementačné rozhodnutia, ktoré sa na projekte udiali.

## 1.2 Zadanie

Po tom, čo počítačový program dokázal poraziť svetového šampióna v šachu sa ukázalo, že porovnávanie športových schopností človeka a stroja môže byť zdrojom veľkého množstva užitočných vedľajších produktov a znalostí, ktoré sú nevyhnuteľným dôsledkom vynaloženého výskumného úsilia. Z podobných pohnútok vznikol aj projekt RoboCup, ktorý je oficiálne definovaný svojim cieľom - do roku 2050 vyvinúť tím humanoidných robotov, ktorí odohrajú zápas s aktuálne najlepším futbalovým tímom. Napriek tomu, že takýto cieľ je úzko špecifický na doménu futbalu, obsahuje RoboCup aj súťaže pre záchranné roboty alebo súťaže v autonómnej navigácii.

Súťaže v rámci RoboCupu sa dajú rozdeliť na dve hlavné podoblasti - simulačné a robotické. Zatiaľ čo v robotickej časti proti sebe súťažia reálni, fyzickí roboti, v simulačných súťažiach sa abstrahuje od elektrotechnických detailov a súťaž beží ako simulácia na serveri. Server poskytuje hráčovi cez UDP spojenie správy, ktoré slúžia na simuláciu senzorov, ktorými by disponoval skutočný robot. Následne mu hráč odošle príkazy, ktoré sú podmnožinou sady inštrukcií poskytovaných serverom. Pri veľkom zjednodušení si môžeme predstaviť pre 3D robota príkazy pre pohyb kĺbov ako takúto sadu inštrukcií. Fundamentálnym pravidlom pre jednotlivých softvérových hráčov je, že spolu nesmú komunikovať iným spôsobom ako pomocou serverom definovaných inštrukcií. Inak povedané, rozhodovanie sa každého jedného robota musí byť autonómne.

Náš tím sa venuje práci na robotovi schopnom zúčastniť sa 3D humanoidnej futbalovej simulácie na konci akademického roku. V dobe písania tohto dokumentu ešte neexistoval fakultný hráč schopný odohrať simulovaný 3D zápas. Úlohou nášho tímu je vylepšiť existujúcu kostru fakultného hráča, pokúsiť sa vylepšiť niektoré aspekty jeho správania a na týchto základoch vybudovať vyššiu logiku, implementujúcu samotné futbalové „rozmýšľanie" hráča. Paralelne s našim tímom sa o podobné ciele usiluje aj iný fakultný tím, čo by malo vyústiť do prvého simulovaného 3D zápasu na našej univerzite.

## 1.3 Štruktúra dokumentu

Na úvod sú obsiahnuté všeobecné informácie, týkajúce sa tohto dokumentu, projektu RoboCup a samotného cieľu tohto projektu

V druhej časti dokumentu analyzujeme existujúce tímy, pochádzajúce z 2D aj 3D simulovaného futbalu, s cieľom hľadať oboznámenia sa s doménou, nájsť v existujúcich tímoch zaujímavé a podnetné myšlienky, ktoré by sme mohli v našom projekte využiť. Dôraz je kladený na analýzu tímu Dream Team, na ktorom sme sa rozhodli nášho hráča vyvíjať, a na tím Agenty007, taktiež fakultným tímom z predchádzajúceho akademického roku, zaujímavým najmä z hľadiska výsledku ich práce - editora pohybov, vďaka ktorému sa zjednodušuje práca spojená s modelovaním pohybov.


Nasleduje špecifikácia požiadaviek na prototyp, ktorý bude výsledkom nášho úsilia v zimnom semestri.

Posledná časť dokumentu obsahuje hrubý návrh zmien, ktoré máme za cieľ uskutočniť počas vývoju prototypu v zimnom semestri.


# 2 Analýza

V tejto časti dokumentu sa venujeme skúmaniu existujúcich 2D aj 3D tímov za účelom zoznámenia sa s problémovou doménou, trendmi prevládajúcimi pri riešení problémov spojených s architektúrou, prehľadávaním stavového priestoru, implementovaním elementárnych pohybov, ako aj za účelom oboznámenia sa s nástrojmi a prostriedkami, ktoré sa pri vývoji robotov používajú.

## 2.1 Analýza servera

Pre kompletnú analýzu servera pozrite prosím dokumentáciu tímu Dream Team [20], V danej časti sú popísané iba zmeny, ktoré nastali oproti staršej verzii.

### 2.1.1 Server 0.6.0 -> 0.6.2

Pri prechode na novú verziu servera sa zmenila množina informácií zasielaných serverom. Zmeny sa týkajú hlavne správy SEE. Tá označuje objekty, ktoré robot vidí. Drobné zmeny nastali aj v tom, kedy je táto informácia zasielaná. Všetky zmeny sú zhrnuté v tabuľke Tab. 1.

| Vlastnosť | Server 0.6.0 | Server 0.6.2 |
|---|---|---|
| **Zasielanie SEE v prvej správe** | Správa obsahuje informácie o bránkach a lopte | Správa obsahuje informácie o objektoch, ktoré robot skutočne vidí |
| **Zasielanie SEE v ďalších správach** | SEE je v každej správe | SEE je v každej treťej správe |
| **Umiestnenie SEE v rámci správy** | SEE je pred kĺbmi | SEE je za kĺbmi |
| **Obsah SEE** | Položka „pol“ určujúca referenčnú pozíciu objektu | Robot vidí každú končatinu zvlášť, dokonca aj svoje vlastné |

Tab. 1 Zmeny v parseri

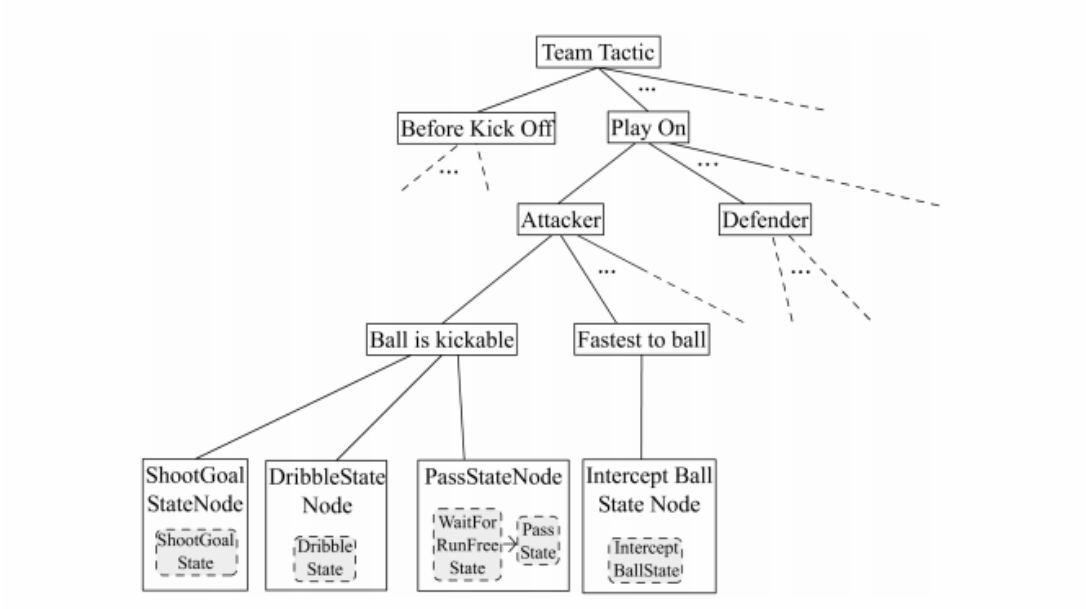
## 2.2 Analýza 2D tímov

Analyzovali sme domáce (12.Hráč, Gang of Six, Jahodoví Princovia) aj zahraničné (DAInamite, UvA Trilearn) tímy. Pri analýze 2D tímov sme sa sústredili na vyššiu logiku, architektúru, spôsob výberu akcie a koordináciu činnosti.

### 2.2.1 DAInamite

DAInamite je nemecký tím, ktorý je zaujímavý tím, že bol ako prvý tím pre RoboCup 2D vyvíjaný pod programovacím jazykom Java. Tím poskytuje základnú sadu zdrojových kódov k dispozícii, chýba iba implementácia vyššieho správania. Po skúsenostiach s ich prvou verziou sa jej autori rozhodli znovu celý framework prepísať s použitím knižnice Spring [6][7], ktorá umožňuje meniť nastavenia a väzby medzi jednotlivými komponentmi v systéme pomocou konfiguračného XML súboru, čo vysoko zvyšuje modularitu hráča.

Architektúra výberu akcie je založená na rozhodovacom strome, na ktorého vrchu sa rozhoduje podľa aktuálneho hracieho módu, na druhej úrovni podľa pridelenej roly hráča (útočník, obranca, brankár). Graficky znázornené je to na Obr. 1




Obr. 1 Rozhodovací strom tímu DAInamite

Tieto akcie sa ohodnocujú v širšom rámci zvanom Tactic. Agent, teda nevyhodnocuje najlepšiu akciu iba na základe aktuálneho stavu modelu sveta, ale berie do úvahy aj svoje predošlé akcie, alebo akcie vyhodnotené ako najlepšie z pohľadu tímu[1].

Toto je zmena oproti pôvodnej koncepcii tímu DAInamite z roku 2007, kedy sa na výber akcie používala kombinácia stavového automatu a modulov ohodnocujúcich pravdepodobnosť a výhodnosť tej-ktorej akcie.

Kľúčovou triedou v architektúre tímu DAInamite je trieda Prophet, ktorá sa na základe údajov z modelu sveta snaží vypočítať pravdepodobný stav sveta o daný počet cyklov. Pri prijatí správy od serveru Prophet vyhodnotí, či správa neobsahuje niektoré nové fakty, ktoré by prerušili a zneplatnili aktuálne vybraný Tactic. Môžu nastať dve situácie:

a) Nové fakty nenarúšajú vybranú taktiku, tým pádom sa vyberie akcia smerujúca k splneniu cieľu vybranej taktiky

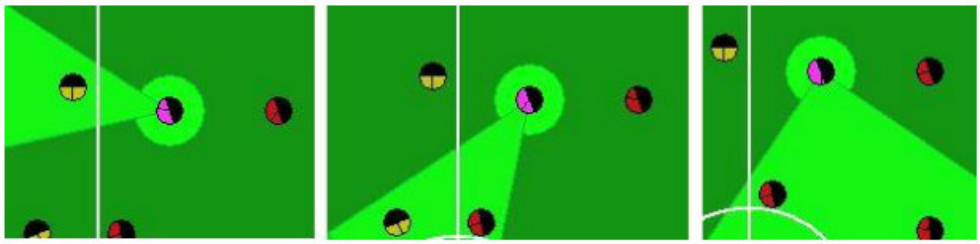
b) Nové fakty obsahujú fakty, ktoré zneplatňujú výber aktuálnej taktiky. V takomto prípade sa musí spustiť algoritmus na výber novej taktiky

Ako vidno, nová taktika sa vyberá iba v prípade, že agent obdržal správu o neočakávanom fakte. To umožňuje ušetriť výpočtové prostriedky a osamostatniť vyhodnocovanie taktiky do osobitného vlákna.

Tím DAInamite prišiel ako prvý s myšlienkou otáčania krku, ktorú už mali implementovanú takmer všetky tímy hrajúce na posledných majstrovstvách sveta [5]. Za normálnych okolností dostáva agent údaje o objektoch v určitom výseku [2] , pričom každý výsek má zadefinovanú periodicitu posielania správ. Čím väčší výsek, tým menej často chodia agentovi informácie. To predstavuje problém, ak sa chce agent zamerať na viacero bodov na ihrisku. Tím DAInamite navrhol a implementoval algoritmus, pomocou ktorého si

[1] Najlepšia akcia z pohľadu tímu sa môže líšiť od najlepšej akcie z pohľadu samotného hráča
[2] 45, 90 alebo 180 stupňovom

agent zaznamenáva „zaujímavé body“ na ihrisku a používa príkazu otáčania krku na ich pokrytie. To umožňuje mať pokrytú väčšiu časť modelu sveta za cenu malej neaktuálnosti informácii v nich obsiahnutých. Algoritmus sa ukázal ako mimoriadne efektívny, čo je aj dôvodom jeho rýchleho rozšírenia. Výstup algoritmus ukazuje sekvencia obrázkov na Obr. 2 [8].



Obr. 2 Skenovanie okolia v 3 po sebe idúcich cykloch

### 2.2.2 12. hráč

Tím 12. hráč je fakultným tímom postaveným nad základmi DAInamite frameworku. Verejne dostupný DAInamite framework, ale neobsahuje prvky vyššej logiky hráča, obsahuje iba základnú sadu týkajúcu sa komunikácie so serverom a modelu sveta. Doprogramovaniu vyššej logiky sa venoval tím 12. hráč. Primárne išlo o prenesenie formácii, ich dynamickej premeny, rozhodovacieho stromu tímu Jahodových Princov do DAInamite frameworku a vylepšenie určitých častí tohto tímu, menovite výkopy brankára a zahrávanie štandardných situácii. Zverejnený DAInamite framework obohatili o rozpoznávanie štandardných situácii a vylepšenie zvukovej komunikácie medzi hráčmi[3]. Tím sa ukázal byť ako veľmi dobre pripravený a zvíťazil v regionálnej súťaži v ročníku 2008/2009.

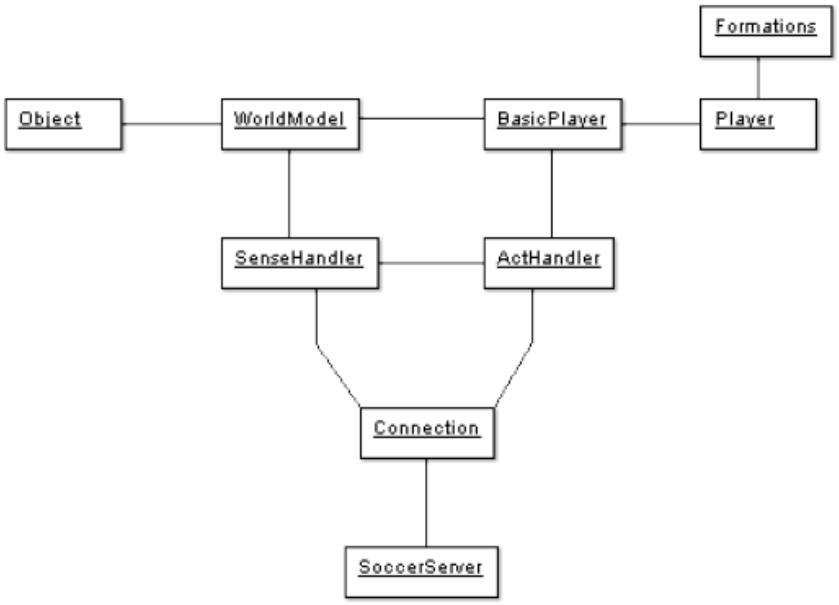
Tím sa zaoberal aj možnosťou implementácie algoritmu prerušovania súperových prihrávok. Keďže deterministický algoritmus by mohol byť úspešný proti jednému tímu a neúspešný voči iným, rozhodli sa jej prvotní autori, tím Brainstormers o trénovanie adaptívnej neurónovej siete na tento účel. Výsledky sa ukázali byť veľmi sľubné, úspešnosť algoritmu dosahovala približne 80% [9].

Bohužiaľ, tím 12. hráč sa nedopracoval k vývoju tejto časti systému, ale v [9] ostáva vypracovaný jej hrubý návrh.

### 2.2.3 UvA Trilearn

Tím vznikol v roku 2001. Autormi tímu sú dvaja študenti z Amsterdamskej Univerzity. tím vytvorili v rámci diplomovej práce. Ich hlavným prínosom je vytvorenie novej architektúry hráča. Tú uvádzame na Obr. 3. Opisujeme iba dve triedy architektúry považované za najdôležitejšie, ide o triedy WorldModel a Object.

[3] Išlo o koncept attentionTo – hráč si mohol vybrať hráča, ktorého zvukové správy prednostne spracovával



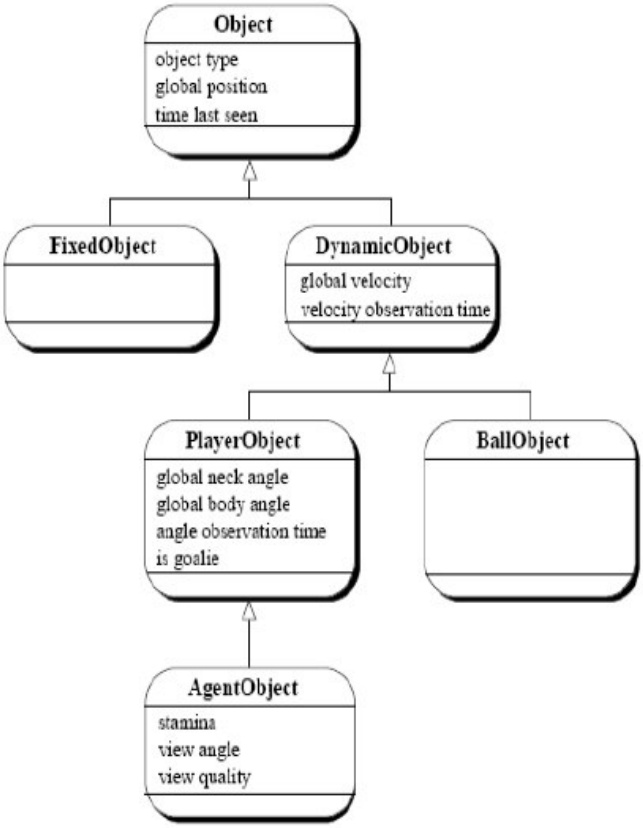

Obr. 3 Architektúra hráča UvA Trilearn so základnými komponentmi

### 2.2.3.1 Trieda Object

Reprezentuje objekty uchovávané v triede WorldModel. Samotná trieda Object je abstraktnou triedou a určuje, že každý objekt má aspoň tri základné informácie: typ, pozíciu a čas kedy bol naposledy videný. Poznáme niekoľko typov objektov, o ktorých si hráč uchováva informácie:

- FixedObject – pevný objekt obsahuje len informácie, ktoré určuje trieda object ako povinné. Medzi pevné objekty patria, tyče bránky a ďalšie významné body ihriska ako stred ihriska, rohy ihriska, a niekoľko orientačných bodov po okraji ihriska.
- Dynamicobject – ide o abstraktnú triedu, ktorá pridáva ďalšie atribúty pre pohyblivé objekty, akými sú hráč a lopta. Ide o atribúty, rýchlosť pohybu a čas kedy bola daná rýchlosť zaznamenaná.
- BallObject – ide o podtriedu dynamických objektov. Do tejto triedy objektov patrí iba lopta a nepridáva žiadne nové atribúty.
- PlayerObject – je triedou dynamických objektov, ktorá v sebe uchováva informácie o jednotlivých hráčoch. Pribúdajú nasledovné informácie: uhol natočenia tela, uhol natočenia hlavy, čas zaznamenania natočenia a informáciu či je hráč brankárom.
- AgentObject – je triedou rozširujúcou triedu PlayerObject, obsahuje informácie o agentovi samotnom, pridáva atribúty o jeho stamine, uhle a kvalite videnia.

Hierarchia objektov je uvedená na Obr. 4.




Obr. 4 Hierarchia tried objektov

### 2.2.3.2 WorldModel

Táto trieda predstavuje hráčov pohľad na svet, v ktorom sa nachádza. Informácie, ktoré sa v ňom nachádzajú možno rozdeliť do 4 skupín:

- Enviroment information – ide o informácie špecifické pre prostredie poskytované serverom, parametre servera, či parametre heterogénnych hráčov.
- Match information – sú informácie o stave zápasu, patria sem aj informácie o čase (počet cyklov), číslo hráča, gólový rozdiel.
- Object Information – informácie o všetkých objektoch uvádzaných v časti 2.2.3.1 Trieda Object.
- Action Information – agent si uchováva informácie o vlastných akciách , ktoré vykonal, aj ktoré vykonať chce. Tento spôsob je vhodný lebo hráč dokáže posúdiť či vykonal akciu, ktorú chcel vykonať.

Trieda WorldModel nám poskytuje možnosti ako pracovať s informáciami, ktoré sa v nej uchovávajú. Máme na to funkcie, ktoré zaraďujeme do 4 kategórií:

- Retrieval – slúžia na získanie konkrétnej informácie, ktorá je potrebná o stave zápasu, či objekte.
- Update – tieto metódy aktualizujú hráčov svet na základe informácií, ktoré získal od servera.


- Prediction – tieto funkcie slúžia na určenie pravdepodobného stavu objektu v niektorom budúcom cykle. Vzhľadom na to, že nevieme akú akciu vykoná hráč používajú sa hlavne na určenie polohy lopty v budúcich cykloch.
- High Skill – ide o funkcie, ktoré dokážu z informácii sveta vyvodiť určité závery. Patria sem metódy na určenie počtu hráčov v určitej oblasti, určenie najbližšieho či najrýchlejšieho hráča k určitému miestu, indikáciu či objekt spĺňa vysoko-úrovňové podmienky (napr.: či je možné kopnúť do lopty), určenie smeru najbezpečnejšej trajektórie pre loptu medzi protihráčmi a.i.

