SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

RoboCup 3D

Tímový projekt
Dokumentácia k produktu

Tím: Androids (tím č. 5)
Vedúci tímu: Ing. Ivan Kapustík
Členovia tímu: Bc. Juraj Belanji
Bc. Miroslav Hruška
Bc. Roman Kováč
Bc. Andrej Minárik
Bc. Veronika Wolfová
Študijný odbor: Softvérové inžinierstvo
Akademický rok: 2010/2011

Obsah

1 Úvod

Tento dokument slúži ako dokumentácia k tímovému projektu tímu Androids, vytvorená v predmete Tímový projekt na tému RoboCup 3D v akademickom roku 2010/2011. V dokumente sa nachádzajú informácie týkajúce sa analýzy, návrhu a implementácie agenta – robota – pre tému simulovaného futbalu RoboCup 3D.

1.1 Zadanie

Téme RoboCup, presnejšie lige simulovaného robotického futbalu, sa naši študenti venujú už jedenásť rokov. Tímy študentov, či už v rámci umelej inteligencie alebo tímového projektu, sa snažia vytvárať a vylepšovať programy, ktoré simulujú správanie sa futbalového agenta. Každý tím sa v rámci obmedzení, určených pravidlami hry futbal a špecifikami simulačného prostredia, snaží vytvoriť čo najlepšieho agenta. Mužstvo, vytvorené z takýchto agentov, by malo vyhrať nad mužstvom súpera.

Simulácia futbalu pôvodne prebiehala iba v dvoch rozmeroch. Pre zvýšenie reálnosti simulácie bolo vytvorené 3D simulačné prostredie, ktoré rozširuje možnosti hry. 3D simulačné prostredie sa pomerne výrazne líši od doposiaľ používaného 2D prostredia, a to jednak spôsobom simulácie, ale hlavne možnosťami, ktoré poskytuje agentom. Na rozdiel od 2D simulácie ide o simuláciu jednotlivých pohybov humanoidného robota.

Hlavným cieľom projektu bude vytvoriť agenta pre 3D simuláciu, ktorý dokáže plnohodnotne využívať možnosti poskytované simulačným prostredím. Úlohou teda bude prevziať jednoduchšieho agenta, vytvoreného na našej fakulte v minulom roku, a doplniť do neho komplexnejšie typy správania, predovšetkým zložitejšie pohyby agenta, ale pozornosť by sa mala venovať už aj rozhodovaniu na vyššej úrovni, taktike a stratégii. Dôležitým faktorom bude využitie moderných prístupov robotiky a umelej inteligencie.

Keďže sa predpokladá ďalšie rozširovanie agenta v ďalších rokoch, dôležitými požiadavkami sú prehľadnosť a ďalšia rozšírovateľnosť, a to na úrovni návrhu aj implementácie. Dôraz pri vytváraní agenta by mal byť kladený najmä na dobre prepracované a odladené schopnosti agenta, ktoré umožnia agentovi efektívne konať v prostredí.

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

1.2 Štruktúra dokumentu

Cieľom tohto dokumentu je vytvoriť dobrý podklad pre implementovanie agenta, ktorý bude úspešne hrať simulovaný futbal. Dokument je rozdelený na 3 základné časti.

Kapitola 2 obsahuje analýzu súčasného stavu. Analyzované sú domáce a zahraničné tímy a na základe získaných poznatkov je vytvorené zhodnotenie. Následne je analyzovaný server používaný na simulovanie futbalu. Vysvetlené a opísané sú efektory a perceptory, ktoré obsahuje agent, a taktiež je opísaný súčasný fyzický robot NAO, ktorého model bude používaný v simuláciách. Ako základ ďalšieho rozširovania a zdokonaľovania RoboCup 3D futbalu je prevzatý agent JIM. Na záver tejto kapitoly sú analyzované ešte niektoré fakty, na ktoré sme prišli počas analýzy.

Po dôkladnej analýze problémovej oblasti nasleduje špecifikácia požiadaviek v kapitole 3. Špecifikované sú základné pohyby, ktoré potrebujeme zdokonaľovať, taktiež sú špecifikované požiadavky na rozšírenie editora pohybov a uvažuje sa aj nad vytváraním editora správania, ktorý bude súčasťou editora pohybov a bude slúžiť na definovanie vyššej logiky agenta. Následne je špecifikované vyššie správanie agenta a na záver je opísaný testovací framework.

V nasledovnej časti je uvedený hrubý návrh prototypu. V kapitole 4 sa teda rozširujú špecifikované požiadavky a určujú sa možné cesty implementácie v prototype. Najprv sa opisujú pohyby a vytváranie vyšších činností pomocou základných pohybov za účelom dosiahnutia nejakého cieľa (dobehnutie k lopte, kop na bránu a pod.). Následne sú navrhnuté úpravy v editore pohybov. Navrhnutý je aj editor správania, ktorý bude pracovať buď na základe produkčného systému, alebo pomocou stavového automatu, zadefinovaného v jazyku XABSL. Na záver je navrhnutý testovací framework pre testovanie schopností agenta.

Kapitola 5 opisuje priebeh a výsledky implementácie prototypu. Do prototypu sme sa rozhodli implementovať niektoré pohyby. Následne sme upravili niektoré funkcionality editora pohybov. Veľkou časťou implementácie je aj testovací framework, ktorý je navrhnuitý a sčasti implementovaný ako tréner pre agenta.

Kapitole 6 opisuje pokračovanie implementácie produktu. V tejto časti sme sa ďalej zaoberali pohybmi agenta. Pokračovali sme v implementácii niektorých častí testovacieho frameworku a následne sme implementovali niektoré testy pre agenta. Závažným bodom implementácie je aj zmena v plánovacom module agenta.

2 Analýza

V tejto časti dokumentu sa venujeme analýze existujúcich 3D tímov, či už svetových alebo domácich, za účelom zoznámenia sa s rôznymi spôsobmi riešenia problémov ovládania simulovaného robota. Taktiež analyzujeme nástroje a prostriedky, ktoré sa pri vývoji robotov používajú.

V prvej časti analýzy poukazujeme na zaujímavosti v riešeniach iných tímov. Následne je urobená analýza servera, na ktorom bude simulácia prebiehať. Nasleduje analýza agenta JIM, na ktorého budeme nadväzovať. V poslednej časti sú identifikované ešte niektoré fakty, ktoré budeme musieť zohľadňovať počas práce na projekte.

2.1 Analýza 3D tímov

Analyzovali sme 12 tímov v RoboCup 3D, a to 6 zahraničných (Kouretes, Nao Team Humboldt, FC Portugal, UT Austin Villa, SEU-3D a Nexus 3D) a 6 domácich tímov (Agenty 007, Critical Error, RoboKit, Hviezdna jedenástka, RoboKopy a Neurotics).

2.1.1 Kouretes

Tím vznikol v roku 2006 na Krétskej Technickej Univerzite (TUC). Všetky informácie o tíme sú prevzaté z publikácií a dokumentácie tímu[1].

Základný prístup tímu je taký, že na agenta (robota) pozerajú ako na zoskupenie agentov [1]. Každý agent je zodpovedný za jednu alebo viac úloh robota. Agent Kouretes tímu pozostáva z nasledovných modulov:

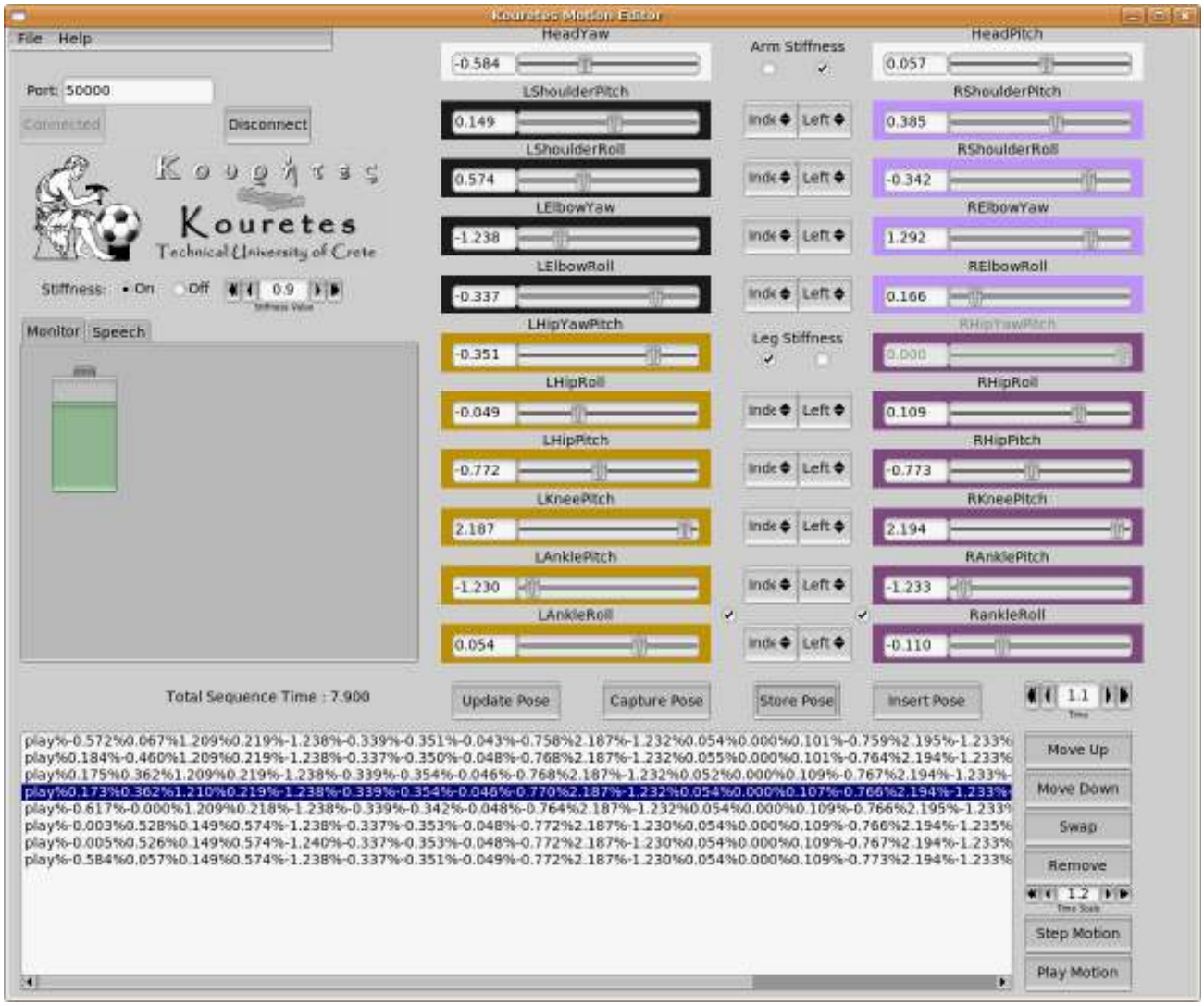
- *Image Processor* – zodpovedný za spracovanie obrazu z kamery, najprv vykonáva segmentáciu farieb a následne deteguje loptu a bránku, a tak aktualizuje vizuálne vnímanie robota.
- *Localization* – zodpovedný za odhad polohy robota v teréne.
- *Behavior* – zodpovedný za správanie a rozhodovanie robota vzhľadom na vnemy.
- *Special Action* – zodpovedný za načítanie a vykonanie pohybov zadefinovaných v pohybovom editore tímu.
- *Communication* – zodpovedný za komunikáciu medzi robotmi.
- *Robot Controller* – zodpovedný za komunikáciu s ovládačom hry a nastavenie stavu robota (štart, stop, cieľ, penalizácia a pod.).

[1] http://www.intelligence.tuc.gr/kouretes/web/

Jednou zo zvláštností práce tímu je použitie upevňujúceho učenia (RL – reinforcement learning) pre učenie dobrých pohybových funkcií pre rôzne úlohy, ktoré má robot spĺňať (chôdza, otáčanie sa na mieste, kop do lopty a pod.). Ich prístup je založený na NAC rámci (natural actor-critic). Tento prístup umožňuje odhadnúť (pomocou pokus-omyl testov) gradient hodnoty funkcie parametrizovaného pohybu ako funkciu pohybových parametrov. Sledovaním gradientu sa prispeje k (lokálnemu) optimálnemu pohybu (postupnosti pohybov).

Tím Kouretes sa v súčasnosti prevažne zaoberá reálnym 3D futbalom a nie simulovaným. Avšak zaujímavosťou tohto tímu je ich editor pohybu.

Zaujímavosťou Kouretes Motion Editora (editor pohybov znázornený na *Obr. 1*) je, že na určovanie pohybov používa možnosti symetrického pohybu kĺbov. Taktiež je výhodné, že obsahuje niektoré preddefinované pohyby navrhnuté tímom Kouretes.



Obr. 1: *Grafický editor pohybu tímu Kouretes*

Ďalšou výhodou editora je, že je robený pre fyzické roboty, a tak sa pohyby môžu nahrávať priamo z pohybov fyzického robota. Všetky pohyby sú zadefinované v XML súboroch. V XML súbore, ktorý editor používa na nastavenie, sa dajú zadefinovať aj iné typy robotov (nie len NAO). Toto umožňuje prispôsobenie editora eventuálnym zmenám v rozvoji robotov. V súbore je možné zadefinovať kĺby, uhly otáčania, počet kĺbov a pod.

2.1.2 Nao Team Humboldt

Dôvod výberu tímu Nao Team Humboldt spočíva v ich úspešnosti na svetovej úrovni. Tím Nao Team Humboldt, ďalej len NTH, nemá verejne k dispozícii podrobnú dokumentáciu, ale má Team Description Paper a množstvo špecifických publikácií. Keďže sa zaoberá fyzickou aj simulačnou humanoidnou verziou RoboCupu, mnoho publikácií sa venuje najmä opisu humanoidnej ligy. V publikáciách môžeme nájsť aplikáciu Constraint based techniques ako iný pohľad na používanie probabilistických metód, akými sú Kalmanove filtre a Monte Carlo metódy. NTH má nízkoúrovňový prístup už zvládnutý na vysokej úrovni, čo sa prejavuje aj na témach publikácií. Pre naše potreby sú dôležité informácie o dvoch softvérových aplikáciách, konkrétne XabslEditor a Simple Soccer Agent.

Extensible Agent Behavior Specification Language[2] (XABSL) je jednoduchý jazyk, pomocou ktorého je možné opísať stavovým automatom správanie sa autonómnych agentov. XabslEditor je open-source softvérová aplikácia, ktorá umožňuje vyjadrenie vysokoúrovňovej logiky spoločne s diagramovou grafickou vizualizáciou písaného kódu v XABSL. Je vytvorená v Jave, mala by byť použiteľná na všetkých platformách. Pri riešení vyššej logiky agenta by bolo vzhľadom na úspechy dosiahnuté pomocou špecifikácie prostredníctvom XABSL vhodné podrobnejšie analyzovať danú možnosť.

Simple Soccer Agent[3] je funkčná aplikácia so zdrojovým kódom v jazyku C++ v prostredí Visual Studio, v ktorej je použitá architektúra, ktorú používa tím NTH. Nevýhodou je, že zdrojové kódy sú len minimálne okomentované. Na stránke je uvedené, že program je v čo najväčšej miere implementovaný minimalisticky a môže slúžiť začiatočníkom ako základ pre vlastné experimenty.

Pri štúdiu informácií o tomto tíme je možné vidieť profesionalitu, ktorá sa za ich prístupom skrýva. Preto nás taktiež zaujala myšlienka experimentovania na ich agentovi. V prípade rozhodnutia tvoriť nového agenta by bolo vhodné podrobne analyzovať architektúru ich agenta najmä z dôvodu, že majú správne logicky oddelenú abstraktnú matematickú od skonkretizovanej fyzikálnej časti.

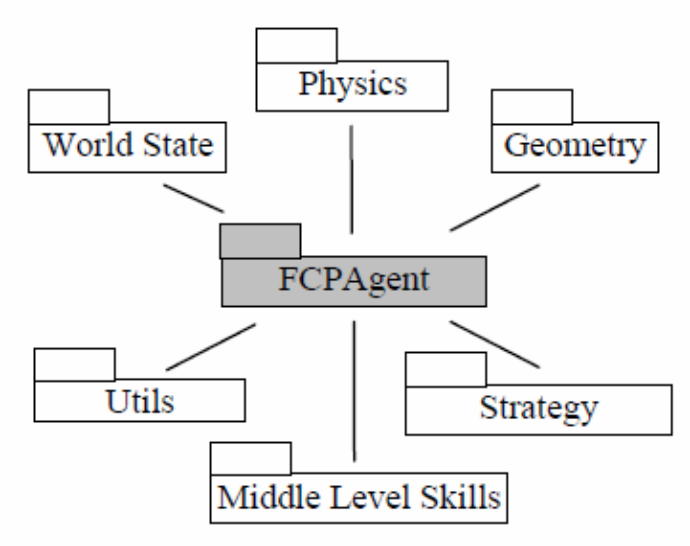
---

[2] http://www.xabsl.de

[3] http://launchpad.net/simplesocceragent/trunk/0.2/+download/SimpleSoccerAgent.pdf

2.1.3 FC Portugal

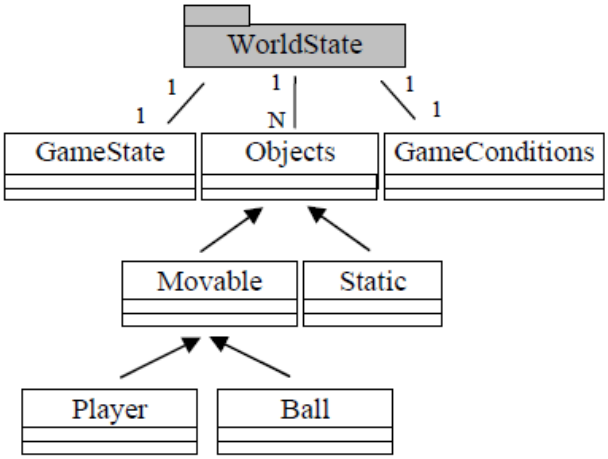
Tím FC Portugal 3D je víťazom súťaže RoboCup World 3D v roku 2006, pričom víťazného agenta aj zverejnil. Zverejnené dokumenty sa zaoberajú predovšetkým stratégiou 2D tímu, ktorý tiež tento tím vyvíjal. Dostupný je však dokument z roku 2004, kde je na dvoch stranách opísaný ich 3D agent. Všetky informácie o tíme sú prevzaté z dokumentácie a publikácií tímu[4]. *Obr. 2* zobrazuje architektúru agenta, ktorá je rozdelená na šesť hlavných modulov/balíčkov.



Obr. 2: *Architektúra agenta*

World State modul je zodpovedný za reprezentáciu aktuálneho stavu sveta. Informácie ako pozícia lopty, ostatných agentov (aj seba samého), pozície bránok, skóre a čas sú uložené v tomto module. *Obr. 3* zobrazuje hlavné triedy v tomto module, kde GameState obsahuje fakty o zápase (skóre, čas, ktorý tím začína hrať,...), GameConditions dáta o rozmere ihriska, teplote a Object informácie o statických (bránky), či dynamických (lopta, agent) objektoch v hre.

[4] http://www.ieeta.pt/robocup/index.htm



Obr. 3: *Schéma reprezentujúca stav sveta*

Strategy modul vychádza z myšlienky, ktorú tím aplikoval v 2D agentovi. Je zodpovedný za priraďovanie vhodných pozícií agentom v závislosti od aktuálnej pozície lopty a aktuálnej situácie. Low Level Skills (na obrázku Middle Level Skills) modul je zodpovedný za základné pohyby ako beh a kopanie do lopty. Physics modul simuluje na strane agenta výpočty vykonávané serverom. Vypočítava rýchlosť, zrýchlenie, brzdné dráhy, sily a pod. Geometry modul je používaný pre vykonávanie jednoduchších geometrických kalkulácií spojených so vzdialenosťami, vektormi a pod. Utils modul vykonáva úlohy ako posielanie/prijímanie správ, písanie logu a ďalšie.

Tím FC Portugal má bohaté skúsenosti s riešením robotického futbalu na 2D aj na 3D úrovni. Navrhnuté metódy z úspešného 2D tímu sa pokúsil preniesť aj do svojho 3D tímu, vďaka čomu je vytvorený agent na vysokej taktickej úrovni. K riešeniu základných pohybov agenta však tím nezverejnil žiadne publikácie. Z tohto dôvodu je vhodné sa inšpirovať iba návrhom architektúry, prípadne samotnou stratégiou agenta, ktorá je v predchádzajúcom texte zhrnutá.

2.1.4 UT Austin Villa

Medzinárodný tím robotického futbalu UT Austin Villa pochádza z USA. Venuje sa rôznym ligám robotického futbalu, v rámci simulovanej ligy hlavnej súťaži aj súťaži trénerov (coach). Všetky informácie o tomto tíme sú prevzaté z dokumentácie tímu[5]. Na svojich webových stránkach poskytujú na stiahnutie zdrojové kódy:

- Agenta typu Coach – ide o agenta, ktorý je schopný poskytovať iným agentom strategické rady (o hre prostredníctvom ručne zadaných pravidiel a o súperovi prostredníctvom pravidiel naučených zo súperovej hry), Linux binary

[5] http://www.cs.utexas.edu/~AustinVilla/

- Agenta typu Coachable player – agenti, ktorí dokážu zapracovať rady od coach agenta do svojej hry, postavení na agentoch tímu UvA Trilearn

Keď hovoríme o coach agentovi, ide o agenta, ktorý sa učí predvídať správanie ďalších agentov na základe pozorovania ich hry v minulosti a automaticky generuje rady, ktoré vedú k zlepšeniu výkonu tímu. Potrebné informácie čerpá z logov minulých hier protiagenta, vytvára model správania a na základe modelu generuje rady [2].

Najskôr identifikuje podstatné udalosti (nahrávky, strely). Potom udalosti a aktuálny stav prostredia v danom čase exportuje do databázy. Databáza sa používa na vytvorenie pravidiel v jazyku CLang. Na začiatku zápasu sa rady kombinujú s niekoľkými ručne zadanými pravidlami.

Keepaway je podúloha robotického futbalu, kde jeden tím sa usiluje udržať si loptu v rámci limitovaného priestoru a druhý tím sa ju snaží ukoristiť. Toto sa deje v rámci simulátora, parametrizovať možno veľkosť priestoru, počet obrancov a útočníkov.

Ide o náročnú úlohu z pohľadu učenia – veľký stavový priestor, každý agent má o ňom len určité informácie, viacerí tímoví spoluagenti sa musia učiť spoločne. Tím UC Austin Villa v tejto úlohe využíva reinforcement learning techniques (učenie s odmenou a trestom).

Half field offense je rozšírením keepaway smerom k učeniu sa tímovému správaniu. Modeluje typický scenár útoku vo futbale, kde ofenzíva jedného tímu sa musí dostať cez defenzívu iného tímu a pokúsiť sa streliť gól. Riešenie, ktoré sa agenti ofenzívy majú naučiť, je tzv. politika (policy), ktorá mapuje stavové premenné na akcie.

Opis tímu je krátka práca, ktorá ale obsahuje architektúru agenta, ktorou je možné sa inšpirovať. Architektúra ovládania obsahuje prvky:

- Stratégia
  - vysokoúrovňové správanie
- Zručnosti
  - chôdza
  - otočenie
  - kop
  - vstávanie
  - pád
- Nízkoúrovňové ovládanie
  - ovládanie kĺbov cez PID ovládače

Najväčším prínosom tímu je výskumná práca v rámci robustnej lokalizácie, robustného videnia, poskytnutia rady na základe modelovania protiagenta (simulovaná liga).

2.1.5 SEU-3D

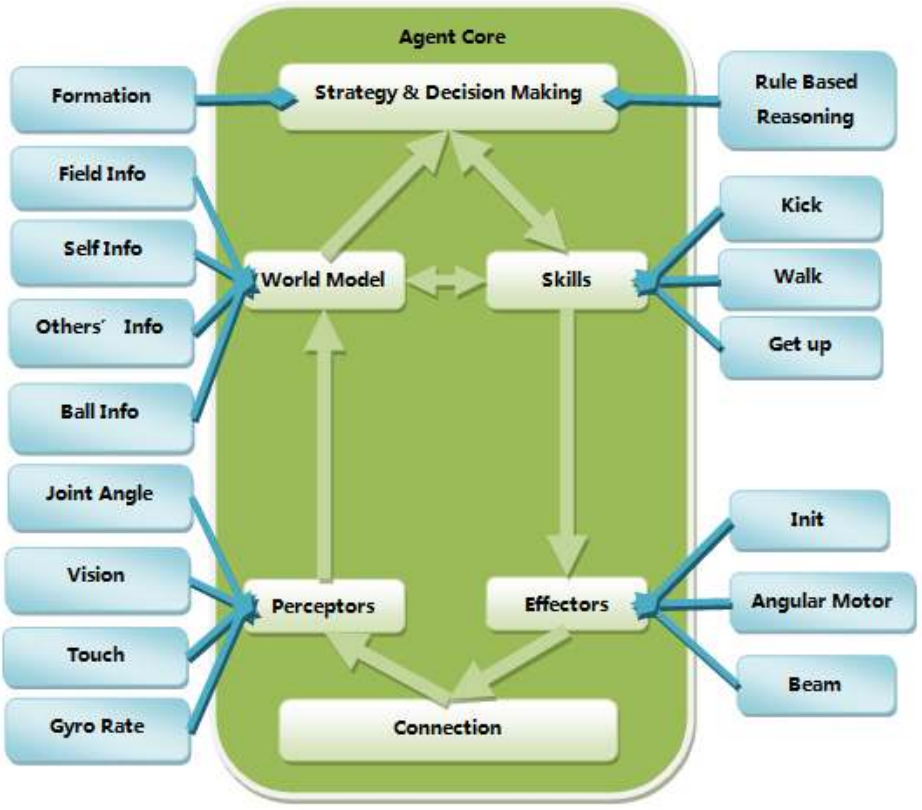
Tím SEU-3D vznikol v roku 2005 na čínskej Southeast University. 3D robotickému futbalu sa tím venuje od roku 2007. V tom istom roku sa umiestnil na treťom mieste vo svetovom pohári tímov robotického futbalu. Tím SEU-3D je zložený z viacerých menších tímov a všetky informácie ohľadom tohto tímu sú prevzaté z tímovej dokumentácie[6].

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

Pohyb agenta je plynulý a zaujímavosťou je, že pri pohybe zapája aj ruky (pri chodení aj pri behu). Pri chôdzi robí agent veľmi malé kroky, dokáže sa pohybovať všetkými štyrmi smermi. Chôdza aj beh sú pomerne rýchle. Vstávanie je realizované posadením sa, rozkročením nôh a súčasným predpažením rúk. Je rýchle, agent pri ňom nestráca stabilitu. Streľba je pomerne nepresná, príprava trvá dlho. Agenti si nakopávajú loptu dopredu, namiesto nahrávky. Radšej potiahnu loptu sami a pokúšajú sa prejsť cez súpera individuálne, čo nie je veľmi vhodné riešenie.

Na *Obr. 4* vidíme architektúru agenta tímu SEU. Zelený obdĺžnik tvorí jadro agenta. Moduly vo vnútri sú implementované podľa vzoru singleton, čo umožňuje ich jednoduchú interakciu. Modré moduly pripojené k jadru boli navrhnuté ako add-ony. Toto umožňuje jednoduché prispôsobenie agenta zmenám vo verziách servera. Každý modul sa ľahko môže vypojiť a následne zlepšiť, upraviť alebo prípadne celkom odstrániť. Agent pritom funguje aj bez odstránených modulov.

[6] http://me.seu.edu.cn/sfzx/rescue/index.html



Obr. 4: *Architektúra agenta tímu SEU-3D*

2.1.6 Nexus 3D

Nexus 3D je iránsky tím s niekoľkoročnou tradíciou v oblasti simulovaného 2D aj 3D futbalu. Počas svojej existencie získal viacero ocenení na miestnej aj medzinárodnej úrovni. Tento tím sa zaoberal zdokonaľovaním základných pohybových schopností agentov, ale aj streľbou, logikou a stratégiami. Pri riešení rozhodovania a streľby využívali fuzzy prístupy. Keďže naši agenti ešte nemajú dotiahnuté základné pohybové schopnosti, je pre nás zaujímavejšia práca tímu Nexus 3D práve v oblasti pohybových schopností robota. Vo svojom agentovi z roku 2007 použili princípy evolúcie na vytvorenie relatívne rýchlej a stabilnej humanoidnej chôdze. Všetky informácie o tíme sú čerpané zo stránky[7] a opisu tímu [3].