#### 2.2.3.3 Zhodnotenie

Architektúra hráča sa ukázala v 2D simulačnej lige ako veľmi úspešná. Pre náš ďalší postup pri vývoji 3D tímu, by bolo vhodné inšpirovať sa niektorými konceptmi, ako metódy WorldModelu aby sme dokázali lepšie rozhodovať o vysoko-úrovňových akciách, ktoré ma hráč vykonávať iba na základe informácií, ktoré má o svete.

### 2.2.4 GangOfSix

Je tím založený na architektúre hráča UvA Trilearn. Tím sa stal víťazom regionálneho turnaja v roku 2007.[19] Za najväčšie prínosy považujem implementáciu koordinácia rozhodovania pomocou koordinačných grafov a fuzzy regulátora.

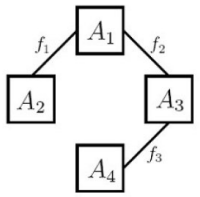
#### 2.2.4.1 Koordinačný graf

Pri hre nejde vždy o to vykonať najlepšiu možnú akciu hráča ako samostatne stojaceho agenta, iná akcia, ktorá sa z jeho pohľadu zdá menej výnosná môže byť pre tím výhodnejšia. Preto by mal hráč svoju činnosť koordinovať s ostatnými agentmi.

Aby nemusel jeden agent koordinovať svoju činnosť s každým agentom boli použité procesy na tvorbu koordinačných grafov, ktoré určujú, ktoré agenty majú svoje akcie spolu koordinovať. Výsledkom je graf bodov(agentov) a spojníc, ktoré určujú nutnosť spolupráce. Je dôležité poznamenať, že každý hráč si dokáže vytvoriť iba čiastočný graf, lebo má obmedzené informácie o svojom okolí.

Na Obr. 5 je znázornený koordinačný graf, ak vrcholy A1, A2, A3, A4 znázorňujú agentov a spojnice f1, f2, f3 potrebu koordinácie činnosti. Môže ísť napríklad o koordinačný graf hráča s loptou (agent A1), ktorý vie o troch hráčoch vo svojom okolí. Na základe koordinačného grafu vie, že musí svoju činnosť koordinovať s ďalšími dvomi hráčmi agentom A2 a A3. Agent A2 koordinuje svoju činnosť s inými hráčmi, o ktorých agent A1 nevie, a s agentom A1. Agent A1 navyše vie, že agent A3 svoju činnosť koordinuje s agentom A4, ktorý je mimo dosahu hráča A1(preto s agentom A4 svoju činnosť nekoordinuje). Ďalšie agenty sú pre hráča A1 nezaujímaví pre nedostatok informácií.

Tím ďalej nadviazal tvorbou algoritmu na elimináciu premenných, ktorý mal rozhodovať na základe koordinačných grafov a výnosových funkciách o vykonaných akciách.



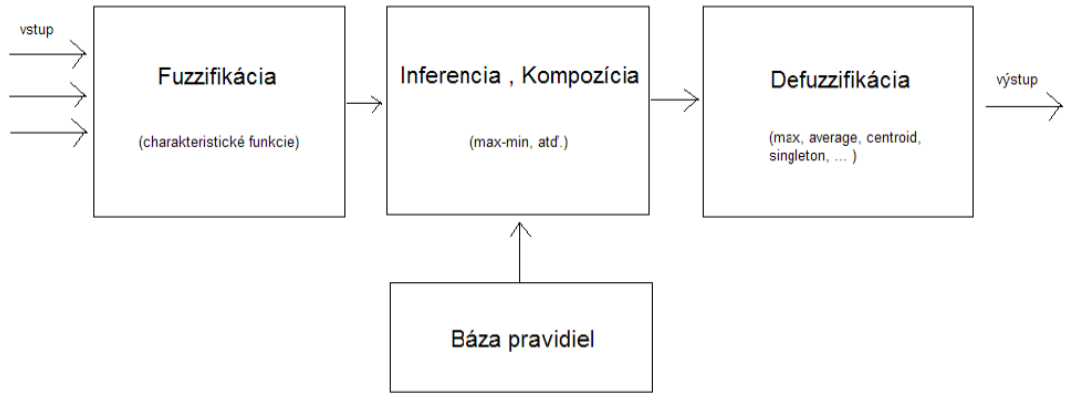

Obr. 5 Koordinačný graf

### 2.2.4.2 Fuzzy Regulátor

Je mechanizmus rozhodovania sa, ktorý nemusí priniesť presné výsledky. Fuzzy regulátor pracuje v nasledovných krokoch:

a) Fuzzifikácia –prevod vstupných hodnôt na stupne príslušnosti jednotlivých lingvistických premenných.
b) Inferencia - určenie výsledného stupňa príslušnosti celej predpokladovej časti pre každé pravidlo. Najčastejšie sa používa metóda *min* (prienik fuzzy množín).
c) Kompozícia - určuje výslednú funkciu príslušnosti výstupov pomocou stupňov príslušnosti predpokladov jednotlivých pravidiel a samotných záverov pravidiel. Najčastejšie sa používa metóda max(zjednotenie množín).
d) Defuzzifikácia - realizuje prevod výslednej funkcie príslušnosti výstupov na ostrú (reálnu) hodnotu. Často používanou metódou defuzzifikácie je metóda ťažiska.

Jednoduchú schému Fuzzy regulátor vidíte na Obr. 6. Báza pravidiel - obsahuje pravidlá správania sa fuzzy regulátora.




Obr. 6 Komponenty Fuzzy Regulátora

Úlohou Fuzzy regulátora je správne a rýchlo sa rozhodnúť. Z implementácie tímu vyplýva, že je skôr vhodné využiť fuzzy regulátor na doladenie parametrov akcie, ktorá sa bude vykonávať než na samostatné rozhodovanie sa hráča.

### 2.2.4.3 Zhodnotenie

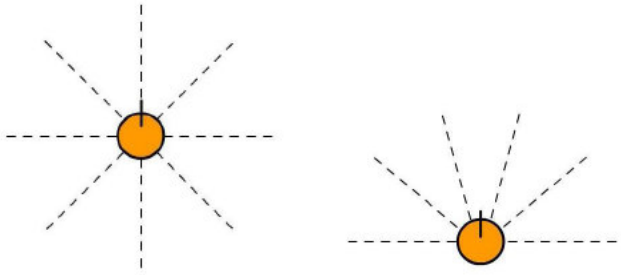
Tím bol na turnaji organizovanom na našej fakulte najúspešnejším v roku 2007. Pre koordinačné grafy nemáme v 3D lige ešte využitie, keďže na ploche sa nepohybujú viac ako traja roboti. Určitú formu fuzzy regulátora by však bolo možné využiť pri niektorých pohyboch, ktorých vykonanie nemusí byť úplne presné. Nie je však jasné či sa takýmito pohybmi budeme zaoberať.

### 2.2.5 Jahodoví princovia

Tento tím je nasledovníkom tímu GangOfSix. Pokračovali v ich práci a doimplementovali algoritmus eliminácie premenných. V roku 2008 vyhrali regionálny turnaj.[18]

#### 2.2.5.1 Rozohrávanie brankára

Tím Gang of Six používal jeden algoritmus prihrávania pre brankára aj hráčov. Hráč rozdeľuje plochu okolo seba pomocou lúčov a následne si vyberá plochu kam kopne loptu. Jahodoví princovia prišli s vylepšením daného algoritmu pre brankára, ten sa pri prihrávaní sústredí iba na plochu pred sebou čo mu umožňuje detailnejšie rozdelenie priestoru a presnejšie prihrávanie.



Obr. 7 Lúče rozdeľujúce plochu pri prihrávaní hráča a brankára

#### 2.2.5.2 Algoritmus eliminácie premenných

Tím implementoval Algoritmus Eliminácie premenných inak nazývaný APE, navrhovaný už tímom Gang of Six. Daný algoritmus slúži na vybranie najvýhodnejšej akcie nie iba pre jedného hráča, ale pre tím ako celok => maximalizácia spoločnej výnosovej funkcie.

Implementovali 9 výnosových funkcií, použitých v APE, ktoré reprezentujú rôzne herné situácie a opisujú celkovú stratégiu tímu.

1 : Zastavenie súperovej akcie

2 : Prihrávka spoluhráčovi

3 : Driblovanie hráča

4 : Odkopnutie lopty

5 : Streľba na bránu

6 : Prihrávka po získaní lopty

7 : Dvojnásobná prihrávka

8 a č.9 : Presun na strategickú pozíciu

Vyhodnotenie algoritmu však trvá tak dlho, že nie je v praxi použiteľný.

#### 2.2.5.3 Zhodnotenie

APE nebol použiteľný v 2D lige, a pre malé množstvo agentov na ploche v 3D lige nemá uplatnenie. Upraviť možnosť prihrávania je správne nie len pre brankára ale aj pre ľubovoľného rozohrávajúceho hrča, ale iba v prípade rozohrávania. Počas hry by mal byť každý agent schopný nahrávať všetkými smermi.


## 2.3 Analyza 3D timov

Analyzovali sme domáce aj svetové tímy. Keďže v 3D simulačnej lige sa hrá len s tromi hráčmi nešlo o analýzu stratégií, ale sústredili sme sa hlavne na nižšie schopnosti agentov, ako chôdza, vstávanie, či kopanie do lopty.

### 2.3.1 Vlastný tím (Angelo) – domáci 3D tím

Tím vytvoril v predošlom semestri náš tímový kolega Jaroslav Chnúrik na predmete Umelá Inteligencia v rámci alternatívneho zadania. Úlohou bolo vytvoriť hráča schopného vykonávať aspoň základné pohyby (kráčanie, kopanie, a.i.).

#### 2.3.1.1 Schopnosti

Medzi schopnosti hráča patrí kráčanie vpred, otáčanie sa, vstávanie, automatická detekcia pádu a inteligentný presun na určené súradnice.

***Kráčanie***

Hráč dokáže kráčať iba vpred. Pri kráčaní iným smerom sa musí najskôr otočiť. Pri tomto pohybe nemusí pokrčiť kolená ako je to u väčšiny hráčov. Pohybuje sa teda vo vztýčenej polohe. Výsledkom je rýchlejšia chôdza spôsobená ušetreným časom pri začiatočnom pohybe. Cenou je ale znížená stabilita. Tá je badateľná prevažne v strednej časti kroku, kde je robot nútený sa „preklopiť“ na druhú nohu.

Krok je rozložený na dve riadiace fázy a osem pohybových fáz. Riadiace fázy slúžia na postavenie robota do polohy vhodnej pre kráčanie a na spätný pohyb, t.j. do stabilnej polohy v stoji. Pohybové fázy reprezentujú krok ľavou a pravou nohou ako jeden plynulý pohyb, sú neoddeliteľné.

Pri kráčaní hráč pohybuje takmer všetkými kĺbmi. Pomáha si tak udržať rovnováhu (napr. rozpažením rúk).

***Otáčanie sa***

Otáčanie je nepresné. Reprezentované je iba konštantným otočením robota. Presnosť otáčania je približne +/- 1 radián. Pri otáčaní je využitá schopnosť robota otáčať bedrovým kĺbom do troch strán. Pri tomto pohybe sa robot predkloní aby preniesol ťažisko.

Keďže server neodosiela informácie o natočení robota, je potrebné zmerať súčasné natočenie. To je vykonaná zaznamenaním pozície robota, jeho predklonením a opätovným zmeraním pozície. Rozdiel medzi meraniami udáva vektor, definujúci smer natočenia robota.

***Vstávanie***

Pred vstávaním hráč najskôr zistí pozíciu robota. Tá môže byť tvárou hore a tvárou dole. V prvom prípade je ťažké sa postaviť, preto sa robot najskôr otočí do druhej pozície.

Meranie pozície je vykonávané vystretím rúk. Ak sa zmení pozícia robota, je natočený tvárou dole. Dôvod je rovnaký ako pri otáčaní a to absencia údajov o natočení robota. Samotné vstávanie z tejto polohy prebieha v troch fázach.

***Detekcia pádu***

Pád je detekovaný opakovaným sledovaním pozície hráča. Pri detekcii pádu sú najskôr dokončené jednoduché pohyby, prerušené všetky komplexné pohyby a do registra pohybov je doplnený na začiatok komplexný pohyb „Vstávanie“ (pozri časť implementácia).


#### 2.3.1.2 Implementácia

Hráč je postavený na serveri verzie 0.6.0.

Základným prvkom tohto hráča je register pohybov. Do toho sú registrované pohyby vo dvoch úrovniach. Prvá úroveň určuje iba postupnosť stavov robota s popisom spôsobu, akým má byť vykonávaná pohybová fáza (dynamika pohybu). Druhá úroveň popisuje komplexné pohyby. Komplexný pohyb je identifikácia cieľa, ktorý má robot dosiahnuť. Hráč vyberá pohyby tak, aby tento cieľ dosiahol. Potom začne sledovať iný cieľ. Príkladom komplexného pohybu je presun na zadané súradnice.

Serveru sú odosielané informácie o rýchlosti jednotlivých kĺbov, nie požadovaný cieľový stav. Je teda potrebné upraviť výpočty tak, aby bol výstup hráča práve tento zoznam. O tento úkon sa stará pohybový výpočtový stroj (z angl. engine, ďalej iba PVS).

Výhodou je, že PVS potrebuje na vstupe iba údaje o dynamike pohybu (rýchlosť, tolerancia, referenčný bod, atď.) a výstupom je rýchlosť pohybu kĺbu. PVS vie počítať iba s elementárnymi dátami ako číslo, bod a vektor, preto musí byť každý pohyb rozdelený na 22 samostatných položiek. Pre každý pohyb je vykonávaných 22 samostatných výpočtov. Teoreticky je možné pohybovať nezávisle každým kĺbom ale pre jednoduchosť nebola táto vlastnosť použitá. PVS podporuje viacero druhov pohybov. Hráč využíva pohyb „čím bližšie, tým pomalšie“, čím vzniká efekt plynulého pohybu.

#### 2.3.1.3 Zhodnotenie

Najväčšou výhodou tohto hráča sú jeho elementárne pohyby vykonávané pohybovým výpočtovým strojom. Bolo by vhodné využiť tento stroj aj pri implementácii vytváraného hráča.

### 2.3.2 Neurotics

Slovenský tím vznikol na Fakulte informatiky a informačných technológií v roku 2007.

Agent bol postavený na základe agenta Zigorata. Pri zdokonaľovaní pohybu využili genetické programovanie. Učenie je však riešené pomocou evolučného algoritmu, čo však nie je efektívne z hľadiska dĺžky učenia. Celková evolúcia jedincov bola príliš komplikovaná a časovo náročná.

Krížením vznikajú pri evolučnom algoritme nové generácie jedincov. Potomkovia sa ohodnocujú pomocou hodnoty fitness. Správanie, teda vykonávanie pohybov je realizované na základe génov, ktoré zdedili od svojich rodičov. Tím vytvoril testovací framework, vďaka ktorému testovali fungovanie základných pohybov. Vykonávanie pohybov, resp. jedincov prebieha sériovo a medzi tým sa hráč odpájaj a pripája na server.

Hráč je rozložený do modulov (správaní). Tie sú reprezentované ako triedy C++.

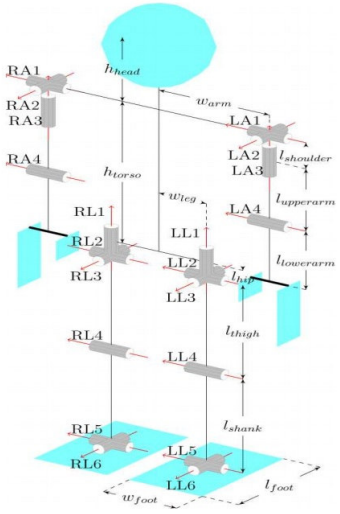
Popis správaní:

- ***AdjustJointBehavior*** – Podľa zadaných hodnôt priamo nastaví uhlovú rýchlosť kĺbu.
- ***AnkleBehavior*** – Nastaví uhol otočenia členku.
- ***BasicBehavior*** – Základná trieda hráča, z ktorej dedia všetky ostatné triedy, nemá žiadnu špeciálnu funkcionalitu.


- ***BeamBehavior*** – Vysiela *beam* príkaz serveru. *Beam* príkaz prenesie hráča na požadovanú pozíciu.
- ***ElbowBehavior*** – Nastaví uhol otočenia lakťa.
- ***HipBehavior*** - Nastaví uhol otočenia bedrového kĺbu.
- ***InitPosBehavior*** – Inicalizuje hráča a nastaví všetko potrebné pre jeho štart.
- ***KneeBehavior*** - Nastaví uhol otočenia kolena.
- ***MainBehavior*** – Trieda rozhodujúca na základe zložitejších procedúr o nasledujúcej postupnosti akcií. V prototype táto trieda vykonáva samotné chodenie – postupnosť jednotlivých menších elementárnych akcií súvisiacich s chodením.
- ***RotateArmBehavior*** - Nastaví uhol otočenia ramena.
- ***RotateLegBehavior*** - Nastaví uhol otočenia nohy.
- ***ShoulderBehavior*** - Nastaví uhol otočenia plecom.
- ***TurnToBehavior*** – Otočí hráča na požadovaný uhol.
- ***TurnToLeftBehavior*** – Otočí hráča na požadovaný uhol smerom doľava.
- ***TurnToRightBehavior*** - Otočí hráča na požadovaný uhol smerom doprava.
- ***WalkBehavior*** – Hráč kráča rovno v smere, v ktorom je otočený.

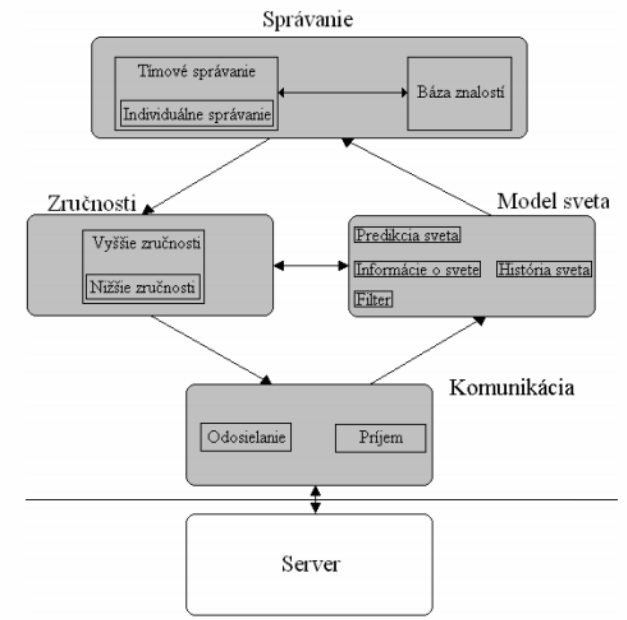
### 2.3.3 Hviezdna jedenástka

Hviezdna jedenástka je domáci tím z FIIT STU, ktorý sa venoval 3D Robocupu. Rozhodli sa pre vývoj hráča typu Humanoid, nakoľko je to novší typ a tím sa obával, že sa časom zastaví vývoj servera s hráčmi Sphere. Nakoľko sa na ihrisku nenachádzalo viac hráčov, hráč nemusel mať implementované schopnosti, ktoré by sa používali pre interakciu s inými hráčmi a preto nevyužíval perceptor See. Ako model hráča si tím zvolil soccerbot056 (Obr. 8).




Obr. 8 Soccerbot056

Tím pri implementácii parsera správ použil nástroj APG a taktiež mali vytvorené logovanie do súboru. Na logovanie, resp. jeho kód použili podmienenú kompiláciu, aby sa hráč mohol skompilovať bez tohto kódu pre logovanie.




Obr. 9 Architektúra hráča

Tím implementoval zručnosti ako vstávanie zo zeme (2 typy), chôdza (tiež 2 typy) a otáčanie sa na mieste. Pri pohybe ale využívali len sekvenciu základných pohybov, nezisťovali žiadne informácie o ťažisku hráča.

### 2.3.3.1 Opis pohybov hráča

#### *Vstávanie z ľahu na bruchu*

Hráč mal pri páde ruky pri tele. Po dopade ich zozadu prevrátil o 180 stupňov, takže ich mal vystreté na zemi pred sebou. S vystretými rukami sa začal podopierať a zároveň pokrčil kolená a sťahoval ich pod trup. Keď už mal dolnú časť nôh na zemi, začal priťahovať ruky bližšie k telu, až nakoniec boli pri kolenách. Následné hráč vyrovnal lakeť a začal trup vystierať kolmo na zem. Z tejto polohy už vstal.

#### *Vstávanie hráča*

V prípade, že hráč padol dopredu, pohybom rúk od zeme sa odrazil a pri pristátí sa skrčil. Následne, po ustálení, sa postavil.

#### *Chôdza*

Tím implementoval dva rôzne typy chôdze. Pri prvom type prvý krok hráča spôsobil, že sa hráč naklonil do strany a musel udržať rovnováhu, následne sa začal malými krokmi, so stále pokrčenými kolenami a rukami vystretými rovnobežne s hornou polovicou dolnej končatiny, pohybovať.

V druhom type chôdze hráč chodil bez ohýbania nohy v kolene. Využíval pri chôdzi ruky, teda napríklad pravá noha dopredu a zároveň pravá ruka dozadu. Pohyb nohy bol oblúkom smerom od tela a vždy po dopade nohy musel malú chvíľku hráč čakať, kým sa ustálil.


***Drepy***

Hráč sa dokázal uviesť do drepu, ktorý pripomínal ľudský drep. Teda súčasne s pokrčením nôh v kolenách, vystrel ruky dopredu. Ruky sa vystierali vždy rovnobežne so stehennou časťou nôh.

***Stojka na hlave***

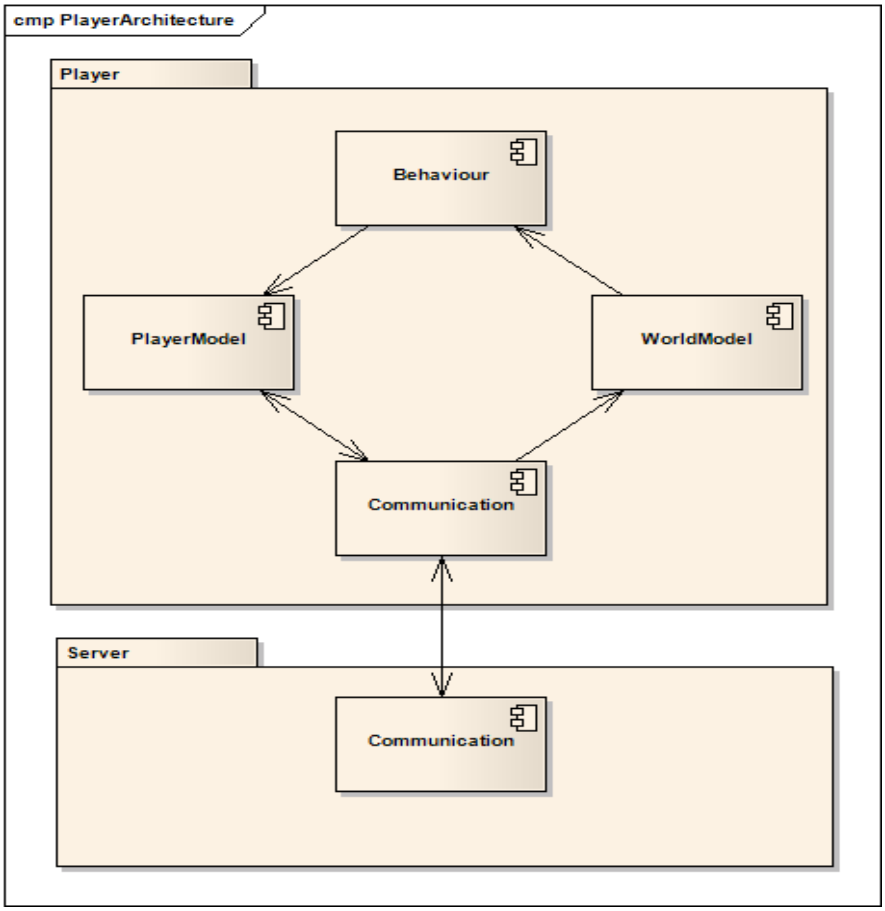
Hráč dokázal aj stáť na hlave. Najprv sa uviedol do drepu, ruky ale udržiaval kolmo k zemi. Následne padol dopredu, ruky totižto dosiahli k zemi a keďže pri drepe bol jemne naklonený dopredu, keď ho ruky jemne nadvihli, spôsobilo to, že sa prevalil dopredu. Po dopade ruky rozpažil (asi 120-130 stupňový uhol), čím sa podoprel a mohol vystrieť nohy.

### 2.3.4 DreamTeam

Riešenie tohto tímu je založené na práci tímu Hviezdna jedenástka, ktorý úspešne implementoval komunikáciu so serverom. Ako prioritu si stanovili implementáciu dynamickej chôdze. To znamená, že robot by mal dokázať využívať svoje receptory tak, aby bol schopný efektívne reagovať na zmenu situácie počas vykonávania nejakej činnosti, najmä na poruchy vnesené do pohybového procesu[20].

#### 2.3.4.1 Architektúra hráča

Rozhodli sme sa pre náš tímový projekt využiť práve architektúru použitú tímom Dream Team.




Obr. 10 Architektúra hráča

Skladá sa zo štyroch komponent:

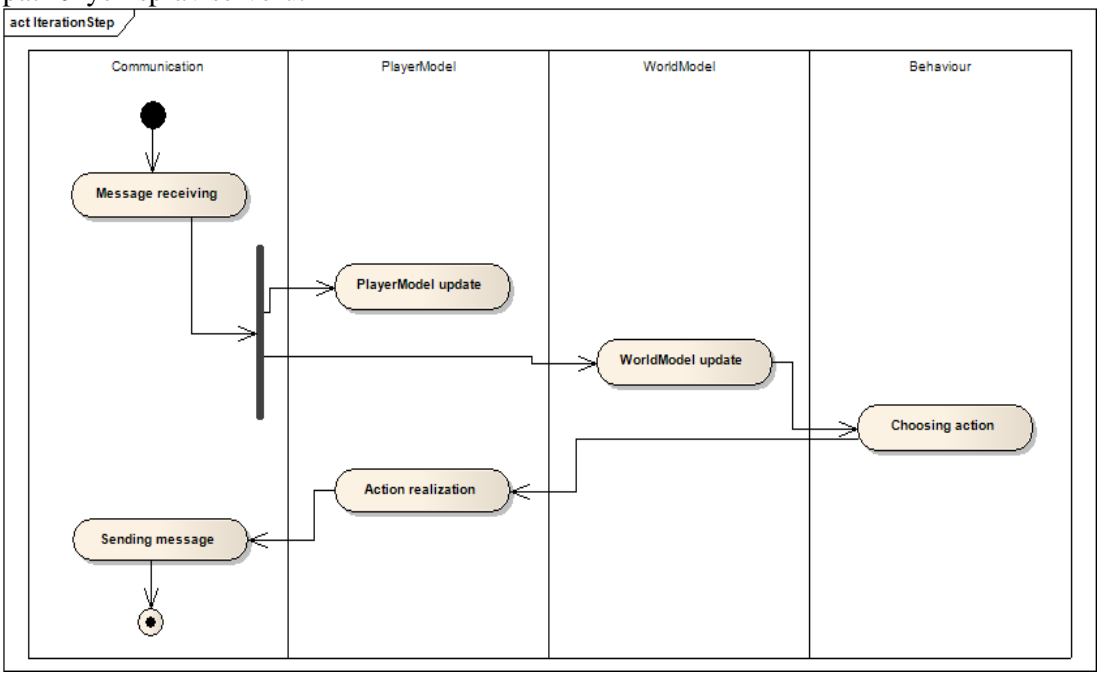
a) *Communication* slúži na výmenu správ medzi serverom a hráčom. Bude používaný vždy na začiatku a konci hlavného cyklu vykonávania agenta. Prijaté správy obsahujú informácie o aktuálnom stave sveta – pozície hráčov, lopty apod. V odoslaných správach informuje hráč server, akú akciu chce vykonať (zväčša oznamuje pohyby jednotlivých kĺbov).

b) *WorldModel* predstavuje model okolitého sveta, t.j. futbalového ihriska, so všetkými dostupnými informáciami o objektoch v ňom (tie získava zo správ od servera). Aktuálny model sveta je základom pri rozhodovaní o ďalšej akcii hráča.

c) *Behaviour* zabezpečuje výber vhodnej akcie na základe informácií o okolitom svete. Spôsob rozhodovania bude popísaný nižšie.

d) *PlayerModel* je komponent, ktorý obsahuje aktuálny model hráča. To znamená údaje o kĺboch a končatinách, vrátane údajov o ich rýchlosti, smeru, a pod. *PlayerModel* slúži na realizáciu akcií, ktoré dostane od komponentu *Behaviour*.

Cyklus vykonávania agenta je zobrazený na Obr. 11. Na začiatku cyklu sa spracujú správy zo servera. Na základe získaných informácií sa aktualizuje model sveta a model hráča. Následne sa vyberie ďalšia akcia, ktorá sa aj vykoná, čo v konečnom dôsledku znamená odoslanie patričných správ serveru.




Obr. 11 Cyklus vykonávania agenta

### 2.3.4.2 Modul správania

Tento modul zabezpečuje výber nasledujúcej akcie a jej vykonanie. Základom pre rozhodovanie je aktuálny model sveta, čiže aktuálna situácia na ihrisku. Tento model sveta sa posúva objektom predstavujúcim akcie, ktoré vrátia vhodnosť vykonať danú akciu v danej situácii. Následne sa vyberie a vykoná najlepšie ohodnotená akcia. To má na starosti tzv. *Dispatcher* modul. Okrem toho má tento modul na starosti aj dlhodobejšie rozhodovanie sa vo forme vytvárania stratégie. To prebieha najmä počas vykonávania pohybov – vtedy nemá zmysel, aby Dispatcher rozhodoval o ďalšej akcii. Vykonanie aktuálne zvolenej akcie má na


starosti objekt reprezentujúci daný pohyb. Zabezpečuje najmä správny prechod medzi jednotlivými fázami pohybu a sleduje odchýlky skutočných hodnôt kĺbov od očakávaných. Výhodou tejto architektúry je, že sa ľahko pridávajú nové zručnosti.

Pre uľahčenie rozhodovania je definovaných niekoľko stavov, ktoré zjednodušene vyjadrujú nejaký stav sveta. Napríklad herný stav *CornerKickRight* znamená, že niektoré družstvo zahráva rohový kop. Dôležité pre rozhodovanie sú však najmä stavy hráča a stavy o okolí. Medzi stavy hráča napr. patrí aj stav určujúci jeho pozíciu v tíme (útočník, obranca, brankár).

### 2.3.4.3 Dynamické vykonávanie činností

Základom pre dynamické vykonávanie činností robota je rozdelenie činností, akcie na tzv. *high skills* (napr. kráčanie, beh, kopnutie do lopty) a *low skills*, ktoré ovládajú pohyby kĺbov (napr. krok, otočenie sa). Každý *low skill* je rozdelený na niekoľko fáz. Vo fázach, v ktorých je hráč v stabilnej polohe, je možné prehodnotiť vykonávanie pohybovej aktivity. *Low skill* obsahuje informácie o potrebných stavoch kĺbov pri vykonávaní danej činnosti. Pokiaľ sa nezhodujú s aktuálnymi informáciami získanými z *player* modelu, robot sa pokúsi uviesť do stabilnej polohy a rozhodnúť sa o ďalšej činnosti.

### 2.3.4.4 Vytváranie inštancií zručností z XML

Ako už bolo spomenuté v predošlej časti, zručnosti agenta sú rozdelené na vyššie zručnosti a nižšie zručnosti. Nižšie zručnosti predstavujú nejaký základný pohyb, ako napr. chôdza. Každá nižšia zručnosť je ďalej rozdelená na niekoľko fáz. Každá fáza musí obsahovať informácie o efektoroch, ktoré majú vykonať pohyb, a o hodnotách, ktoré majú tieto efektory v danej fáze dosiahnuť. Vrámci pohybu musia byť definované prechody medzi jednotlivými fázami. DreamTeam definoval jednoduchý XML súbor, ktorý obsahuje všetky spomenuté údaje. Tento XML súbor si hráč rozparsuje a pre každú vyššiu zručnosť vytvorí inštanciu vo svojom modeli, ktorú následne môže používať v zápase. Definovanie novej akcie je tak výrazne jednoduchšie a rýchlejšie (pri testovaní netreba kompilovať kód pri každej zmene).

Na nasledujúcej ukážke je vybratá iba časť súboru, ktorá obsahuje definíciu jednej vyššej zručnosti a jednu fázu. Celý súbor však obsahuje definíciu všetkých zručností robota.


```
<robot>
	<high_skills>
		<high_skill name="walk_to_ball">
			<use_low_skill skill="walking"/>
		 </high_skill>
	</high_skills>

	<low_skills>
		<low_skill name="walking">
			<initial_phase name="chodzapriprava"/>
		</low_skill>
	</low_skills>

	<phases>
		<phase name="chodzapriprava" next="wageRight" isFinal="false">
			<efectors>
				<efector name="lle2">
					<start>0</start>
					<end>-5</end>
				</efector>
				<efector name="rle2">
					<start>0</start>
					<end>5</end>
				</efector>
			</efectors>
			<finalization_phase>ROLLBACK</finalization_phase>
			<rescue_movement>PROCEED</rescue_movement>
			<speed_constant>1</speed_constant>
		</phase>
	</phases>
</robot>
```

Na začiatku sa definujú všetky vyššie zručnosti (elementy <high_skill>). Definícia vyššej zručnosti obsahuje zoznam nižších zručností, ktoré môže vyššia zručnosť využiť. Napríklad kopnutie do lopty môže mať ako nižšiu zručnosť kopnutie rovno a kopnutie do boku. Akým spôsobom sa potom nižšia zručnosť vyberie, si povieme neskôr.

Po definícii vyšších zručností nasleduje definícia nižších zručností (elementy <low_skill>). Definíca nižšej zručnosti obsahuje iba názov počiatočnej fázy pohybu. Ako posledné sú definované fázy (elementy <phase>). Každá fáza obsahuje názov nasledujúcej fázy (atribút *next*), prípadne indikátor, či je daná fáza posledná (atribút *isFinal*). Hlavnou časťou však predstavuje zoznam efektorov, ktoré majú vykonať nejaký pohyb. Pre každý efektor je definovaná počiatočná a konečná hodnota, ktorá sa má dosiahnuť v danej fáze. Okrem toho je ešte pre každú fázu definované meno finalizačnej fázy, záchranného pohybu a rýchlostná konštanta.

Samotný XML súbor však nestačí pre vytvorenie nového pohybu. Pre novú zručnosť treba vytvoriť špeciálnu triedu v kóde a implementovať jej metódy, ako metóda pre určenie užitočnosti daného pohybu pre daný stav, či metóda, ktorá riadi vykonávanie samotného pohybu.


### 2.3.4.5 Zhodnotenie

Hlavným prínosom DreamTeamu je implementácia dynamických pohybov, ktorá znižuje riziko pádu hráča. Využiť sa dá aj modulárna architektúra, ktorá umožňuje ľahko pridať nové pohyby, prípadne upraviť existujúce vďaka využitiu XML súboru pri definovaní pohybov.

## 2.3.5 Agenty 007

Tím Agenty 007 vznikol na Fakulte informatiky a informačných technológií STU v roku 2008/2009.[4] V tom istom roku vyhral miestny turnaj v 3D robotickom futbale.

### 2.3.5.1 Schopnosti hráča

Hráč sa dokáže pohybovať všetkými smermi, vstávať, strieľať na bránu. Tím však zatiaľ nemá stratégiu, teda nie je schopný samostatnej hry.

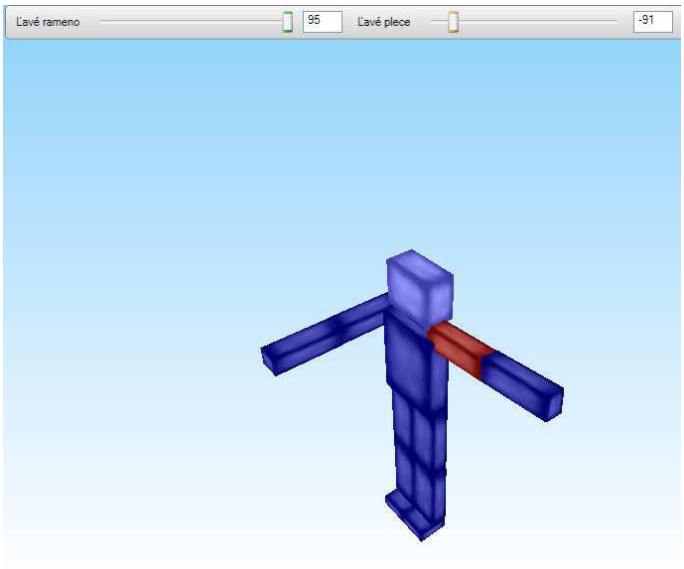
### 2.3.5.2 Editor pohybov

Editor pohybov umožňuje jednoduchým spôsobom vytvárať pohyby a následne poskytuje možnosť ich priamej simulácie (prehrávania). Je prepojiteľný so serverom a s hráčom. Editor vyskladá zadané parametre a tie pošle agentovi pri vytváraní nového procesu.

Existujú dve možnosti vytvárania pohybov:

a) Zadaním parametrov natočenia kĺbov

b) Natočením kĺbov priamo na modeli robota poskytovanom editorom




Obr. 12 Editor pohybov tímu Agenty 007

Editor umožňuje vytvorenie a uloženie akéhokoľvek pohybu, či už elementárneho (natočenie kĺbu), alebo zloženého (beh, vstávanie, chôdza).

Tím vytvoril sadu základných pohybov, ktoré sú k dispozícii v editore.


#### 2.3.5.3 Parser správ zo servera

Tím Agenty 007 implementoval parser správ umožňujúci spracovanie a následné prehratie akéhokoľvek zaznamenaného pohybu agenta na serveri Simspark.

Parser zaznamenáva správy, ktoré prijme hráč zo serveru. Uložené správy sú ďalej spracovávané v editore pohybov, ktorý rozpozná vykonané pohyby a uloží ich do pripravených dátových štruktúr. Následne je možné celý pohyb spätne prehrať a analyzovať.

#### 2.3.5.4 Zhodnotenie

Tím sa sústredil na vytvorenie pomocných nástrojov pre ďalší vývoj agentov. Parser a editor výrazne uľahčujú vývoj a testovanie nových pohybov.

Neboli implementované žiadne výrazné zmeny u samotného hráča, tým stále nemá stratégiu, takže nie je schopný samostatnej hry.

### 2.3.6 The Boldhearts

Tento tím skončil ako druhý na posledných majstrovstvách sveta (Graz 2009). Bohužiaľ, v dobe písania tohto dokumentu nebol ešte publikovaný popis tímu, stránka tímu neposkytovala možnosť pozrieť si tento dokument. Preto táto analýza je založená iba na videách zo zápasov tímu na majstrovstvách sveta.

#### 2.3.6.1 Tímová hra

Hráč sa správal pomerne individualisticky, tímová súhra bola minimálna. Hráči sa držali svojich rolí. Brankár ostával na svojom mieste a hráči sa snažili hýbať sa po „svojej“ časti ihriska, ako pravé a ľavé krídlo. Nahrávky medzi členmi tímu boli minimálne a nevyzerali byť cielené.

#### 2.3.6.2 Brankár

Brankár, napriek tomu, že očividne sa mu venovalo menej pozornosti ako hráčom v poli, bol pomerne efektívny v blokovaní striel súperových hráčov. Za normálnych okolností stál v rovnakom postoji ako hráči, v konštantnej vzdialenosti od bránky, na spojnici lopty s bránkou. Mimo stretu s iným hráčom sa držal vzpriamene a dobrovoľne nepadal na zem.

#### 2.3.6.3 Základný postoj a chôdza

Hráč tohto tímu používa postoj so skrčenými nohami a s rukami pevne pri tele ako základný, je ho vidieť na Obr. 13 Chôdza prebiehala z tohto základného postoja, pričom charakteristiky postoja, skrčené nohy a ruky pri tele sa počas chôdze nemenili. Chôdza bola statická, počas celého jej trvania ostávalo ťažisko prakticky nemenné. Rýchlosť chôdze bola uspokojivá, ale nedosahovala rýchlosti tímu SEU. Rovnako ako u podobných tímov, mal hráč implementované aj cúvanie a chôdzu do bokov.

Obr. 13 Základný postoj robota tímu BoldHearts

### 2.3.6.4 Vstávanie

Hráč sa dokázal sekvenciou pohybov zobrazených na Obr. 14 postaviť relatívne rýchlo. Nevýhodou tejto sekvencie bolo, že hráč netestoval, či prechádzajúce pohyby dosiahli svoj cieľ. Ako dôsledok sa stávalo, že hráč zbytočne vykonával všetky pohyby, pričom po vykonaní celej sekvencie bol stále na zemi.

Obr. 14 Sekvencia pohybov pri vstávaní

### 2.3.6.5 Streľba

Streľba bola u hráča tímu BoldHearts relatívne presná, ale trvalo veľmi dlho, kým sa hráč dostal do presnej pozície potrebnej k streľbe. Samotná streľba bola iba pohybom stehenného kĺbu vysokou uhlovou rýchlosťou. Hráč nepoužíval napriahnutie sa. Priemerný dostrel bol približne na úrovni 1/3 dĺžky ihriska. V mnohých prípadoch nebola strela optimálnym riešením a manuálne ťahanie lopty by bolo efektívnejšie.

### 2.3.7 Fantasia

Ide o úspešný čínsky tím, ktorý sa pravidelne zúčastňuje svetových turnajov. Projekt vznikol v roku 2005 a v roku 2006 sa stali víťazmi v 3D simulačnej lige.[15]

#### 2.3.7.1 Udržanie rovnováhy

Tím používa tzv. statickú chôdzu, ide o princíp kedy sa robot v každej fáze kroku nachádza v stabilnej polohe. Bod nulového momentu (Zero moment point) je bod na zemi (podklade), ktorý dostaneme sčítaním všetkých síl pôsobiacich na teleso. Ak sa tento bod nachádza v stabilnej oblasti (teda v oblasti kde sa robot dotýka podkladu) znamená to, že sa nachádza v stabilnej polohe, a to mu umožňuje pohybovať sa.