Chôdza v základe pozostáva z troch fáz – presun váhy na jednu nohu, pokrčenie nohy, ktorá nie je oporná a posun tejto nohy v smere chôdze. Ďalej je pri chôdzi nutné zabezpečiť udržanie rovnováhy. Obľúbeným prístupom plánovania trajektórie kĺbov riešiacim problém stability je využitie indikátora stability ZMP (Zero Moment Point).

[7] http://nexus.um.ac.ir

Je zrejmé, že chôdza robota po rovine je realizovaná ako periodicky sa opakujúca postupnosť čiastkových pohybov, ktorá je navyše rovnaká pre krok pravou aj ľavou nohou len s rozdielom časového posunu. Práve vďaka periodicite chôdze je možné pohyb každého kĺbu opísať pomocou skráteného trigonometrického Fourierovho radu. Na určenie koeficientov Fourierovho radu použil tím Nexus 3D genetický algoritmus. Na ovládanie kĺbových motorčekov implementovali jednoduchý PD regulátor, aj keď vďaka adaptívnemu charakteru evolučných metód je možné použiť aj iné regulátory. Uhly kĺbov môžeme formulovať použitím Fourierovho radu. Fourierov rad periodickej funkcie času $f(t)$ môžeme zapísať ako:

$$f(t) = a_0 + \sum_{n=1}^{\infty}\left(a_n \cos\frac{2n\pi}{T} + b_n \sin\frac{2n\pi}{T} t \right)$$

kde a_n a b_n sú konštantné koeficienty a t je perióda. Periódu môžeme vypočítať zo želanej základnej frekvencie ω ako $t = 2\pi/\omega$.

Členovia Nexus 3D vychádzali z predpokladu, že môžu zanedbať vyššie frekvencie bez zhoršenia výkonu, keďže tieto frekvencie nemôžu prispievať k vykonávanému pohybu, lebo použité servo motory slúžia ako nízko priepustné filtre. Preto použili skrátený Fourierov rad obsahujúci len prvé tri členy trigonometrickej formy Fourierovho radu:

$$f(t) = A\left[a_0 + \sum_{n=1}^{m}\left(a_n \cos\frac{2n\pi}{T} + b_n \sin\frac{2n\pi}{T} t \right)\right]$$

kde a_n, b_n a t sú rovnaké ako v predchádzajúcom vzorci, A je parameter amplitúdy určujúci dĺžku kroku a parameter m určuje počet členov Fourierovho radu. Použitie tohto vzorca výrazne zmenšuje výpočtovú náročnosť hľadania optimálneho riešenia pomocou genetického algoritmu. Taktiež rytmus, rýchlosť a typ chôdze je možné meniť za behu prestavením len jedného alebo dvoch parametrov.

Genetický algoritmus bol použitý na určenie optimálnych koeficientov skráteného Fourierovho radu. Fáza učenia prebiehala offline. Frekvencia vo Fourierovom rade je považovaná za konštantu a môže byť zmenená po dokončení fázy offline učenia. Takto môže byť Fourierov rad pre každý kĺb reprezentovaný siedmimi reálnymi číslami a pre model robota HOAP-2, ktorý použil tím Nexus 3D pri vývoji tejto metódy, bolo potrebných 25 takýchto radov (jeden pre každý kĺb). Keďže takto vytvorení jedinci boli veľmi veľkí, tím navrhol ďalšie vylepšenia metódy, aby bolo učenie efektívnejšie.

Priama chôdza je symetrická, preto bolo možné neuvažovať samostatné Fourierove rady pre pravú stranu chromozómu, ktorého dĺžka tým klesla skoro na polovicu. Ďalej nastavenie členku robota

bolo možné odvodiť od nastavenia bedra a kolena tak, aby vždy bolo rovnobežne s povrchom. Pre ďalšie zjednodušenie zanedbali kĺby hlavy a niektoré ďalšie nepotrebné kĺby. Výsledkom bol chromozóm šiestich reálnych čísel pre každý z 12 kľúčových kĺbov.

Vzhľadom na veľkosť chromozómu pracoval algoritmus s 300 jedincami v každej z 900 generácií. V ďalšom postupe sa jedinci vyjadrení vyššie opísaným chromozómom medzi sebou krížili a mutovali. Bolo použité jednoduché dvojbodové kríženie spolu so zavedením operátora, ktorý náhodne zväčšil alebo zmenšil niektoré reálne čísla chromozómu o malú hodnotu medzi 0 a 5. Pravdepodobnosť kríženia bola 0.6 a pravdepodobnosť mutácie 0.05. Rozsah koeficientov bol tiež obmedzený a to na hodnoty od -50 do 50. Pokusy s rôznymi hodnotami frekvencie ukazujú, že takmer všetky vygenerované chôdze počas fázy učenia konvergujú k človeku podobnej chôdzi s podobnými vzormi pohybu rúk a pásu.

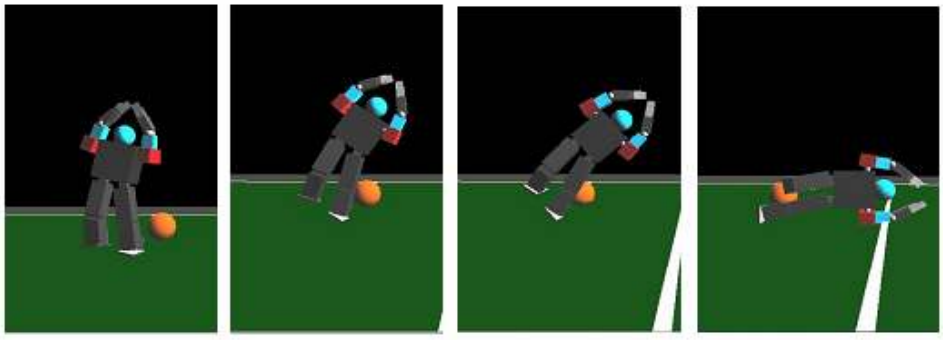
Pohyby niektorých kĺbov boli vo všetkých chôdzach podobné, čo viedlo k odstráneniu ďalších nadbytočných parametrov. Výsledky pokusov ukazujú, že stabilná priama chôdza môže byť vygenerovaná použitím len dvoch kĺbov – bedrového 3 a kolenného. Aj fitness funkcia bola výrazne zjednodušená, aby rýchlejšie chôdze boli ohodnotené lepšie ako stabilné, ale pomalé chôdze. Ďalej bola pri fitness funkcii do úvahy zobratá odchýlka od priameho smeru, aby rovné chôdze boli ohodnotené lepšie ako zakrivené. Zjednodušená fitness funkcia vyzerá nasledovne:

```
ak (aktuálny čas testovania < celkový čas testovania / 2),
potom fitness := čas * vzdialenosť
inak fitness := vzdialenosť – priemerná odchýlka
```

Úpravou frekvencie a zosilnenia pohybu kĺbu je možné dynamicky meniť rýchlosť chôdze. Miernou zmenou polohy kĺbu nohy 1, zodpovedného za rotáciu stehna okolo osi Z, je možné zmeniť smer chôdze o niekoľko stupňov.

Pred kopnutím do lopty robot najskôr ukončí chôdzu a zastaví. Následne presunie váhu na podpornú nohu, druhú nohu pokrčí, napriahne dozadu a zrýchli smerom dopredu. Kopajúca noha dosiahne svoju maximálnu rýchlosť, keď sa dostane pred robota.

Brankár chytá loptu prevážením sa na bok. Pri tom presunie ťažisko, natočí trup na želanú stranu, pokrčí nohy a akonáhle sa preváži, začne telo postupne znova narovnávať. Uvedený postup je zobrazený na *Obr. 5*.



Obr. 5: *Brankár v akcii*

2.1.7 Agenty 007

Tím Agenty007 je tím z akademického roku 2008/2009. Vyhral TPCUP v roku 2009. Všetky informácie o tíme vychádzajú z finálnej dokumentácie tímu [4].

Agent sa dokáže pohybovať všetkými smermi, vstávať, strieľať na bránu. Tím sa nezaoberal stratégiou hrania. Najviac sa zaoberali rovnováhou agenta a na vyriešenie tohto problému navrhli „rovnovážny modul“. Tento modul je navrhnutý na riešenie problémov pri pohybe robota, teda rieši nepresnosti v správach zo servera. Agent sa pri spracovaní správy často môže nachádzať v inej polohe, než v akej by sa v skutočnosti mal nachádzať. Tím sa preto sústredil na vypočítanie ťažiska robota a navrhol použitie nasledovnej funkcie:

[Xn, Yn, Zn] ziskajNovuPolohuBodu(Xos,Yos, Zos, Xp, Yp, Zp, alfa, beta, gama)

Vstupy:

Xos,Yos, Zos – súradnice bodu, cez ktorý prechádzajú osi otáčania
Xp, Yp, Zp – počiatočné súradnice bodu
alfa (os Z), beta(os Y), gama(os X) – uhly natočenia bodu okolo daných osí

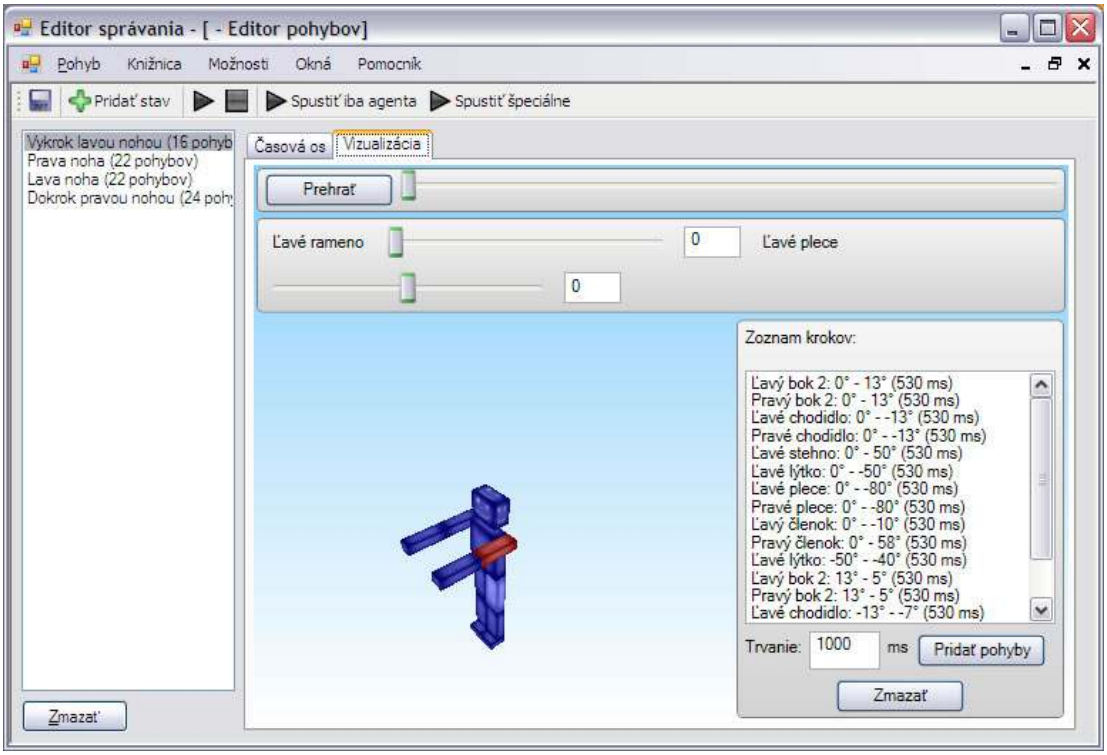
Výstup:

Xn, Yn, Zn – koncové súradnice bodu po aplikovaní natočení okolo jednotlivých osí

V rovnovážnom module tak sledujú pohyb ťažiska v čase. Samotná implementácia má zopár problémov. Neboli urobené väčšie zmeny na samotných agentoch vzhľadom na tímy z predchádzajúcich rokov. Tím sa najviac sústredil na vytvorenie pomocných nástrojov na spracovanie pohybov agenta, akými sú Editor pohybov a Parser správ zo servera.

Tím vytvoril jednoduchý editor pohybov. Editor umožňuje jednoduchým spôsobom vytvárať pohyby a následne poskytuje možnosť ich priamej simulácie (prehrávania). Editor je priamo prepojiteľný so serverom a taktiež aj s agentom.

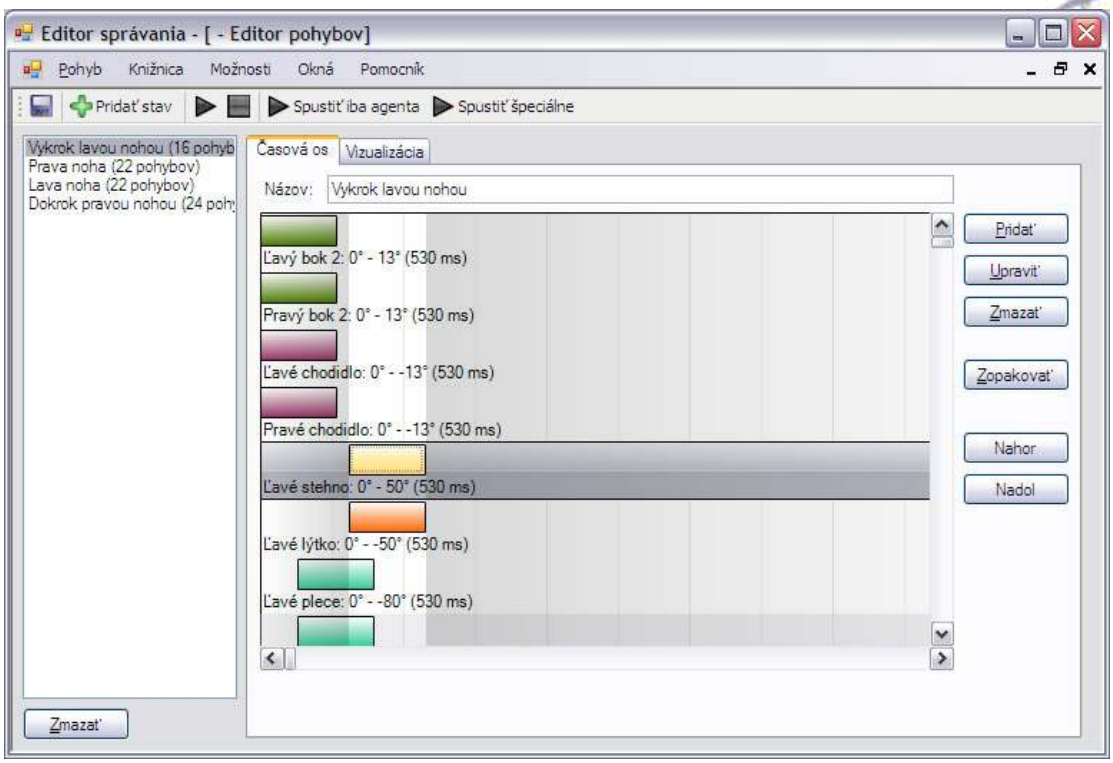
Pohyby v editore sú reprezentované otočením kĺbov, ako je možné vidieť na *Obr. 6*. Editor umožňuje vytvorenie akéhokoľvek pohybu (vstávanie, beh, chôdza, otočenie,...). Samotné pohyby je možné exportovať do RMO súborov. Tento typ súboru využíva zápis údajov, ktorý je ľahko parsovateľný. Taktiež je možné súbor priamo meniť v jednoduchom textovom editore.



Obr. 6: *Grafický editor pohybov*

Ďalšou zaujímavosťou editora je časová os (*Obr. 7*), pomocou ktorej je umožnené sledovanie následnosti otočenia kĺbov. Možné je určiť čas otočenia kĺbov, rýchlosť, následnosť a synchronizované pohybovanie kĺbov.

Parser správ, ktorý tím implementoval, umožňuje spracovanie a prehranie akéhokoľvek zaznamenaného pohybu agenta na serveri. Samotný parser pracuje tak, že po prijatí správy zo servera sa táto vyparsuje a rozpoznané pohyby sa vložia do pripravených dátových štruktúr. Následne sa dané pohyby môžu prehrať a analyzovať.



Obr. 7: *Časová os pohybov*

Hlavným prínosom tímu bolo vytvorenie editora na spracovanie pohybov. Je vhodné pracovať na tomto editore a pokúsiť sa ho zlepšiť a doplniť o ďalšiu funkcionalitu.

2.1.8 Critical Error

Critical Error je tím, ktorý vytvoril agenta menom Sirius. Tento tím bol aktívny v akademickom roku 2009/2010. Všetky informácie o tíme vychádzajú z jeho finálnej dokumentácie a SVN repozitára tímu[8].

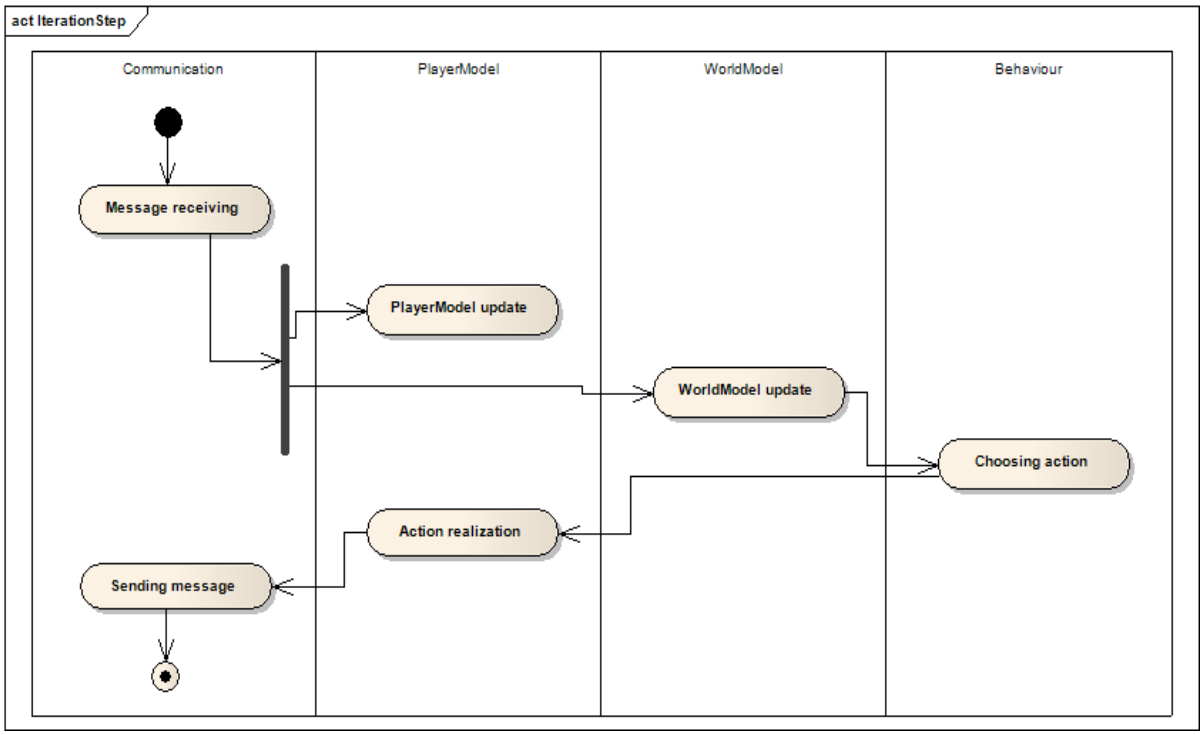
Tím nadviazal na prácu tímu Dream Team. Zo špecifikácie požiadaviek sa dozvedáme, že tím sa zaoberal:

- Pridaním funkcionality do editora pohybov na import a export pohybov vo formáte XML
- Parsovaním (dôvodom bola nekompatibilita pôvodného parsera s novou verziou servera)
- Logovaním agenta
- Pohybmi agenta
- Záznamami o zápase

[8] http://labss2.fiit.stuba.sk/TeamProject/2009/team17is-si/index.html

Tím vychádza zo zdrojových kódov Dream Teamu, z čoho vyplýva aj rovnaká architektúra. Architektúru je možné rozdeliť na štyri komponenty (základný cyklus komunikácie je znázornený na *Obr. 8*):

- Communication
- WorldModel
- Behaviour
- PlayerModel



Obr. 8: *Cyklus komunikácie*

Modul Communication slúži na komunikáciu so serverom. V module WorldModel je reprezentovaný model okolitého sveta. PlayerModel je komponent obsahujúci aktuálny model agenta, ktorý slúži na realizáciu akcií. Akcie dostáva od modulu Behaviour, ktorý je zodpovedný za jej výber v závislosti od súčasnej situácie v hre.

Zdrojový kód agenta Critical Error je vytvorený v C++ v prostredí Visual Studio. Jeho dobrou stránkou je fakt, že je refaktorovaný a štruktúrovaný. Veľkým nedostatkom je však len minimálna miera dokumentovania. Negatívnou vlastnosťou je aj absencia testovacieho frameworku, čo sa však pri C++ očakávalo.

Agent JIM vznikol prerobením agenta tímu Critical Error do Javy. Kód tohto agenta má však veľký nedostatok, a síce že je takmer úplne nezdokumentovaný. Avšak v prípade Agenta JIM je veľkou výhodou testovací framework, vďaka ktorému je možné bezpečne napredovať.

Prínosom tímu je aj zdokonalenie editora pohybov, vytvoreného tímom Agenty007, ktorý bol obohatený o import a export vo formáte XML. Dôvodom je iný formát zápisu pohybov týchto dvoch tímov. Editor pohybov je vytvorený v jazyku C# v prostredí Microsoft Visual Studio. Kód je refaktorovaný a dobre organizovaný, avšak znovu chýba testovací framework a k zmenám existuje len minimálna dokumentácia.

Tím taktiež upravil parser komunikácie so serverom (prispôsobil ho novej verzii servera). Logovanie bolo vytvorené pomocou samostatnej triedy. V prípade agenta Sirius bol použitý preprocesor AspectC++, vďaka čomu je logovanie vysoko konfigurovateľné. V prípade agenta JIM je pre logovanie tiež vytvorená samostatná trieda.

Prínosom tímu je aj vytvorenie súboru *soccer.dll*. Keďže sa tím zaoberal kopmi, bolo potrebné upraviť server, aby umožňoval testovať kopy. Tento súbor predstavuje úpravu servera, ktorá spolupracuje s verziou 0.6.2, ale nie sú problémy ani s použitím s verziou 0.6.3.

Analýzou záznamov o zápase sa tím pokúsil prebrať efektívnu chôdzu tímu SEU RedSun. Vzhľadom na časovanie logu o zápase a šum vnášaný do komunikácie sa tímu nepodarilo v dobrej miere rekonštruovať ich princíp pohybu. Tím uvádza, že aj po dostatočne intenzívnom vylaďovaní sa nepodarilo dosiahnuť to, čo očakávali.

Pri implementácií sa tím zaoberal na low-level úrovni zlepšovaním výpočtu pozícií. Z uvedených informácií možno potom predpovedať pozíciu iných objektov, uvažovať o rýchlosti lopty a pod. Dané výpočty podporujú posun riešenia na vyššej úrovni. Súčasne tím implementoval systém rozhodovania v reálnom čase. V praxi to znamená, že môžeme meniť správanie sa robota bez potreby rekompilácie projektu. Riešenie je realizované pomocou použitia dynamických knižníc. Na vytvorenie vlastnej knižnice správania je vytvorený zdrojový súbor obsahujúci kostru kódu, ktorá tvorí dokumentačný súbor.

Kvalita kódu oboch agentov tímu Critical Error je na dobrej úrovni, dalo by sa v práci na nich pokračovať. Taktiež vytvorené podporné prostriedky ako parsovanie a logovanie nám umožňujú v projekte rozvíjať vyššiu logiku agenta. Jedným z problémov by mohla byť absencia testov v jazyku C++, ktorá by nám mohla spôsobovať problémy pri implementácii novej funkcionality. V prípade, že by sme možnosť testovania chceli využiť, bolo by vhodné pokračovať v rozvoji agenta JIM.

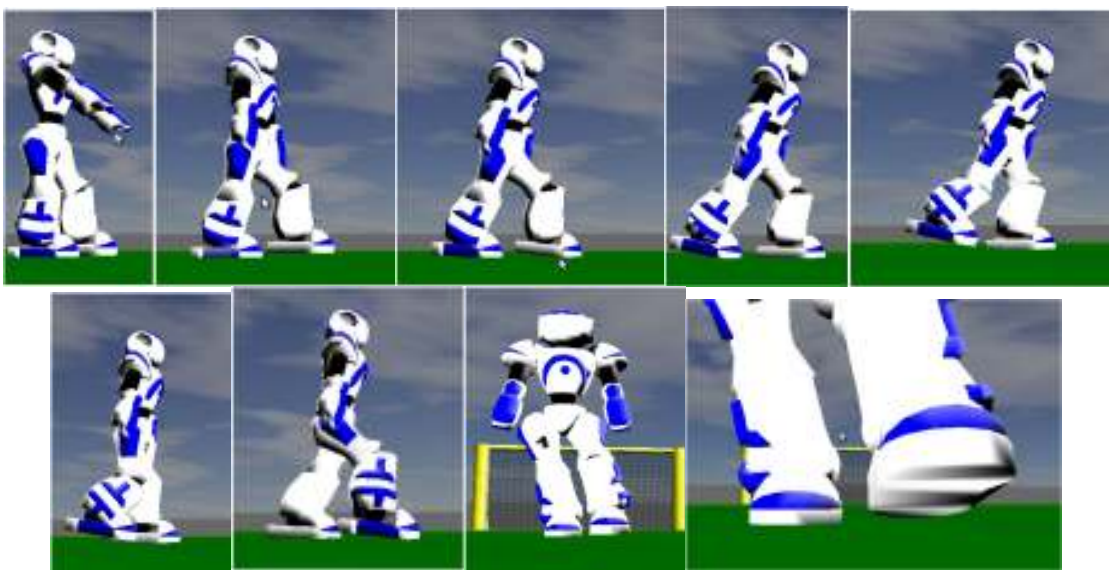
2.1.9 RoboKit

Ako základ si zvolili skorú verziu tímu RoboKopy, kde opravili základné chyby a migrovali na novú verziu servera 0.6.3. Pri tvorbe pohybov bol použitý editor pohybov tímu Agenty007. Vzorom pre jednotlivé pohyby bol človek a jeho pohyby, avšak nie všetky reálne pohyby mohli použiť z dôvodu inej fyziológie robota.

Chôdza je symetrická, takže sa sústredili len na rozbor jedného kroku. Tento pohyb sa skladá z dvoch častí – vykročenie a dokrok. Vykročenie prebieha nasledovne:

- Naváženie sa na pravú stranu (opačnú ako zdvíhaná noha).
- Po navážení zodvihnutie ľavej nohy, čo sa deje zároveň s pokrčením pravej nohy (aby sa horná časť tela nepresúvala). Toto človek uskutoční zdvihnutím stehna, pokrčením kolena, členka a aj kĺbu na špičke nohy.
- Simultánne nastáva navažovanie dopredu pravou nohou.
- Dôležitým bodom je bod, kedy koleno zdvíhanej nohy (ľavej) dosiahne maximálny ohyb a začne sa znova narovnávať. Tu je robot naklonený zhruba tak v polovici dĺžky kroku.
- Členok má zodpovedajúci uhol, aby robot nezakopol o zem.
- Pokračuje sa dosadnutím ľavej nohy na zem. To sa deje vystieraním pravej nohy.

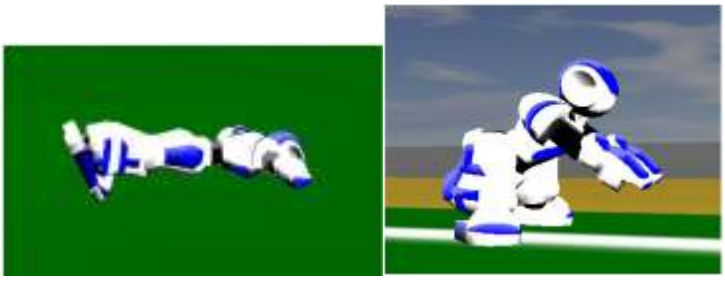
V časti dokrok prisunieme pravú nohu dopredu k ľavej nohe. Keďže sme naváženi na ľavej nohe môžeme jednoducho otočiť stehenným kĺbom dopredu. Musíme pokrčiť koleno a členok tak, aby sme nezakopli chodidlom. Zároveň vystierame ľavú nohu, aby sme nemuseli neprimerane skrčiť pravú nohu. Celú túto sekvenciu opakujeme dokola symetricky pre druhú nohu s tým, že sa zmenil fakt, že sme nemuseli zdvihnúť pravú nohu, ktorá je stále vo vzduchu z predošlého pohybu (*Obr. 9*).