#### 2.3.7.2 Základné pohyby

##### *Chôdza*

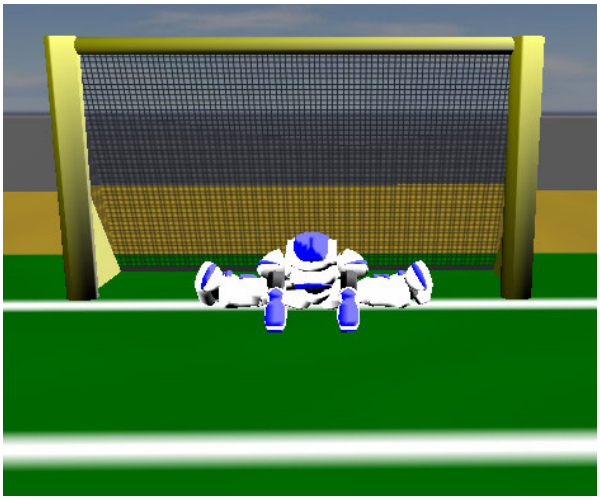
Chôdza robota je statická čo zabezpečuje jej stabilitu, medzi existujúcimi tímami však nie je najrýchlejšia. Robot pri chôdzi nepohybuje rukami.

##### *Vstávanie*

Vstávanie robota možno rozložiť do troch fáz, V prvej fáze robot upaží, predpaží a roznoží čo má v prípade, že je na bruchu za následok zmenu polohy ťažiska a robot potom pokračuje do druhej fázy kedy nohy podsunie pod seba a vstane z drepu. V prípade, že ležal na chrbte ho pohyb, ktorý vykonal presunie do sedu a opäť pokračuje do fázy tri a vstáva z drepu.

##### *Chytanie brankára*

Na Obr. 15 je možné vidieť polohu chytajúceho brankára tímu Fantasia. Vzhľadom na to, že väčšina striel ide po zemi považujeme dané chytanie za výhodné, keďže ním robot pokryje veľkú časť brány a dokáže z neho pomerne rýchlo vstať.



Obr. 15 Poloha brankára tímu Fantasia pri chytaní

#### 2.3.7.3 Zhodnotenie

Tím používa zaujímavý spôsob chytania brankára, ktorý by bolo vhodné naučiť aj nášho robota a vyskúšať jeho úspešnosť oproti spôsobu chytania používanému inými tímami.

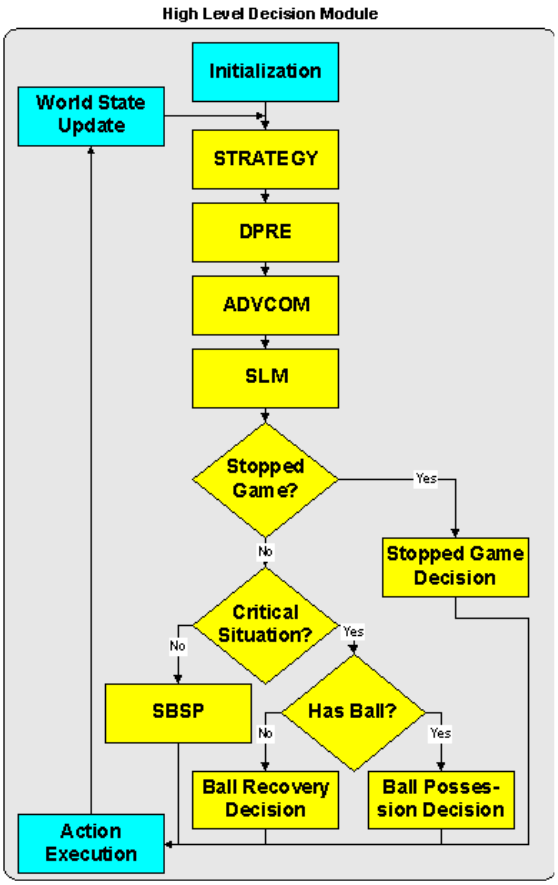
### 2.3.8 FC Portugal

FC Portugal je dlhoročný tím, ktorý zaznamenal niekoľko významných úspechov v 2D aj 3D simulovanom robotickom futbale. Základom rozhodovania hráča je tzv. HLDM modul (High-Level Decision Module).[16]

Cieľom HLDM (Obr. 16) nie je iba rozhodnúť o nasledujúcej akcii, ale tiež stratégii, formácii, či type daného hráča v daný moment. Hráči dokážu medzi sebou komunikovať a vymieňať si správy, čo je veľmi dôležitý prvok aj v reálnom futbale. HLDM dokonca zahŕňa aj rozhodovanie, ktorým smerom sa má hráč pozerať, čo pri 3D simulovanom robotickom futbale zatiaľ nemá význam, pretože hráči vidia 360 stupňov. Rozhodovanie potom závisí od aktuálnej situácie.

#### 2.3.8.1 Značkovacia technika

Ide o techniku, ktorú FC Portugal prvý krát využili pri 2D robotickom futbale v roku 2000, keď im zabezpečila prvé miesto na majstrovstvách sveta. Táto metóda sa snaží prekaziť súperove prihrávky tak, že najprv určí, kade by mohli potenciálne prihrávky viesť a v prípade, že sa hráč nachádza v blízkosti niektorej z možných prihrávok, pokúsi sa ju prerušiť tým, že vojde lopte do cesty. Táto metóda je zvlášť úspešná pri rozohrávaní súperovým brankárom (kedy je hra pozastavená).



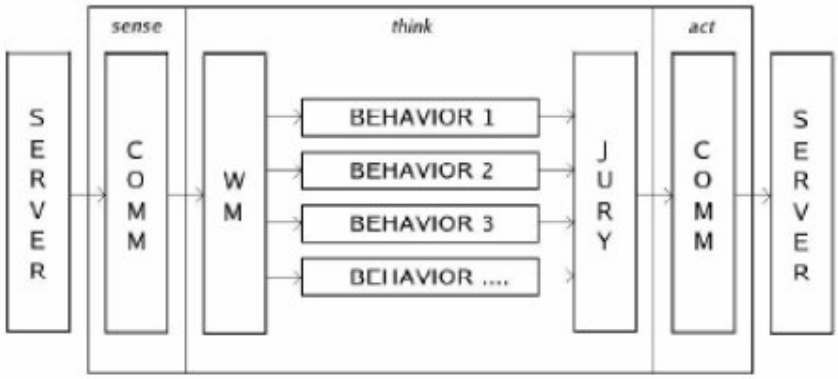

Obr. 16 HLDM modul hráča tímu FC Portugal


### 2.3.8.2 Zhodnotenie

Hlavným prínosom tímu FC Portugal je vysoká úroveň rozhodovania ich hráča, ktorá zahŕňa viacero pokročilých techník. Významná je značkovacia technika potencionálnych prihrávok, ktorá zabezpečila tímu víťazstvo na majstrovstvách sveta.

## 2.3.9 Little green bats

Holandský tím, ktorý sa začal venovať RoboCupu v roku 2005.[17] V roku 2006 sa už zúčastnili majstrovstiev sveta v Brémach. Odvtedy sú pravidelnými účastníkmi majstrovstiev sveta. V modeli sveta WM (World model), sa nachádzajú všetky informácie o hráčoch a lopte. Správanie je realizované ako nezávislé triedy. Správania medzi sebou nekomunikujú ale je možné komunikovať s modelom sveta, od ktorého správanie dostáva všetky potrebné informácie. Hráč pozostáva z troch vrstiev a komunikácie zo serverom. Základné vrstvy by sa dali vyjadriť ako „sense, think, act“. Architektúra hráča je zobrazená na obrázku Obr. 17.




Obr. 17 Architektúra hráča Little Green Bats

Vrstva JURY určuje, ktoré činnosti by sa mali vykonávať priamo počas hernej činnosti. Posúdi, ktoré správanie, prípadne skupinu správaní je vhodné v danej situácii využiť.

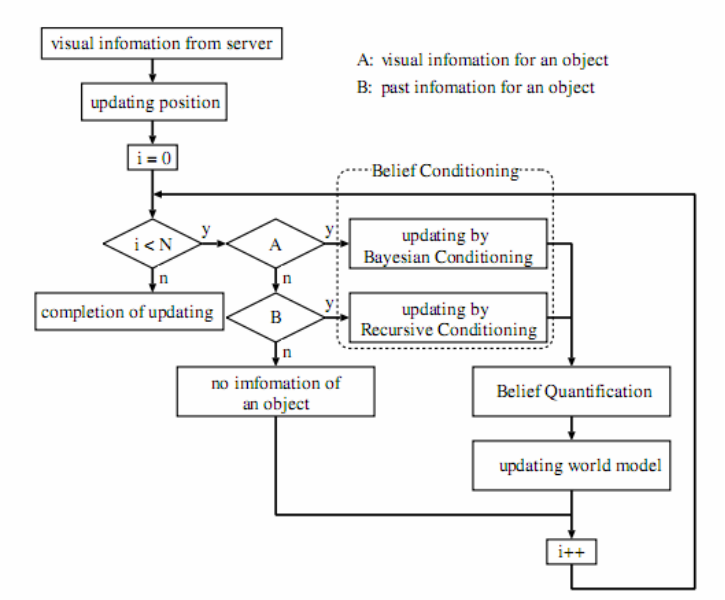
Hráč je vybavený niekoľkými zadefinovanými správaniami a to:

- ballTowardGoal – ťahanie na bránu a streľba
- stayInField – bránenie hráčovi opustiť hraciu plochu
- stayInformation – zabezpečenie zotrvania hráča vo formácii
- awayFromTeammates – zabezpečuje aby hráči pokryli väčšiu časť ihriska
- Pass – prihrávka lepšie postavenému spoluhráčovi
- Dribble – kopanie si lopty pred sebou.

Ich architektúra ďalej umožňuje programátorovi zaraďovať správania do flexibilnej hierarchie a rozhodovať, ktorá hierarchia bude najvhodnejšia pre použitie v danej situácie. Little green bats týmto predstavili „Hierarchiu správania“.

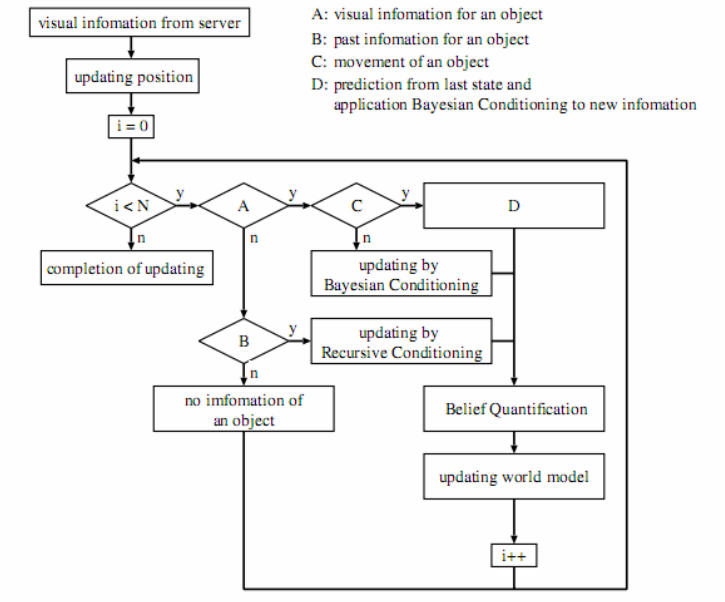
### 2.3.10 NAITO Strikers

Tím NAITO Strikers pochádza z Japonska z Technickej univerzity v Nagoji. Zamerali sa na model podľa M. Saxena a T. Guputa (Obr. 18) a nakoľko tento model mal určité nedostatky, rozhodli sa ho vylepšiť.




Obr. 18 Model podľa Saxena a Guputa

Tento model ale mal určité problémy, ako napríklad, že s pohyblivými objektmi sa nakladalo ako s nepohyblivými a že presnosť informácií o objekte, ktorý sa neaktualizoval dlhší čas, bola veľmi nízka. Na Obr. 19 je znázornený model NAITO Strikers tímu.




Obr. 19 Model podľa NAITO Strikers

V tomto modeli sa pre pohyblivé a nepohyblivé objekty použili 2 rôzne techniky.


#### 2.3.10.1 Pohyblivé objekty

Pri pohyblivých objektoch, agent predvída súčasný stav z predchádzajúceho. Zoberie sa pôvodný stav a nové dáta a pomocou Bayesovej úpravy sa aktualizujú údaje o objekte.

#### 2.3.10.2 Nepohyblivé objekty

V prípade nepohyblivých objektov sa údaje o objekte aktualizujú pomocou klasického Saxenovho a Guputovho modelovania.

#### 2.3.10.3 Pohyb hráča

Hráčova chôdza sa veľmi podobá ľudskej, dĺžka krokov nie je ani príliš malá, ani veľká, hráč používa pri chôdzi ruky a nenakláňa sa príliš do strán. Chôdza je plynulá, bez nutnosti krátkych páuz pre stabilizovanie hráča.

### 2.3.11 SEU Red Sun (China)

Tím SEU Red Sun vznikol v roku 2005 na čínskej Southeast University.[3] 3D robotickému futbalu sa tím venuje od roku 2007. V tom istom roku sa umiestnil na treťom mieste vo svetovom pohári tímov robotického futbalu.

#### 2.3.11.1 Schopnosti hráča

Hráč sa dokáže pohybovať všetkými smermi, vstávať, strieľať na bránu a rozohrávať. Tím je schopný samostatnej hry.

***Chôdza/Beh***

Pohyb hráča je plynulý, za zmienku stojí, že na udržanie rovnováhy používa ruky (pri chodení aj pri behu). Pri chôdzi robí hráč veľmi malé kroky, dokáže sa pohybovať všetkými štyrmi smermi. Chôdza aj beh sú pomerne rýchle.

***Vstávanie***

Vstávanie je realizované posadením sa, rozkročením nôh a súčasným predpažením rúk. Je rýchle, agent pri ňom nestráca stabilitu.

***Streľba***

Streľba je pomerne nepresná, príprava trvá dlho. Agent kope celou nohou, nevyužíva len časť od kolena dolu.

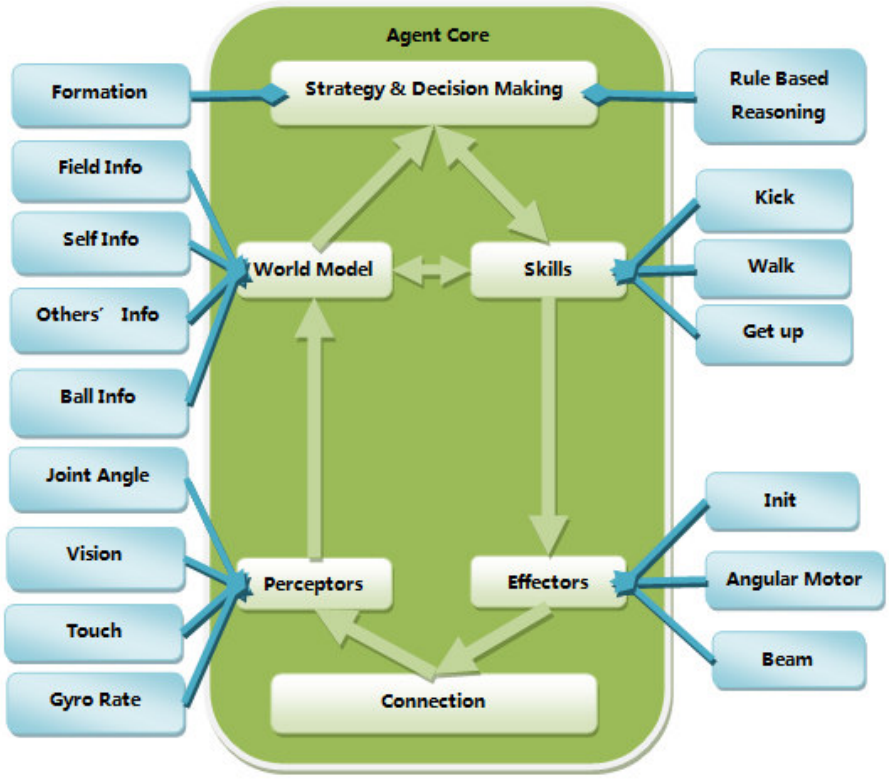
***Rozohrávka***

Hráči nakopávajú lopty dopredu, namiesto nahrávky. Radšej potiahnu loptu sami a pokúšajú sa prejsť cez súpera individuálne.

#### 2.3.11.2 Architektúra agenta

Na obrázku vidíme architektúru agenta tímu SEU Red Sun. Zelený obdĺžnik tvorí jadro agenta. Moduly vo vnútri sú implementované podľa vzoru *singelton*, čo umožňuje ich jednoduchú interakciu. Modré moduly pripojené k jadru boli navrhnuté ako *add-ony* (sú ľahko vymeniteľné).

Architektúra nie je striktne rozdelená na vrstvy, je použitý mechanizmus *plug-inov*, z toho vyplýva jednoduché prispôsobovanie agenta zmenám serveru, ktoré nastávajú pri jeho vývoji.




Obr. 20 Architektúra hráča tímu SEU Red Sun

### 2.3.11.3 Zhodnotenie

Tím sa sústredil na nadštandardné prevedenie pohybov bez lopty (chôdza, beh, vstávanie), na úkor streľby a nahrávok, tie sú nižšej kvality. Vďaka vysokej rýchlosti sa tím ale dokáže presadiť.


### 2.3.12 UI-AI – zahraničný 3D tím

Druhým analyzovaným tímom je UI-AI.[1][2] Tím vznikol na univerzite v Isfahane v Iráne. Pravidelne sa zúčastňujú simulačnej ligy RoboCup už od roku 2003.

#### 2.3.12.1 Schopnosti

Hráč tímu UI-AI dokáže kráčať všetkými smermi (vpred, vzad, do strany), otáčať sa, vstávať, kopať do lopty a merať pozíciu iných objektov na hracej ploche.

***Kráčanie***

Hráč pri kráčaní pokrčí nohy v kolenách a pri chôdzi postupne prechádza troma fázami:

a) Naklonenie na jednu stranu (prenesenie ťažiska)

b) Pohyb nohou vpred

c) Prenesenie váhy robota vpred aby bolo možné cyklus opakovať s druhou nohou

Tento pohyb je stabilný ale nepodobá sa reálnej chôdzi a je pomalý.

***Vstávanie***

Pri vstávaní je použitý jeden pohyb, ktorý je aplikovateľný na všetky situácie. V predošlých verziách bola použitá metóda využívajúca neprirodzených vlastností robota. Presnejší popis vstávania nie je uvedený v žiadnom z dostupných zdrojov.

#### 2.3.12.2 Meranie pozícií

Výpočet vlastnej pozície je triviálny, pretože server poskytuje všetky potrebné údaje bez odchýlky. Problém nastáva pri meraní pozícií iných objektov. V minulých verziách servera, kde boli roboti reprezentovaní guľami, bolo jednoduché vypočítať pozície. Novšia verzia servera reprezentuje robotov oveľa zložitejšie, čo sťažuje orientáciu pomocou zraku.

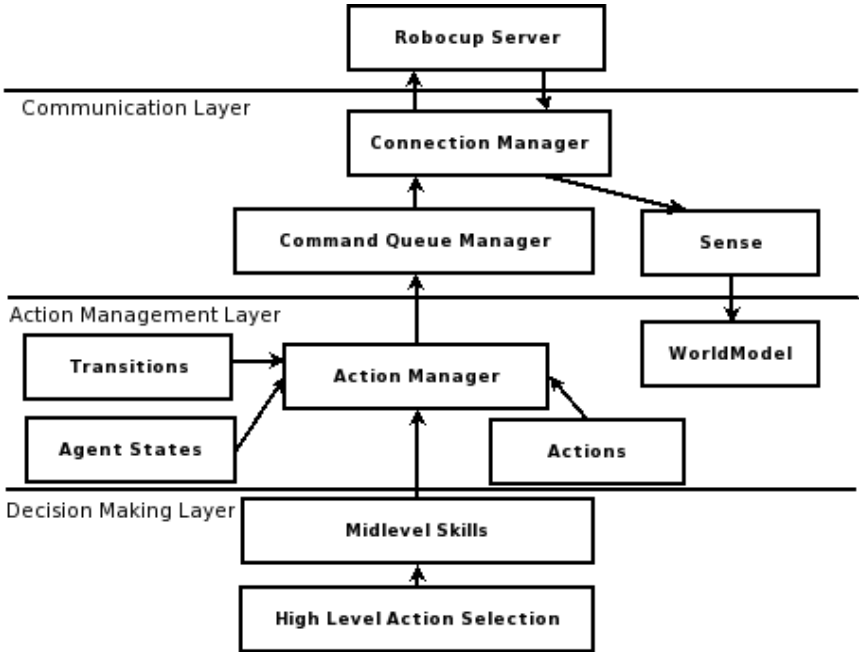
Hráč využíva na meranie pozícií trojrozmerné transformácie bodov pomocou rotačných matíc.

#### 2.3.12.3 Implementácia

Hráč je implementovaný v troch vrstvách:

- Komunikačná vrstva (Communication Layer)
- Vrstva manažmentu akcií (Action Management Layer)
- Rozhodovacia vrstva (Decition Making Layer)

Celkový pohľad na architektúru hráča je znázornený na obrázku 1.




Obr. 21 Architektúra hráča UI-AI ([1], p.2)

***Komunikačná vrstva***

Táto vrstva je zodpovedná za komunikáciu so serverom a za synchronizáciu vláken so serverom. Po prijatí správy od servera aktualizuje model sveta a iniciuje výpočtové procesy na nasledovnej vrstve.

***Vrstva manažmentu akcií***

Úlohou druhej vrstvy je poskytovať prístup ku pohybom hráča na vyššej úrovni abstrakcie. Zapuzdruje tak základné pohybové schopnosti hráča. Z dôvodu zložitosti bola vrstva rozdelená na štyri časti:

- Stavy – zahŕňa informácie o natočení kĺbov robota a ďalšie atribúty jeho polohy
- Transakcie – činnosť vedúca od jedného stavu do iného
- Akcie – postupnosť stavov a transakcií definujúcich zložitejší pohyb
- Manažment akcií – zabezpečuje rozhodovanie na úrovni akcií (kráčať ďalej alebo ukončiť krok?)

***Rozhodovacia vrstva***

Tu je implementovaná inteligencia hráča. Rozhoduje o správaní hráča v závislosti od modelu sveta, iných hráčov a lopty.

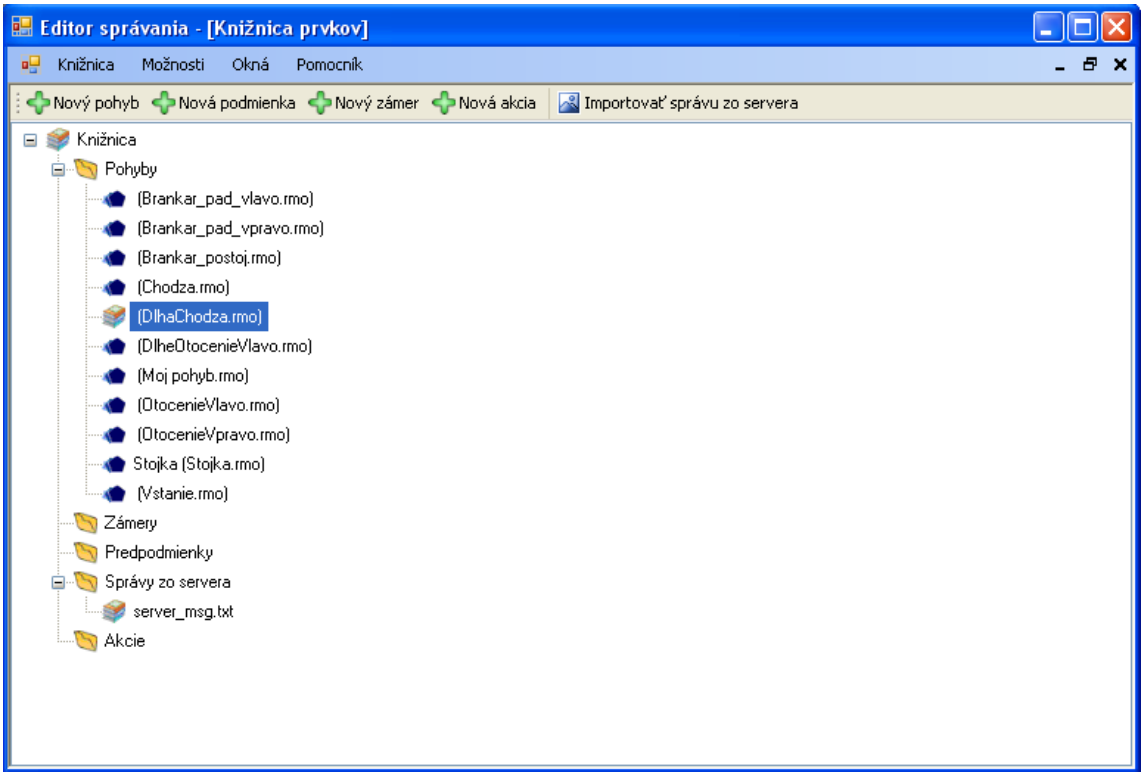
### 2.3.12.4 Zhodnotenie

Za zaujímavý prvok na tomto hráčovi považujeme architektúru jeho modulov, ktorá umožňuje modifikovať niektoré jeho časti a pritom zachovať funkcionalitu ostatných častí.

## 2.4 Analýza editora správania

Proces vytvárania a testovania pohybov agenta je komplexná úloha, vyžadujúca si spravidla veľké množstvo úsilia. Editor správania vytvorený tímom Agenty 007 v akademickom roku 2008/2009 je nástroj, ktorý tento proces výrazne uľahčuje. Ponúka možnosť jednoducho namodelovať nový pohyb a vzápätí ho vizualizovať na testovacom robotovi, bez nutnosti spustiť server, alebo agenta. Avšak jej tvorcovia umožnili aj testovanie priamo na serveri spustiteľné priamo z editora po vytvorení nového pohybu, alebo načítaní už vytvoreného.

Vytvorené pohyby sa ukladajú do knižnice pohybov, odkiaľ je možné ich spätne načítať.

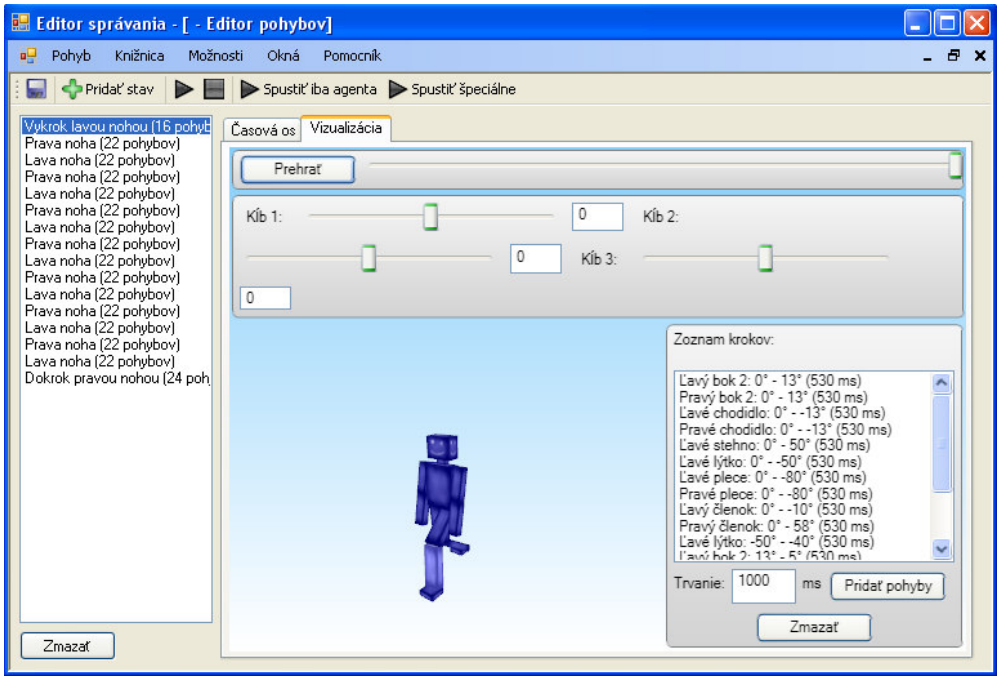



Obr. 22 Editor správania – knižnica pohybov

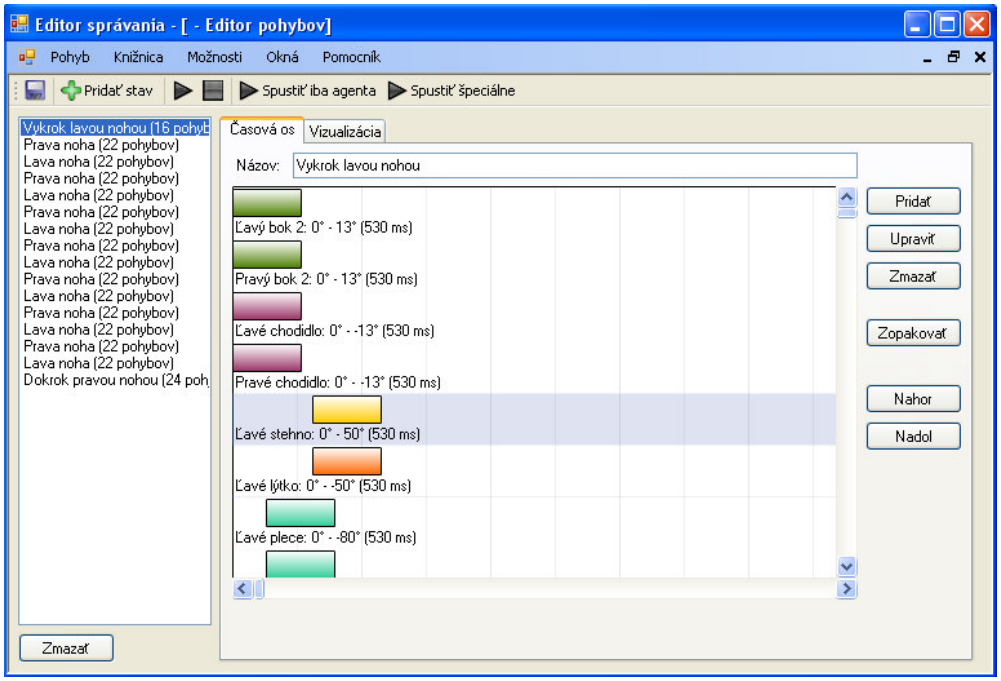
Na obrázku Obr. 22 je knižnica pohybov editora správania, odkiaľ je možné načítavať vytvorené pohyby pre ďalšiu editáciu, alebo simuláciu.

Na obrázku Obr. 23 je základné okno editora pohybov s načítaným pohybom. V ľavej časti sa nachádza zoznam stavov, z ktorých je pohyb zložený a počet jeho elementárnych pohybov. Každý z týchto stavov sa skladá z krokov (otočení jednotlivých kĺbov) rozdelených do fáz. Kroky sú zobrazené v pravej časti okna v Zozname krokov. Pri každom z nich sa nachádzajú informácie určujúce o ktorý kĺb ide, počiatočný a koncový uhol jeho natočenia a čas potrebný na jeho natočenie do požadovanej polohy.

V hornej časti okna sa nachádza tlačidlo slúžiace na prehratie vizualizácie na testovacom robotovi, ktorý sa nachádza v centrálnej časti okna. Nad tlačidlom prehrávania sú tlačidlá umožňujúce simuláciu pohybu na serveri, prípadne spustenie iba samotného agenta a tlačidlo pre špeciálne spustenie. Tento mód umožňuje spomalené, alebo zrýchlené vykonávanie pohybov, či nastavenie prestávok.

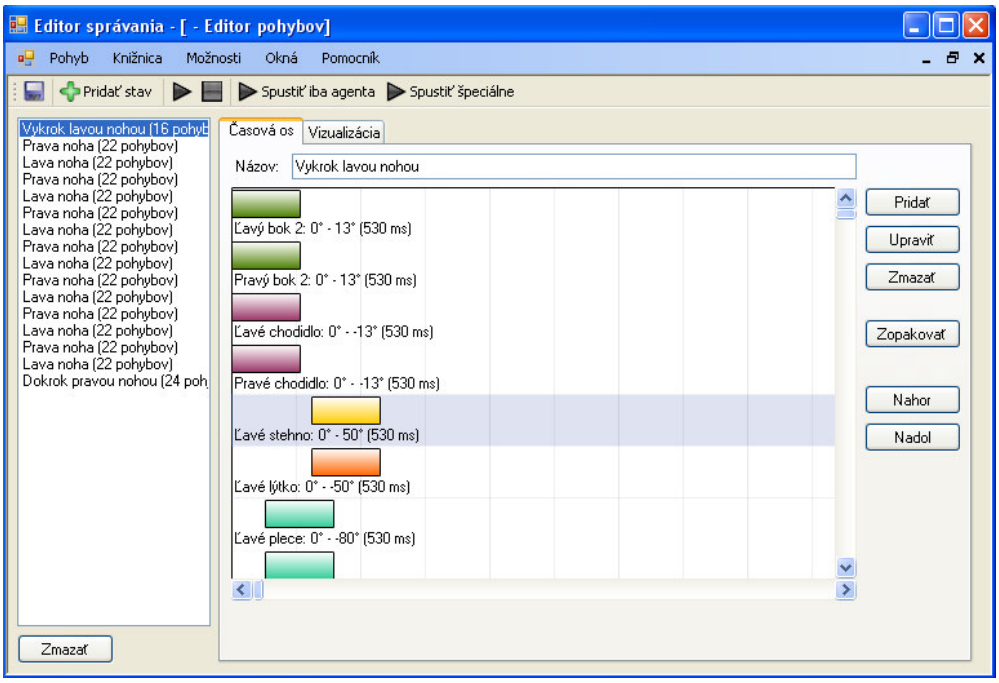



Obr. 23 Editor správania – načítaný pohyb vizualizovaný v editore




Obr. 24 Editor Správania – pohyb znázornený na časovej osi

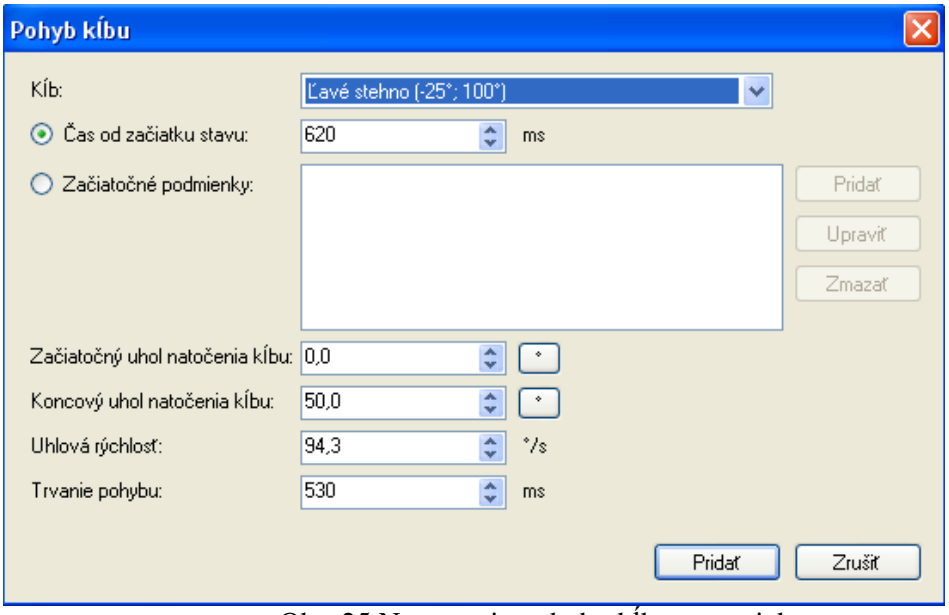
Na obrázku



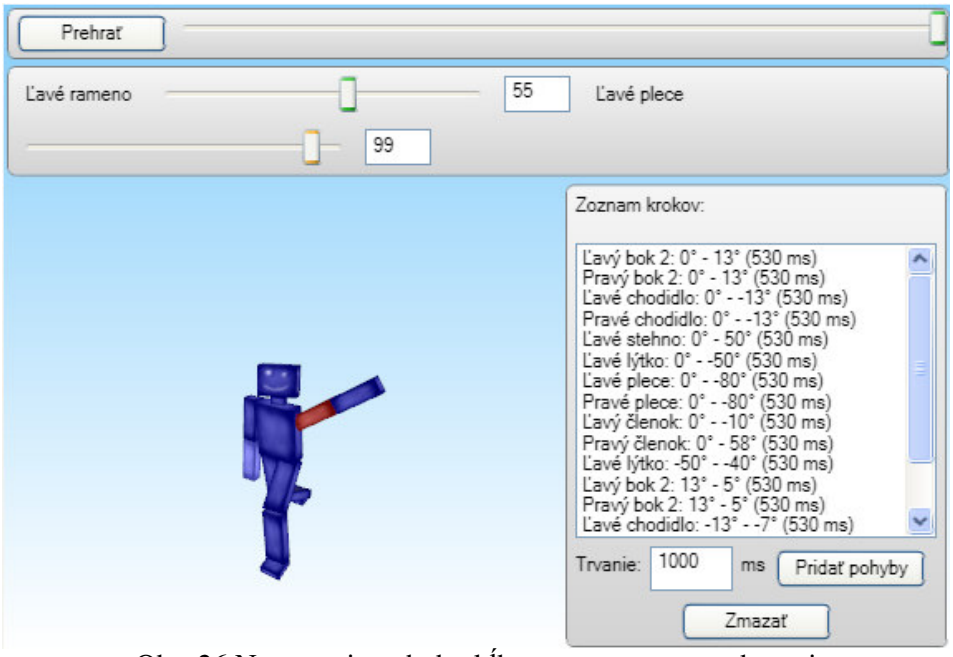

Obr. 24 sa nachádza časová os Stavu 1 (Výkrok ľavou nohou). Farebné obdĺžniky sú natočenia jednotlivých kĺbov. Pri každom sú údaje obsahujúce jeho názov, počiatočný a konečný uhol natočenia a čas potrebný na jeho vykonanie. Natočenia je možné v rámci osi ľubovoľne posúvať alebo meniť ich parametre.

Editor umožňuje vyskladať a simulovať jednoduché pohyby (napríklad zdvihnutie nohy), ale tiež komplexné série pohybov (napríklad chôdza). Vytvorenie elementárneho pohybu je možné dvomi spôsobmi:

a) Zadaním parametrov natočenia zvoleného kĺbu v numerickom tvare (vidieť na Obr. 25). Je nutné zadať čas od začiatku stavu, uhly natočenia a uhlovú rýchlosť alebo trvanie pohybu.
b) Zvolením kĺbu na testovacom robotovi a následné zadanie parametrov pomocou posuvníka (vidieť na Obr. 26), pričom výsledok sa zobrazí okamžite.




Obr. 25 Nastavenie pohybu kĺbu numericky




Obr. 26 Nastavenie pohybu kĺbu na testovacom robotovi

Editor umožňuje možnosť importu pohybu zo súboru (s príponou .rmo). Aplikácia je implementovaná v jazyku C#.


# 3 Špecifikácia

V časti špecifikácia sa venujeme opisu a detailnému rozobratiu požiadaviek súvisiacich s vývojom prototypu na zimný semester. Každá zo spomenutých požiadaviek je ďalej spomínaná v sekcii Návrh, kde sa venujeme týmto požiadavkám z hľadiska možnej budúcej implementácie.

## 3.1 Import a export XML z editora pohybov

Tím Dream Team, z ktorého sme sa rozhodli vychádzať, používa pre modelovanie pohybov XML súbor s presne definovaným formátom. Tím Agenty 007 používa na modelovanie pohybov editor, ktorého výstupom je textový súbor, ktorý dokáže rozpoznať iba aplikácia ich hráča a samotný editor. Prirodzenou požiadavkou je premostiť tieto dva formáty súborov tak, aby buď editor pohybov alebo hráč tímu Dream Team dokázali rozpoznať oba tieto formáty. Keďže sme sa rozhodli vylepšovať hráča tímu Dream Team, je nutné zapracovať do editoru pohybov možnosť importu tohto XML súboru. Po experimentovaní a úprave elementárnych kĺbových pohybov následne umožniť používateľovi exportovať tieto pohyby späť do XML súboru.

Aby sme toto dosiahli, je nutné prekonať paradigmatický rozdiel medzi ponímaním pohybov u oboch tímov. Tím Agenty 007 chápe pohyby ako jednoduchú sekvenciu elementárnych pohybov kĺbov, zatiaľ čo u tímu Dream Team je pohyb rozdelený najprv do fáz, z ktorých každá obsahuje tieto elementárne pohyby. XML obsahuje súbor všetkých pohybových schopností, ktorými hráč disponuje, zatiaľ čo editor pohybov umožňuje modelovať iba jednu sekvenciu. Preto pri importe je nutné používateľovi umožniť vybrať jednu z nízko úrovňových schopností v XML súbore a iba tú modelovať. Následne sa pri exporte do toho istého súboru táto sekvencia musí prepísať, nie iba pridať na koniec, pretože by mohli vzniknúť 2 rovnako pomenované schopnosti.

## 3.2 Zmeny v parseri kvôli novému typu servera

Dream Team používa automaticky generovaný „ABNF Parser Generator“-om [10]. Ten vytvorí C++ kód, ktorý funguje ako parser. Obsahuje syntaktickú aj sémantickú analýzu.

Hlavnou výhodou je, že na vytvorenie parsera je potrebné zadefinovať iba „čo“ bude robiť, nie „ako“. Funkcionalita je popísaná textovým súborom s príponou *bnf*. Obsahuje formuly jazyka ABNF (Augmented Backus-Naur Form) podľa RFC 4234 [11].

Nevýhodou ale je, že takto vygenerovaný kód nie je čitateľný. Časť funkcionality je dokonca reprezentovaná poľom hodnôt, ktoré predstavuje spustiteľný kód. Všetky úpravy nad parserom je možné robiť iba na úrovni popisu funkcionality. Tým vzniká isté riziko, pretože je potrebné vždy vygenerovať nový parser a po čase sa môže stať, že generátor už nebude kompatibilný s prostredím.