Obr. 9: *Vykročenie a dokrok robota*

Zdvíhanie robota zo zeme je znázornené na *Obr. 10*. Robot leží na zemi a rozpaží ruky tak, aby sa mohol rukami zdvihnúť zo zeme. Následne sklopí ruky v horizontálnom smere. Zároveň s týmto pohybom sklopí stehenné kĺby a roztiahne nohy, pokrčí kolená a členky, takže nakoniec skončí sčupený na zemi. Je dôležité, aby sa nepreklopil na chrbát. Toto sa nestane, pretože má ruky pred sebou mierne smerom nad hlavu. Avšak ako je robot rozkývaný, môže sa stať, že prepadne dopredu,

preto ihneď ruky opäť rozpaží. V tejto pozícii chvíľu zotrvá, aby nepodľahol oscilácii a nespadol. Následne pritiahne nohy k sebe a vystrie kolená, čím sa postaví do vzpriamenej polohy.



Obr. 10: *Vstávanie*

Pokiaľ ide o vstávanie z chrbta, robot sa najprv otočí na brucho. Robot sa najprv potrebuje dostať do sedu. Keďže robot má väčšiu časť hmoty v hornej časti tela, mohol by to byť problém. Preto sa postupuje tak, že sa najprv pokrčia ruky, čím sa robot odtláča od zeme. Tento pohyb sa vykoná tak rýchlo, že sa mierne odrazí od zeme. Následne sa sklopia stehenné kĺby, čím sa robot dostane do sedu. Ruky má v tejto polohe pred sebou. Následne roztiahne nohy a preklopí sa dopredu.

Kop do lopty pozostáva z viacerých fáz – prenesenie váhy na ľavú nohu, dostatie pravej nohy do vzduchu, napriahnutie pravej nohy a strela pravou nohou, nespadnutie (*Obr. 11*).

Obr. 11: *Kop do lopty*

Tím RoboKit demonštroval vhodnú implementáciu pohybov aj na minuloročnom školskom turnaji, ktorý vyhral. Pri návrhu pohybov bol zvolený kompromis medzi rýchlosťou a stabilitou, pričom sa inšpirovali pohybmi človeka, vďaka čomu vytvorené pohyby vyzerajú dostatočne reálne a esteticky.

2.1.10 Hviezdna jedenástka

Tím Hviezdna jedenástka pôsobil v predmete Tímový projekt v akademickom roku 2007/2008. Všetky informácie o tíme sú prevzaté zo stránky[9] a finálnej dokumentácie tímu [5].

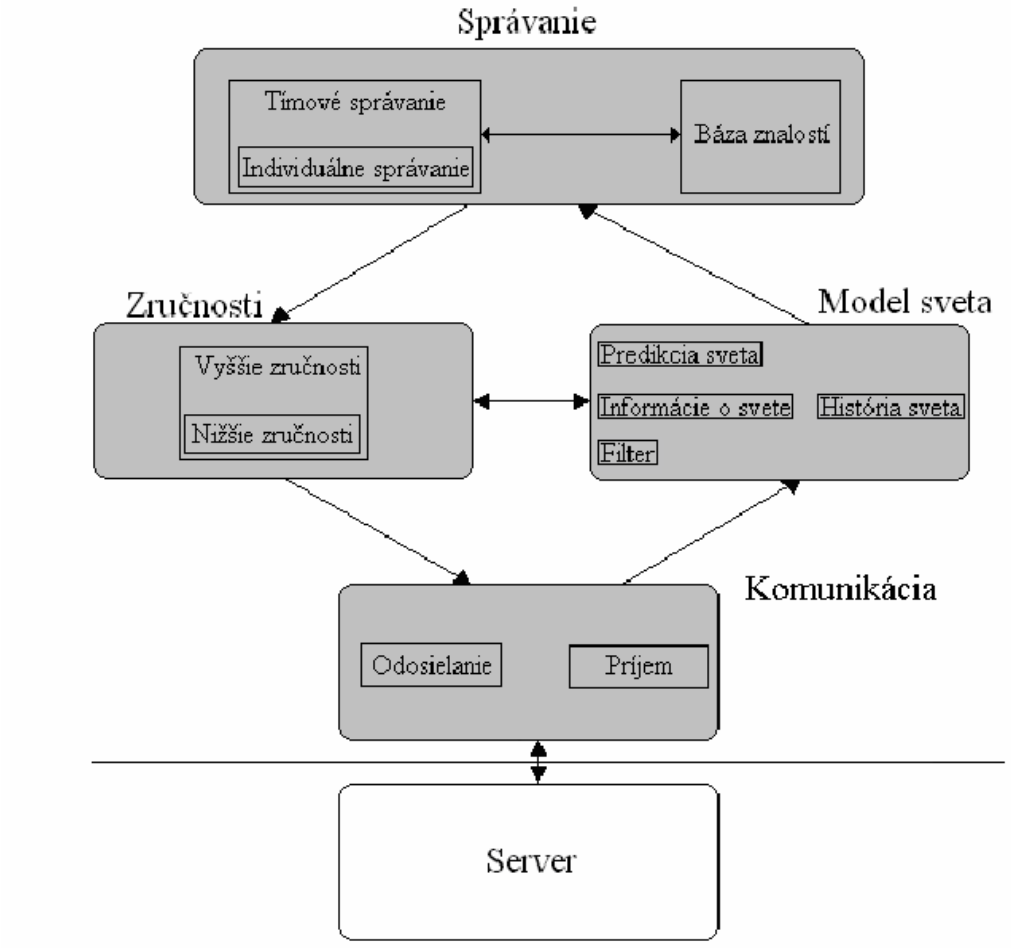
[9] http://labss2.fiit.stuba.sk/TeamProject/2007/team11is-si/

V analýze sa tento tím venoval najmä výberu servera, pre ktorý budú vytvárať svojho agenta (2D, 3D sphere, 3D humanoid). Množinu stanovených požiadaviek tvorili komunikácia agenta so serverom (spoľahlivý TCP verzus rýchly UDP protokol), práca s údajmi (vhodne uložené, odstraňovanie nepresností), správanie (tímové správanie, nekonfliktnosť výpočtov správania a pohybu – vlákna – rýchlosť verzus správnosť), architektúra (s ohľadom na ďalšie rozširovanie, moduly). Architektúra agenta, znázornená na *Obr. 12*, je rozdelená nasledovne:

- Správanie:
  - báza znalostí
  - tímové správanie
  - individuálne správanie
- Zručnosti:
  - vyššie – napríklad krok, beh, kop
  - nižšie – napríklad pohyby jednotlivých kĺbov
- Model sveta:
  - predikcia sveta – predvídanie správania
  - informácie o svete – aktuálny stav sveta
  - história sveta – predchádzajúci stav modelu sveta
  - filter – odstraňovanie šumu z prijímaných údajov
- Komunikácia:
  - odosielanie
  - príjem

Pôvodným plánom tímu bolo pokračovať v agentovi Zigorat, no nakoniec sa rozhodli začať budovať agenta od základov z dôvodu priveľkej previazanosti kódu pôvodne uvažovaného agenta. Cieľom prototypovania tohto tímu teda bolo najmä vytvorenie základnej štruktúry agenta a overenie ovládania pohybov kĺbov. Agenta sa rozhodli implementovať v programovacom jazyku C++ s dôrazom na prenositeľnosť medzi platformami Windows, Linux a Mac OS.

Keďže komunikácia prebieha pomocou TCP aj UDP protokolov, pre agenta boli implementované TCP a UDP sokety – využité boli POSIX sokety. Komunikácia agenta bola oddelená do samostatného vlákna, aby agent mohol komunikovať so serverom nezávisle od práve vykonávaných iných činností. Na zachytenie správ prichádzajúcich zo servera tím vytvoril parser správ. Parser nespracúva informácie o ďalších agentoch na ihrisku (v riešení tohto tímu to nebolo potrebné). Spracúva informácie zo všetkých perceptorov, ktoré boli potrebné pri vývoji agenta (neuvádzajú zoznam). Pri tvorbe využili APG (ABNF parser generator) nástroj, ktorý zo vstupnej gramatiky generuje kód parsera. Ten je ešte potrebné doplniť o funkcionalitu vpisovanú do vopred pripravených callback funkcií (definovať, čo sa má s prijatými vyparsovanými dátami urobiť).



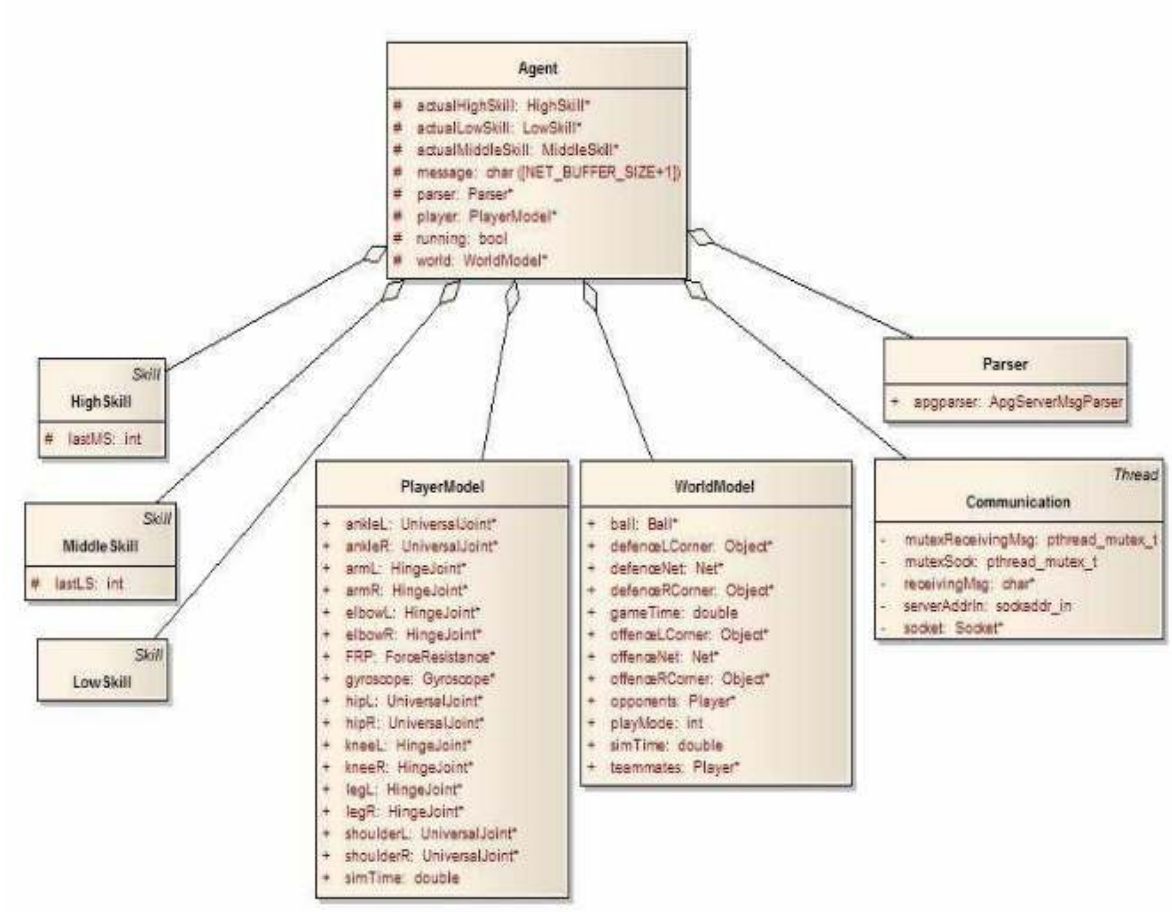
Obr. 12: *Navrhnutá architektúra agenta tímu Hviezdna jedenástka*

Na komunikáciu boli implemetované triedy TCPSocket a UDPSocket, aby agent mohol pomocou daných protokolov so serverom komunikovať (v samostatnom vlákne). Bola vytvorená aj spoločná trieda Socket, od ktorej obe spomínané triedy dedia. Komunikácia prebieha pomocou správ v určenom formáte – prvé štyri bajty predstavovali dĺžku správy a network order a zvyšné boli určené na obsah (s-výrazy – môže ich tvoriť reťazec alebo zoznam ďalších s-výrazov).

Model agenta pozostáva z kĺbov, gyroskopu umiestneného v trupe a silového perceptora umiestneného v chodidlách. Východiskom pre jeho vytvorenie bol štandardný model *soccerbot056* (14 kĺbov). Všetky informácie o modeli agenta obsahuje trieda PlayerModel.

Model sveta predstavuje trieda WorldModel. Ide o všetky objekty, ktoré môže agent vidieť a všetky potrebné informácie pre to, aby sa vedel rozhodovať podľa aktuálnej situácie. Obsahuje informácie o lopte, bránkach, rohoch ihriska, spoluagentoch a protiagentoch, atribúty času, režimu hry a metódu update (získava aktuálne informácie o objektoch v modeli sveta, vstupom sú údaje z parsera, volá sa v každej simulačnej slučke).

Výsledná architektúra agenta je znázornená na *Obr. 13*.



Obr. 13: *Výsledná architektúra agenta tímu Hviezdna jedenástka*

2.1.11 RoboKopy

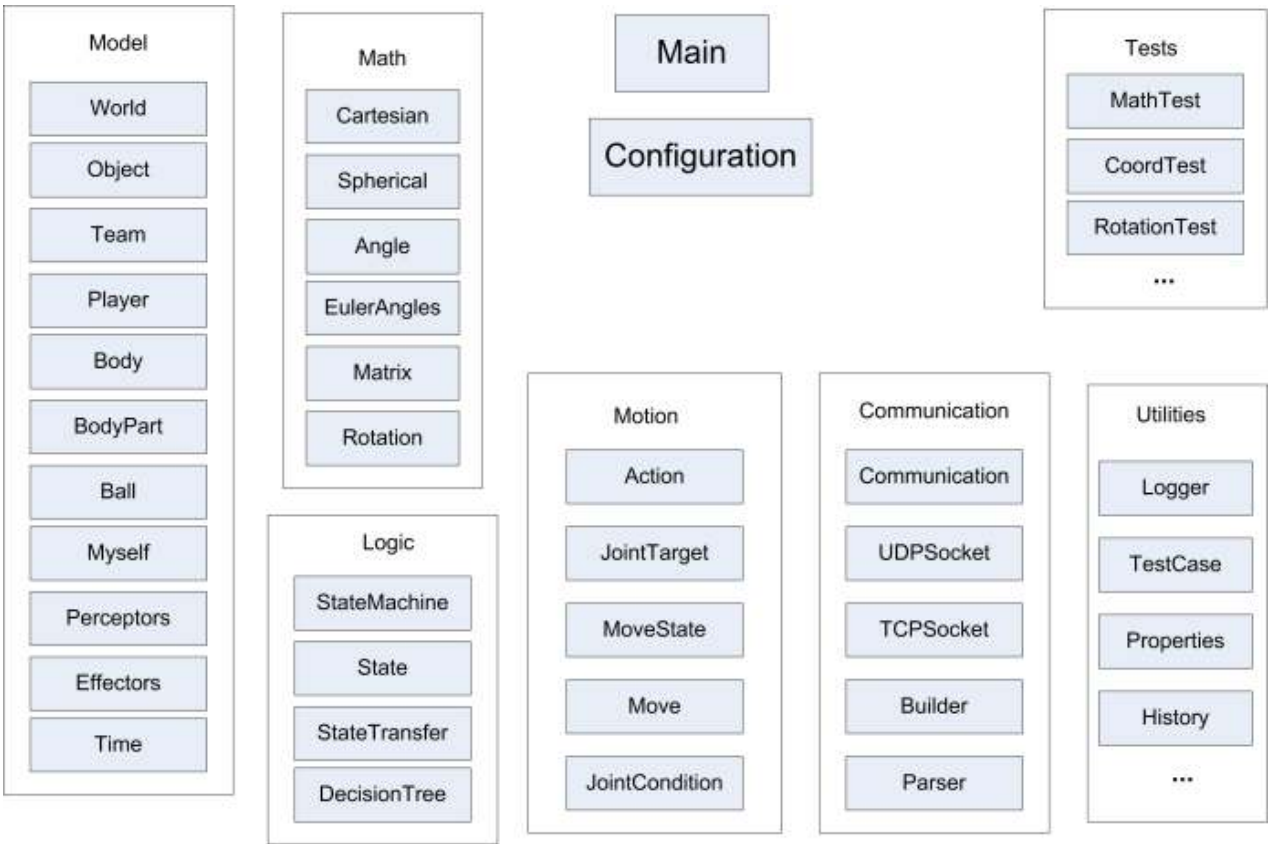
Tím RoboKopy pôsobil na fakulte v akademickom roku 2009/2010. Najväčším prínosom tímu bolo rozšírenie editora pohybov tímu Agenty 007 o strednú logiku. Všetky informácie sú prevzaté zo stránky[10] a finálnej dokumentácie tímu [6].

Samotný projekt bol rozdelený do niekoľkých modulov (*Obr. 14*):

- Model – uchováva informácie o stave sveta a objektov – pozície, rýchlosti, zrýchlenia
- Math – sú tu umiestnené triedy s matematickými štruktúrami a operáciami
- Logic – obsahuje stavový automat a prechodové funkcie reprezentované rozhodovacím stromom

[10] http://labss2.fiit.stuba.sk/TeamProject/2009/team15is-si/

- Motion – obsahuje triedy slúžiace na vykonávanie pohybov
- Communication – sprostredkuje komunikáciu so serverom protokolom TCP alebo UDP, stará sa o parsovanie a skladanie správ pomocou s-výrazov
- Utilities – obsahuje pomocné triedy na vedľajšie činnosti ako parsovanie konfiguračných súborov, logovanie a pod.
- Tests – obsahuje testy niektorých tried a algoritmov vo forme TestCase-ov



Obr. 14: *Diagram tried*

Z navrhnutých častí tím implementoval:

- Matematický model – výpočet orientácie a pozície kamery (na základe videných objektov a GyroRatePerceptor), prevod bodov z kamerovej do globálnej sústavy, výpočet rýchlostí a zrýchlení objektov, súradníc ťažiska a častí tela a jednoduchú predikciu pohybu na niekoľko simulačných krokov
- Históriu kinematických údajov pre každý dynamický objekt
- Správanie pomocou stavového automatu
- Predpodmienky pre pohyby – rozšírenie formátu RMO (editor tímu Agenty 007), načítanie a procedurálne vykonávanie predpodmienok
- Pomocné funkcie – logovanie, testovanie a pod.

Ako už bolo povedané najväčším prínosom tímu je editor strednej logiky. Rozšírenie editora spočívalo v troch bodoch:

1. Modul elementárnych pohybov
2. Modul strednej logiky
3. Modul modelovania strednej logiky

V editore pribudla možnosť modelovať strednú logiku agenta, čo je umožnené samostatným robustným komplexným modelom editora strednej logiky. Modul modelovania strednej logiky umožňuje jednoduchým a efektívnym spôsobom vytvárať logiku, ako aj editovať existujúcu logiku. V rámci jedného súboru je možné meniť ľubovoľné parametre strednej logiky.

Ďalšou časťou, ktorá pribudla do editora, bol komplexný modul elementárnych pohybov. V tomto module sa nachádzajú pohyby, ktoré je možné spájať za sebou, a tak vytvárať komplexné pohyby, pozostávajúce z ľubovoľného počtu jednoduchých pohybov.

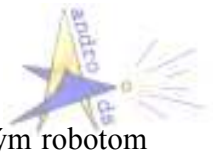
Posledným rozšírením editora bolo vytvorenie modulu strednej logiky, ktorý umožňuje jednoduchým spôsobom spúšťať strednú logiku namodelovanú pomocou modulu na modelovanie strednej logiky.

2.1.12 Neurotics

Neurotics je tím, ktorý sa v školskom roku 2007/2008 pod vedením Ing. Mariána Lekavého, PhD. venoval práci na agentovi pre 3D simulovaný robotický futbal RoboCup 3D. Vo svojej práci nadviazali na dvoch agentov – agenta Hazard (od tímu 6th sense, ktorý sa na FIIT téme venoval rok pred nimi) a agenta Zigorat. Prvý agent mal podľa tímu Neurotics výhodnú architektúru, lebo bola ľahko rozšíriteľná a flexibilná vďaka organizácii do vrstiev z hľadiska úrovne správania a ďalej do modulov. Tento agent bol postavený na staršom type servera, ktorý nepoužíval humanoidné roboty, preto Neurotics použili aj agenta Zigorat, ktorý mal podľa nich výborný model komunikácie so serverom. Všetky informácie ohľadom tímu sú prevzaté z tímovej dokumentácie na stránke tímu[11].

Tím Neurotics pracoval so serverom vo verzii 0.5.6 implementovaným systémom SPARK bez použitia SPADES. Tento server umožňoval simuláciu rôznych typov robotov. Neurotics použili typ robota s názvom *soccerbot056*. Na komunikáciu so serverom sú použiteľné protokoly TCP a UDP, pričom klient (agent) dostáva správy s informáciami z perceptorov robota v diskrétnych časových intervaloch 20 ms. Zrakový perceptor použitého robota umožňoval 360° videnie, objekty boli reprezentované hmotnými bodmi a poloha objektu bola vyjadrená relatívne od robota vo sférických súradniciach. Stred sférickej sústavy bol v ťažisku v hornej časti trupu robota.

[11] http://labss2.fiit.stuba.sk/TeamProject/2007/team17is-si/



Keďže fakultní predchodcovia tímu Neurotics sa venovali 3D simulácii so zjednodušeným robotom v tvare gule, ktorý bol výrazne odlišný od humanoidného robota, tím Neurotics si za svoj hlavný cieľ zvolil naučiť humanoidného robota vstávať do polohy, z ktorej sa dá najefektívnejšie pokračovať v dosahovaní cieľa, ktorý mal robot pred spadnutím. Túto úlohu sa rozhodli riešiť použitím evolučného algoritmu. Samotný agent Neurotik má štyri základné časti:

1. Hlavný vykonávací cyklus
2. Komunikačný modul
3. Pohľad na svet
4. Modely správania

Komunikačný modul tím Neurotics prebral z agenta Zigorat, pričom ho náležite upravil pre svoje potreby. Pohľad na svet bol tiež prebraný zo Zigorata, avšak len s minimálnymi zmenami. Štruktúra modelu správania pochádza z agenta Hazard.

Keďže dĺžka chromozómov uchovávajúcich informácie o postupnosti základných pohybov (pohybov kĺbov) nie je pevne daná a vopred určená, jedinci sú reprezentovaní pomocou tzv. „Messy"-chromozómov. Chromozóm je zložený z usporiadaných dvojíc – index a hodnota génu, pričom index určuje čas a hodnota génu vstupy pre metódy zabezpečujúce pohyb kĺbov. Aby pohyby boli relatívne vzhľadom na informácie o okolí, ktoré sú premenlivé a nie vopred známe, príslušné závislosti sú tiež predmetom evolúcie. Gén obsahuje čas, identifikátor a parametre správania vo forme koreňových stromov, ktoré sú reprezentované Readovým lineárnym kódom a vektorom ohodnotení vrcholov.

Fitness funkcia na ohodnotenie jedincov má u tímu Neurotics päť zložiek, z ktorých každá sa podieľa na výslednej hodnote normovanou hodnotou z intervalu <0;1> prenásobenou váhou zložky.

Pre výber jedincov na reprodukciu si tím zvolil stratégiu ruletového výberu, ktorá imituje náhodnosť udalostí v živote jedincov. Dvaja jedinci sa krížia na úrovni chromozómov aj na úrovni génov. Na úrovni chromozómov dochádza k jednoduchému jednobodovému kríženiu. Na úrovni génov sa krížia stromy s rovnakým indexom a to tak, že si vymenia podstromy. Mutácia na úrovni chromozómu spočíva v pridaní alebo odobratí náhodného génu. Mutácia na úrovni génov je zmena času génu a vymenenie náhodného podstromu za iný náhodne vygenerovaný podstrom. Do novej generácie vstupujú všetci potomkovia rodičov z predchádzajúcej generácie a najlepší jedinci z predchádzajúcej generácie.

Tím Neurotics vytvoril vlastnú klient-server aplikáciu, ktorá umožnila distribuovanú evolúciu agenta. Pomocou tejto aplikácie si agenti posielajú počas evolúcie migrujúcich jedincov v kruhu a zároveň majú možnosť zálohovať a obnovovať vlastné generácie zo servera.

Komunikácia je implementovaná pomocou soketov a je odolná voči náhodnému odpojeniu klienta. Server si uchováva logickú mriežku klientov a ich prepojení. Zároveň si uchováva aj informácie o niekedy pripojených a aktuálne pripojených klientoch. Ak sa nejaký klient odpojí, zostane

v mriežke, ale prepojenia medzi pripojenými klientmi sa upravia tak, aby ho obišli a nečakali na jeho príspevok do komunikácie.

Tím Neurotics preskúmal možnosti využitia evolučného algoritmu pre učenie agenta vykonávať základné pohyby a akcie (postavenie sa). Evolučné algoritmy v kombinácii s rôznymi inými prístupmi boli použité u viacerých zahraničných tímov, preto sa javia byť perspektívne s ohľadom na ďalší vývoj v oblasti RoboCup 3D.

2.1.13 Zhodnotenie tímov

Na záver uvedieme najzaujímavejšie časti analyzovaných tímov a identifikujeme možné cesty špecifikácie a návrhu, a v konečnom dôsledku aj implementácie. Identifikovali sme niekoľko zaujímavých častí, ktoré by sme mohli použiť pri vývoji nášho agenta.

Kouretes. Najväčším prínosom tohto tímu je editor pohybov, nad ktorým by sme sa mohli pozastaviť a implementovať v terajšom editore pohybov tímu Agenty 007 (upravenom tímom Critical Error) nejakú dodatočnú funkcionalitu ako napríklad symetrické pohyby kĺbov a pod.

Nao Team Humboldt. Od tohto tímu by sme mohli prebrať a bližšie sa pozrieť na jazyk XABSL, používaný na vytváranie vyššej logiky agenta.

FC Portugal. Pri tomto tíme nás najmä zaujala architektúra ich agenta. Bolo by vhodné nejakým spôsobom porovnať a spojiť architektúru tohto tímu s už vytvorenou architektúrou agenta, na ktorého budeme nadväzovať.

UT Austin Villa. Tento tím nás zaujal najmä agentom Coach, teda trénerom, ktorý sa zaoberá trénovaním ostatných agentov. V ďalšej práci na projekte by sme sa mohli zamyslieť nad vypracovaním jedného takéhoto agenta, ktorý bude v podstate „rozkazovať“ ostatným agentom a usmerňovať ich hru.

SEU-3D. Najväčším prínosom tohto tímu je plug-inové prepojenie modulov, ktoré zabezpečuje, že agent môže byť nezávisle zdokonaľovaný viacerými tímami naraz. Bolo by vhodné napríklad niektoré nové funkcionality v editore pohybov implementovať ako plug-in, vďaka čomu by sa daný modul mohol voľne pripájať či odpájať od hlavnej aplikácie.

Nexus 3D. Najzaujímavejším aspektom analýzy tohto tímu je ich spôsob riešenia základných pohybov pomocou genetických algoritmov. Pri vyhotovení dobrého testovacieho framewroku by bolo vhodné použiť genetický algoritmus tohto tímu.

Agenty 007. Od tohto tímu preberieme editor pohybov a prípadne ho budeme zdokonaľovať.

Critical Error. Po analýze tohto tímu sme zistili, že bude najpravdepodobnejšie tímom, na ktorý budeme nadväzovať. Tento tím v podstate vytvoril dvoch agentov. Agenta Sirius, vytvoreného

v C++, a agenta JIM, vytvoreného v Jave. Ďalším prínosom tímu je aj možnosť zadefinovania pohybov v XML súbore a následná implementácia podpory v editore pohybov, ktorý aj my budeme používať.

RoboKit. Tento tím sa najmä zaoberal spracovaním kvalitných pohybov, a preto by bolo vhodné sčasti prebrať ich sekvencie.

Hviezdna jedenástka. Tento tím pôsobil v rámci RoboCup 3D príliš dávno a založený je na staršom modeli robota, takže jediným zaujímavým aspektom je možné použitie niektorých komunikačných modulov agenta.

RoboKopy. Tento tím vytvoril editor strednej logiky, ktorý je založený na stavovom automate. Pokiaľ sa rozhodneme definovať vyššiu logiku agenta týmto spôsobom, bolo by vhodné použiť a prípadne rozšíriť tento editor.

Neurotics. Tento tím sa prevažne zaoberal implementovaním evolučných algoritmov na zdokonaľovanie a vytváranie pohybov. Uvedený prístup by sme sa mohli pokúsiť prevziať a vylepšiť, ako už bolo spomenuté.

2.2 Analýza servera

Server SimSpark je simulačné prostredie, v ktorom prebieha simulácia robotického 3D futbalu. Všetky informácie ohľadom analýzy servera sú prebraté zo zdroja [7].

2.2.1 Architektúra servera

Server je postavený na rámci zeitgeist. Časti servera sú vytvorené najmä v jazykoch C++ a Ruby. Server podporuje nahrávanie plug-inov, avšak každý plug-in musí byť vytvorený v jazyku Ruby. Použité sú dve knižnice na vizualizáciu, a to OpenGL a SDL. Najdôležitejšou časťou servera je vrstva Oxygen, ktorá obsahuje scénu a sú v nej zapuzdrené transformácie, geometria prostredia a objektov. Táto vrstva je taktiež zodpovedná za monitoring kolízií objektov a prepojenie so samotnými agentmi (robotmi). Cez túto vrstvu sa potom dajú pridávať plug-iny do servera a to tak, že vrstva Oxygen obsahuje cyklickú prechodovú funkciu, ktorá v každom svojom behu prejde a vykoná vybrané funkcie z plug-inov.

2.2.2 Simulácia na serveri

Server SimSpark pracuje sekvenčne. V každom simulačnom cykle server zozbiera informácie z každého senzoru každého agenta a vyhodnotí všetky vykonané akcie efektorov jednotlivých agentov (robotov). Simulácia sledujeme na monitore. Tento monitor dostáva od servera informácie o zmenách a znázorňuje ich, teda renderuje ich. Monitor servera môže aj čítať záznamy, a tak sa dajú prehrávať už odohrané zápasy (alebo simulácie). Tieto záznamy sú zapísané v súboroch, ktoré nazývame logy (logy zápasov). Aby sa prehral nejaký log, je potrebné spustiť monitor s prepínačom,

ktorého argumentom je súbor na prehratie (--logfile <názov súboru>). Najaktuálnejšia verzia servera je 0.6.4.

Po spustení SimSpark servera program načíta potrebné zásuvné moduly pre vyhodnocovanie efektorov, perceptorov a realizáciu pohybov. Monitor k serveru sa spúšťa automaticky, avšak niekedy je vhodné ho pre lepší beh simulácie spustiť na inom počítači. Následne je potrebné k serveru pripojiť jednotlivých agentov, a to pomocou programu agentspark.exe. Po tomto je možné simuláciu spustiť. Monitor je možné spustiť interne alebo externe, a to nastavením premennej *$enableInternalMonitor* v súbore *simspark.rb*, na *true* pre interný alebo *false* pre externý monitor.

Na komunikáciu medzi serverom a agentom sa používajú s-výrazy (je to buď reťazec, alebo zoznam ďalších s-výrazov). Správy posielané medzi serverom a agentmi sú kódované v ASCII. Každá správa je prefixovaná svojou dĺžkou (32 bitové bezznamienkové číslo vo formáte Big Endian).

Simulácia RoboCup 3D začína vytvorením agentov, priradením do tímu a ich umiestnením na ihrisko. Každý agent na ihrisku vidí značky, ktoré toto ihrisko ohraničujú. Tieto značky sa nachádzajú v každom rohu ihriska a na oboch okrajoch brány. Agent taktiež vidí relatívnu pozíciu lopty a ostatných agentov. Základné pohyby, ktoré agent potrebuje ovládať, sú chôdza, beh, postavenie sa po páde, kop do lopty a pod. Keďže agent vie, kde sa nachádza lopta, bránky a ostatní agenti, je potrebné, aby tieto informácie vedel využiť k streleniu gólu.

Správanie každého agenta sa ukladá do tried, ktoré sú odvodené od triedy Behavior (triedy zodpovednej za správanie). Dve základné metódy týchto tried sú inicializácia, ktorá slúži na spustenie agenta a zároveň na uloženie na vhodnú pozíciu na ihrisku, a myslenie, ktorá od serveru získava údaje, vyhodnocuje ich a následne určuje ďalšie správanie agenta.

2.2.3 Perceptory

Perceptory slúžia v RoboCup 3D na vnímanie okolia pre jednotlivých agentov. Server pomocou nich posiela každému agentovi špecifickú správu o jeho pozícií v prostredí, na základe ktorej sa hráč môže rozhodovať. Pozastavme sa pri jednotlivých perceptoroch, ktoré môžeme rozdeliť na dve skupiny – základné a futbalové.

Základné perceptory opisujú správanie, ktoré je špecifické v danom prostredí a nie je spojené priamo s futbalovými schopnosťami agenta.

Keďže potrebujeme, aby agent mal aj futbalové schopnosti, musí mať aj perceptory, ktoré to umožňujú. Tieto perceptory sú prístupné iba v simulovaní futbalu.

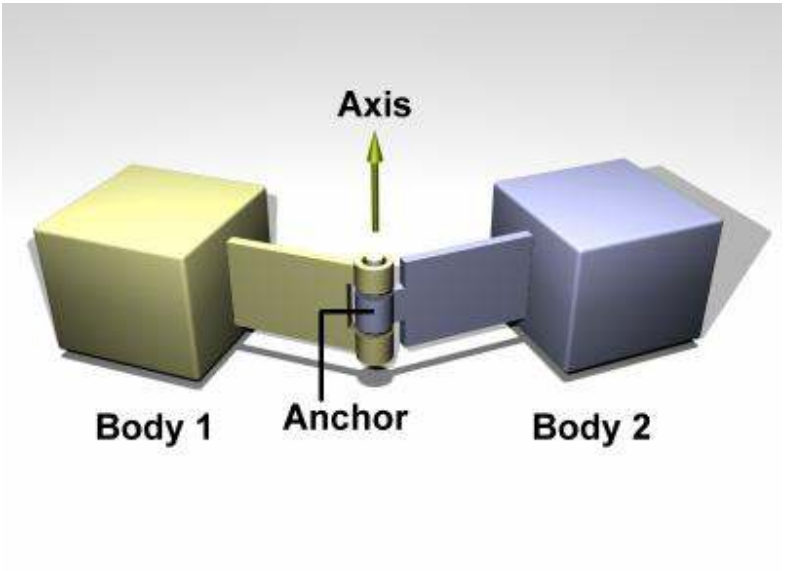
GyroRate Perceptor. Perceptor GyroRate slúži na opísanie orientácie tela hráča. Údaje sa prenášajú pomocou správy, ktorá obsahuje GYR identifikátor a názov časti tela, ku ktorej patrí. Ďalej obsahuje tri hodnoty rotačných uhlov. Práve tieto tri uhly určujú celkovú polohu vzhľadom k súradnicovej

sústave. Robot NAO má tento perceptor umiestnený v hornej časti tela. Formát správy pre tento perceptor vyzerá nasledovne:

Formát správy:	(GYR (n <name>) (rt <x> <y> <z>))
<name>:	Časť tela
<x> <y> <z>:	Uhlová rýchlosť v smere troch osí voľnosti v stupňoch za sekundu
Príklad:	(GYR (n torso) (rt 0.01 0.07 0.46))
Frekvencia zasielania:	Každý cyklus

HingeJoint Perceptor. HingeJoint perceptor určuje, o koľko stupňov je ohnutý daný kĺb robota. Kĺb je zobrazený na *Obr. 15*. Formát správy vyzerá nasledovne:

Formát správy:	(HJ (n <name>) (ax <ax>))
<name>:	Meno korešpondujúceho HingeJoint
<ax>:	Uhol zohnutia daného kĺbu (0 = kĺb je vystretý)
Príklad:	(HJ (n laj3) (ax -1.02))
Frekvencia zasielania:	Každý cyklus



Obr. 15: *Ukážka kĺbu HingeJoint*

UniversalJoint Perceptor. UniversalJoint perceptor sa už v robote NAO nevyskytuje. Nahradili ho dva HingeJoint perceptory, pomocou ktorých hráč ohne kĺby v dvoch smeroch. Formát správy pre tento perceptor vyzeral nasledovne:

Formát správy:	(UJ (n <name>) (ax1 <ax1>) (ax2 <ax2>))
<name>:	Názov kĺbu
<ax>:	Uhly ohybu
Príklad:	(UJ (n laj1 2) (ax1 -1.32) (ax2 2.00))
Frekvencia zasielania:	Každý cyklus

Touch Perceptor. Tento perceptor slúži na oznámenie kolízie jednotlivých hráčov. Toto oznámenie sa vykonáva pomocou binárnych hodnôt 0 a 1. Hodnota 0 označuje, že kolízia nenastala a hodnota 1 predstavuje stav, že prišlo ku kolízii. Tento perceptor sa na novej verzii servera (0.6.4) nepoužíva. Formát správy je nasledovný:

Formát správy:	(TCH n <name> val <bit>)
Príklad:	(TCH n bumper val 1)

ForceResistance Perceptor. Tento perceptor slúži na oznámenie pôsobenia sily a jej vektora. Súradnice *c* určujú bod, na ktorý sila pôsobí, a súradnice *f* určujú práve vektor tejto sily. Robot NAO má dva takéto perceptory a nachádzajú sa v nohách robota. Formát správy pre tento perceptor vyzerá nasledovne:

Formát správy:	(FRP (n <name>) (c <px> <py> <pz>) (f <fx> <fy> <fz>))
<name>:	Názov časti tela
<px> <py> <pz>:	Súradnice, kde nastalo pôsobenie sily
<fx> <fy> <fz>:	Súčasti vektora sily
Príklad:	(FRP (n lf) (c -0.14 0.08 -0.05) (f 1.12 -0.26 13.07))
Frekvencia zasielania:	Každý cyklus

Accelerometer. Tento perceptor počíta zrýchlenie, ktoré dostáva. Robot NAO má tento perceptor umiestnený v hornej časti tela. Formát správy je nasledovný:

Formát správy:	(ACC (n <name>) (a <x> <y> <z>))
<name>:	Názov časti tela
<x> <y> <z>:	Súčasné zrýchlenie
Príklad:	(ACC (n torso) (a 0.00 0.00 9.81))
Frekvencia zasielania:	Každý cyklus

Vision Perceptor. Aby hráč mohol využívať svoje futbalové schopnosti, je nutné, aby vedel, kde sa nachádza, a videl ostatných hráčov, loptu a brány. Na to mu slúži práve tento perceptor, ktorý zachytáva 90° uhol. Na začiatku hry je hráč natočený automaticky na súperovu stranu ihriska, ale musí sa vedieť natočiť aj na opačnú stranu.

GameState Perceptor. Tento perceptor sa využíva hlavne na začiatku, keďže pomocou neho hráč zistí veľkosť ihriska a lopty. Počas hry sa však využíva tiež, nakoľko hráčovi hovorí, aký je čas zápasu a v akom stave sa hra nachádza. Formát správy vyzerá nasledovne:

Formát správy:	(GS (t <time>) (pm <playmode>))
<time>:	Čas v zápase
<playmode>:	Stav zápasu
Príklad:	(GS (t 0.00) (pm BeforeKickOff))
Frekvencia zasielania:	Každý cyklus

AgentState Perceptor. Tento perceptor ukazuje stav batérie v percentách a teplotu agenta v stupňoch.

Hear Perceptor. Tento perceptor slúži na komunikáciu medzi hráčmi. Táto komunikácia však neprebieha priamo, ale len cez server. Hráč taktiež nemôže počuť všetko, ale len do vzdialenosti, ktorú určuje server.

2.2.4 Efektory

Efektory sa požívajú na všetky akcie, ktoré chceme, aby náš agent vykonal. Pomocou efektorov posielame serveru správy o zmenách polohy jednotlivých častí agenta alebo o činnostiach, ktoré následne agent vykoná. Efektory sú rovnako ako perceptory rozdelené do dvoch skupín – základné a futbalové.

Základné efektory slúžia na určenie základného správania agenta, ktoré tesne súvisí s prostredím a nie je späté s futbalovými schopnosťami agenta.

Futbalové efektory ovládajú perceptory, ktoré sú špecifické pre simulovaný futbal. Tieto efektory sú prístupné iba počas simulácie zápasu.

Create Effector. Pomocou tohto efektoru sa odovzdá agentovi názov súboru, ktorý obsahuje opis hráča. Je dostupný po pripojení agenta k serveru a po ňom sa očakáva efektor Init, ktorý hráča priradí k vybranému tímu.

HingeJoint Effector. K pohybu jednotlivými kĺbmi hráča používame HingeJoint efektor, ktorý nám umožňuje zadať názov kĺbu, s ktorým chceme hýbať, a uhol, o ktorý chceme daný kĺb ohnúť.

UniversalJoint Effector. Tento efektor už neexistuje na novom type robota (robot NAO). Slúžil na pohyb kĺbu v smere dvoch osí.

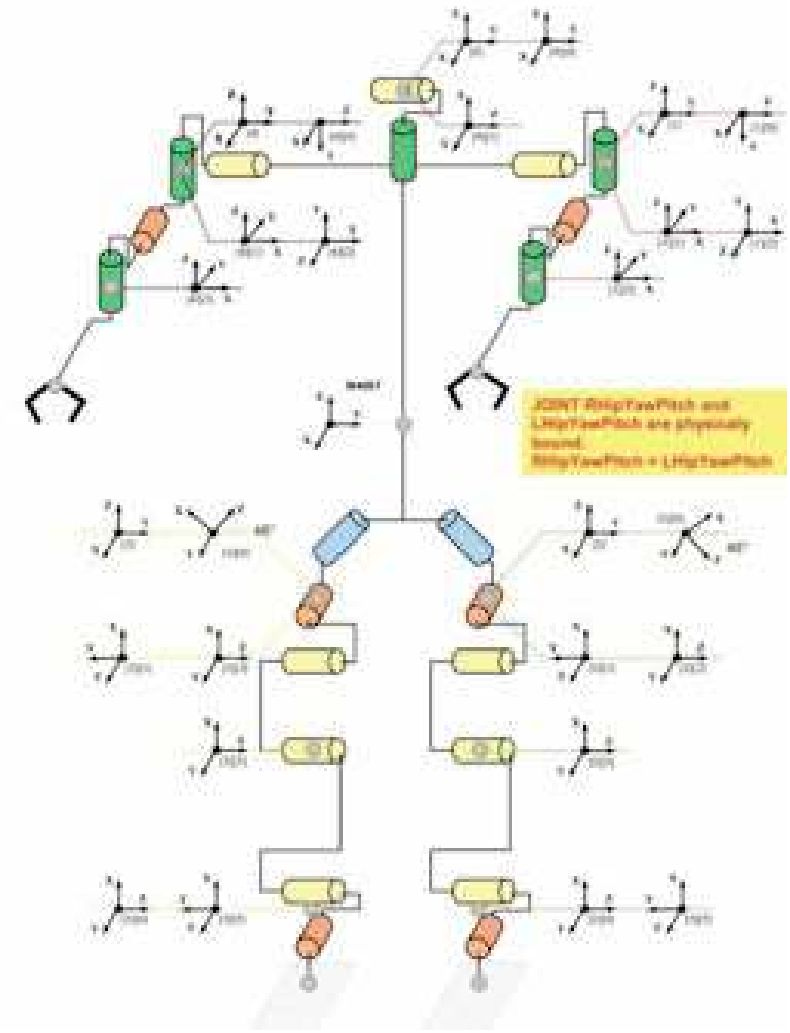
Init Effector. Init efektor sa spúšťa zvyčajne po Create efektore a priradí hráča k vybranému tímu.

Beam Effector. Tento efektor musí byť zavolaný ešte pred začiatkom hry a určuje umiestnenie hráča na hraciu plochu po jeho inicializácií.

Say Effector. Say efektor sa používa na odoslanie správy ostatným hráčom. Správa sa však neposiela priamo, ale cez server. Správa je kódovaná v ASCII kóde.

2.2.5 Robot NAO

Model robota, s ktorým budeme pracovať v projekte RoboCup 3D, je Robot NAO. Jeho výška je 57 cm a váha 4,5 kg. Obsahuje 22 kĺbov. Na *Obr. 16* sú znázornené kĺby a osi robota NAO.



Obr. 16: *Kĺby a osi robota NAO*

2.3 Agent JIM

Agent JIM je fakultný agent, ktorý vznikol prenesením agenta Sirius tímu Critical Error do Javy. Vzhľadom na otvorené zdrojové kódy je dobré zhodnotiť, na akej sa nachádza úrovni a aké sú predpoklady na jeho úspešné prevzatie. Vyhľadávaním informácií a komunikáciou so samotnými autormi sme zistili, že zdrojové kódy agenta sú k dispozícií v SVN repozitári, a to v dvoch verziách:

- Jim (hlavná línia vývoja)
- JimTP (experimentálna línia, vhodné pre Tímové projekty)

Po komunikácii s autormi a predbežnej analýze zdrojového kódu môžeme stav agenta zhrnúť nasledovne:

- Vývojové prostredie Eclipse, jazyk Java
  - Logika a konfigurácie v Ruby
  - Komunikácia, parser, model sveta, predpovedací modul v Jave
- Vývoj aspektovo-orientovane rozšírený pomocou AspectJ
- Vyvíjaný technikou Test Driven Development
  - podľa slov autorov je 90% kódu pod testami
- Zmena načítania pohybov a správania za behu
- Aktívne vyvíjaný projekt, postupne pribúdajúca dokumentácia
- Pohyby - vstávanie z brucha a z chrbta, prototyp chôdze

Po analýze zdrojového kódu sme dospeli k záverom, že kód je:

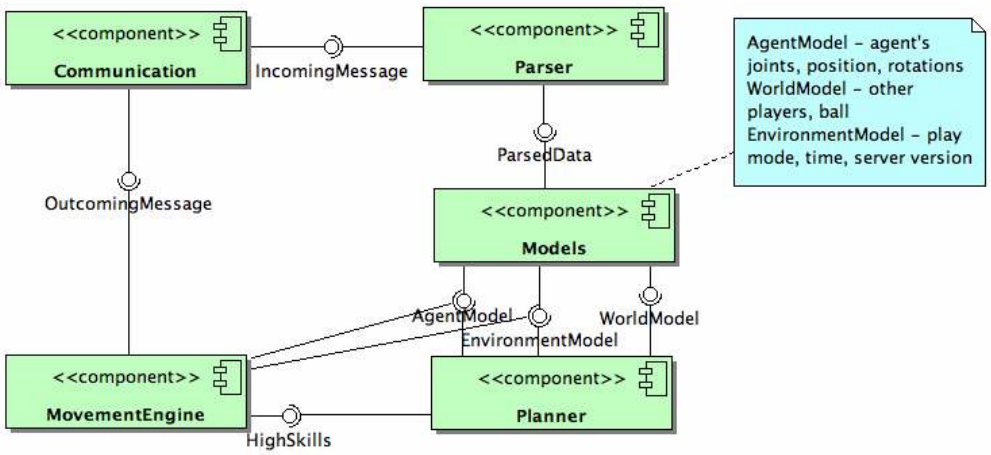
- Štruktúrovaný, vhodne organizovaný
- Obsahujúci moderné techniky - anotácie, testy
- Dokumentovaný na úrovni súborov, nie konkrétnych metód

2.3.1 Architektúra agenta

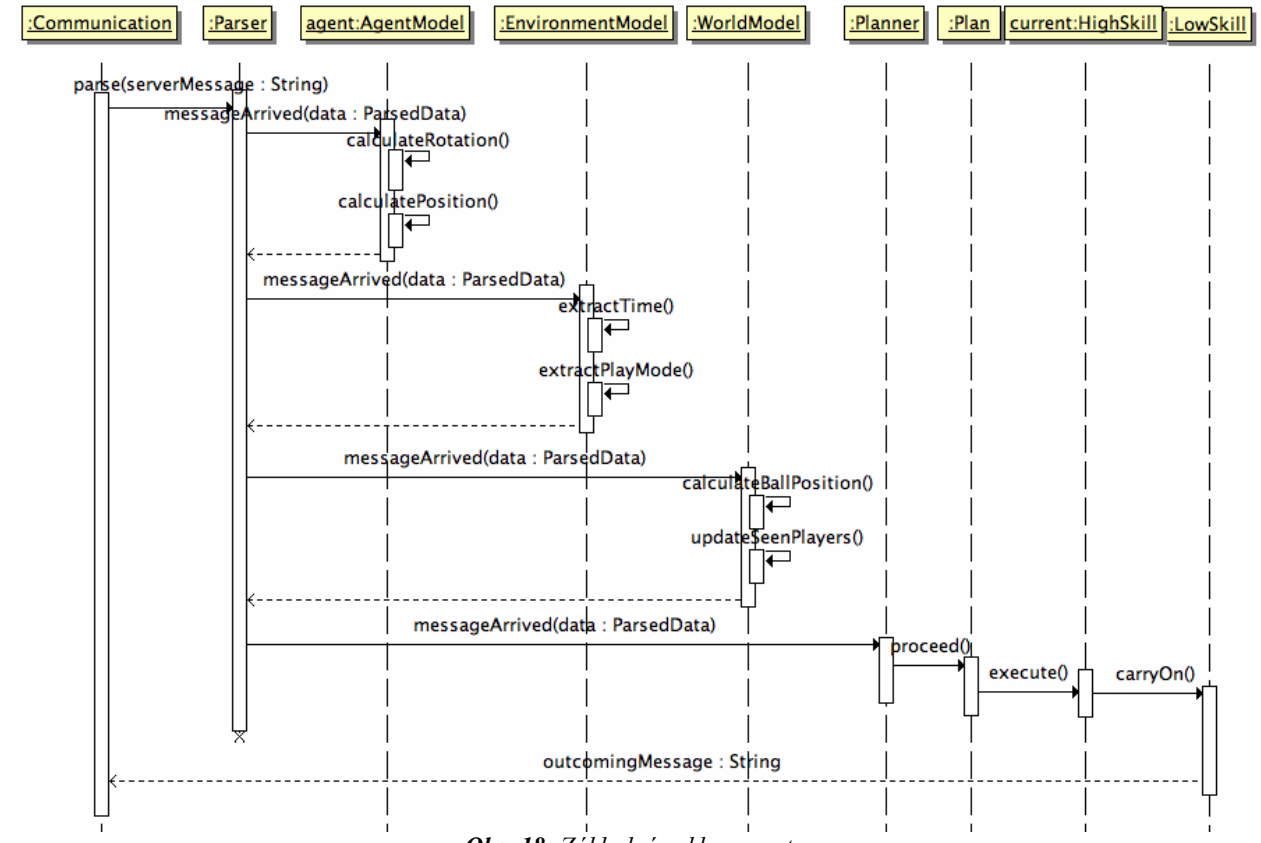
V nasledujúcej časti na základe informácií z repozitára prehľadne zhodnotíme agenta.

Komponenty agenta. Agenta je možné rozložiť na nasledovné komponenty – Communication, Parser, Models, Planner a Movement Engine. Keďže agent vychádza z princípov agenta Sirius, je možné pozrieť si práve jeho architektúru pre bližšie pochopenie vzťahov. Pre dobrú predstavu však postačí *Obr. 17*, kde sú názorne vyjadrené vzťahy medzi komponentmi.

Hlavný cyklus komunikácie. V predchádzajúcej časti bolo ukázané, z akých komponentov sa skladá systém. *Obr. 18* znázorňuje, v akom vzťahu s jednotlivými komponentmi sa nachádza hlavný cyklus komunikácie agenta so serverom.



Obr. 17: *Komponentová dekompozícia agenta JIM*



Obr. 18: *Základný cyklus agenta*

2.3.2 Zhodnotenie

Nakoľko JIM je open-source projekt, ktorý je funkčný na dobrej úrovni a obsahuje množstvo testov, je vhodné vziať do úvahy pokračovanie na tomto agentovi. Taktiež technológie použité na vývoj sú tímu dobre známe a znamenajú pre náš tím pozitívne predpoklady na pokračovanie v agentovi.

2.4 Doplňujúce informácie k analýze

Pri analýze sme odhalili ešte dva problémy, ktorými sa budeme zaoberať:

- Prostredie sa tvári ako agent
- Stabilita hráča je kľúčovým prvkom dobrého agenta

2.4.1 Multiagentové systémy

Potrebujeme sa ešte zmieniť, že sme počas analýzy zistili, že sa multiagentový systém RoboCup 3D, pozostávajúci z *n* agentov (robotov) v simulačnom prostredí v skutočnosti správa ako *n+1* multiagentový systém, pričom je tento systém zostavený z *n* agentov a navyše prostredia.

Zistili sme teda, že sa samotné prostredie správa ako agent a túto okolnosť sme sa rozhodli využiť. Toto sa dá využiť najmä pri pohyboch agenta v simulačnom prostredí (chôdza, vstávanie a pod.). Tu ide o princíp, že sú agenti naučení správať sa iba zadefinovanom prostredí, keďže neobsahujú žiadnu formu vyššej inteligencie, ktorá by im dovolila prispôsobovať sa prostrediu. V našom prípade určite využijeme túto skutočnosť pri implementácii agenta.

2.4.2 Využitie gyroskopu na vylepšenie stability agenta

Gyroskop (GyroRate Perceptor), ako už bolo povedané, slúži na získavanie informácií o zmene v orientácii tela robota, doručuje informácie o uhlových rýchlostiach v smere troch rotačných osí (x, y, z). Server poskytuje hodnoty relatívne k polohe v čase inicializácie robota. Formát správy tvorí identifikátor GYR, názov časti tela, z ktorej údaje pochádzajú a uhlové rýchlosti v smere troch osí:

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

Tím Agenty 007 navrhol rovnovážny modul, ktorý by dokázal určiť aktuálne ťažisko robota a umožnil by detegovať situáciu, kedy robot stráca rovnováhu, a vykonať pohyby, ktoré by robota stabilizovali a zabránili tak pádu. Rovnovážny modul navrhli pomocou goniometrických funkcií.

Implementovaná metóda výpočtov na základe gyroskopu tímu Agenty 007 však nie je úplne presná, keďže sú výpočty zaťažené nepresnosťami vnášanými serverom, a teda ide sa o predikciu. Využívať informácie z gyroskopu by vzhľadom na nepresnosť a zložitosť výpočtov (v prípade, že by sme chceli rovnovážny modul upraviť) nebolo vhodné. Potrebujeme navrhnúť iný spôsob kontrolovania stability agenta (napríklad ZMP použitý tímom Nexus 3D).

3 Špecifikácia požiadaviek

Táto kapitola dokumentu je určená na špecifikáciu požiadaviek a funkcionality, ktorú by sme chceli navrhnúť. Zamerali sme sa na päť základných oblasti. V prvej časti sa budeme zaoberať samotnými pohybmi, lebo ako vieme, ani najlepšia taktika nemá veľkú šancu byť úspešná, ak agent nie je pánom svojho tela, preto je návrh a implementácia čo najdokonalejších pohybov jedným z hlavných problémov RoboCup 3D.

V druhej časti bližšie špecifikujeme vyššie schopnosti agenta a následne aj stratégiu hrania. Tieto vyššie schopnosti sa budú vykonávať pomocou nižších schopností, s tým rozdielom, že budú vykonávané za nejakým účelom. Následne sa kombináciou vyšších schopností budú môcť plánovať útoky alebo obrana tímu.

Nasleduje špecifikácia požiadaviek na rozšírenie editora pohybov tímu Agenty 007 o ďalšiu funkcionalitu, prípadne aj pridanie novej časti, ktorá sa bude zaoberať vyššou logikou, teda plánovaním a stratégiou.

V nasledujúcej časti sa zaoberáme špecifikáciou možných použití jazyka na vytváranie vyššej logiky XABSL. Tento jazyk je založený na stavovom automate.

V poslednej časti sa zaoberáme špecifikáciou testovacieho frameworku, ktorý bude mať za účel testovať schopnosti nášho agenta.

3.1 Pohyby agenta

Z hľadiska pohybov sa môžeme zamyslieť nad piatimi základnými pohybmi:

- Vstávanie
- Chôdza
- Otáčanie
- Kop do lopty
- Bránenie gólom

3.1.1 Vstávanie

Keď robot spadne na zem, ostane ležať na bruchu, na chrbte alebo na boku. Pozície pre inicializáciu vstávania je vhodné obmedziť na vstávanie z chrbta a z brucha, prípadne len z chrbta alebo len z brucha, pričom z ostatných pozícií sa robot pred samotným vstávaním najskôr dostane do zvolenej východiskovej pozície.

Väčšina svetových tímov má implementované vstávanie z brucha aj z chrbta, pričom obe sú si dosť podobné – robot sa rozkročí, pokrčí nohy a podsunie ich pod seba, pričom využije, že mu prostredie umožňuje sunúť nohy po zemi. V našom agentovi by sme chceli implementovať oba štandardné

spôsoby vstávania a pridať ďalší spôsob, ktorý by sa nespoliehal na možnosť sunúť nohy po „trávniku“. Predpokladáme, že posledný spomínaný spôsob bude pomalší ako oba štandardné, ale bude pravdepodobne použiteľný aj v reálnom fyzickom robote na drsnom povrchu.