Pri generovaní parsera tímu *Dream Team* bola použitá verzia generátora 5.0 z Októbra 2007. Dostupná je už verzia 6.0 z Júna 2009.


## 3.3 Logovanie

Nakoľko je logovanie v robocupe dôležitý pomocník pri analyzovaní či už deja na ihrisku, pohybov hráča a podobne, je dôležité zamyslieť sa nad možnosťami vytvorenia logovania, ktoré by čo najmenej zaťažovalo samotný beh aplikácie a ktoré by bolo jednoducho upraviteľné a nezasahovalo by do kódu hráča, agenta a ostatných častí aplikácie, čím sa môže predísť neprehľadnému kódu.

Tím Hviezdna jedenástka implementoval logovanie vo forme samostatnej časti programu, ktorá sa kompilovala podmienene, aby v prípade, že logovanie nie je žiaduce, mohli hráča skompilovať bez logovacej časti. Logovacia časť aplikácie sa starala o logovanie viacerých záležitostí, napríklad logovanie komunikácie, vytváranie vlákien, pozícia lopty, kĺbov hráča a iné. Čo logovať a čo nie sa nastavovalo pomocou prepínačov [12].

```
void Logger::SetSwitches() {

     switches[LOG_RECIEVED_DATA] = false;
     switches[LOG_SENT_DATA] = false;
     switches[LOG_PARSE_TEST] = false;
     switches[LOG_COMMUNICATION] = false;
     switches[LOG_CONFIG] = false;
     switches[LOG_THREADS] = false;
     switches[LOG_TEMP] = true;

     switches[LOG_WORLD_BALL_POSITION] = false;

     switches[LOG_PLAYER_SIMTIME] = false;
     switches[LOG_PLAYER_GYROSCOPE] = false;
     switches[LOG_PLAYER_FRP] = false;

     switches[LOG_PLAYER_ANKLES] = false;
     switches[LOG_PLAYER_KNEES] = false;
     switches[LOG_PLAYER_LEGS] = false;
     switches[LOG_PLAYER_HIPS] = false;
     switches[LOG_PLAYER_SHOULDERS] = false;
     switches[LOG_PLAYER_ELBOWS] = false;
     switches[LOG_PLAYER_ARMS] = false;
}
```

V samotnej aplikácii sa logovanie volalo pomocou nasledovného kódu, ktorý zobrazuje logovanie samotného hráča [12].

```
Logger::logoutput(Logger::LOG_PLAYER_GYROSCOPE,
log_playermodel_gyroscope, gyroscope->absolute.x, gyroscope-
>absolute.y, gyroscope->absolute.z);
```

Vidíme, že logovanie zasahuje do kódu aplikácie, pretože je potrebné zavolať funkciu logovania v každej časti programu. V prípade, že by sme upravili logiku logovania a potrebovali vykonať zmeny aj vo volaní logovania, boli by sme nútení prejsť všetky zdrojové kódy vo všetkých súboroch, kde voláme logovací mechanizmus.


## 3.4 Pohyby hráča

V projekte Robocup3D sa miera dôležitosti prenáša na samotného agenta teda na tím robotických hráčov. Práve preto je prioritou zdokonalenie správanie jednotlivých hráčov. Požiadavky sú teda v tomto smere jednoznačné: zdokonaliť pohyb a správanie hráčov.

Náš tím sa rozhodol upravovať hráča tímu DreamTeam. Preto práve na tomto hráčovi budeme uplatňovať vylepšenia v oblasti pohybov. Hráčom sa pridajú nové alebo modifikované akcie v činnosti pohybovej ako aj hernej. Špeciálne by sa mali vytvoriť pohyby pre brankára, u ktorého by sa takto mohla zefektívniť jeho činnosť.

Jednotlivé pohyby hráčov by sa mali vytvárať pomocou editora pohybov, ktorý bude spolupracovať s XML súbormi. Náš hráč by mal mať zdokonalenú stabilitu a chôdzu. Pomocou daných pohybov by mohlo byť možné brániť pádom. Zdokonalíme vstávanie aj koordináciu. Chôdza bude riešená drobčením nakoľko, je pre hráča stabilnejšia.

Dostatok pozornosti budeme venovať brankárovi, ktorého správanie by sme chceli viac oddeliť od správania hráčov a tým znásobiť je potrebu v tíme.

## 3.5 Záznam zápasu

Rcssmonitor neslúži len na vizualizáciu aktuálne simulovaných zápasov. Dokáže prehrať aj záznam staršieho zápasu. Najnovšie dostupné záznamy zápasov pochádzajú z Majstrovstiev sveta 2008. Sú dostupné na [14]. Pri analyzovaní svetových tímov sme použili aj tieto záznamy.

Pri návrhu a vytváraní nových pohybov sa určite budem inšpirovať aj pohybmi ktoré používajú iné tímy. Avšak ak by sme boli schopný vytvoriť parser, ktorý dokáže získať natočenie kĺbov zo záznamu generovaného serverom naša práca by sa zjednodušila len na odladťovanie existujúcich pohybov, a dovolilo by nám to vytvoriť oveľa viac vlastných pohybov či už iných alebo podobných ako používajú iné tímy.

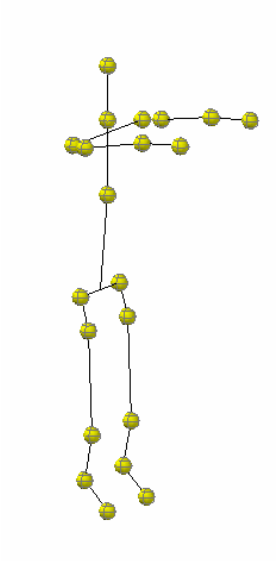
Pri analyzovaní záznamu servera sme zatiaľ prišli na niekoľko dôležitých informácií. Server si uchováva správy v pevnej štruktúre ohraničené delimitermi. Aj v prípade, že je obsah správy prázdny, delimitery sa uvádzajú v zázname. Správa obsahuje informáciu iba ak došlo k zmene stavu daného objektu. Každá správa, ak nejde o stav hry, obsahuje čas záznamu a potom nasledujú informácie o jednotlivých objektoch. Jeden robot sa skladá z 23 objektov.

Po prvotnej analýze záznamu servera sme dospeli k záveru, že pravdepodobne bude možné z tohto zdroja získať pohyby hráčov. Ďalšou analýzou bolo zistené čo znamenajú jednotlivé zo 16 hodnôt v časti SLT, ktoré definujú jeden objekt. Prvých 12 hodnôt definuje pravdepodobne otočenie objektu, ale touto informáciou nie sme si istý. Hodnoty 13, 14, 15 určujú pozíciu objektu sú to súradnice, ktoré budú pre ďalšie výpočty najdôležitejšie. Posledná hodnota udáva mierku v akej má byť objekt vykreslení a preto je pre nás nezaujímavá. Ďalšia časť v zázname určuje typ objektu, jeho rozmery a použité textúry.

Ukážka záznamu o jednom objekte:

(nd TRF (SLT 0.000186502 -1 9.1806e-005 0 1 0.000186498 -4.85457e-005 0 4.85285e-005 9.18151e-005 1 0 -5.17997 3.092 0.458662 1)(nd StaticMesh (load models/rupperarm.obj) (sSc 0.07 0.07 0.07)(resetMaterials matLeft naoblack naowhite)))

Na základe vizuálnej analýzy boli identifikované jednotlivé objekty a ich pozície pri základnom postavení hráča.(Obr. 27)



Obr. 27 Vizualizácia polohy bodov jednotlivých objektov v základnom postavení hráča


# 4 Návrh

Sekcia návrh pojednáva o možnostiach implementácie želaných požiadaviek. Rozpracováva detailnejšie požiadavky z hľadiska existujúcich prostriedkov a existujúcej infraštruktúry zdrojových kódov a snaží sa navrhnúť, akým spôsobom úspešne docieliť splnenie týchto požiadaviek.

V tejto časti bližšie opisujeme zmeny a vylepšenia, ktoré sme sa rozhodli uskutočniť a vyskúšať na prototype. Patria sem: Import a Export XML súboru z Editora správania, Vytvorenie parsera pre spracovanie záznamu zo servera, Úprava parsera správ prichádzajúcich zo servera, Vytvorenie nových pohybov hráča, Vytvorenie aspektovo-orientovaného logovania hráča. Ako základ nášho projektu sme sa rozhodli použiť existujúceho hráča Dream Teamu z našej fakulty.

## *4.1 Import XML do editora pohybov*

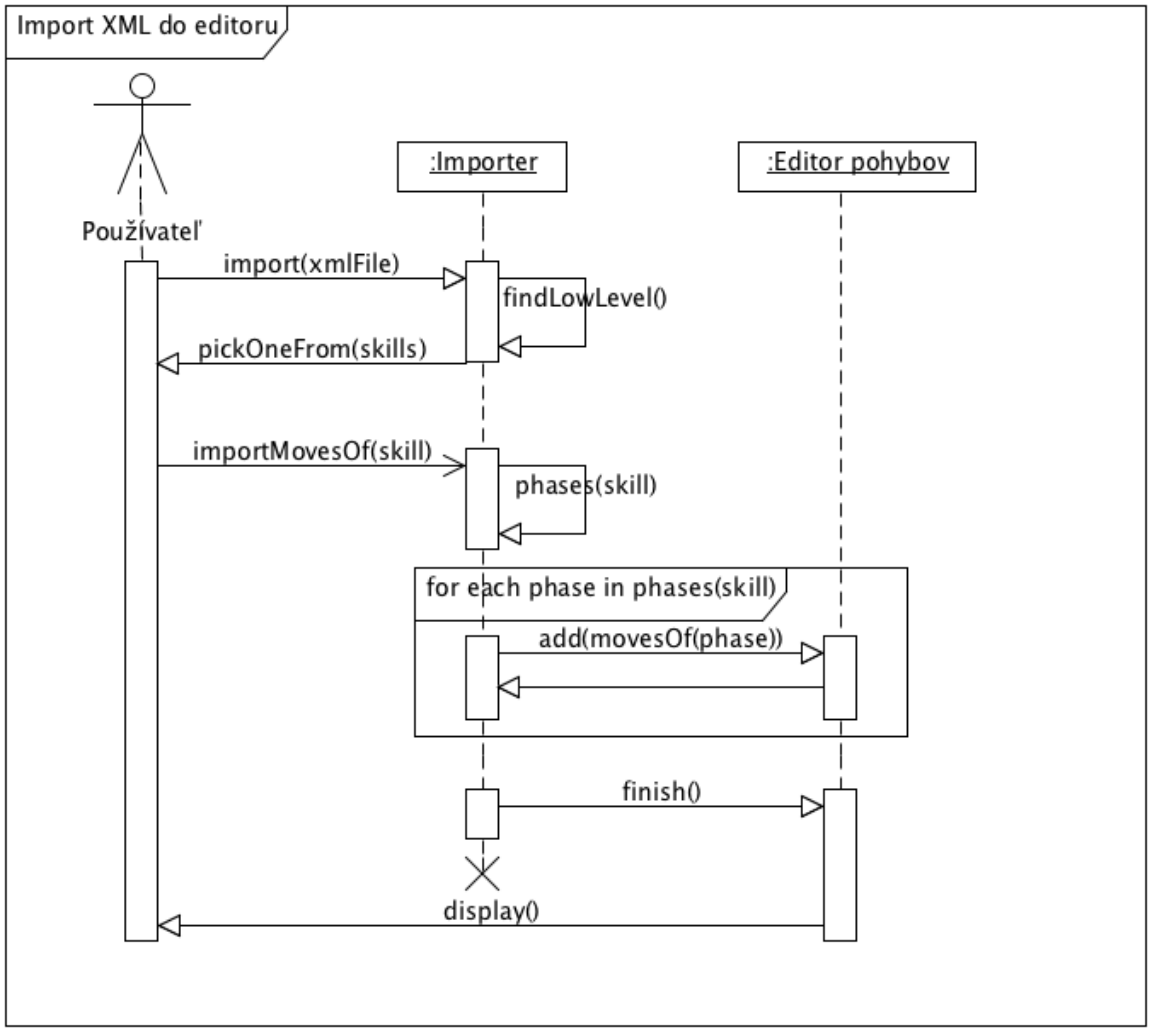
XML dokument s definíciou pohybov pozostáva z týchto kľúčových elementov:

high_ skill – abstrakcia vysoko-úrovňového správania, napríklad kráčania. Tento element je zaujímavý iba z hľadiska architektúry tímu Dream Team, pre import do editoru pohybov je nazaujímavý. Pozostáva práve z jedného elementu low_skill

low_skill – tento element je dôležitý z hľadiska importu, pretože začínajúc od tohto elementu dokážeme skonštruovať celú sekvenciu elementárnych pohybov. Práve zoznam týchto elementov bude rozparsovaný ako prvý a používateľovi sa dá na výber práve jeden low_skill na modelovanie. Obsahuje práve jednu inicializačnú fázu, pomocou ktorej sa dá vystopovať celá sekvencia elementárnych pohybov.

phase – tento element obsahuje zoznam želaných pohybov pre kĺby za zadaný čas. Tieto pohyby sa už dajú priamo mapovať na pohyby kĺbov v editore pohybov. Mimoriadne dôležitý je atribút next_phase, pomocou ktorého dokážeme naviazať ďalšiu fázu na spracovanie, vďaka čomu opäť môžeme pridať ďalšie kĺbové pohyby do editoru, atď. Keďže niektoré low_skill-y obsahujú cyklické opakovanie fáz, pohyby z každej fázy sa naimportujú práve raz. Dcérske elementy finalization_phase a rescue_movement budú v prototype ignorované.

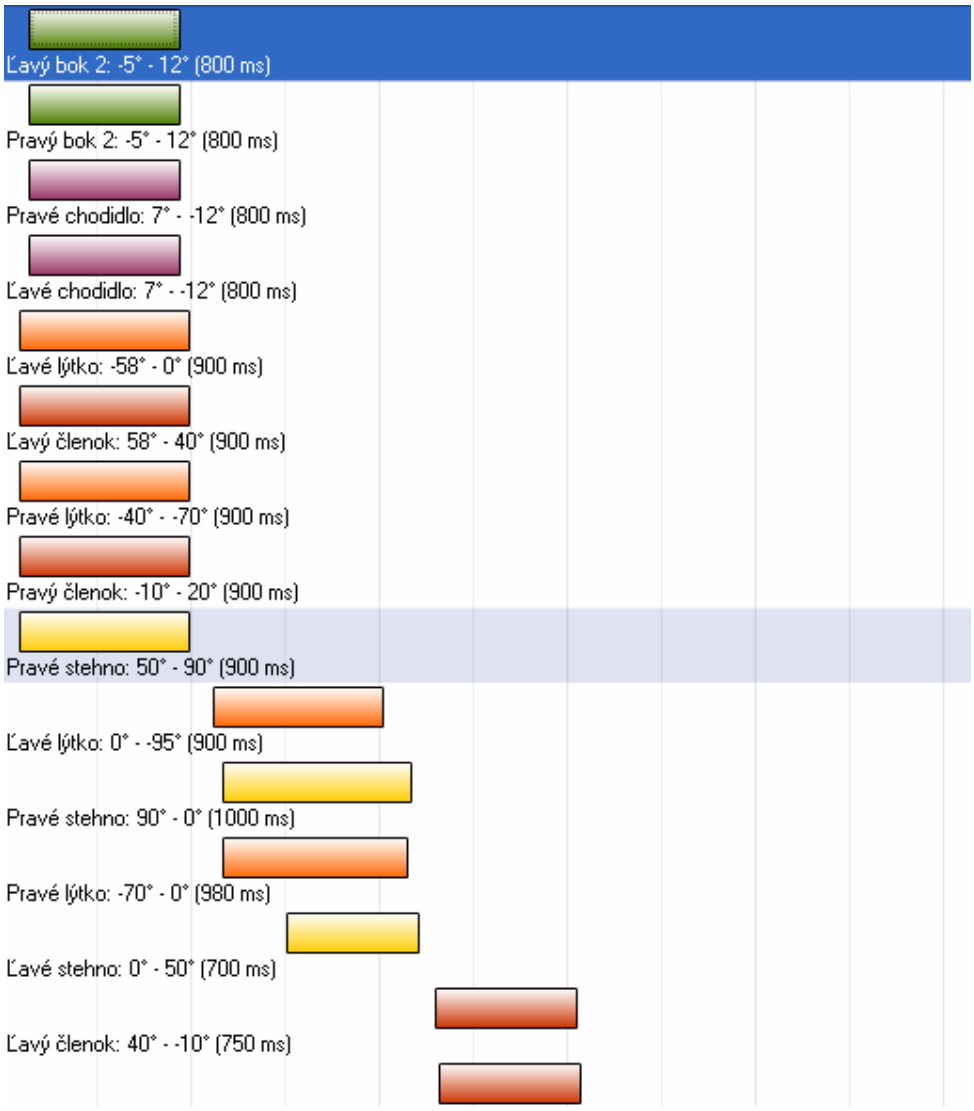
V sekvenčnom diagrame by teda import vyzeral ako na Obr. 28.




Obr. 28 Sekvenčný diagram importu XML do editoru pohybov

## 4.2 Export XML z editora pohybov

Pri opačnom postupe, t.j. exporte pohybov do XML súboru musíme vyriešiť, ako rozsekať sekvenciu pohybov na fázy, a tie následne spojiť do elementu low_skill. Reprezentáciu a usporiadanie elementárnych v čase v editore ukazuje Obr. 29.




Obr. 29 Usporiadanie pohybov v editore

Ako je na obrázku vidno, jednotlivé pohyby začínajú a končia v rôznych časových okamihoch, zatiaľ čo v XML konfigurácii tímu Dream Team má každá fáza jasne pridelené trvanie a pozíciu, ktorú majú kĺby obsadiť po skončení fázy. Na premostenie týchto dvoch konceptov je nutné rozsekať jeden plynúci pohyb na viacero fáz tak, aby po spojení týchto fáz vznikol opäť plynulý pohyb. Tiež je nutné naimplementovať "hluchú fázu", počas ktorej sa nebude vykonávať žiaden pohyb. Názvy fáz budú generované automaticky podľa ich poradia a názvu low_skillu, ktorý ich bude obsahovať.


Na ilustráciu roztriedenia na fázy by sa 2. štvorica pohybov na Obr. 29 rozbila na 3 fázy:

- prvá krátka 50 milisekundová fáza obsahujúca 2. štvoricu kĺbov
- druhá fáza, trvajúca 800ms, obsahujúca prvých 8 kĺbov
- tretia fáza, 50ms dlhá, obsahujúca 2. štvoricu kĺbov

## 4.3 Potrebné úpravy v parseri používanom Dream Teamom

Tá časť, ktorá sa pri novom serveri zmenila je reprezentovaná kódom [10]. Ten nepočíta s videním každej končatiny ako samostatného elementu. Rozšírením tohto kódu o potrebné elementy bude možné vygenerovať parser pre nový server.

Ako možnosť bolo zvážené použiť novšiu verziu generátora (v.6.0). Vzhľadom na rozsah potrebných úprav bude použitá pôvodná verzia (v.5.0). Prechod na iný generátor by priniesol zbytočné riziko v oblasti kompatibility.

***Ukážka kódu*** [10]*: ABNF kód na spracovanie správy SEE parserom*

```
;;; SEE - example: (See (F1L (pol 10.40 135.00 -17.78)) ... (P (team Robolog) (id 5) (pol 5.05
10.01 -8.11)))
MsgSee = "(" SeeStr *SP *SeeObject ")"

SeeObject = SeeOneObject / SeeOnePlayer
SeeOneObject = "(" ObjectName *SP "(" "pol" *SP SEEx *SP SEEy *SP SEEz "))" *SP
SeeOnePlayer = "(" PStr *SP "(" "team" *SP SEETeamName ")" *SP "(" "id" *SP
SEEPlayerID ")" *SP "(" "pol" *SP SEEPx *SP SEEPy *SP SEEPz "))" *SP

SeeStr = "SEE"
PStr = "P"
ObjectName  = identifier
SEETeamName = identifier
SEEPlayerID = 1*DIGIT
;;;kvoli dvojcifernym cislam - zistit ako sa to robi cez DIGIT
;;;SEEPlayerID = 1*DIGIT2

SEEx  = realNr
SEEy  = realNr
SEEz  = realNr
SEEPx  = realNr
SEEPy  = realNr
SEEPz  = realNr
```

## 4.4 Návrh logovania

Nakoľko by sme chceli oddeliť logovanie od kódu aplikácie, rozhodli sme sa implementovať ho aspektovo pomocou AspectC++. To nám umožní oddeliť kód logovania od kódu aplikácie a jednoduché zmeny v prípade, že bude potrebné pozmeniť formu alebo obsah logovania.

Vytvorením logovania pomocou aspektovo-orientovaného prístupu umožníme, aby bola logovacia časť aplikácie plne oddelená od samotnej aplikácie a jednoducho nasadená v inom programe.


Ukážka aspektu v AspectC++ [13]

```
aspect Counter{
     static int m_count;
     Counting():m_count(0){}

     pointcut counted()="Circle"||"Polygon";

     advice counted():classHelper{
          Helper(){Counter::m_count++;}
     }m_counter;

     advice execution("%main(...)"):after(){
          cout<<"Final count:"<<m_count<<"objects"<<endl;
     }
};
```

V aspektovo-orientovanom prístupe poznáme bodové prierezy a s nimi súvisiace exponované body spájania. V našom prípade týmito bodmi spájania budú napríklad volania funkcií, ktoré nastavujú kĺby hráča. Samozrejme sa budú zachytávať aj ostatné funkcie programu.

Kompilátor pre AspectC++ je implementovaný ako preprocesor a prevádza kód v AspecC++ do klasického C++ kódu. Pomocou neho vytvoríme súbor, ktorý ďalej predáme C++ kompilátoru. Nakoľko následne, až do zmien v logike logovania, nie je potrebné používať kompilátor pre AspectC++, nepredstavuje použitie aspektov v programe problém, ktorým by bola nutnosť vždy kompilovať najprv logovaciu časť pomocou AspectC++ a následne celý program pomocou kompilátora C++.

## 4.5 Pohyby hráča

Pri vytváraní pohybov budeme vychádzať z hráča tímu DreamTeam. Pohyby sa však budú realizovať v editore pohybov a následne export do XML.

Hlavné pohyby, ktoré budeme zdokonaľovať sú:

- chôdza
- pohyb brankára
- kop do lopty

*Chôdza* je navrhnutá vo forme drobčenia. Aj keď je potrebné viac pohybovať kĺbmi, je táto technika z hľadiska hráča oveľa stabilnejšia.

Jednotlivé kroky však v tomto prípade budú musieť byť vykonávané rýchlejšie.

*Pohyb brankára* bude výrazne pozmenený v zmysle zabránenia strely ak je brankár na zemi. Brankár ma v tomto momente možnosť pohybovať sa ťahaním svojho vlastného tela po zemi, teda plazenie. Týmto by mohol vykryť ohrozenú časť brány aj bez nutnosti vstať.

*Kop do lopty* je navrhnutý trochu z odlišného hľadiska ako bol zaužívané. Hráč pred kopom otočí chodidlo o 90 stupňom a tým kopne do lopty bokom končatiny. Úmerne s


veľkosťou plochy, ktorou hráč kopne do lopty by sa mala zvyšovať aj presnosť konkrétneho kopu.

## 4.6 Získanie pohybu zo záznamu

Cieľom je získať nové pohyby pre hráča vybratím daných pohybov zo záznamu. Celý proces bude odteraz prebiehať v niekoľkých krokoch.

Bude dokončená analýza záznamu, dátová aj vizuálna aby sme vedeli, ktoré informácie o kĺboch dokážeme získať priamo, a ktoré musíme vypočítať.

Po dokončení analýzy bude implementovaný parser schopný získať informácie o jednotlivých hráčoch, potrebné pre výpočet natočenia ich kĺbov. Parser by mal byť schopný detekovať pád hráča a na základe neho môžeme celý jeho pohyb po hracej ploche rozčleniť na menšie časti.

Parser bude exportovať pohyb jedného hráča počas celého zápasu do jediného XML súboru, kde budú zaznamenané všetky stavy kĺbov hráča počas zápasu. Takto spracovaný pohyb budeme schopný importovať do editora správania a môžeme z neho následne vybrať požadované časti pohybov a odladiť ich, keďže v zázname nie je pohyb ktorý hráč chcel vykonať, ale pohyb, ktorý skutočne vykonal.


# 5 Prototyp

Pri prototypovaní sa nám podarilo úspešne implementovať, či odskúšať navrhované zmeny a vylepšenia. Okrem toho sme vykonali ďalšie potrebné zmeny v kóde.

## 5.1 Import XML do editora

Ako bolo v špecifikácii a návrhu spomenuté, jedna z úloh bola spojiť efektivitu XML konfigurácie Dream Teamu s editorom od tímu Agenty 007. Závažným problémom sa ukázal prevod *speedConstant* z XML na čas, koľko samotný pohyb trvá. Agent Dream Teamu si určuje uhlovú rýchlosť pohybu kĺbu ako rozdiel medzi aktuálnou pozíciou kĺbu a požadovanou pozíciou na konci fázy. Tento rozdiel je následne vynásobený konštantou *speedConstant*, čo v praci znamená, že čím vyššia *speedConstant*, tým rýchlejší je pohyb kĺbu. Údaje v XML sú ale dané v uhlovej stupnici, pričom príkazu serveru by mali obsahovať uhlovú rýchlosť vyjadrenú v radiánoch. Pri skúmaní zdrojového kódu agenta Dream Teamu sa vzťah zjednodušil nasledovne:

$$time = \frac{1000 \cdot |desiredAngle - actualAngle|}{speedConstant \cdot |desiredAngle - actualAngle| \cdot 180} \frac{\pi}{} = \frac{1000\,\pi}{180\,speedConstant} \approx \frac{17{,}45}{speedConstant}$$

Tento vzťah určuje čas v sekundách, ktorý trvá fáza so zadaným *speedConstant*. Vzťah je ale iba aproximáciou trvania fázy – skutočné trvanie záleží od počiatočného nastavenia kĺbov. Pre získanie skutočných dĺžok fáz by bolo nutné simulovať pohyby vo fyzikálnom simulátore. Najväčšia pozorovaná odchýlka medzi skutočnou dĺžkou pohybu a aproximovanou bola 3%[4].

Okrem tejto nepresnosti sa pohyby úspešne naimportujú do editoru pohybov a sú pripravené na ďalšiu editáciu používateľom.

## 5.2 Export XML z editora

Úlohou bolo zabezpečiť export pohybov z editora do XML rovnakej štruktúry, ako má konfiguračné XML Dreamteamu. V editore sú jednotlivé množiny pohybov kĺbov rozdelené do stavov, v XML Dreamteamu namiesto stavov figurujú na seba nadväzujúce fázy. Pohyby v stavoch teda bolo potrebné rozsekať na jednotlivé fázy na základe začiatočných a konečných časov elementárnych pohybov kĺbov. Taktiež bolo potrebné dopočítať rýchlostnú konštantu na základe začiatočného natočenia kĺbu, konečného natočenia kĺbu a doby trvania pohybu.

Pri tvorbe fáz vznikol problém s takzvanou hluchou fázou, ktorá vzniká ak na seba dve po sebe nasledujúce fázy nenadväzujú hneď, ale je medzi nimi prázdny časový interval. Vo výstupnom XML hluché fázy nefigurujú, druhá fáza je posunutá tak, aby začínala v okamihu ukončenia predchádzajúcej.

Import z XML do editoru a export opačným smerom bol otestovaný, výstupom importovaného XML súboru a jeho následného exportu je XML súbor identický s pôvodným súborom.

4 436 / 421 milisekúnd


## 5.3 Parser

Keďže s príchodom novej verzie servera sa zmenil komunikačný protokol, museli sme vytvoriť aj nový parser. Komunikačný protokol je v tomto prípade reprezentovaný formátom a obsahom textovej správy odosielanej klientovi.

Parser bol generovaný nástrojom APG. Ten vygeneruje zdrojové kódy vykonávajúce parsovanie podľa stanovených pravidiel. Výsledný kód funguje tak, že pri nájdení jedného zo zvolených slov alebo premenných v zdrojovom súbore, spustí príslušnú funkciu na zmenu stavu. Tieto funkcie teda postupne zapisujú načítavané informácie o videných objektoch, pohyboch a pod.

Oproti pôvodnému parseru bolo potrebné rozšíriť definíciu videných objektov. V pôvodnej verzii hráč dostal od servera pre každý objekt jeden bod, na ktorom ho vidí (aj hráč bol reprezentovaný jedným bodom). Nové správy, ktoré posiela server, obsahujú informácie o každej časti objektu (končatiny hráča). Hráč bol teda rozšírený tak, že zoznam videných objektov je dynamický.

## 5.4 Zmeny v projekte Dream Teamu

Počas implementácie zmien boli identifikované vážne nedostatky v programe Dream Teamu. Nedostatky sa týkali prevažne únikov pamäte (z angl. memory leak), neefektívnej práce s objektami (odovzdávanie hodnoty kopírovaním namiesto referencie), nesprávnej enkapsulácie a niekoľkých prípadov nebezpečného kódu (bol volaný operátor delete nad objektom, ktorý nemusel byť vždy alokovaný dynamicky).

Pravdepodobne najväčší nedostatok bolo použitie zásobníkov. Tie boli realizované ako pole objektov (čo nie je odporúčané) s dodatočnou hodnotou, určujúcou, koľko objektov z poľa je aktuálne používaných.

Celý projekt bol preto refaktorizovaný a boli odstránené všetky identifikované nedostatky. Pôvodné „zásobníky“ (pole objektov + počet použitých) boli nahradené neštandardným zásobníkom, ktorý zabezpečuje správnu prácu s poľami aj efektivitu alokácie.

## 5.5 Logovanie

Logovanie je implementované ako samostatná trieda, ktorej funkcie pre zaznamenanie požadovanej informácie sú volané z rôznych miest aplikácie. Pre nastavenie parametrov logovania sa využíva textový súbor, ktorý obsahuje potrebné nastavenia. Pomocou neho sa nastavuje úroveň logovania ako log, debug alebo error. Implementované logovanie umožňuje aj zvolenie si vlastného názvu výstupného súboru. Táto informácia o tvare mena výstupného logovacieho súboru musí byť spolu s ostatnými nastaveniami zaznamenaná v konfiguračnom súbore. Konštruktor triedy Log nastaví tieto údaje len v prípade, že ešte nebola vytvorená žiadna jeho inštancia.

Logovanie informácií je umožnené buď do konzoly alebo do súboru. Je umožnené, aby sa logovacie dáta nezapisovali ani nezobrazovali na žiadnom mieste. Konfiguračný súbor umožňuje pridávanie komentárov, tieto ale musia byť v odlišnom riadku ako sú nastavenia samotných parametrov logovania.

Ukážka konfiguračného súboru:

#Komentár

PARAMETER = HODNOTA


Je nutné aby riadky s nastaveniami parametrov neobsahovali žiadne iné informácie ako tie, ktoré sú požadované. V prípade nekorektne nastavených parametrov sa pre logovanie použijú predvolené nastavenia.

Samotné inicializovanie triedy Log, okrem načítania parametrov logovania, použije ID hráča v názve výstupného logovacieho súboru. Týmto je zabezpečené aby každý hráč mal vlastný výstupný súbor pre logovanie.

Jednotlivé úrovne logovania, ktoré sú nazvané ako log, debug a error budú zaznamenávať informácie podľa ich typu, obyčajné údaje, údaje potrebné pre ladenie a chybové údaje. Každý typ logovania sa musí zavolať samostatne na požadovanom mieste v programe.

V nastavení parametrov logovania sa nachádzajú aj informácie o tom, ktoré časti programu sa budú zaznamenávať. Sú to napríklad prijaté dáta, odoslané dáta, pozície kĺbov, fáza pohybu, typ pohybu a iné. Počet a typ zaznamenávaných údajov sa ešte môže zmeniť v súvislosti s ďalším vývojom a podľa neskorších potrieb logovania.

## 5.6 Úprava servera

Predpokladom na otestovanie kopov do lopty bolo upraviť server tak, aby sa hráč mohol teleportovať[5]. Pri pokuse o kop mimo zápas bol totiž hráč serverom okamžite prenesený späť na svoju hraciu polovicu. To isté nastalo pri pokuse o prenesenie sa na súperovu polovicu. Na prekonanie týchto komplikácii bolo nutné

a) vypnúť kontrolu súradníc pri prijímaní beam príkazu

b) umožniť použitie tohto príkazu v ľubovoľnom hracom móde

c) upraviť pravidlá serveru tak, aby neodhadzoval hráča späť

Pri dosiahnutí týchto cieľov nám veľmi pomohol Ing. Marián Buchta[6], a to jednak pomocou návodu na stránkach serveru, ako aj osobne, radami a informáciami pri kompilácii.

Samotný návod [21] na stránke serveru zrozumiteľne objasňuje postup pre nastavenie prostredia, preto spomenieme iba poznatky, ktoré nie sú z tohto postupu evidentné.

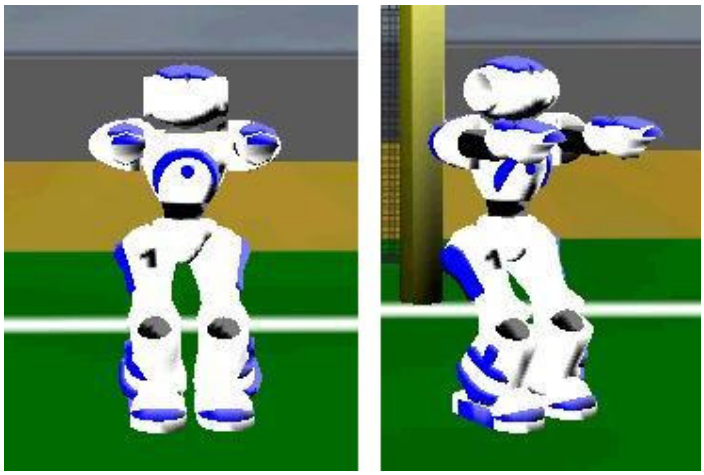
- Pri generovaní projektu očakáva vývojové prostredie knižnice z projektu libboost. Tieto knižnice majú mená začínajúce na „libboost", ale linker očakáva knižnice začínajúce sa na reťazec „boost". Premenovanie súborov vyriešilo tento problém.
- Projekt je nutné skompilovať ako Release verziu. Debug verzie projektu sú nekompatibilné so zvyškom servera
- Vytvorenie .exe súboru samotného servera – simspark.exe samo osebe prakticky nič nerieši. Pre väčšinu praktických zmien je treba vytvoriť knižnicu pravidiel – soccer.dll, a touto knižnicou nahradiť existujúcu v lib/rcssserver3d

Výsledkom bol server, ktorý umožňoval hráčovi prekročiť poliacu čiaru v ľubovoľnom hracom móde a premiestňovať sa podľa potreby, čo umožnilo otestovať a kvantifikovať atribúty rôznych druhov kopov, ktoré sme v rámci prototypu vytvorili.

5 Pomocou príkazu „beam"
6 Jeden z ľudí pracujúcich na vývoji servera pre RoboCup3D

## 5.7 Pohyby hráča

Všetky pohyby vychádzajú zo základného postavenia hráča, ktoré je zobrazené na Obr. 30. Po vykonaní pohybu sa do tohto postavenia musia zase vrátiť. Všetky pohyby sú implementované do obidvoch strán.



Obr. 30 Základné postavenie hráča

Pri implementácií pohybov pomocou XML súboru, ale aj všeobecne, sa oplatí posúvať kĺby skôr v menších uhloch. Pri použití väčších uhlov obvykle nastáva problém s ladením pohybu, pretože každá zmena rýchlosti alebo inej fázy môže robota značne vychýliť. Samozrejme, nie vždy sa dá použitiu väčšieho uhla vyhnúť, napr. pri kopaní do lopty. Priemerný pohyb kĺbov v rámci jednej fázy je v našom prípade okolo 30 až 40 stupňov.

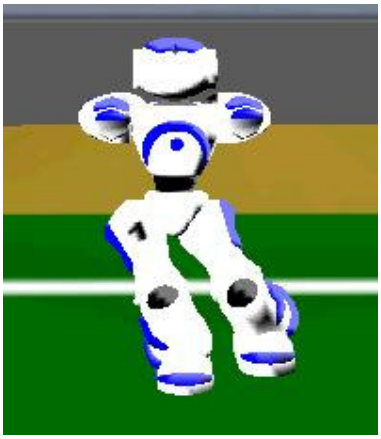
Základom pri tvorbe pohybov je udržanie stability. To sa dosahuje rôznym presúvaním ťažiska. Všeobecne platí, že čím nižšie ťažisko, tým väčšia stabilita. Preto má robot v základnom postavení pokrčené kolená. Presúvanie ťažiska sa najčastejšie robí nakláňaním robota na stranu, pomôcť pri tom môžu aj ruky, ktoré sa inak veľmi nepoužívajú.

Pri tvorbe pohybov je najlepšie postupovať fázu po fáze, teda vždy čo najviac vyladiť aktuálnu fázu, až potom prejsť na ďalšiu. Je to najmä preto, že po dokončení pohybu býva ladenie celkom zložité a zmena v niektorých skorších fázach ľahko naruší celý pohyb (prejaviť sa dokážu i naozaj malé zmeny uhlov alebo rýchlosti).

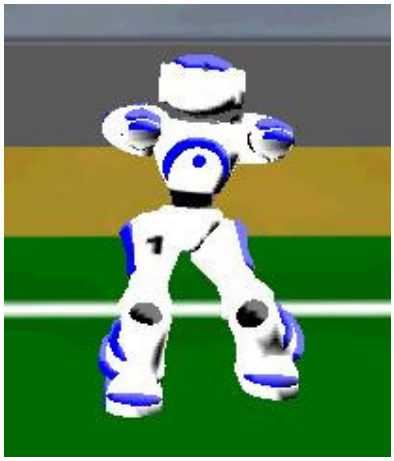
### 5.7.1 Chodenie do boku

Predpokladajme, že agent bude chcieť kráčať doľava. Na začiatku prenesie svoje ťažisko na pravú nohu (nakloní sa doprava) a zdvihne ľavú nohu. Následne zmení svoje naklonenie na pravej nohe do opačnej strany, čím sa vlastne vykoná samotný krok. Robot teraz stojí rozkročmo. Pohyb pokračuje naklonením a presunutím ťažiska na ľavú nohu, pritiahnutím pravej nohy a vyrovnaním do základného postavenia. Následne sa cyklus opakuje. Jednotlivé fázy ilustrujú Obr. 31 až Obr. 33.

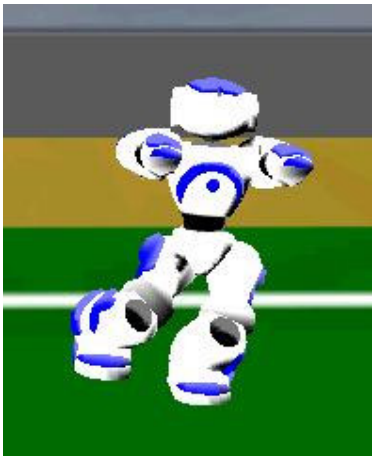
Tento pohyb sa vyznačuje vysokou náchylnosťou na zmenu rýchlosti a uhlov otočenia jednotlivých kĺbov. I malá zmena môže v horšom prípade znamenať pád, v lepšom vychýlenie smeru.



Obr. 31 Presunutie ťažiska na pravú nohu a zdvihnutie ľavej nohy



Obr. 32 Presunutie na pravej nohe



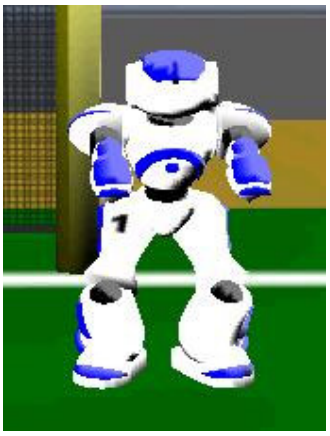
Obr. 33 Pritiahnutie pravej nohy a vyrovnanie

### 5.7.2 Otáčanie

Opäť predpokladajme, že sa agent bude chcieť otočiť smerom doľava. Na začiatku vytočí svoje nohy tak, aby smerovali od seba, čím sa zároveň predkloní. Tento pohyb predstavuje mierne zneužitie simulačného prostredia, keďže agent pri ňom šúcha nohy po

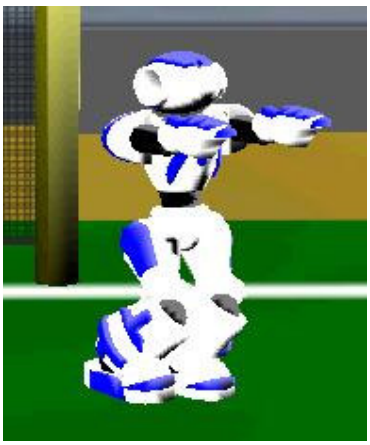
zemi, čo by v reálnom prostredí nebolo možné. Ďalej agent pokračuje naklonením doľava, zdvihnutím pravej nohy a otočením sa na ľavej nohe. Následne sa vráti do základného postavenia a cyklus začína odznovu.

V jednom cykle sa agent otočí približne o 30 stupňov. Otočenie o 360 stupňov trvá približne 12 sekúnd a zaberie približne 7 cyklov. Toto otáčanie je vhodné najmä pre menšie uhly (menej ako 90 stupňov), no dá sa použiť aj pre väčšie.