Pri všetkých troch typoch vstávania bude úlohou robota postaviť sa do základného postoja, v ktorom je stabilný a z ktorého môže pokračovať ďalšími akciami, preto pre overenie úspešnosti nami implementovaných spôsobov vstávania bude rozhodujúci pomer postavení do stabilného postoja k celkovému počtu pokusov.

3.1.2 Chôdza

Ak vie robot kráčať iba dopredu, môže to byť v mnohých situáciách nevýhodné, keďže pre zmenu smeru sa musí najskôr otočiť. Preto by sme okrem chôdze dopredu chceli implementovať u nášho agenta aj schopnosť robiť úkroky, teda kráčať do boku, a schopnosť cúvať, prípadne ísť šikmo.

Ďalej by bolo pre chôdzu dopredu rozumné implementovať rýchlu a pomalú chôdzu. Je predpoklad, že rýchla chôdza bude mať väčšie odchýlky od priameho smeru a bude menej stabilná, preto sa bude používať na prekonávanie veľkých vzdialeností. Naopak pomalá, „presná“ chôdza bude slúžiť na prekonávanie krátkych vzdialeností pri približovaní sa k lopte, aby sme si ju nechtiac neodkopli.

Dve základné podoby chôdze sú drobčenie a humanoidné kráčanie. Väčšina tímov používa chôdzu drobčením (malými krokmi) bez zapojenia pohybu rúk, ktoré ostávajú obvykle vystreté pozdĺž tela robota, preto by sme sa mohli pokúsiť pri našich typoch chôdze využiť aj ruky na ešte vernejšie priblíženie ľuďom. Vyššou úrovňou chôdze, ktorú by sme možno mohli tiež skúsiť implementovať, je plnohodnotný beh, ktorý zatiaľ tímy nevyužívajú.

Úspešnosť vytvorenia stabilných typov chôdze bude meraná priemerným časom, počas ktorého robot kráčal a nespadol. Druhým meradlom úspešnosti bude rýchlosť jednotlivých typov chôdze, teda priemerný čas, za ktorý robot prejde celé ihrisko alebo jeho časť. Posledným kritériom bude odchýlka od želaného smeru.

3.1.3 Otáčanie

Otáčanie by malo byť skutočne rýchle a stabilné, aby sa ním nezaberalo priveľa času. Tiež by malo byť presné, teda bude dobré, ak sa robot bude vedieť otáčať postupne po malých uhloch. Otáčanie implementujeme do oboch smerov symetricky. Taktiež je možné implementovať ďalšie typy otáčania – otočenie o 45°, 90°, 135°, 180°, „vojenské“ otočenie o 90°.

Základom je postupné otáčanie, preto o ostatných typoch budeme uvažovať a pokúsime sa ich implementovať podľa toho, ako rýchlo budú postupovať práce na pohyboch vyššej priority.

3.1.4 Kop do lopty

Základným spôsobom kopania je kop dopredu, ktorý je možné implementovať ako kop špičkou alebo stranou chodidla. Kop pootočeným chodidlom má predpoklady byť presnejší, ale kop špičkou je jednoduchší na vytvorenie, preto implementujeme oba spôsoby a porovnáme ich. Pri kope dopredu môže robot pracovať s celým telom alebo len s dolnou končatinou. Aj keď je zložitejšia práca s celým telom, lebo si vyžaduje zladenie viacerých pohybov, vyskúšame oba postupy, pretože kop s napriahnutím je podobnejší ľudskému.

Ďalej, pre ušetrenie času potrebného na to, aby sa robot dostal do správnej pozície na použitie kopu dopredu, by sme chceli implementovať a odladiť aj kop do boku, pričom tento by bol realizovaný stranou chodidla, nie špičkou. Ďalším možným kopnutím je kop dozadu alebo kop pod určitým uhlom (napr. 45°). Týmito ďalšími spôsobmi kopu do lopty sa budeme zaoberať podľa postupu prác na dôležitejších pohyboch.

Overením úspešnosti implementácie spôsobov kopania do lopty bude pomer kopnutí bez spadnutia ku všetkým kopnutiam daným spôsobom. Ďalším kritériom bude presnosť a dosah kopnutia.

3.1.5 Bránenie

Na svetovej úrovni sa používajú predovšetkým dva spôsoby bránenia – pádom brankára s vystretým telom na bok alebo sadnutím si na zem s rozkročenými nohami. Prvý spôsob je bežnejší, ale má nevýhodu, že keď z neho robot vstáva, tak stráca loptu z dohľadu. Zo sedu rozkročmo je možné pomerne rýchlo sa postaviť (je to totiž jedna z fáz vstávania) a brankár stále vidí situáciu pred sebou a pozíciu lopty.

Implementujeme a vyskúšame oba spôsoby bránenia, napriek tomu, že druhý je pre človeka v podstate nerealizovateľný a využíva vlastnosť prostredia, že je ihrisko hladkou plochou, po ktorej je možné šúchať nohami. Ďalšou výhodou druhého zo spomínaných spôsobov bránenia je totiž fakt, že ak je brankár priamo v ceste lopte, tak pádom na bok uvoľní lopte cestu, kým rozkročením a dosadnutím ostane v ceste lopte a pokryje aj pomerne široký priestor na obe strany od trajektórie lopty, čím zabráni gólu pri jej miernom vychýlení.

Úspešnosť bude určovať pomer chytených striel na bránu ku všetkým strelám.

3.2 Vyššia logika

Cieľom Robocupu je vytvorenie agenta schopného hrať futbal. Aby bolo možné tento cieľ splniť, je potrebné, aby agent disponoval aj vyššími schopnosťami. Po analýze agenta JIM, na ktorého nadväzujeme, sme zistili, že agent v súčasnosti nedisponuje vyššími schopnosťami, a tak je požiadavkou ich implementovať. Medzi vyššie schopnosti agenta sme zaradili nasledovné:

1. *Chôdza* – predstavuje schopnosť dostať sa na určené miesto. Budú využité nižšie schopnosti agenta ako pohyb dopredu, či otočenie sa.
2. *Vstanie* – predstavuje schopnosť vstať z polohy ležmo. Budú použité nižšie schopnosti vstávania z chrbta a vstávania z brucha.
3. *Kop do lopty* - predstavuje schopnosť kopnúť do lopty na stanovený cieľ. Týmto cieľom môže byť bránka, spoluagent, prípadne akýkoľvek bod na ihrisku. Použijeme nižšie schopnosti rôznych typov kopania do lopty a otáčania.
4. *Vedenie lopty* - prestavuje schopnosť vedenia lopty jedným agentom. Bude pozostávať zo schopností kopu do lopty a chôdze.
5. *Zorientovanie sa* – predstavuje schopnosť zistiť svoju vlastnú polohu na základe polohy bránok a polohu lopty. Táto schopnosť bude pozostávať zo schopnosti otáčania sa.
6. *Blokovanie strely* – predstavuje schopnosť zablokovať strelu správnym postavením agenta a následným zablokovaním strely nižšími schopnosťami ako pádom, či rozkročením. Túto schopnosť bude využívať predovšetkým brankár.

Spomenuté vyššie schopnosti agenta nám pomôžu abstrahovať od detailov nižších schopností agenta, a tak zjednodušiť ich výber a postupnosť. Ich počet však nie je konečný a v prípade potreby, je možné ich kombinovať a vytvoriť tak schopnosť agenta na ešte vyššej úrovni, napríklad útok. Takto by sa následne mohli urobiť kombinované pohyby vyššej úrovne, ktoré by sa mohli využiť pri plánovaní stratégií.

3.3 Editor pohybov a správania

Tím Critical Error, z ktorého sme sa rozhodli vychádzať, používa pre modelovanie pohybov XML súbor s presne definovaným formátom. Nový editor pohybov, ktorý rozpracoval tento tím od tímu Agenty 007, dokáže už brať ako vstup nie len textový súbor predpísanej štruktúry, ale aj exportovať a importovať XML súbory, v ktorých sú pohyby zadefinované.

Ďalším krokom pri práci bude úprava XML súboru tak, aby obsahoval aj nami požadované informácie, teda aby sa neriadil rýchlosťou pohybu (taktami), ale samotným časom priebehu akcie. Zároveň potrebujeme nastaviť nekontrolovanie ukončenia pohybu, a tak umožniť, aby sa pohyb, ak sa náhodou neukončí v zadanom čase, presunul do nasledovnej sekvencie. Takáto štruktúra je čiastočne navrhnutá v agentovi JIM, na ktorého budeme nadväzovať.

Vhodné by bolo upraviť existujúci modul pre logovanie, ktorý tím Critical Error používal od tímu Hviezdna jedenástka. Toto by bolo vhodné zakomponovať do editora tak, aby pomocou vypísaných logov pohybov, mohol editor vytvoriť XML súbor tých pohybov. Toto najmä pomôže budúcim riešiteľom RoboCup 3D. Logovanie je nateraz vyriešené tak, že sa volá v každej časti kódu samé za seba. Bolo by vhodné toto zhrnúť do osobitného modulu. Na riešenie tohto problému by sa mohli použiť možnosti aspektovo-orientovaného programovania a jazyka AspectJ, ktorý je v častiach agenta JIM už použitý.

Pri realizácii pohybov vidno, že niektoré časti tela sa pohybujú symetricky v rovnakom smere alebo symetricky v opačnom smere. Toto by sa dalo využiť tak, že by sa do editora pridala možnosť zdvojených pohybov, ako to má implementované tím Kouretes v ich editore. Napríklad pre ramenný kĺb (ľavý a pravý) by bolo vhodné pridať tri možnosti:

- Samostatný pohyb – kĺb sa pohybuje samostatne
- Spojený (symetrický) pohyb – kĺby sa pohybujú spolu, lineárne v čase (ľavé a pravé rameno)
- Zrkadlový (opačný pohyb) – kĺby sa pohybujú po zrkadlových trajektóriách

Keď sa pozrieme o kúsok ďalej, teda na spracovávanie vyššej logiky agenta a plánovania stratégie, dobrým nápadom by bolo pridať k editoru pohybov aj editor na upravovanie stratégie. Toto by sa dalo riešiť implementovaním produkčného systému. Na produkčný systém by bol vytvorený samostatný modul, ktorý by sa správal ako plug-in k terajšiemu editoru. Tým by sa zabezpečilo to, že by sa v budúcnosti v prípade potreby tento modul jednoducho mohol odpojiť a nepoužívať alebo by sa mohol pripojiť k inému editoru.

3.4 Definovanie vyššej logiky pomocou XABSL

Ako alternatívu produkčného systému by bolo možné použiť jazyk na definovanie vyššej logiky XABSL. V konečnom dôsledku by sa aj tento strategický pohľad na hranie agentov mohol implementovať do editora pohybov.

XABSL (*Extensible Agent Behavior Specification Language*) je jazyk vhodný na opísanie vyššej logiky nášho hráča. Vzhľadom na nadviazanie na agenta implementovaného v jazyku Java je výhodou, že je k dispozícii aj Java XABSL library. Využiť opis správania pomocou XABSL môže byť výhodné najmä, ak komplexnosť správania začne rásť a implementovanie takéhoto správania v jazyku Java by začalo byť neefektívne. Pri opise správania v XABSL budeme klásť dôraz na ďalšiu rozšíriteľnosť.

Správanie agenta sa tak bude môcť opísať pomocou .xabsl súborov. XABSL je dialekt jazyka XML, s ktorým budeme pracovať aj z dôvodu definovania pohybov nášho agenta. Prvky nižšieho správania implementujeme v jazyku Java, no stavy konečného automatu sa na ne budú odkazovať, preto musíme prototypy funkcií a definície parametrov špecifikovať v XABSL súbore.

3.5 Framework pre efektívne testovanie schopností agenta

Predpoklady agenta simulovaného robotického futbalu na úspešnosť môžeme hodnotiť z dvoch pohľadov:

- architektúra
- parametre

Keďže sa agent rozhoduje v reálnom čase a v prostredí, ktorého dôveryhodnosť je cielene znižovaná vnášaním chýb, je dôležité, aby agent bol na vysokej úrovni z obidvoch hľadísk. Kým o architektúre môžeme uvažovať ako o vhodnej, pri parametroch môžeme ísť ďalej a uvažovať o nich ako o správnych.

Posúdenie správnosti parametrov je však problém najmä z časového hľadiska, keďže simulácia odzrkadľuje reálny svet. Ako príklad môžeme uvažovať chôdzu agenta, kde parametre predstavujú natočenia a pozíciu kĺbov v konkrétnych časoch. V prípade, že sa zameriame na logiku vyššej úrovne, môžeme pod parametrami uvažovať pojmy ako blízkosť lopty, či napr. taktiku.

Cieľom testovacieho frameworku je vytvoriť rámec, v ktorom sa explicitne určí, čo sa od agenta/agentov očakáva, následne sa vyhodnotí, či sa daná podmienka splnila. Cieľom je tieto procesy simulácie a vyhodnotenia robiť automatizovane, zároveň pri zrýchlení simulácie do maximálnej miery.

3.5.1 Špecifikácia hlavných funkcií

Vzhľadom na hore uvedenú všeobecnú špecifikáciu je možné zhrnúť funkcie produktu do nasledovných bodov:

1. Stanovenie očakávaných udalostí
2. Vyhodnotenie vzťahu výslednej udalosti a očakávanej udalosti
3. Prehľadávanie priestoru parametrov pre minimalizáciu rozdielu medzi výslednou udalosťou a očakávanou udalosťou

Ako príklad uvedieme testovanie vstávania agenta zo zeme. Mapovaná na predchádzajúce kroky bude realizácia špecifikácie pozostávať z:

1. Stanovenia výslednej pozície, v ktorej sa má agent nachádzať
2. Vyhodnotenia, v akej sa nachádza a za aký čas
3. V prípade potreby optimalizácie pomocou techník umelej inteligencie hľadať vhodnejšie parametre modelu

4 Návrh prototypu

Táto kapitola má za cieľ navrhnúť spôsob implementácie špecifikovaných požiadaviek z kapitoly 3. Špecifikácia je vytvorená rozsiahlejšie, pričom implementovať budeme len časti.

V prvej časti návrhu sú opísané možné zmeny v zadefinovaných pohyboch agenta. Nasleduje časť, ktorá je tesne spätá s pohybmi a to definovanie vyššej logiky pohybov. Tu je navrhnutý spôsob ako sa agent bude rozhodovať pri použití jednotlivých pohybov.

4.1 Hrubý návrh implementácie pohybov

Napriek tomu, že sa implementácia pohybov môže zo špecifikácie zdať jednoduchá, dosiahnuť skutočne stabilné a zároveň rýchle pohyby vôbec nie je triviálna úloha, preto sme sa do prototypu rozhodli z každej triedy dôležitých pohybov zahrnúť len niektoré základné. Pohyby budeme vytvárať v prostredí editora pohybov buď úpravou pohybov agenta JIM, alebo návrhom vlastných pohybov od začiatku.

4.1.1 Vstávanie

Spomínané boli tri rôzne možnosti, ako implementovať vstávanie, pričom posledný z nich – vstávanie bez „využitia“ prostredia – nepatrí medzi priority, preto ho do prototypu zahŕňať nebudeme a radšej sa zameriame na zdokonalenie a testovanie prvých dvoch spôsobov vstávania, ktoré by podľa správ od našich predchodcov o stave agenta JIM mali byť už vytvorené.

4.1.2 Chôdza

Chôdza je jeden z najzložitejších pohybov, pokiaľ ide o stabilitu, preto budeme tomuto pohybu venovať najväčšiu pozornosť. Agent JIM má implementovaný prototyp chôdze, ktorú ale treba ešte vylepšiť a otestovať. V prototype sa zameriame na chôdzu dopredu, dozadu a do strán (úkroky). Posledná chôdza je dôležitá hlavne pre brankára, aby mal stále loptu v zornom poli. Drobčenie je považované za všeobecne stabilnejšie než humanoidné kráčanie, ale v prototype sa budeme zaoberať oboma spôsobmi pohybu.

4.1.3 Otáčanie

Existujú XML súbory pre otáčanie do oboch strán, preto v prototype budeme tieto pohyby testovať a vylepšovať. Z vlastných navrhovaných spôsobov otáčania v prototype budeme pracovať na otočení o 90°, 180° a na „vojenskom“ otočení. Tým posledným sa budeme zaoberať hlavne z toho dôvodu, že ho považujeme za najrýchlejší spôsob otočenia. Problémom však je udržanie stability agenta. Pokúsime sa implementovať a otestovať tento spôsob otáčania tak, aby bol čo najstabilnejší a za úspech budeme považovať, ak pri testovaní z 10 pokusov zostane agent 9-krát stáť a bude môcť úspešne pokračovať v hre.

4.1.4 Kop do lopty

Kopanie do lopty je ďalším na stabilitu veľmi náročným úkonom. Zatiaľ pre kopanie neexistujú žiadne XML súbory, preto ho budeme vyvíjať od začiatku. V prototype sa zameriame hlavne na kopanie dopredu špičkou aj stranou nohy a na presnosť a dosah týchto kopnutí. Porovnáme kopanie využívajúce prácu celého tela s kopaním len za pomoci kĺbov spodnej končatiny.

4.1.5 Bránenie

Agent JIM zatiaľ nemá implementované pohyby brankára, takže bránenie gólom budeme vyvíjať od základov. Spôsoby bránenia gólom sme sa rozhodli zahrnúť oba, keďže ich implementácia vyzerá byť zvládnuteľná pomerne rýchlo.

4.2 Vyššia logika pohybov agenta

Návrh jednotlivých vyšších schopností agenta súvisí s už existujúcimi časťami implementovanými v agentovi JIM. Ide predovšetkým o časti, ktoré sú zodpovedné za reprezentáciu sveta a plánovaciu triedu *Planner*. Pre ilustráciu návrhu vyšších schopností agenta budú použité diagramy činností, ktoré znázornia výber správnej akcie na vykonanie v jednom cykle simulácie. Navyše pri každej schopnosti uvedieme, aké sú vstupné parametre pre danú schopnosť. Zo spomenutých schopností bolo vynechané blokovanie strely, ktoré bude pravdepodobne navrhnuté v letnom semestri.

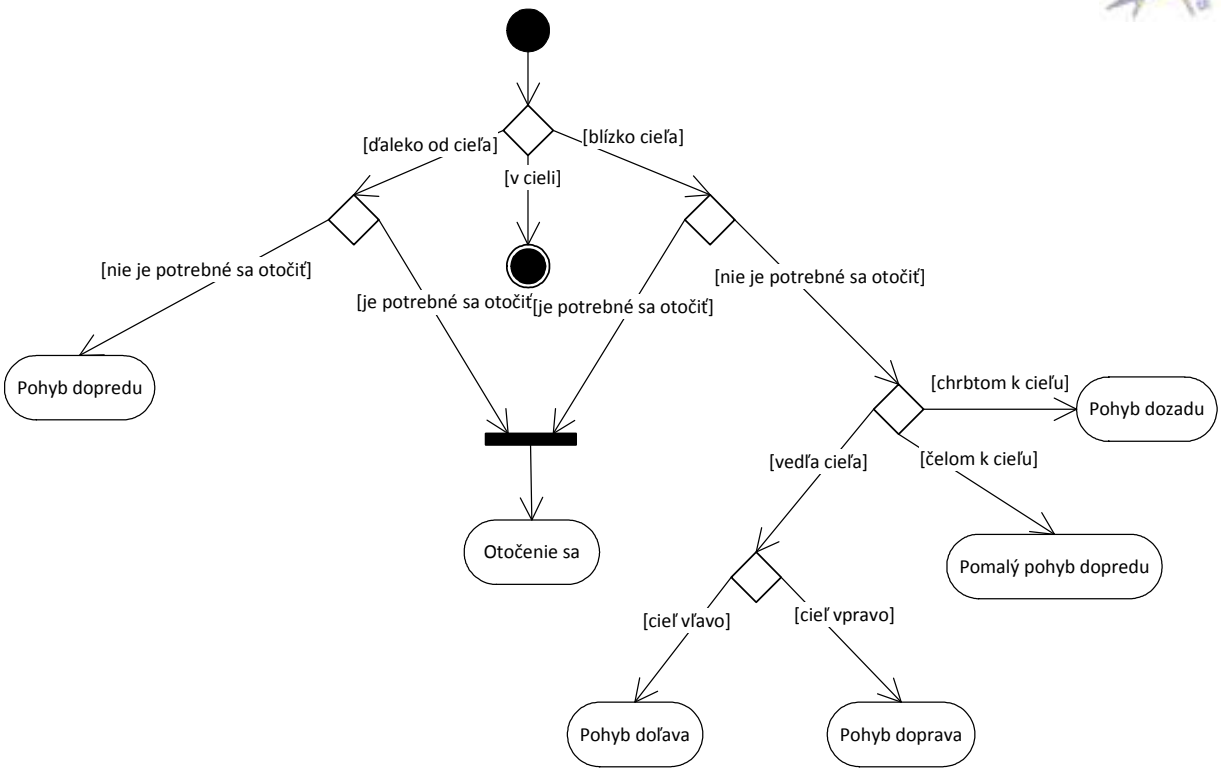
4.2.1 Chôdza

Ako už bolo spomenuté v časti venovanej špecifikácií vyšších schopností, chôdza zabezpečuje pohyb z jedného bodu na druhý. Pri inicializácií tohto pohybu je preto potrebné poznať bod, do ktorého má agent ísť.

Tabuľka 1: *Vstupné parametre chôdze*

Parameter	**Opis**
Cieľ	Bod, kam agent smeruje

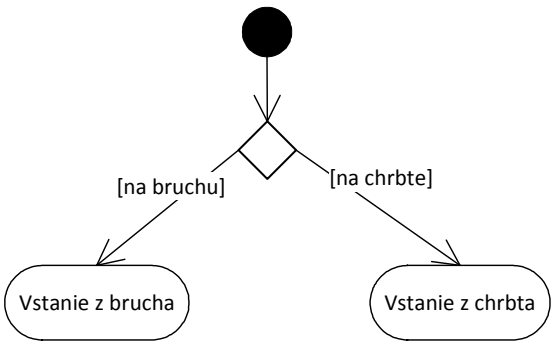
Návrh chôdze vychádza z požiadavky minimalizovať jej trvanie. Z tohto dôvodu je výber akcií závislý od vzdialenosti agenta od cieľa. Ak je táto vzdialenosť dostatočne malá, agent bude môcť využiť chôdzu do strán, či dozadu. Inak sa agent otočí na požadovaný uhol a pôjde rovno. Postup je znázornený na *Obr. 19*.



Obr. 19: *Chôdza agenta*

4.2.2 Vstávanie

Agent v prípade vstávania nepotrebuje ďalšie parametre na výber vhodnej akcie. Predpokladáme, že agent má informáciu o svojom aktuálnom stave, t.j. či leží na chrbte alebo na bruchu. Výber akcie je teda jednoduchý a zobrazuje ho *Obr. 20*.



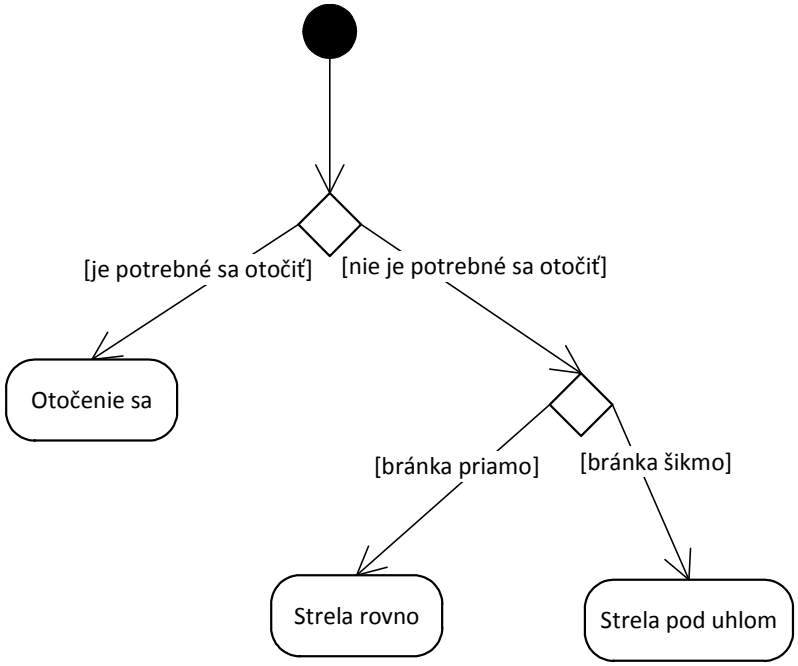
Obr. 20: *Vstávanie agenta*

4.2.3 Kop do lopty

Pri tejto schopnosti vychádzame z toho, že agent disponuje zatiaľ iba dvomi druhmi kopu – kop rovno a kop šikmo. Medzi vstupné parametre patrí okrem cieľu kopu aj sila, akou má agent do lopty kopnúť. Výber akcie je znázornený na *Obr. 21*.

Tabuľka 2: *Vstupné parametre kopu do lopty*

Parameter	Opis
Cieľ	Bod, na ktorý bude smerovať kop
Sila	Sila, akou agent kopne do lopty (maximálna alebo presná)



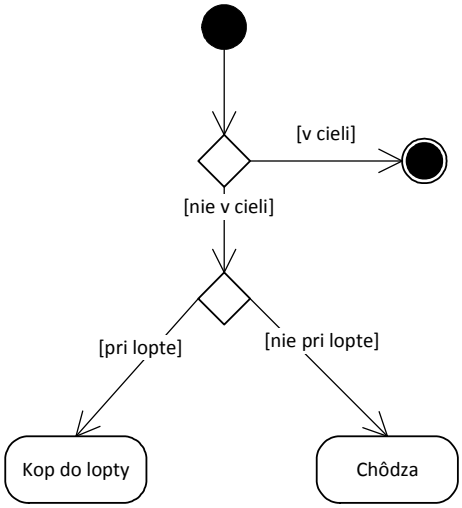
Obr. 21: *Kop do lopty*

4.2.4 Vedenie lopty

Vedenie lopty bude riešené predkopávaním si lopty smerom k stanovenému cieľu. Využijeme tu spomenuté vyššie schopnosti agenta – chôdza a beh, aby sa agent dostal k lopte. Diagram výberu akcie je znázornený na *Obr. 22*.

Tabuľka 3: *Vstupné parametre vedenia lopty*

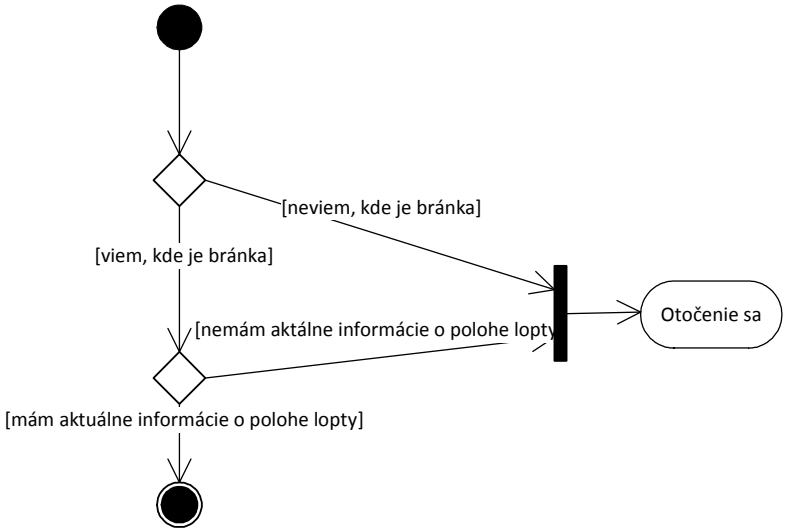
Parameter	Opis
Cieľ	Bod, kam agent s loptou smeruje



Obr. 22: *Vedenie lopty*

4.2.5 Zorientovanie sa

Táto schopnosť slúži na zistenie svojej polohy a polohy lopty. V prípade, že obe tieto informácie sú známe a aktuálne, táto schopnosť nemá zmysel. Postupnosť akcií agenta je znázornená na *Obr. 23*.



Obr. 23: *Zorientovanie sa*

4.3 Návrh editora správania

Rozšírenie editora pohybov vytvoreného tímom Agenty 007 a rozpracovaného tímom Critical Error, má dva štádiá. Prvým je práca na samotnom editore pohybov a zlepšenie jeho funkcionality a druhým je implementovanie editora správania, ktorý by bol plug-inom tohto editora.