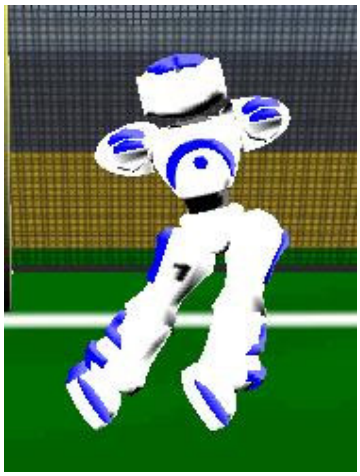
Obr. 34 Predklon a rozkročenie

Obr. 35 Presunutie ťažiska na ľavú nohu a zdvihnutie pravej

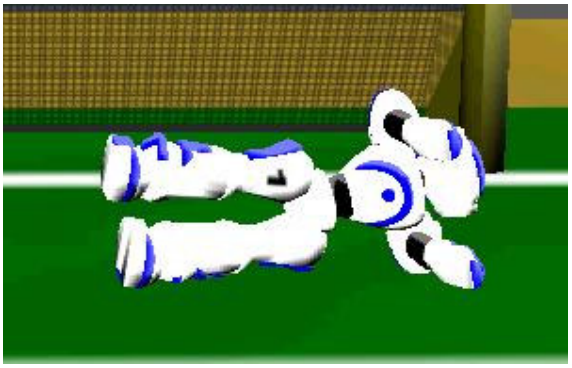


Obr. 36 Otočenie na ľavej nohe a vyrovnanie

### 5.7.3 Padanie brankára



Obr. 37 Preváženie na stranu



Obr. 38 Brankár na zemi

### 5.7.4 Kopy robota

V rámci prototypovania sa nám podarilo navrhnúť a implementovať 3 nové verzie kopu do lopty pričom celkových kopov bolo vytvorených 5:

- Kopnutie do lopty bokom chodidla ľavou nohou v smere cca 45° (Bocny_kop_lava_noha)
- Kopnutie do lopty bokom chodila pravou nohou v smere cca 45° (Bocny_kop_prava_noha)
- Kopnutie do lopty bokom chodidla ľavou nohou v smere 90° (DoBoku_kop_lava_noha)
- Kopnutie do lopty bokom chodidla ľavou nohou v smere 90° (DoBoku_kop_prava_noha)
- Kopnutie do lopty v smere dozadu ľavou nohou (DoZadu_kop_lava_noha)

Rôzne typy kopov boli implementované pre zdokonalenie kopacej techniky. Hráč môže rôznymi kopmi zminimalizovať čas potrebný na kopnutie do lopty. Podstata je v tom, že sa nemusí otáčať do potrebného smeru. Ak potrebuje vyslať loptu trochu šikmo použije bočný kop. Ak ju potrebuje poslať doprava či doľava použije Kop do doku a paralelne kop dozadu.

Štatistika jednotlivých kopov je zobrazená v tabuľke Tab. 2, kde je zobrazený priemerný čas vykonania kopu. Nasleduje diaľka kopu, ktorá však závisí na správnom postavení hráča pred loptou. Diaľka je vyjadrená v zlomkoch ihriska. Nakoniec pravdepodobnosť pádu, ktorá sa vypočítala jednoduchým vydelením počtu pádov desiatim (10 pokusov na každý pohyb).

Tab. 2 Štatistika kopov

| **Typ kopu** | **Čas** | **Diaľka** | **Pravdepodobnosť pádu** |
|---|---|---|---|
| Bočný kop | 2s | 2/10 ihriska | 20% |
| Kop do Boku | 1s | 2/10 ihriska | 10% |
| Kop dozadu | 1,5s | 3/10 ihriska | 20% |

### 5.7.4.1 Bočný kop

Bočný kop ma veľkú výhodu pri potrebe kopnutia lopty cca v 45 stupňovom uhle. Pri tomto kope sa hráč nemusí natáčať smerom kam chce kopnúť a takisto takáto strela na bránu môže byť prekvapujúca pre brankára.

Hráč v jednotlivých fázach daného pohybu najprv premiestni ťažisko na pravú nohu. Ľavú nohu zdvihne v smere od tela do boku. Následne zmení na točenie bedrového kĺbu čo spôsobí, že noha smeruje k lopte bokom chodidla. Hráč už len natočí nohu v kolene a premiestni nohu v smere dopredu pričom zasiahne loptu.

Analogicky je implementovaný pohyb aj pre pravú nohu. Medzi týmito pohybmi nie je rozdiel ani v čase ani v kvalite kopu.

Bočný kop je znázornený na Obr. 39. Obrázok je zachytený tesne po kopnutí do lopty.

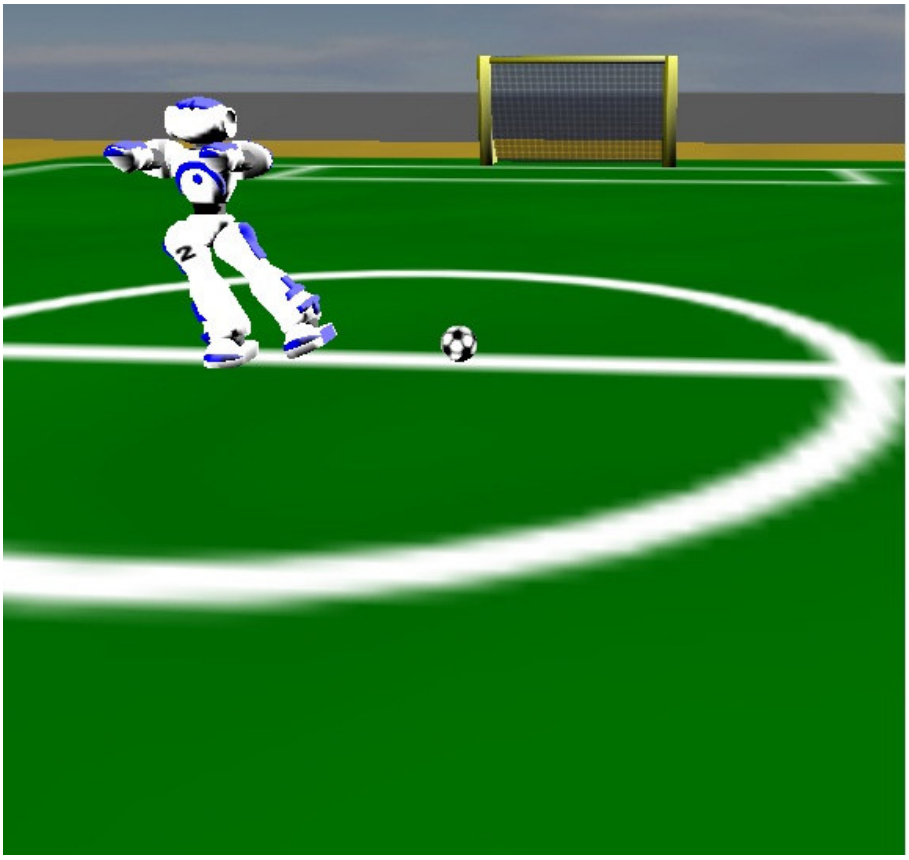
Obr. 39 Bočný kop

### 5.7.4.2 Kop do Boku

Kop do boku je myslený skôr ako prihrávka ale môže sa využiť taktiež ako strela na bránu. Má dobrú stabilitu, hráč pri ňom nepadá a takisto je to veľmi rýchly kop. Hráč sa pri kope v 90 stupňov uhle nepotrebuje zdĺhavo presúvať pred loptu stačí keď sa postaví vedľa nej a použije kop do boku.

Samotný kop je veľmi jednoduchý a teda neobsahuje veľmi veľa pohybov. V prvej fáze hráč natočí svoje členky a bedrové kĺby tak aby dosiahol prenesenie ťažiska na pravú nohu. Následne keď je ľavá noha odľahčená a nenesie žiadnu váhu vystrelí smerom do boku pričom zasiahne loptu.

Analogicky je implementovaný pohyb aj pre pravú nohu. Medzi týmito pohybmi nie je rozdiel ani v čase ani v kvalite tento kop je znázornený na Obr. 40. Obrázok je zachytený tesne po kopnutí do lopty.



Obr. 40 Kop do boku

### 5.7.4.3 Kop dozadu

Pri predpoklade, že hráč bude bývať otočený smerom k súperovej bráne tento kop by sa mal využívať ako nahrávka, kde by prihral hráčovi za sebou. Teoreticky je možné však využitie tohto kopu aj smerom na bránu nakoľko tento kop ma zo všetkých nových kopov najväčšiu razanciu a dosahuje najväčšiu diaľku.

Hráč sa, takisto ako pri ostatných kopoch, nahne doprava a prenesie ťažisko na pravú nohu. Nasleduje zdvihnutie ľavej nohy a zároveň jej ohnutie v kolene. V poslednej fáze len takto ohnutú nohu prenesie po celom obvode uhlu bedrového kĺbu čím sa spôsobí kopnutie do lopty spodnou časťou chodidla v smere dozadu.

Tento kop je znázornený na Obr. 41. Obrázok je zachytený tesne po kopnutí do lopty.

Obr. 41 Kop dozadu

## 5.8 Parser logu servera

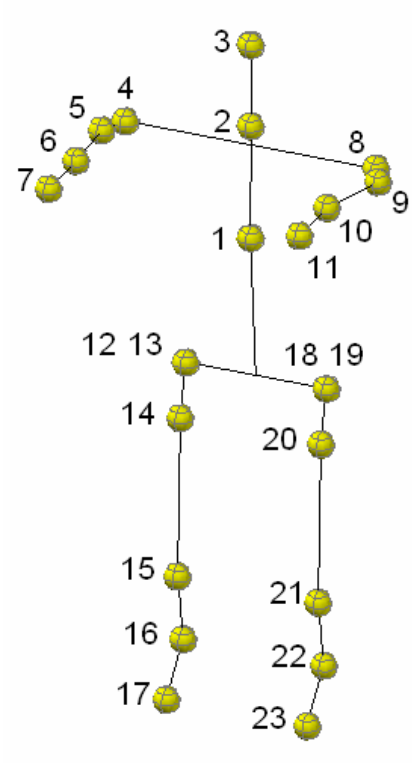
Na základe analýzy bol vypracovaný návrh parsera logu, ktorý mal z dostupných informácií v logu servera vypočítať natočenie kĺbov robota.

Robot má 22 kĺbov a skladá sa z 23 objektov. Všetky kĺby a objekty sú opísané v nasledujúcich tabuľkách a zobrazené na obrázkoch. Na Obr. 43 sú označené jednotlivé kĺby tak ako boli číslované pri výpočtoch. Na Obr. 42 sú objekty, z ktorých sa robot skladá redukované na gule s polomerom 0,01 určujú iba bod kde sa objekt nachádza neobsahujú informáciu o natočení objektu. Tab. 4 opisuje objekty, z ktorých sa skladá robot a obsahuje aj súradnice pri pripojení robota, keď stojí v základnom postavení[7]. V tabuľke Tab. 3 sú kĺby, je uvedené ich označenie podľa editora správania a približná poloha vzhľadom na objekty.

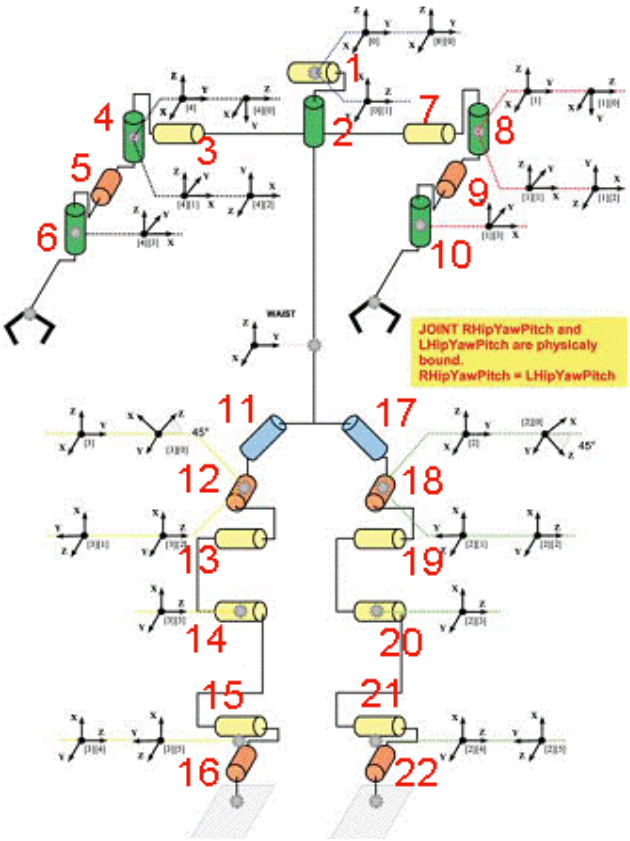
Keďže sa nám nepodarilo identifikovať čo znamená prvých 12 číslic definujúcich objekt, počítali sme pomocou analytickej geometrie podľa bodov čiar a plôch. Podarilo sa nám vypočítať natočenia kĺbov výsledky však nie sú úplne presné. Ich presnosť je však postačujúca pre ďalšie použitie.

Hlavným nedostatkom, ktorý najvýraznejšie vplýva na výkyvy vo výpočtoch je predpoklad, že niektoré objekty sú pevne spojené, ale v niektorých momentoch sa odchýlia a spôsobí to zmenu v natočení niektorých čiar a plôch. Táto chyba však býva nanajvýš 1-2 stupne čo nie je považované za dostatočný pohyb kĺbu. Keďže takáto zmena nemá výrazný vplyv na robota.

[7] Základné postavenie je postavenie keď je natočenie všetkých kĺbov 0.




Obr. 42 Pozície objektov robota




Obr. 43 KÍby robota


Tab. 3 Zoznam kĺbov

| Id Kĺbu | Označenie v Parseri | Približná poloha kĺbu z pohľadu objektov (táto informácie nie je presne podložená ide len o približnú informáciu na základe odhadu a výpočtov) |
|---|---|---|
| 1 | HE2 | Nebola presnejšie určená, ale mal by sa nachádzať až nad bodom krku |
| 2 | HE1 | Nebola presnejšie určená, ale mal by byť zhodný s predchádzajúcim bodom |
| 3 | RAE1 | Kĺb by sa mal nachádzať v pravom pleci |
| 4 | RAE2 | Kĺb by sa mal nachádzať v pravom pleci |
| 5 | RAE3 | Kĺb by sa mal nachádzať v pravom lakti |
| 6 | RAE4 | Kĺb by sa mal nachádzať v pravom lakti |
| 7 | LAE1 | Kĺb by sa mal nachádzať v ľavom pleci |
| 8 | LAE2 | Kĺb by sa mal nachádzať v ľavom pleci |
| 9 | LAE3 | Kĺb by sa mal nachádzať v ľavom lakti |
| 10 | LAE4 | Kĺb by sa mal nachádzať v ľavom lakti |
| 11 | RLE1 | Kĺb by sa mal nachádzať v pravom bedrovom kĺbe |
| 12 | RLE2 | Kĺb by sa mal nachádzať v pravom bedrovom kĺbe |
| 13 | RLE3 | Kĺb by sa mal nachádzať v pravom bedrovom kĺbe |
| 14 | RLE4 | Kĺb nie je presne určený pre výpočet je použitá približná pozícia, ktorá vychádza z priesečníku osy stehna a lýtka |
| 15 | RLE5 | Kĺb by sa mal nachádzať v pravom členku |
| 16 | RLE6 | Kĺb by sa mal nachádzať v pravom členku |
| 17 | LLE1 | Kĺb by sa mal nachádzať v pravom bedrovom kĺbe |
| 18 | LLE2 | Kĺb by sa mal nachádzať v pravom bedrovom kĺbe |
| 19 | LLE3 | Kĺb by sa mal nachádzať v pravom bedrovom kĺbe |
| 20 | LLE4 | Kĺb nie je presne určený pre výpočet je použitá približná pozícia, ktorá vychádza z priesečníku osy stehna a lýtka |
| 21 | LLE5 | Kĺb by sa mal nachádzať v ľavom členku |
| 22 | LLE6 | Kĺb by sa mal nachádzať v ľavom členku |


Tab. 4 Zoznam objektov

| ID Objektu | Slovné označenie | Súradnica X | Súradnica Y | Súradnica Z |
|---|---|---|---|---|
| 1 | Torso | -5,2 | 3,2 | 0,3837 |
| 2 | Krk | -5,2 | 3,2 | 0,4737 |
| 3 | Hlava | -5,2 | 3,2 | 0,5387 |
| 4 | Pravé plece (kĺb) | -5,2 | 3,102 | 0,4587 |
| 5 | Pravé rameno | -5,18 | 3,092 | 0,4587 |
| 6 | Pravý lakeť (kĺb) | -5,11 | 3,102 | 0,4677 |
| 7 | Pravé predlaktie | -5.06 | 3,102 | 0,4677 |
| 8 | Ľavé plece (kĺb) | -5,2 | 3,298 | 0,4586 |
| 9 | Ľavé rameno | -5,18 | 3,308 | 0,4586 |
| 10 | Ľavý lakeť (kĺb) | -5,11 | 3,298 | 0,4676 |
| 11 | Ľavé predlaktie | -5,06 | 3,298 | 0.4676 |
| 12 | Pravý bedrový kĺb | -5,21 | 3,145 | 0,2687 |
| 13 | Pravý bedrový kĺb | -5,21 | 3,145 | 0,2687 |
| 14 | Pravé stehno | -5,2 | 3,145 | 0,2287 |
| 15 | Pravé lýtko | -5,195 | 3,145 | 0,1037 |
| 16 | Pravý členok (kĺb) | -5,205 | 3,145 | 0,0487 |
| 17 | Pravá šľapa | -5,175 | 3,145 | 0,0138 |
| 18 | Ľavý bedrový kĺb | -5,21 | 3,255 | 0,2687 |
| 19 | Ľavý bedrový kĺb | -5,21 | 3,255 | 0,2687 |
| 20 | Ľavé stehno | -5,2 | 3,255 | 0,2287 |
| 21 | Ľavé lýtko | -5,195 | 3,255 | 0,1037 |
| 22 | Ľavý členok (kĺb) | -5,205 | 3,255 | 0,0487 |
| 23 | Ľavá šľapa | -5,175 | 3,255 | 0,0138 |

Pri vytváraní parseru sme si overili či je možné pomocou dostupných informácií z logu vypočítať natočenie kĺbov. V prototype sa nám podarilo vyriešiť natočenie 10 z 12 kĺbov na nohách. Implementovali sme parser do editora pohybov a z výsledkov sme vytvorili následnosť pohybov, ktoré môžme v editore ďalej upravovať testovať a ladiť. Existuje predpoklad, že pomocou zvyšných 12 čísel definujúcich objekt dokážeme vypočítať natočenie objektu, čo by nám mohlo umožniť zmeniť metódu pre výpočet uhlov kĺbov.


# 6 Použité zdroje

V tejto sekcii uvádzame použité zdroje. Pri internetových zdrojoch, ak nie je uvedené inak tak sú z 1.novembra 2009.

[1] Team UI-AI, UI-AI3D 2007 Team Description, 2007, posledný prístup 18.10.2009, www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/uiai2007TDP.pdf
[2] GHAHFARROKH B. S., FAGHRI F. a i., UI-AI 2008 Mixed Reality Team, 2008, posledný prístup 18.10.2009, http://www-lehre.informatik.uos.de/~jesmeyer/paper/TDP-MR-RC2008-UI-AI.pdf
[3] SEU Red Sun, http://me.seu.edu.cn/sfzx/rescue/index.html (15.10.2009)
[4] Agenty 007, http://labss2.fiit.stuba.sk/TeamProject/2008/team07is-si/dokumentacia.html (15.10.2009)
[5] ENDERT H.,WETZKER R., KARBE T. HESSLER A., BUTTNER P.,BROSSMAN F. : The Dainamite Agent Framework, TU Berlin Faculty of Electrical Engineering and Computer Science
[6] WALLS, E.: Spring in Action, Manning © 2008, ISBN: 1-933988-13-4
[7] http://www.springsource.org/ : oficiálna stránka Spring Frameworku
[8] URBANEC, M: Rozhodovanie sa Hráča s Loptou, 2009
[9] LIGOCKÝ, J., POLÁK, M., HRUBÝ, M., PÁN, G., HRIC, J.: 12.hráč – Technická Dokumentácia, 2009
[10] Lowell D. Thomas, APG – An ABNF Parser Generator, verzia 5.0, Október 2007, posledný prístup 1.11.2009, http://www.coasttocoastresearch.com/description-50.php
[11] RFC 4234, The Internet Engineering Task Force (IETF), dostupné zo stránky IETF, http://www.ietf.org/rfc.html
[12] Hviezdna jedenástka, zdrojové kódy finálnej aplikácie. Dostupné na internete: http://labss2.fiit.stuba.sk/TeamProject/2007/team11is-si/download/sirius.zip
[13] AspectC++ language reference. Dostupné na internete: http://www.aspectc.org/fileadmin/documentation/ac-languageref.pdf
[14] Logfily z RC 08 http://robocup.martenvdsanden.net/index.php
[15] Team Description Paper tímu Fantasia http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/fantasia2007tdp.pdf
[16] Domovská stránka tímu FC Portugal http://www.ieeta.pt/robocup/
[17] Dokumentácia k tímu BATS https://sourceforge.net/projects/littlegreenbats/
[18] Stránka tímu Jahodoví Princovia, 2007, http://labss2.fiit.stuba.sk/TeamProject/2007/team16is-si/
[19] Stránka tímu Gang Of Six, 2006, http://labss2.fiit.stuba.sk/TeamProject/2006/team03/
[20] Stránka tímu Dream Team, 2008 ,http://labss2.fiit.stuba.sk/TeamProject/2008/team01is-si/
[21] Postup inštalácie servera na Windows: http://simspark.sourceforge.net/wiki/index.php/Installation_on_Windows (4.12.2009)