4.3.1 Editor pohybov

V špecifikácii boli identifikované niektoré prvky, ktoré by mohli byť implementované na zlepšenie editora pohybov tímu Agenty 007 (a aktuálnej verzie prepracovanej tímom Critical Error). V editore na základe špecifikácie bude pridaný panel, v ktorom si používateľ bude môcť vybrať, či chce daný kĺb ohýbať samostatne alebo spolu s jeho „bratom“ (symetrickým, párovým kĺbom), a taktiež bude môcť zvoliť, či chce, aby sa ten pohyb konal symetricky alebo po zrkadlovej trajektórii.

Navrhnutý je aj refactoring kódu editora a doplnenie používateľského rozhrania o ďalšiu funkcionalitu, ktorá bude umožňovať zložitejšie pohyby dvoch a viacerých kĺbov odrazu.

4.3.2 Editor správania

Návrh editora správania, ako už bolo spomenuté, je možné urobiť dvomi spôsobmi. Buď to bude samostatný produkčný systém, ktorý bude pracovať z faktami okolitého sveta a na základe stavu agenta sa bude rozhodovať, akú akciu agent urobí, alebo druhým spôsobom je implementovanie stavového automatu a definovanie schopností pomocou jazyka XABSL.

Na návrh algoritmu práce produkčného systému sme sa inšpirovali predmetom umelá inteligencia a navrhli sme ho nasledovne:

1. načítaj súbor s pravidlami (súbor, ktorý bude mať špecifickú štruktúr a bude obsahovať fakty, ktoré sa budú vzťahovať na stav okolitého sveta)
2. vyber pravidlo z bázy poznatkov (toto pravidlo bude zhruba obsahovať podmienky a akcie, ktoré majú byť vykonané pri splnení podmienok)
3. pre zvolené pravidlo urob:
    - nájdi všetky kombinácie naviazaní premenných v podmienkovej časti
    - pre každé naviazanie urob
        - naviaž akciu na pravidlo (pravá strana)
        - odfiltruj z akcie nesprávne operácie
        - zaraď akciu medzi akcie na vykonanie (ak akcia má aspoň jednu operáciu)
4. ak neexistuje žiadna aplikovateľná akcia, tak ukonči proces, inak choď na krok 5.
5. vykonaj prvú akciu a choď na krok 2.

Samotný editor správania bude implementovaný v jazyku Java a bude vytvorený ako plug-in modul do vyššie spomenutého editora pohybov. Samostatné akcie a fakty budú implementované pomocou konštrukcií jazyka Java, najpravdepodobnejšie pomocou spájaného zoznamu (LinkedList). Editor správania bude umožňovať exportovanie taktických rozhodnutí agenta, najpravdepodobnejšie do XML súboru.

Jednotlivé pravidlá budú odvodené z analýzy tímov a návrhu vyššej logiky hrania agenta opísanej v predošlej časti.

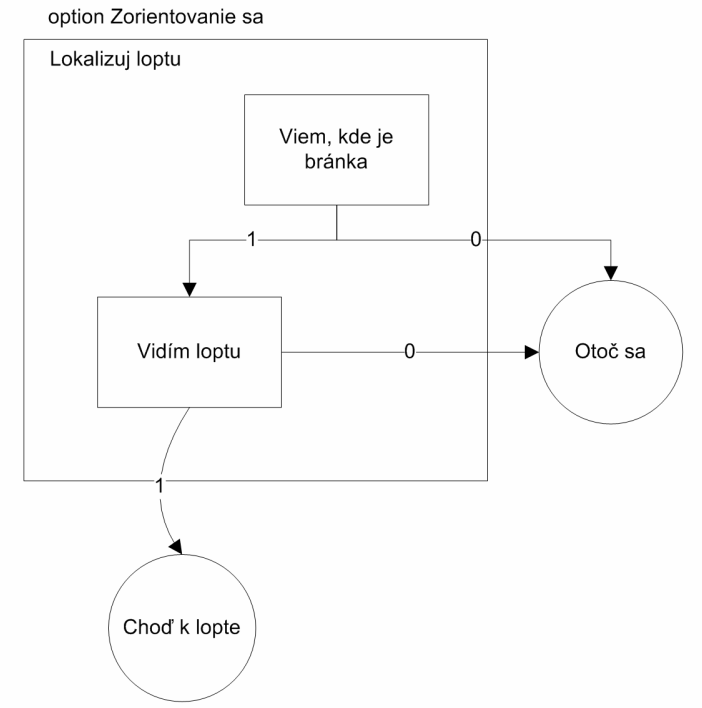
4.4 Vyššie správanie v XABSL

Vyššie správanie v XABSL sa opisuje ako hierarchický konečný stavový automat, preto musíme zvolenú vyššiu logiku vyjadriť v tejto forme.

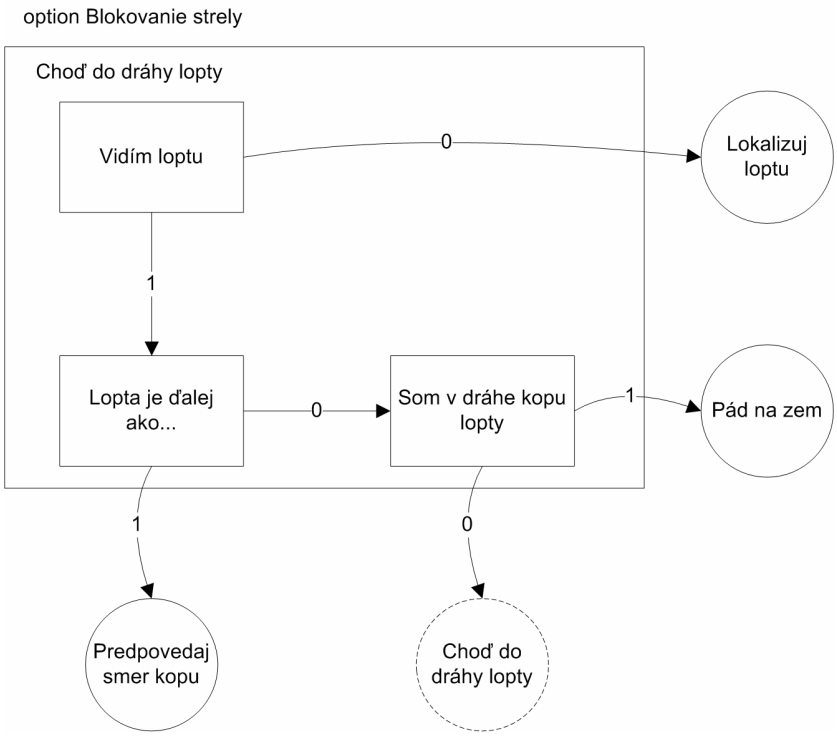
Agent v XABSL pozostáva z menších celkov – tzv. *options,* čo sú vlastne moduly správania, z ktorých každý je konečným automatom. V našom prípade pôjde o:

- zorientovanie sa
- blokovanie strely

Tieto sú zoskupené v strome, ktorého koncové uzly tvoria prvky nižšieho správania, ktoré vyvolajú požadovanú akciu. Prechody stavov realizujeme ako rozhodovacie stromy. Návrh tohto správania v XABSL je možno vidieť na *Obr. 24* a *Obr. 25*. Treba upozorniť, že nižšie činnosti implementujeme v Jave.



Obr. 24: *Zorientovanie sa (návrh pre jazyk XABSL)*



Obr. 25: *Blokovanie strely (návrh pre jazyk XABSL)*

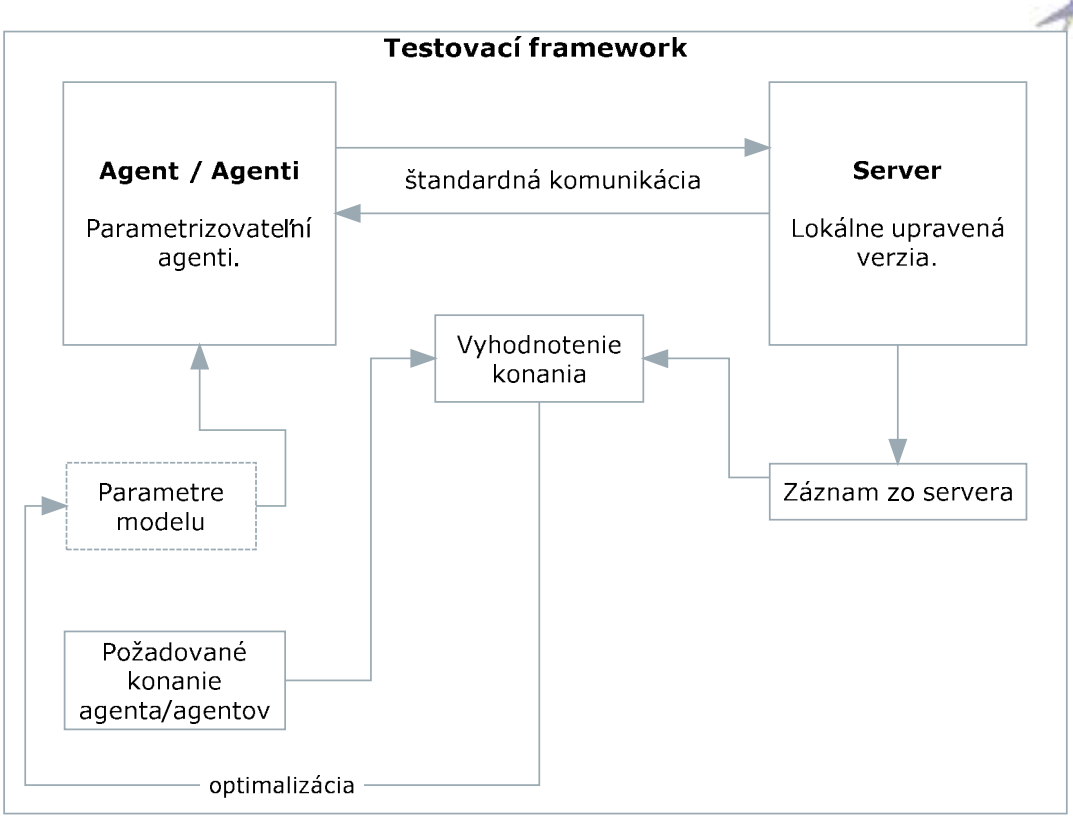
4.5 Návrh testovacieho frameworku

Na úspešné splnenie požiadaviek uvedených v špecifikácií je nutné vykonať zásah do servera aj agenta, keďže je potrebné do maximálnej miery zrýchliť simuláciu. Nakoľko server spoločne s jeho fyzikálnym modulom je prístupný so zdrojovými kódmi, táto úprava je realizovateľná. Úpravou fyzikálnych parametrov je možné meniť rýchlosť simulácie, a tým aj docieliť efektívnejšie testovanie agenta.

4.5.1 Architektúra testovacieho frameworku

Nakoľko požadovaná funkcionalita pracuje na úrovni nad samotným agentom, testovací framework musí umožňovať vloženie agenta do systému. To platí v prípade testovania schopností nižšej úrovne. V prípade testovania vyššej logiky, musí framework umožňovať vloženie viacerých agentov do systému.

Testovanie agenta vyžaduje špecifikovanie očakávaného výsledku. Následne po uskutočnení simulácie sa pomocou záznamu o zápase vyhodnotí, či výsledok dosiahol požadovanú úroveň. Ak nie, framework umožní technikami umelej inteligencie hľadať lepšie nastavenie parametrov. Vzhľadom na povahu problému bude vhodné použiť genetické algoritmy. Na *Obr. 26* je architektúra systému znázornená diagramom.



Obr. 26: *Architektúra testovacieho frameworku*

4.5.2 Opis blokov architektúry

V nasledujúcej časti sú opísané netriviálne bloky, ktoré sa nachádzajú na *Obr. 26*:

Parametre modelu. Blok špecifikuje jednoducho meniteľné parametre agenta, resp. agentov, ktoré budú môcť byť ovplyvňované v rámci optimalizácie. Parametre modelu predstavujú v skutočnosti komunikačný protokol pre použité genetické algoritmy a agenta.

Záznam zo servera. Záznam zo servera slúži na získanie informácií o simulácii. Informácie, ktoré sa týmto spôsobom získajú, je následne potrebné pretransformovať do opisu, ktorý bude kompatibilný s opisom v bloku Požadované konanie agenta/agentov.

Požadované konanie agenta/agentov. Požadované konanie je blok, do ktorého zadáva informácie používateľ. Tieto informácie sú opisom očakávaných udalostí, ktoré sa v bloku Vyhodnotenie konania budú porovnávať s informáciami získanými zo Záznamu zo servera.

Vyhodnotenie konania. V predchádzajúcich opisoch bol uvedený účel bloku Vyhodnotenie konania. Je potrebné uvedomiť si, že jeho zložitosť závisí od miery rozdielu medzi informáciami získanými od používateľa a informáciami, ktoré získava blok zo Záznamu zo servera. Nakoľko informácie zo servera nie sú rýdzo deterministické, je potrebné vytvoriť opis, ktorý bude vždy môcť uvažovať s vopred stanovenou mierou nepresnosti.

4.5.3 Spoločný jazyk pre *požadované konanie* a *záznam zo servera*

Nakoľko systém musí umožňovať vyhodnocovanie konania, je potrebné vytvoriť systém na opis udalostí, ktoré sa stali na ihrisku. Pri nižšej logike bude opis požadovaného stavu spočívať v špecifikovaní pozície kľúčových kĺbov, pri vyššej logike pôjde o pozície agentov.

4.5.4 Informácie k implementácii

Implementácia návrhu bude realizovaná v jazyku Java. Rovnako ako aj pri serveri, je potrebné upraviť v agentovi ponímanie času tak, aby bol adaptabilný na zrýchlenú simuláciu. Pri použití umelej inteligencie vo forme genetických algoritmov je možné použiť bežne dostupné knižnice pre tento účel, napr. JGAP alebo JAGA.

5 Prototyp

Táto kapitola sa zaoberá implementáciou prototypu v rámci projektu RoboCup 3D. Rozhodli sme sa do prototypu implementovať tri časti. Prvá závažná časť je rozsiahlejší návrh a čiastočná implementácia testovacieho frameworku. Tento fraamework pozostáva z dvoch častí: parsera správ a samotného frameworku, ktorý bude slúžiť na testovanie schopností agenta. Tento framework bude svojím spôsobom fungovať ako tréner pre agenta.

V druhej časti sú opísane pohyby, ktoré sme implementovali pre agenta JIM.

Posledná časť sa zaoberá zmenami v editore pohybov. Pre naše účely sme sa rozhodli, pre ľahšiu prácu na ďalšom vývoji, urobiť niektoré zmeny v editore pohybov tímu Agenty 007.

5.1 Implementácia testovacieho frameworku

Testovací framework je aplikácia, ktorá slúži na posielanie príkazov agentom a ich plnenie vyhodnocuje na základe údajov získaných zo servera. Nakoľko testovanie má obsiahnuť viaceré udalosti na ihrisku, bolo nutné presne špecifikovať požiadavky na možnosti testovacieho frameworku.

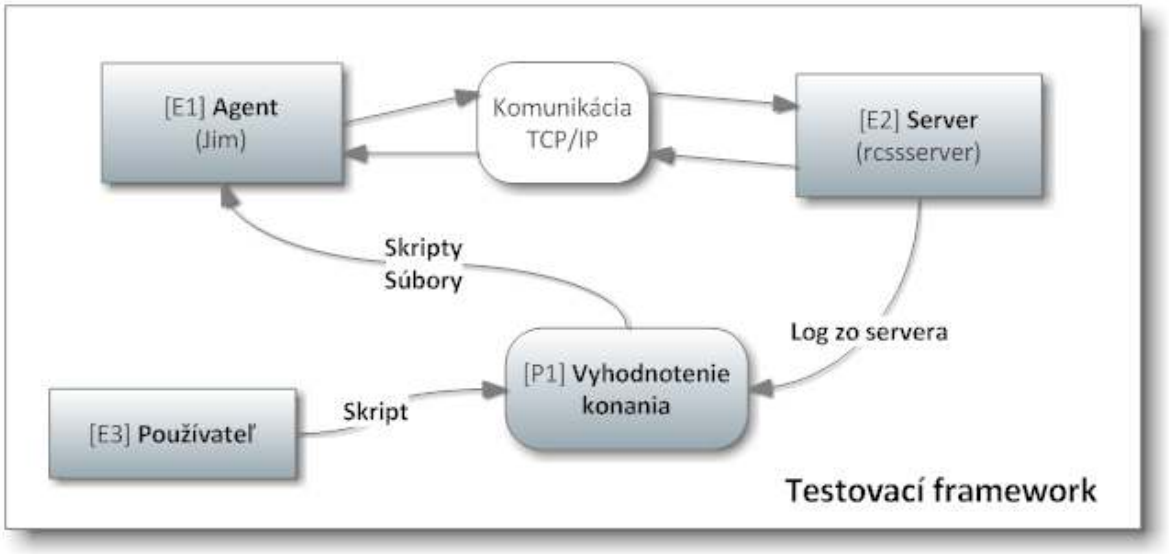
Testovací framework musí umožňovať:

- Testovanie nižších aj vyšších schopností hráča
- Konfiguráciu udalostí na ihrisku
- Testovanie viacerých hráčov a ich kooperácie
- Umožniť optimalizáciu skrze hľadanie lepších parametrov

Aplikácia musí umožňovať komunikáciu s agentom aj serverom. Vychádzajúc zo špecifikácie sa dospelo k nasledovnému zaradeniu frameworku do kontextu agenta a servera, ktoré dobre vystihuje diagram toku dát na *Obr. 27*.

[E1] Agent a [E2] Server

Medzi agentom a serverom prebieha štandardná komunikácia, ktorá je špecifická pre virtuálneho hráča robotického futbalu. Pri implementácii riešenia bolo agenta nutné upraviť do stavu, aby prijímal príkazy z vonku. Nakoľko testovací framework má slúžiť na testovanie pomerne širokého spektra aktivít, bolo nutné pre tento účel nájsť spôsob, ako presne definovať, čo má agent robiť, a akým spôsobom to bude možné vyhodnocovať. Po zvážení viacerých možností tím dospel k použitiu skriptovacieho jazyka Ruby s tým, že agent bude prijímať skripty v stanovenom tvare a spúšťať ich na svojej strane.



Obr. 27: *Diagram toku dát. Zaradenie Testovacieho frameworku v kontexte agentov a servera.*

Zo špecifikácie požiadaviek na testovanie, z netriviálnej tvorby jednotlivých testov a z architektúry JIM-a vyplynula nutnosť prenášať súbory medzi frameworkom a agentom. Nakoľko testovanie má obsiahnuť aj udalosti týkajúce sa viacerých hráčov, ktorí sa môžu nachádzať na rôznych počítačoch, komunikácia je typu klient - server. Konkrétne bol implementovaný open-source TFTP server na strane agenta a TFTP klient na strane frameworku. TFTP server bol navyše upravený tak, aby pri prijatí súboru so špecifickým menom („ruby.exec“) tento súbor neprijímal štandardne, ale aby ho po prijatí vykonal ako Ruby skript.

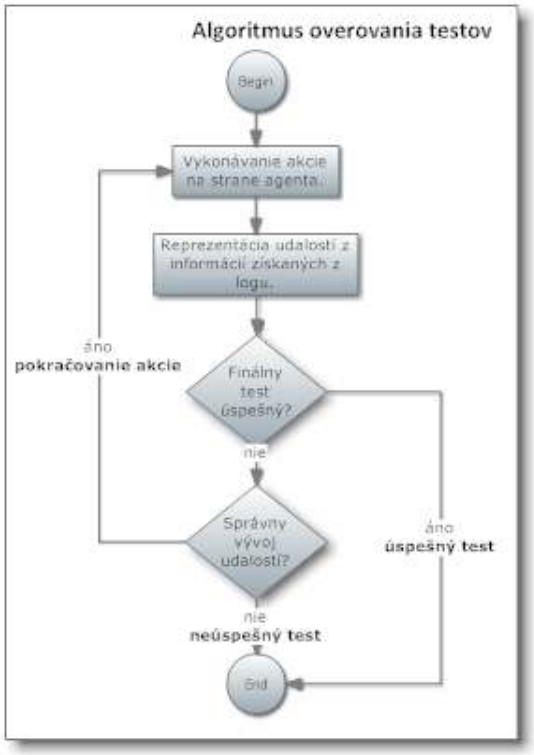
Na strane servera robotického futbalu nebolo potrebné vykonať žiadne zmeny. To je dôležité z dôvodu, že vývoj servera napreduje a zmeny by bolo nutné vykonávať pri každej novej verzii. Na druhú stranu je veľmi dôležité povedať, že server bol na zaradenie do aplikácií typu Testovací framework pripravený.

[E3] Používateľ

Používateľ do testovacieho frameworku zadá skript v jazyku Ruby, ktorý obsahuje:

- čo sa má vykonať
- časovo závislý test, ktorý určuje, či výsledný test môže dopadnúť pozitívne
- takýto typ testu je dôležité do skriptu zaradiť z hľadiska efektívneho využitia času, kedy je zrejmé, že test by v konečnom dôsledku dopadol neúspechom (ako príklad môžeme uviesť nahrávku – ak už má zlý smer, tak je zrejmé, že nahrávka nepríde do požadovanej pozície)
- test, ktorý hovorí, či výsledok akcie bol úspešný

Pre názornejšiu predstavu algoritmus overovania úspešnosti testu je znázornený na *Obr. 28*.



Obr. 28: *Algoritmus overovania testov.*

[P1] Vyhodnotenie konania

Blok spracúva vstupy od používateľa [C], a servera [B] a následne odosiela špecifickým agentom [A] súbory (napr. XML súbory, či Ruby skripty so zmenenými parametrami) a skript v Ruby, ktorý si vyžiada okamžité vykonanie na strane agenta (napr. preplánovanie budúcich akcií). Je dôležité poznamenať, že potenciál na zmenu súborov a teda aj správania umožňuje zaviesť do frameworku princípy umelej inteligencie.

5.1.1 Ukážka používateľom zadaného skriptu

Nasledujúca ukážka reprezentuje približnú formu skriptu zadaného používateľom. V uvedenom príklade ide konkrétne o testovanie prihrávky.

```
def reset
  eps = epsilon.new(2, 2)
  ball.position = position.new(100, 102, eps)
  agents[0].position = position.new(100, 100, eps)
  agents[1].position = position.new(150, 100, eps)
  initial_distance = distance(ball, agents[1])
end
```

```
def is_test_passed
  if (is_near(ball, agents[1]) && is_slow(ball))
    return true, time_passed, distance(ball, agents[1])
  else
    return false
  end
end

def is_situation_still_ok
  if (is_too_far(ball, agents[1], initial_distance) && (is_slow(ball))
    return false
  elsif (time_passed > 20)
    return false
  else
    return true
end

def action
  ball_distance = 0.3
  angle = Math.Pi / 6
  agents[0].plan << move(ball, ball_distance, angle)
  agents[0].plan << pass(agents[1])
end
```

Pre potreby zavedenia umelej inteligencie bude navyše potrebné v letnom semestri vymyslieť rozumný systém označovania parametrov v skripte a ich potenciálny rozsah. Počas optimalizácie metódami umelej inteligencie budeme následne strojovo hľadať optimálne nastavenie týchto parametrov.

5.1.2 Logovanie servera

Server v prípade potreby môže logovať priebeh simulácie. Takýto log pozostáva z S-výrazov, ktoré sú buď obyčajné reťazce alebo zoznam ďalších S-výrazov. Logovanie celej simulácie prebieha nasledovne:

- Zápis informácií o prostredí
- Zápis informácií o stave hry
- Periodický zápis čiastočných informácií o stave hry

Informácia o prostredí má nasledujúcu štruktúru:

((<EnvironmentInformation>)(<SceneGraphHeader>)(<SceneGraph>))

Prvá časť informácie *(EnvironmentInformation)* môže vyzerať nasledovne:

```
((FieldLength 18)(FieldWidth 12)(FieldHeight 40)
(GoalWidth 2.1)(GoalDepth 0.6)(GoalHeight 0.8)
(FreeKickDistance 1.3)(WaitBeforeKickOff 2)
(AgentRadius 0.4)(BallRadius 0.042)(BallMass 0.026)
(RuleGoalPauseTime 3)(RuleKickInPauseTime 1)(RuleHalfTime 300)
(play_modes BeforeKickOff KickOff_Left KickOff_Right PlayOn
KickIn_Left KickIn_Right corner_kick_left corner_kick_right
goal_kick_left goal_kick_right offside_left offside_right
GameOver Goal_Left Goal_Right free_kick_left free_kick_right)
)
```

Prvá časť S-výrazu opisuje názov informácie, ktorá sa nachádza za týmto názvom[12].

Informácia o stave hry má podobnú štruktúru ako tá o prostredí a vyzerá nasledovne:

((<GameState>)(<SceneGraphHeader>)(<SceneGraph>))

Príkladom prvej časti je nasledujúci výraz:

```
((time 0)(half 1)(score_left 0)(score_right 0)(play_mode 0))
```

V tomto prípade nie sú všetky časti S-výrazu povinné. Povinnou časťou je iba informácia o čase.

Hlavička grafu scény obsahuje informácie o kompletnosti grafu scény a jej verzie. Jej štruktúra vyzerá nasledovne:

(Name Version Subversion)

Name môže nadobúdať v súčasnosti 2 hodnoty:

- RSG – indikuje, že graf scény obsahuje plný opis prostredia
- RDS – graf scény obsahuje iba čiastočný opis prostredia, t.j. obsahuje iba informácie o zmenách

[12] http://simspark.sourceforge.net/wiki/index.php/Network_Protocol

Version je hlavná verzia grafu scény (v súčasnosti vždy 0). **Subversion** je vedľajšia verzia grafu scény (v súčasnosti vždy 1). Príklad tejto hlavičky grafu scény vyzerá nasledovne:

```
(RSG 0 1)
```

Graf scény (SceneGraph) predstavuje logickú a priestorovú reprezentáciu scény. Je štruktúrovaný do stromu s koreňom na pozícií <0,0,0> bez rotácie. Každý vrchol v strome obsahuje jedného alebo viac nasledovníkov. Pozícia a rotácia každého nasledovníka je násobkom všetkých transformačných matíc z koreňa až k nasledovníkovi. Každý uzol, ktorý nie je zároveň aj listom je označený skratkou *nd*.

V prípade, že ide o záznam, v ktorom sú len zmeny v grafe scény (RDS), log obsahuje veľa prázdnych uzlov až na uzly, ktoré sa zmenili. Existuje viacero typov uzlov:

- **Základný uzol** (Base node), ktorý má nasledujúcu formu (v logu však takýto uzol nebol nájdený):

  (nd BN <contents>)

- **Transformačný uzol** (Transform node), ktorý reprezentuje 4x4 transformačnú maticu[13], ktorá sa používa na reprezentáciu geometrických transformácií. Mapuje z jednej súradnicovej sústavy do druhej. Jej forma je nasledovná s príslušným S-výrazom:

$$\begin{bmatrix} nx & ox & ax & Px \\ ny & oy & ay & Py \\ nz & oz & az & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  (nd TRF (SLT nx ny nz 0 ox oy oz 0 ax ay az 0 Px Py Pz 1))

- **Geometrický uzol** (Geometric node), ktorý opisuje tvar objektu. Tento uzol je vždy listom v strome, neobsahuje nasledovníkov. Môže mať viacero typov:
  - *Statická sieť* (Static mesh), ktorá je načítaná zo súboru a jej štruktúra je nasledovná:

(nd StaticMesh (load <model>) (sSc <x> <y> <z>) (setVisible 1) (setTransparent) (resetMaterials <material-list>))

[13] http://en.wikipedia.org/wiki/Transformation_matrix

Dôležitým parametrom je sSc, ktorý určuje rozsah objektu. Príklad na takýto uzol je nasledovný:

```
(nd StaticMesh (load models/rlowerarm.obj) (sSc 0.05 0.05 0.05)(resetMaterials matLeft
```

- *SMN,* čo je akási preddefinovaná sieť (momentálne iba krabica, valec a guľa) s nasledujúcou štruktúrou:

(nd SMN (load <type> <params>) (sSc <x> <y> <z>) (setVisible 1) (setTransparent) (sMat <material-name>))

Príklad na takýto uzol je nasledovný:

```
(nd SMN (load StdUnitBox) (sSc 1 31 1) (sMat matGrey))
```

- **Svetelný uzol**, ktorý v našom prípade nie je podstatný.

5.1.3 Parsovanie správ zo servera

Monitorovacia časť riešenia stará sa o prijímanie informácií o priebehu simulácie zo servera. Tieto informácie server poskytuje na porte 3200 pre externé monitory vo forme S-výrazov, ktoré sú opísané v časti analýzy.

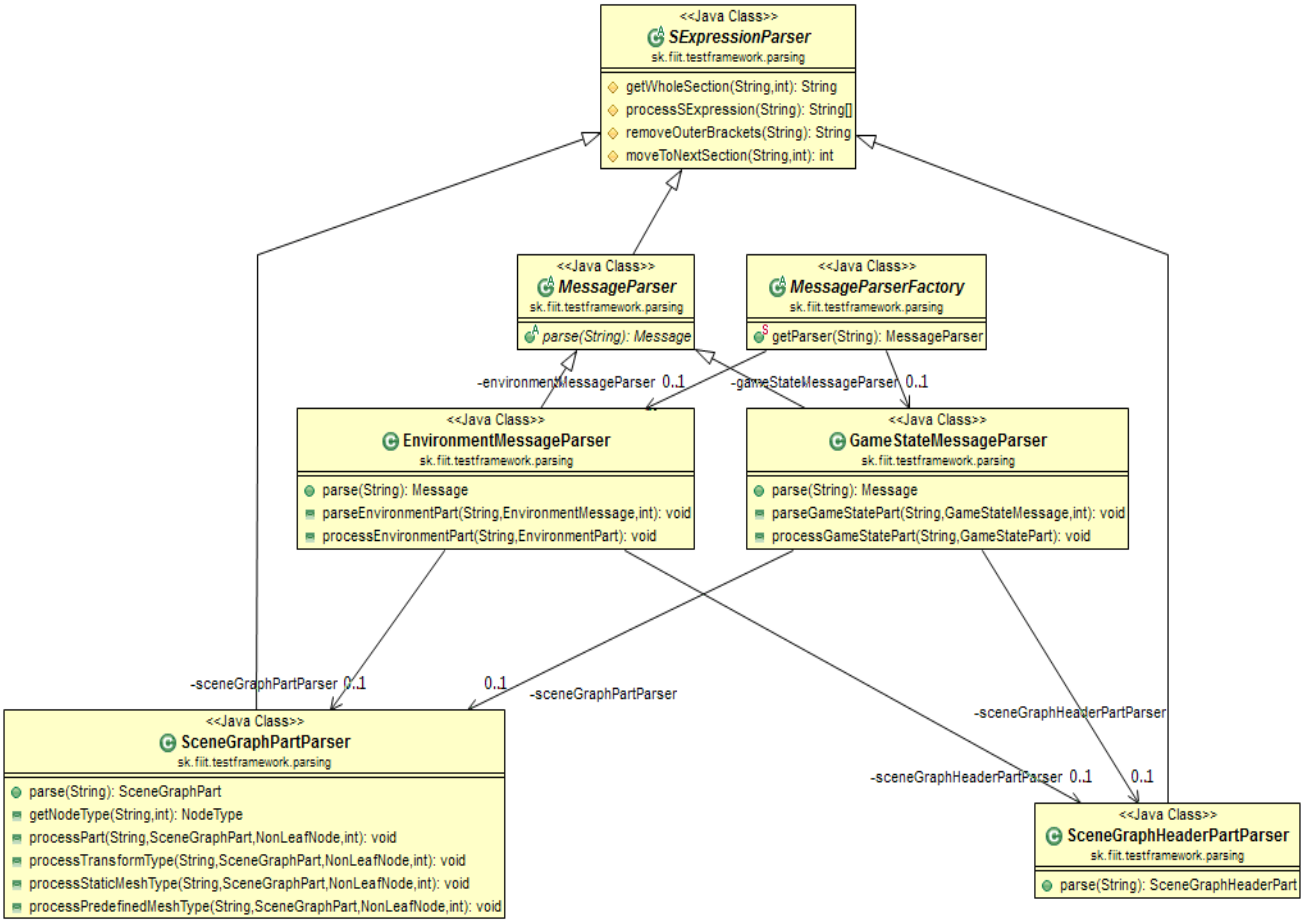
Parsovacia časť riešenia má za úlohu spracovávať informácie o priebehu simulácie poskytované serverom a vytvoriť z nich objektový model, ktorý bude následne spracovaný časťou zodpovednou za reprezentáciu simulácie. Skladá sa zo 4 balíčkov:

1. *sk.fiit.testframework.parsing* – obsahuje triedy zodpovedné za samotné parsovanie informácií
2. *sk.fiit.testframework.parsing.models* – obsahuje triedy, ktoré tvoria objektový model informácií
3. *sk.fiit.testframework.parsing.models.messages* – obsahuje triedy, ktoré reprezentujú prijaté správy zo servera
4. *sk.fiit.testframework.parsing.models.nodes* – obsahuje triedy, ktoré reprezentujú uzly v grafe scény

Najdôležitejšími triedami tejto časti sú triedy v balíčku *sk.fiit.testframework.parsing*, kde sa nachádzajú jednotlivé parsery. Triedy v tomto balíčku sú opísané v *tabuľke 4*.

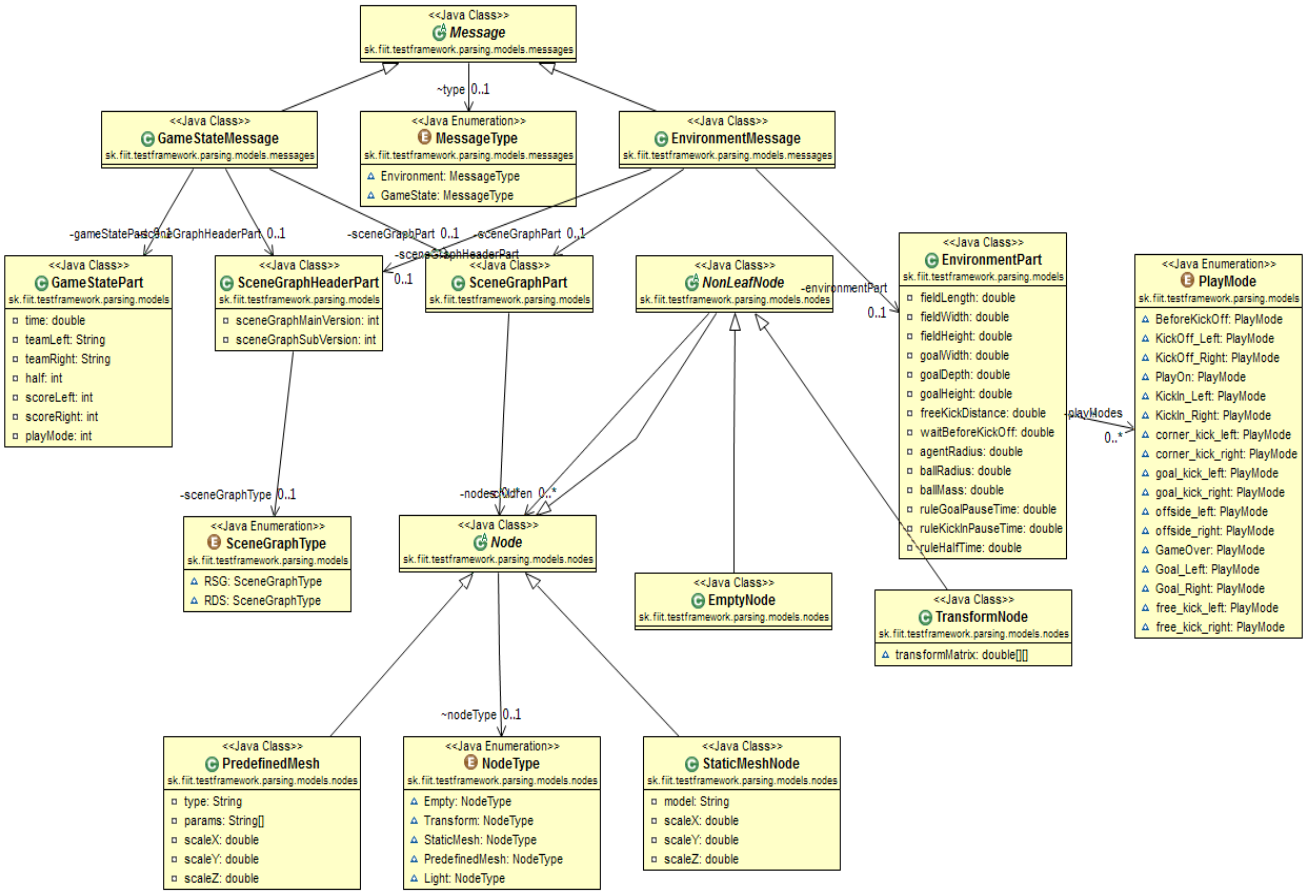
Tabuľka 4: *Triedy balíka sk.fiit.testframework.parsing*

Názov	Opis
SExpressionParser	Abstraktná trieda, ktorá obsahuje základné metódy pre prácu s S-výrazmi a dedia od nej všetky ostatné parsery.
MessageParser	Abstraktná trieda, od ktorej dedia iba hlavné parsery správ.
MessageParserFactory	Na základe prijatej správy sa rozhoduje, aký parser použije.
EnvironmentMessageParser	Hlavný parser pre parsovanie informácií o prostredí.
GameStateMessageParser	Hlavný parser pre parsovanie informácií o stave hry.
SceneGraphPartParser	Používajú ho oba hlavné parsery pre parsovanie časti s grafom scény.
SceneGraphHeaderPartParser	Používajú ho oba hlavné parsery pre parsovanie časti s hlavičkou grafu scény.



Obr. 29: *Diagram tried balíka sk.fiit.testframework.parsing*

Na *Obr. 29* sú znázornené triedy tohto balíčka. Triedy na obrázku generujú objektovú reprezentáciu informácií posielaných zo servera. Táto objektová reprezentácia informácií zo servera sa nachádza v ostatných 3 balíčkoch a triedy v nich sú zobrazené v diagrame tried na *Obr. 30*.

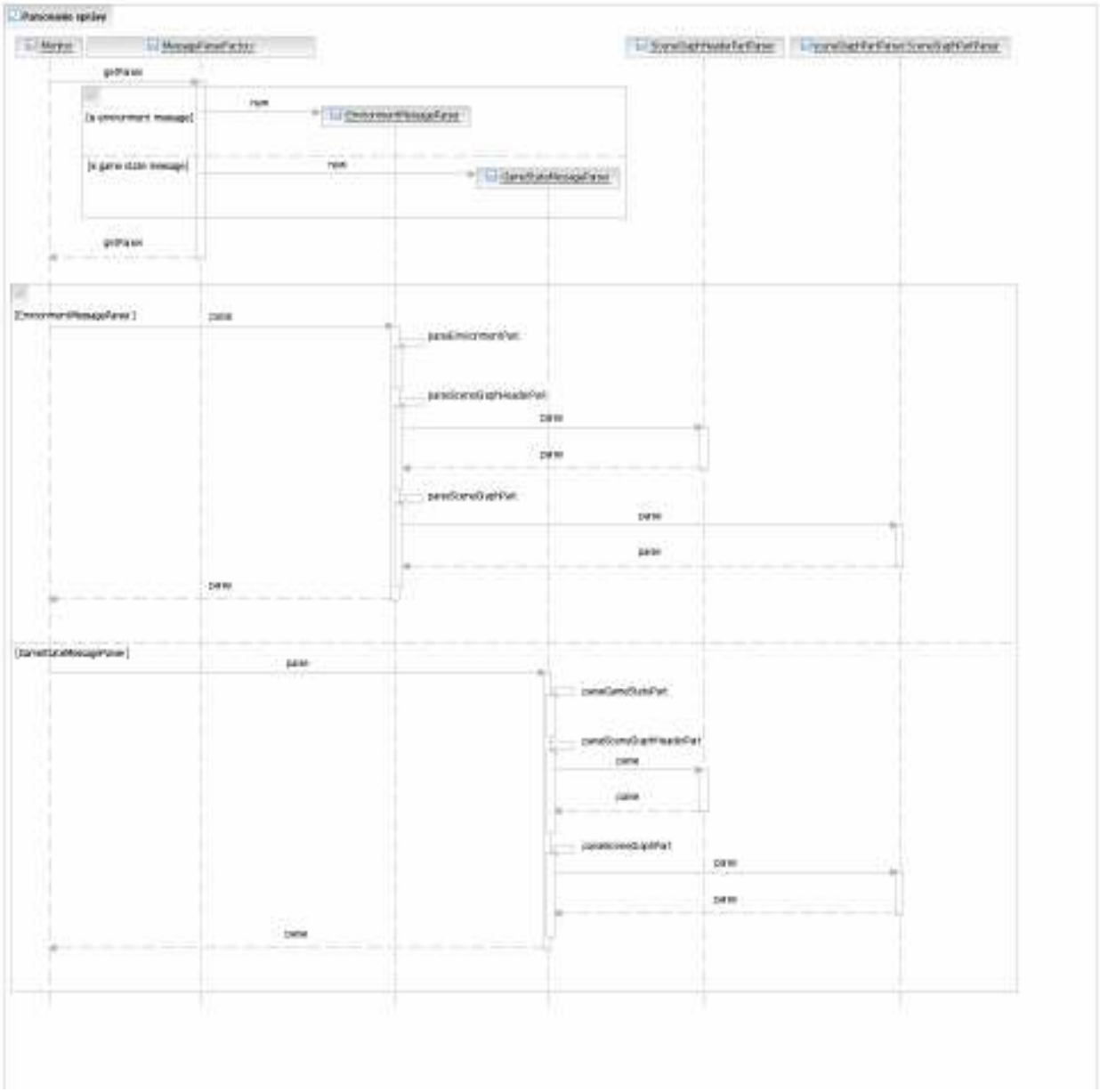


Obr. 30: *Diagram tried balíkov parsera*

Na *Obr. 30* je možné vidieť 2 základné typy správ:

1. *GameStateMessage* – správa prichádzajúca periodicky s informáciami o stave hry. Skladá sa z 3 častí: stav hry (*GameStatePart*), hlavička grafu scény (*SceneGraphHeaderPart*) a graf scény (*SceneGraphPart*).
2. *EnvironmentMessage* – správa, ktorá príde hneď po pripojení sa na server s informáciami o prostredí. Rovnako sa skladá z 3 častí: informácie o prostredí (*EnvironmentPart*) , hlavička grafu scény (*SceneGraphHeaderPart*) a graf scény (*SceneGraphPart*).

Spôsob, akým spolupracujú triedy v balíčku *sk.fiit.testframework.parsing*, je zachytený v sekvenčnom diagrame na *Obr. 31*.



Obr. 31: *Sekvenčný diagram spolupráce tried v balíku sk.fiit.testframework.parsing*

5.2 Pohyby agenta

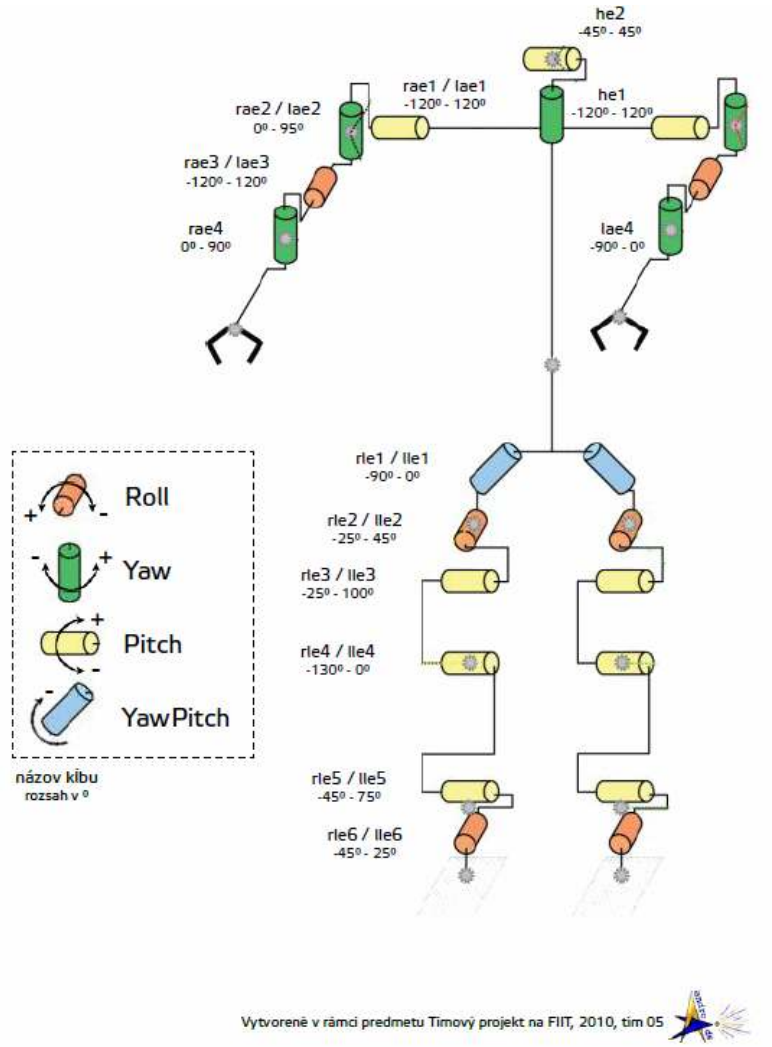
V tejto časti sa zaoberáme pohybmi agenta JIM. Rozhodli sme sa niektoré pohyby prevziať a upraviť a niektoré vytvoriť celkom nové.

5.2.1 Pomôcka pri vytváraní pohybov

Potreba vytvoriť prehľad anatómie agenta JIM vyvstala pri vývoji pohybov priamym vytváraním XML súborov. Pri takomto postupe je potrebná nejaká vizualizácia (aspoň kým autor nemá názvy kĺbov a ich možnosti celkom zautomatizované).

Cieľom bolo vytvoriť pomôcku, ktorá v začiatkoch tvorby pohybov umožní prekonať počiatočné ťažkosti pri identifikovaní kĺbov a zisťovaní ich možností. Naša pomôcka teda obsahuje tri základné informácie o každom kĺbe, ktoré sú potrebné pri vývoji pohybu:

- Názov kĺbu
- Orientácia otáčania (kladná / záporná, naznačená šípkami)
- Rozsah v stupňoch



Obr. 32: *Anatómia robota NAO – pomôcka pri vývoji pohybov*

Schéma agenta pochádza zo stránok venovaných serveru Simspark[14]. V letnom semestri môže k tejto pomôcke pribudnúť stručný návod s tipmi, ako efektívne začať s vývojom pohybov. Schéma je znázornená na *Obr. 32*.

5.2.2 Základné pohyby

Nakoniec sme v prototype neimplementovali všetky pohyby, ktoré sme uviedli v návrhu prototypu. Je to spôsobené hlavne tým, že sa vynorili viaceré komplikácie pri tvorbe pohybov, čo spôsobilo, že náš odhad náročnosti sa ukázal byť dosť nepresný. Napriek tomu sme implementovali veľkú časť plánovaných pohybov na veľmi dobrej alebo dobrej úrovni. Niektoré z nich sú v štádiu „rozpracovaný“, čo znamená, že sa na nich ešte bude s najvyššou pravdepodobnosťou pracovať.

5.2.2.1 Vstávanie

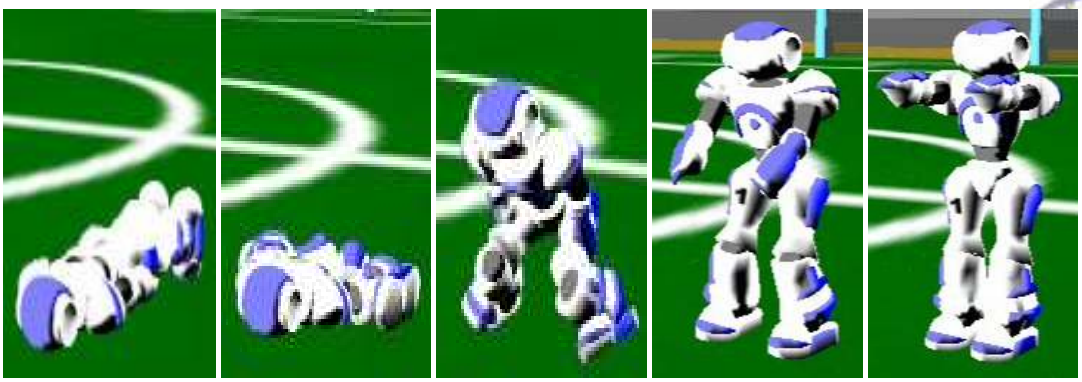
Vytvorili sme oba plánované spôsoby vstávania, vstávanie z chrbta aj z brucha, pričom sme ako základ využili vstávanie oboma spôsobmi od autorov JIM-a, ktoré sme upravili do výslednej podoby. Navyše sme implementovali aj pomocné pohyby pád vpred a vzad, ktoré sme používali pri testovaní vstávania.

Vstávanie z brucha

Hlavnou úpravou pôvodného zdedeného pohybu bolo odstránenie dlhých čakacích fáz a niekoľkých zbytočných pohybov kĺbov, ktoré už pred vykonaním zmeny boli v koncovej polohe vďaka niektorej skoršej fáze pohybu. Ďalšou významnou zmenou bolo dokončenie záveru pohybu tak, aby hráč po jeho vykonaní ostal stáť v základnom postoji, do ktorého ho nastaví server po nabeamovaní, keďže pôvodný pohyb končil pádom hráča späť na zem. Pohyb prebieha v nasledovných krokoch, ktoré znázorňuje *Obr. 33*.

- najskôr hráč uvedie všetky kĺby do neutrálnej polohy, okrem ramenných, ktoré pripaží
- následne rozkročí a úplne skrčí nohy
- v ďalšej fáze postupne prinožuje a vystiera nohy, čím zároveň vstáva
- v koncovej fáze vystierania hráč predpaží ruky, ktoré až do tejto fázy boli pripažené

[14] http://simspark.sourceforge.net/wiki/index.php/Image:Models_NaoAnatomy.png



Obr. 33: *Vstávanie z brucha*

Overenie

Pohyb dosiahol 100% úspešnosť vo všetkých 30 pokusoch. Pre počiatočnú polohu na bruchu trvá 3100 ms a pre polohu na chrbte je nepoužiteľný. Trvanie je pomerne dlhé, preto sa v letnom semestri pokúsime tento pohyb zrýchliť, avšak požiadavka stability bola splnená, takže pohyb je použiteľný aj vo svojom momentálnom stave.

Vstávanie z chrbta

U tohto pohybu boli vykonané rovnaké úpravy ako u vstávania z brucha a aj priebeh pohybu je takmer totožný s výnimkou začiatku. Na *Obr. 34* zachytáva časť pohybu, ktorou sa vstávanie z chrbta líši od vstávania z brucha. Fázy pohybu sú nasledovné:

- vystretie tela a pripaženie rúk
- rýchle predpaženie a čiastočné rozkročenie, vďaka čomu sa hráč čiastočne posadí
- dokončenie rozkročovania a pokrčenia nôh, a zároveň opätovné pripaženie, čím sa hráč prevráti na brucho a skončí v polohe, ktorú opisuje druhá odrážka v opise vstávania z brucha
- následne pohyb prebieha ako vstávanie z brucha

Obr. 34*: Odlišný začiatok vstávania z chrbta oproti vstávaniu z brucha*

Overenie

Aj toto vstávanie bolo 100% úspešné vo všetkých 30 testovacích pokusoch a navyše je použiteľné aj pre vstávanie z brucha, pričom pre túto počiatočnú polohu bolo tiež testované 30-krát so 100% úspešnosťou. Jeho nevýhodou je podstatne dlhšie trvanie, keďže celý pohyb trvá 4320 ms, avšak sme si istí, že sa nám ho ďalšími úpravami podarí výrazne skrátiť tak, aby bolo výhodné tento pohyb uprednostniť pre vstávanie z brucha, a tým odstrániť jeden rozhodovací krok, a síce ktoré zo vstávaní použiť.

5.2.2.2 Pád vpred a vzad

Tieto pomocné pohyby sú založené na jednoduchej rýchlej zmene uhla kĺbu členku, ktorý je zodpovedný napnutie a pritiahnutie nártu. Boli vyvinuté čisto pre potreby testovania vstávania a neplánujeme ich využiť v samotnej hre, keďže pády hráčov počas hry nie sú žiaduce. Výnimkou je brankár, ale ten má sadu vlastných špeciálnych pohybov.

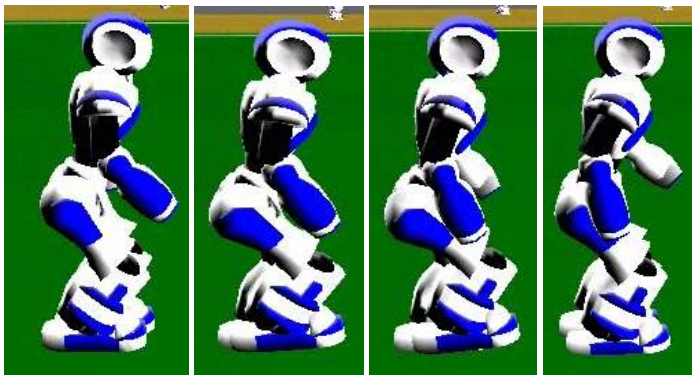
5.2.2.3 Chôdza

Z pôvodného návrhu prototypu sme implementovali veľmi drobnú, pomalú chôdzu dopredu a dozadu, pričom obe chôdze vznikli úpravou spomínaného prototypu chôdze, ktorý sme prevzali spolu s agentom JIM. Ďalej sme úpravou zdedeného prototypu chôdze vytvorili aj našu verziu rýchlej chôdze dopredu, ktorá je založená na drobčení. Z pôvodného návrhu sme neimplementovali chôdzu do strán (úkroky) a ani humanoidné verzie chôdze dopredu a dozadu.

Drobná pomalá chôdza dopredu

Oproti zdedenej chôdzi je tento typ chôdze vzpriamenejší a podstatne drobnejší. Dĺžka kroku je približne pätina dĺžky hráčovho chodidla. Táto chôdza je veľmi pomalá a jej účelom je predovšetkým presné priblíženie k lopte bez jej odkopnutia. Pri tejto aj ostatných našich chôdzach hráč výrazne pracuje s rukami, keďže ruky sú dôležitým prvkom v udržiavaní rovnováhy a prenášaní ťažiska. Na začiatku kráčania robot pokrčí ruky aj nohy, aby zvýšil svoju stabilitu. Krok pravou nohou je zachytený na *Obr. 35*. Jeden krok následne pozostáva z dvoch fáz:

- hráč prenesie váhu na opornú nohu a druhú nohu zdvihne a mierne vystrie v kolene, čím ju predsunie pred opornú nohu, zároveň rukou na strane opornej nohy pohne v ramene dopredu
- dostúpi na predsunutú nohu a prenesie na ňu váhu kvôli ďalšiemu kroku



Obr. 35: *Drobný krok pravou nohou vpred*

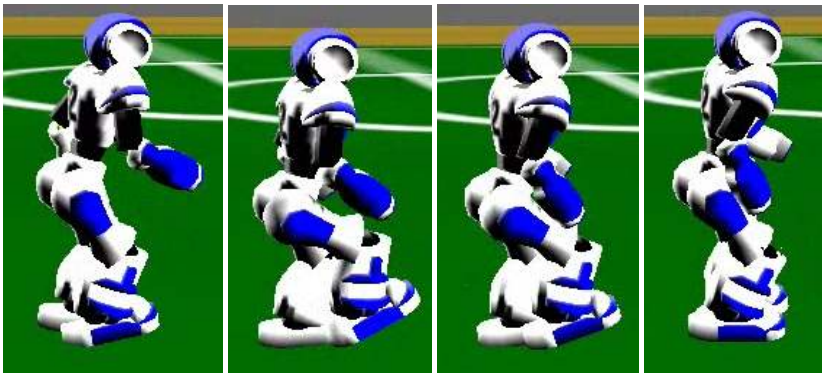
Overenie

Chôdza je stabilná a hráč nespadol ani pri výrazne viditeľných chybách zavedených serverom počas 10 testovacích kôl, z ktorých každé trvalo 3 minúty. Jeden krok trvá 300 ms, inicializačná fáza 700 ms. Priamočiarosť chôdze nie je možné v prípade staticky definovaných pohybov kvôli chybám od servera ovplyvniť, ale táto chôdza približne 90% testovacieho času svoj smer výrazne nemenila.

Drobná pomalá chôdza dozadu

Táto chôdza vznikla náhodne jednou z mnohých úprav zdedenej chôdze dopredu v snahe získať stabilnejšiu rýchlu chôdzu vpred. Tento fakt spôsobuje, že pri kráčaní dopredu hráč výrazne predkopáva nohy a mierne poskakuje. Práca rúk je rovnaká ako v predošlom opísanom spôsobe chôdze. Dĺžka kroku je približne tretina chodidla. Krok chôdze dozadu znázorňuje *Obr. 36* a pozostáva z nasledovných úkonov:

- mierne odrazenie sa z päty pri predkopnutí nohy, lebo päta sa jemne zachytí o zem
- mierne zanoženie pri vrátení predkopnutej nohy späť na zem



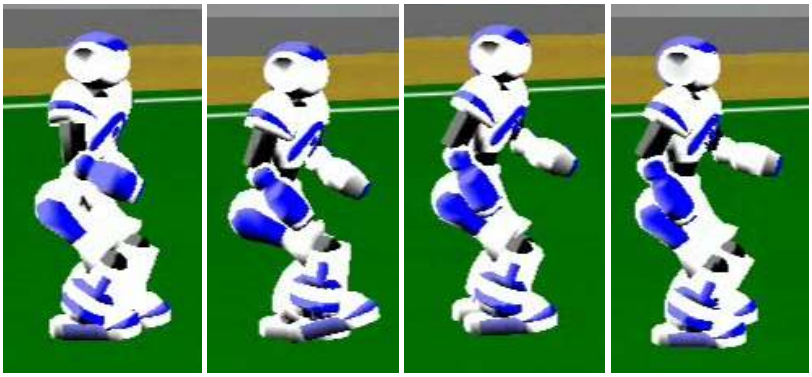
Obr. 36: *Drobný krok pravou nohou vzad*

Overenie

Aj táto chôdza vykazuje výraznú stabilitu a pri testovaní v 10 kolách po 3 minúty hráč ani raz nespadol. Krok trvá 300 ms, inicializácia 700 ms. Vzhľadom na svoju povahu je táto chôdza výraznejšie ovplyvnená chybami od servera a výraznejšie mení smer.

Drobčivá rýchla chôdza dopredu

Podobá sa drobnej chôdzi dozadu, hráč pri nej mierne nadskakuje. Rozdielom voči zdedenému prototypu je hlavne menší predklon hráča, vďaka čomu sa nestáva, že by padol na nos. Táto chôdza má rovnaké fázy ako predošlé dve chôdze, odlišuje sa len v hodnotách otočení jednotlivých kĺbov a je pri nej výraznejšie prenášanie váhy na opornú nohu pri kroku. Je určená hlavne na prekonávanie väčších vzdialeností. Jeden krok je zachytený na *Obr. 37*.



Obr. 37: *Krok vpred pravou nohou*

Overenie

Naša rýchla chôdza dopredu je skoro rovnako stabilná ako ostatné dva typy chôdze, ktoré sme v prototype implementovali. Počas rovnakých testov hráč ani raz nespadol a pád sa vyskytol až pri nahrávaní videa tohto pohybu, ktoré vzhľadom na technické parametre použitého laptopu výrazne ovplyvňovalo kvalitu simulácie. Krok trvá 300 ms a inicializácia 700 ms. V približne 80% pokusov sa hráčovi napriek chybám zo strany servera podarilo udržať relatívne priamy smer a prešiel ihrisko po dĺžke.

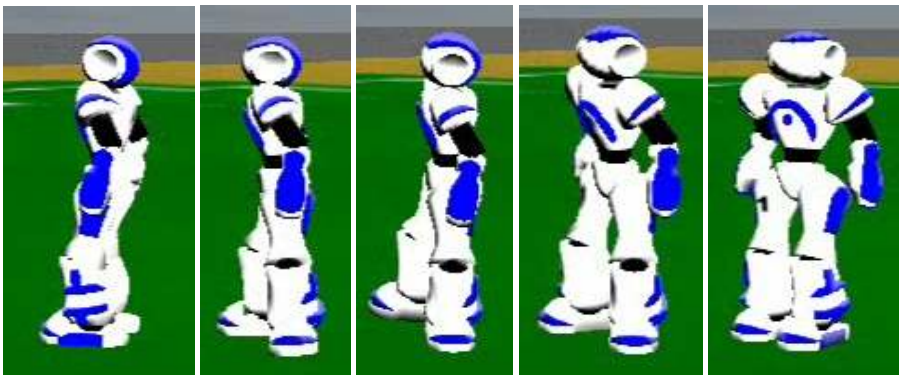
5.2.2.4 Otáčanie

Z navrhovaných spôsobov otáčania sme implementovali len otočenie o 90° do oboch smerov, ktoré je v stave „rozpracovaný“, a testovali otáčania zdedené od predchodcov. Otáčania od predchodcov nám nevyhovovali, keďže hráč pri nich bol výrazne naväzený dozadu a pri väčších chybách od servera často padal dozadu. Navyše tieto otáčania hráč nevykonával na mieste, ale opisoval pri nich výrazný kruh, čo je nežiadúce. Napriek tomu nezavrhujeme možnosť, že s nimi budeme v druhom semestri pracovať, keďže sú pomerne rýchle.

Otočenie o 90°

Toto otočenie okrem fázy inicializácie pracuje len s kĺbmi nôh. Zatiaľ nie je dokončené a vyžaduje si ešte prácu na správnom prenášaní váhy v počiatočnej fáze. Je implementované do oboch smerov a založené na humanoidnom spôsobe otáčania. Priebeh jedného cyklu otočenia je znázornený na *Obr. 38*. Prebieha v nasledovných krokoch:

- mierne pokrčenie nôh a rúk v inicializačnej fáze pre zvýšenie stability priblížením ťažiska k zemi
- prenesenie váhy na opornú nohu a otočenie druhej nohy v bedrovom kĺbe úplne na stranu
- prenesenie váhy na otočenú nohu a prinoženie pôvodnej opornej nohy
- uvedenie pohybovaných kĺbov do pozície, v ktorej boli na konci inicializačnej fázy pred ďalším pokračovaním



Obr. 38: *Jeden cyklus otočenia doľava*

Overenie

Pohyb je stabilný, keďže ani počas 10 minút opakovaného vykonávania cyklu otočenia hráč nejavil žiadne známky náchylnosti na pád. Jeden cyklus trvá 4000 ms a inicializačná fáza, ktorá sa vykoná len na začiatku, 500 ms. Pri úspešne vykonaných cykloch sa hráč takmer nepohne z miesta, čo je výrazné zlepšenie v porovnaní so zdedeným otáčaním.

Problémom je nedeterminizmus cyklu, keďže kvôli nedoladenému prenosu váhy v počiatočných fázach hráč približne v 50% vykonaných cyklov nezmení svoje natočenie. Tento problém sa pokúsime odstrániť počas ďalších prác na pohyboch. Ďalším nedostatkom je fakt, že ani pri úspešnom vykonaní cyklu nie je výsledný uhol otočenia 90°, ale len približne 70°. Toto je pravdepodobne tiež spôsobené chybou v prenose váhy počas cyklu.

Napriek uvedeným nedostatkom je pohyb stabilný a takmer nemení polohu hráča na ihrisku, čo bolo naším hlavným cieľom pri tvorbe otáčania.

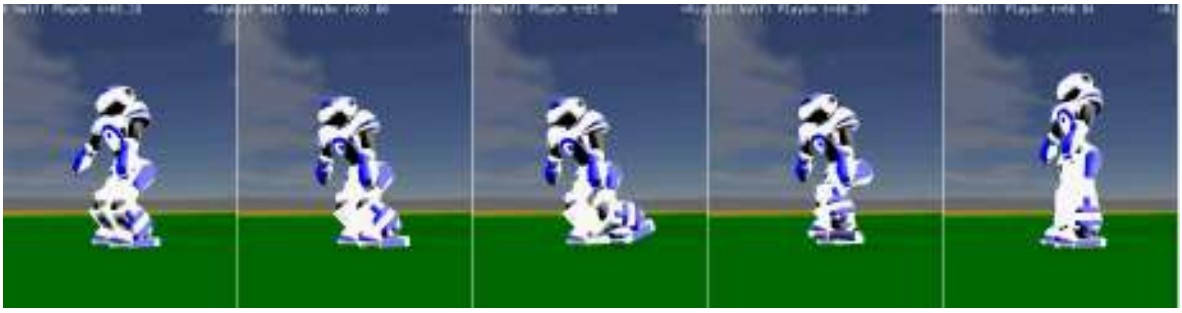
5.2.2.5 Kop do lopty

Do prototypu sme naimplementovali dva typy kopu:

- Priamy kop do lopty
- Kop do lopty hranou chodidla

Priamy kop do lopty

Priamy kop je jeden z viacerých druhov pohybov, ktoré sme sa rozhodli pre hráča JIM vytvoriť. Ide o kop špicou chodidla pri zapojení bedrového, kolenného kĺbu a členku. Ide o kop, ktorý poslúži na ďaleké prihrávky a kopy na bránu. Priebeh kopu je znázornený na *Obr. 39*.



Obr. 39: *Jeden cyklus priameho kopu do lopty*

Pohyb tvorí päť fáz:

- Počiatočná fáza prikrčenia sa dostane robota do polohy, z ktorej môže kop vykonať čo najstabilnejšie. Prikrčenie posunie ťažisko robota nižšie a zvýši tak stabilitu. Do tejto fázy sa zapájajú kĺby bedier, kolien a členkov.
- Druhá fáza vychýlenia tela do strany získa pre agenta priestor na samotný kop a zabezpečí, že po zdvihnutí nohy (opačnej ako je smer naklonenia agenta) sa agent nepreváži na opačnú stranu. V tejto časti pohybu sa okrem kĺbov členkov a bedier zapájajú aj ramenné kĺby ruky.
- Treťou fázou je zodvihnutie nohy. Ide o spoluprácu bedrového, kolenného kĺbu a kĺbu členka s cieľom nezavadiť nohou počas zdvíhania o podložku.
- Štvrtá fáza je samotný výkop so zapojením troch kĺbov – členku, kolena a bedra. Cieľom bolo zapojiť väčší počet kĺbov a dosiahnuť tak sčítanie momentov.
- Posledná fáza je stabilizačná, slúži na návrat agenta do vzpriamenej polohy.

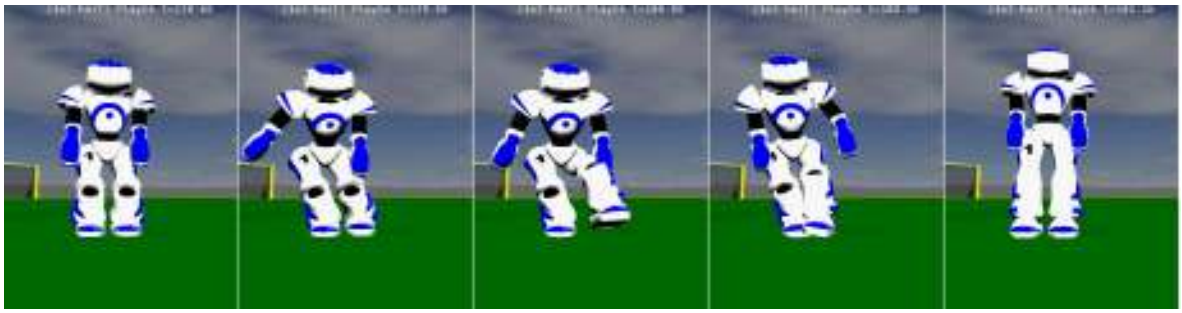
Overenie

Zatiaľ bolo možné overiť iba rýchlosť a stabilitu pohybu, v letnom semestri pristúpime aj k meraniu presnosti kopu.

Z opakovaných 30 pokusov agent ani raz nespadol a takisto ani jedenkrát nezavadil o podložku. Hoci pád po odladení pohybu nenastal, počas testovania rýchlosti pohybu jeden alebo dvakrát došlo k zachyteniu podložky špicou nohy agenta, čo spôsobilo, že sa kop nevydaril – išlo však o náhodnú a ťažko „dosiahnuteľnú“ situáciu. Celý pohyb trvá 1900 ms.

Kop do lopty hranou chodidla

V tomto prípade ide o kop do lopty hranou chodidla agenta. Nejde však o kop dopredu, ale do boku. Tento kop umožní agentovi nahrávku do strany bez nutnosti zdĺhavo sa nastavovať k lopte. Vzhľadom na možnosti a anatómiu agenta nejde o silný kop – na kop možno využiť iba jediný kĺb. Cyklus kopu je znázornený na *Obr. 40*.



Obr. 40: *Jeden cyklus kopu do lopty hranou chodidla*

Tento pohyb má fázy:

- Prvá fáza je založená na prvej fáze priameho kopu – agent sa prikrčí do polosedu, aby tak získal lepšiu stabilitu.
- V druhej fáze sa agent naváži na jednu stranu, aby tak udržal rovnováhu po zodvihnutí opačnej nohy, ktoré nasleduje v ďalšej fáze. Zapája kĺby členkov, bedier, torzo sa mierne nakláňa dopredu kvôli stabilite a rovnováha sa vyvažuje aj rukou.
- V ďalšej fáze agent zapojením kĺbov bedra, kolena a členku vysunie nohu pred telo a zároveň mierne do boku, aby si tak prichystal vhodnú pozíciu na kop.
- Štvrtá fáza je samotný kop, ktorý realizujú bedrové kĺby a zároveň ho vyvažujú ruky (kĺby ramena a lakťa).
- Posledná fáza je stabilizačná, slúži na návrat agenta do vzpriamenej polohy.

Overenie

Takisto ako pri priamom kope, zatiaľ bolo možné overiť iba rýchlosť a stabilitu pohybu, v letnom semestri pristúpime aj k meraniu presnosti kopu. Z tridsiatich pokusov vykonal agent všetky úspešne (bez straty stability a následného pádu). Celý pohyb trvá 2700 ms.

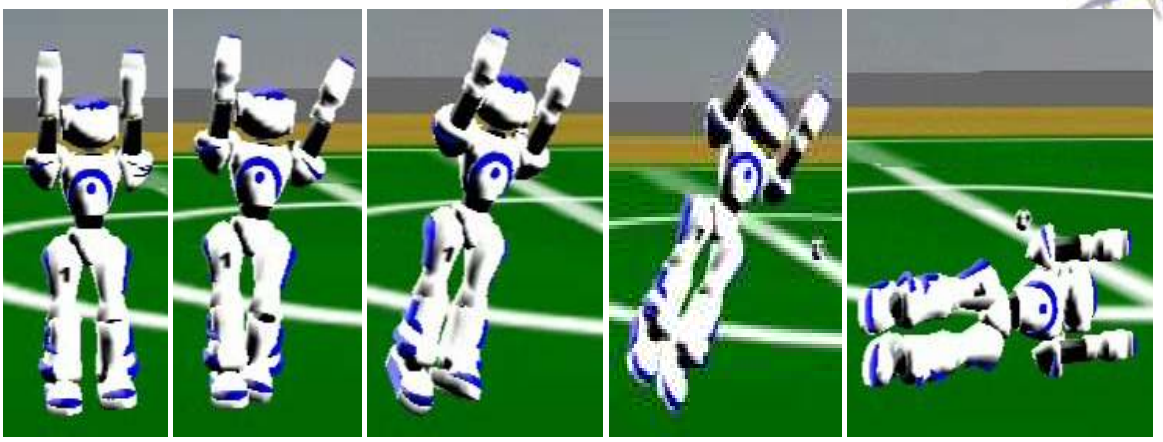
5.2.2.6 Bránenie

V prototype sme implementovali pád brankára na bok do oboch strán. Navyše sme zistili, že budeme potrebovať pomocné pohyby prevrátenia na chrbát pre vstávanie z polohy na boku, lebo použitím vstávania z chrbta by sme museli vstávať na dvakrát, preto sme implementovali aj tieto pomocné pohyby pre obe strany.

Pád brankára na bok

Je to jeden z možných spôsobov zabránenia gólu, ktorý je vhodné použiť, ak brankár nestojí v ceste strele na bránu, ale je v bode, z ktorého pri páde na bok pretne svojím telom trajektóriu strely. V tomto prípade nie je nutné, aby sa brankár zdĺhavo presúval do nového bodu a riskoval, že niektorý z pohybov nevyjde kvôli chybe zavedenej serverom alebo že sa nepostaví presne do dráhy strele. Pohyb je implementovaný pre ľavú aj pravú stranu. Tento pohyb môžeme rozdeliť do nasledovných fáz a ilustruje ho *Obr. 41*:

- v prvej fáze hráč zdvihne ruky šikmo nad hlavu, čo spôsobí, že jeho ťažisko sa presunie vyššie nad zem a bude ľahšie docieliť samotný pád, a zároveň mu to pri dopade pomôže ostať ležať na boku miesto prevrátenia na chrbát alebo na brucho
- v ďalšej fáze pokrčí nohu na strane, na ktorú má spadnúť a zároveň ruku na tej strane vychýli mierne do boku, čím presunie ťažisko na stranu
- následne sa pohybom členku vystretej nohy odrazí dostatočne na to, aby vyvolal samotný pád a miernym napnutím členku na pokrčenej nohe zabezpečí, že bude padať priamo na bok a nepretočí sa počas pádu tvárou k zemi
- počas pádu vychýlenú ruku vráti do polohy rovnobežnej s druhou rukou a napne oba členky, čím ešte kúsok zväčší priestor, ktorý bude svojím telom kryť



***Obr. 41**: Pád brankára vľavo*

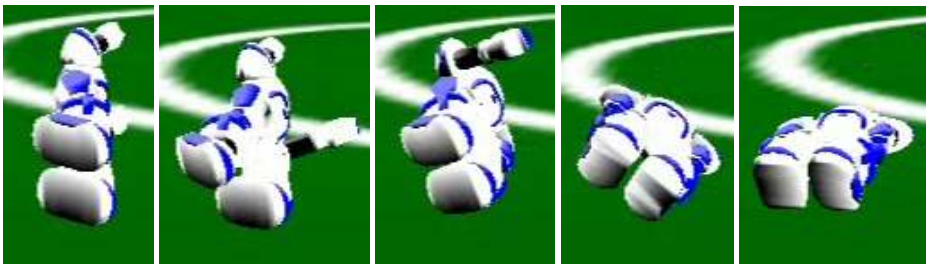
Overenie

Zatiaľ sme netestovali pohyb priamo v akcii, ale pri testoch spoľahlivosti sa ani v jednom z 30 prípadov hráč neprevrátil na chrbát alebo na brucho a vychýlenie z priameho smeru do strany bolo tiež len minimálne. Celý pohyb trvá 2100 ms, pričom dopad na zem nastane približne 400 ms po dokončení pohybu.

Prevrátenie na chrbát

Je to pomocný pohyb pre uvedenie brankára do polohy, z ktorej dokáže vstať po tom, čo použil pohyb *pád na bok*. Po jeho použití ostane ležať na chrbte takmer rovnobežne so smerom, ktorým bol orientovaný v polohe na boku. Priebeh pohybu je zobrazený na *Obr. 42* a pohyb je implementovaný pre použitie z ľavého aj pravého boku. Pohyb pozostáva z dvoch fáz:

- v prvej hráč presunie nezaťaženú ruku a nohu za telo a zaťaženú ruku smerom pred telo, čo spôsobí, že sa prevráti na chrbát
- v druhej fáze pripaží a vráti bedrové kĺby do neutrálneho uhla



***Obr. 42**: Prevrátenie na chrbát po páde na ľavý bok*

Overenie

Pohyb trvá 200 ms a v každom z 30 testovacích pokusov bol úspešný. Dopad na zem nastane približne 200 ms po jeho ukončení.

5.3 Úpravy v editore pohybov

Pre účely ďalšej práce na projekte a vývoji agenta sme potrebovali upraviť niektoré funkcie editora pohybov, ktorý sme prevzali od tímu Agenty 007. Prvou úpravou je XML export pohybov v tvare, ktorý je vhodný pre agenta JIM. Taktiež sme sa rozhodli rozšíriť editor o funkcionalitu symetrických a previazaných kĺbov.

5.3.1 Úprava XML exportu pohybov

Editor pohybov, ktorý možno využiť na vytváranie súborov obsahujúcich definované pohyby, obsahoval možnosť exportu do formátu XML. Žiaľ, exportované XML malo formát určený hráčovi Sirius (tím Critical Error) a nebolo možné exportovať pohyby do XML formátu pre hráča JIM, na ktorého sme sa rozhodli nadviazať.

Pre vývoj pohybov v editore bolo teda nutné implementovať a do editora pridať možnosť exportu XML vo formáte, v akom ho dokáže prečítať náš agent. Toto bolo realizované úpravou pôvodného XML exportu (trieda XmlExporter).

XML export

Celý editor pohybov je napísaný v jazyku C#, preto je na generovanie XML súboru použitá systémová knižnica System.Xml. Na dosiahnutie nového formátu XML exportu boli väčšinou menené volania metód triedy XmlTextWriter v súbore Formats/XmlExporter.cs tak, aby zodpovedali novému formátu. Išlo o metódy WriteStartElement(),WriteEndElement() a WriteAttributeString().

Hráč JIM využíva XML súbory definujúce jednotlivé pohyby s nasledovným formátom:

```
<?xml version="1.0" encoding="UTF-8"?>
<robot xsi:noNamespaceSchemaLocation="moves.xsd">
      <constants>
            <constant name="nazov_konstanty" value="hodnota_konstanty" />

      </constants>

      <low_skills>
            <low_skill name="nazov_pohybu" firstPhase="prva_faza"/>
      </low_skills>

      <phases>
            <phase name="prva_faza" next="dalsia_faza">
                  <effectors>
                        <rle1 end="0" />
```

```
                        <lle3 end="40"/>
                        ...
                  </effectors>
                  <duration>trvanie_fazy</duration>
            </phase>
            <phase name="posledna_faza" isFinal="true">
                  <effectors>
                  ...
                  </effectors>
                  <duration>trvanie_fazy</duration>
                  <finalize>stand_normal</finalize>
            </phase>
      </phases>
</robot>
```

Kompletný formát je zachytený pomocou XSD schémy, ktorá sa nachádza v adresári moves hráča JIM.

Hlavička. Za hlavičku považujeme tag kódovania súboru a otvárací tag <robot>. Tieto boli v pôvodnom XML v inom tvare, preto bol zdrojový kód pozmenený tak, aby generoval novú hlavičku.

Constants. Časť constants obsahuje parametre, ktoré možno využívať namiesto číselných hodnôt. Možno pomocou nich napríklad hromadne meniť polohu kĺbov.

Táto časť sa v pôvodnom XML exporte nenachádzala, nové generované XML obsahuje časť s konštantami pripravenú na ručnú editáciu (keďže ani samotný editor nepodporuje vývoj pohybov pomocou konštánt).

Low_skills. Časť low_skills obsahuje v tagu <low_skill> názov pohybu. Tento musí byť jedinečný a keďže editor nepodporuje pomenúvanie pohybov, rozhodli sme sa odvodiť názov pohybu od názvu ukladaného XML súboru (teda napríklad súbor kick_left.xml obsahuje low_skill s názvom kick_left)

Phases. Medzi tagmi <phases> a </phases> sa nachádzajú jednotlivé fázy pohybu tak, ako boli vytvorené v editore. Každá fáza musí mať svoj jedinečný názov nie len z pohľadu jedného súboru s pohybom, ale aj medzi súbormi navzájom, inak hráč pri načítaní a vykonávaní pohybu jednotlivé fázy navzájom prepisuje a mieša. Problémom bolo, že editor nazýval fázy jedného pohybu automaticky v tvare Phase<číslo> bez možnosti voľby iných názvov. V novom XML exporte sú preto názvy fáz v tvare nazov_pohybu_phase<číslo>, aby sme tak zabezpečili jedinečnosť názvov fáz.

Jednotlivé efektory oproti pôvodnému formátu neobsahujú zatvárací tag a hodnotu počiatočného natočenia kĺbu. Nový tvar efektorov tak obsahuje už iba názov a koncovú hodnotu natočenia príslušného kĺbu.

Trvanie fázy v pôvodnom formáte nefigurovalo (pôvodný formát využíval tzv. speed constant). V interných výpočtoch pôvodného exportu však figurovali hodnoty minEndTime a minStartTime,

ktoré označovali koniec a začiatok fázy – ich odpočítaním sme teda získali trvanie konkrétnej fázy, ktorého hodnotu vkladáme medzi tagy <duration> a </duration>.

V poslednej vygenerovanej fáze nepridávame do tagu <phase> atribút next a naopak pridávame atribút isFinal s hodnotou true. V tejto fáze takisto vkladáme tagy <finalize></finalize>, ktoré slúžia na označenie fázy, ktorou má pohyb skončiť (zväčša ide o fázu stabilizácie robota).

Overenie

Pomocou upraveného XML exportu sme vygenerovali XML súbor, ktorý sme porovnali s ostatnými XML súbormi hráča JIM, aby sme tak zistili odlišnosti. Vygenerovaný súbor zodpovedal XML súborom, ktoré hráč JIM bežne využíva.

Správne generovanie hodnoty duration sme overili porovnaním s časovou osou v editore – trvanie identifikovaných fáz súhlasilo s trvaním pohybov na časovej osi. Nakoniec sme vytvorený pohyb naplánovali hráčovi JIM a ten ho bez problémov vykonal.

5.3.2 Symetria pohybov

U robotov, rovnako ako aj u ľudí, vidno, že existuje niekoľko symetrií. Napríklad ľavý a pravý ramenný kĺb (rae1 a lae1) sú symetrické kĺby. Inšpirovaní tímom Kouretes sme sa pre tento účel rozhodli do editora pohybov implementovať políčko, v ktorom sa bude voliť, či chceme, aby sa symetrické kĺby pohybovali spoločne. Políčko určuje, či pohyby budú:

- Nezávislé – každý kĺb sa pohybuje nezávisle od svojho symetrického spoločníka
- Symetrické – dva symetrické kĺby sa musia pohybovať po symetrických trajektóriách
- Zrkadlové – dva symetrické kĺby sa musia pohybovať po zrkadlovej trajektórii

Je dôležité pochopiť, že keď hovoríme o symetrickom alebo zrkadlovom pohybe, nehovoríme iba o zmene hodnoty z kladnej na zápornú alebo iba o prepísaní uhla symetrického kĺbu. Pre zadefinovanie úspešného pohybu editor jednoducho posunie symetrický kĺb v tom istom (symetrickom) alebo opačnom (zrkadlovom) smere o uhol, o ktorý sa posunie základný kĺb.

Napríklad ak pohybujeme pravým ramenným kĺbom a chceme pohybovať aj k nemu symetrickým kĺbom, teda ľavým ramenným kĺbom, a ten pohyb presahuje dovolenú uhlovú hranicu, ľavý ramenný kĺb sa posunie iba po dovolenú hranicu. Tým sa zabráni zadávaniu nezmyselných hodnôt mimo rozsahu kĺbov do XML súboru.

Keď sa kĺby zviažu výberom v políčku, je jedno, ktorým z kĺbov sa bude pohybovať. Teda nie je zadefinovaný dominantný kĺb. Dominantný kĺb je ten, ktorým sa používateľ rozhodne hýbať v editore.

V prototype sme implementovali symetriu nasledovných kĺbov:

- Ramenné kĺby – rae1 + lae1, rae2 + lae2
- Ručné kĺby – rae3 + lae3, rae4 + lae4
- Nožné kĺby – rle1 + lle1, rle2 + lle2, rle3 + lle3, rle4 + lle4, rle5 + lle5, rle6 + lle6

Všetky úpravy ohľadom symetrie kĺbov boli robené v triedach balíka *MotionEditor*.

Overenie

Overenie upravenej funkcionality editora pohybov prebiehalo tak, že sme najprv vytvorili symetrický pohyb a následne vyexportovali XML súbor, ktorý sme potom testovali na agentovi JIM. Všetky pokusy prebehli úspešne. Boli pokusy urobiť pohyb, ktorý posunie viazaný kĺb mimo rozsah, ale kĺb sa zastavil v maximálnej povolenej hodnote, ako sme predpokladali.

5.3.3 Previazanie kĺbov

Počas práce na projekte sme zistili, že by bolo vhodné urobiť previazanie niektorých kĺbov pre uľahčenie tvorby pohybov. Vyskytla sa požiadavka rozšíriť editor pohybov o novú funkcionalitu *coupledWith*, ktorá bude predstavovať previazanie dvoch alebo viacerých kĺbov. Je zrejmá previazanosť symetrických kĺbov vo funkcionalite symetrie pohybov, ale chceli sme byť schopní synchronizovať napríklad aj pohyb bedrového kĺbu s kolenným. Táto možnosť je zaujímavá najmä pri tvorbe chôdze alebo rôznych typov kopov.

Počas implementácie sme však narazili na problém. Samotný editor nepodporuje takéto komplikovanejšie previazanie a už nejde iba o zmenu hodnoty, ako to bolo pri symetrických (či zrkadlových) pohyboch. Je nutné navrhnúť, implementovať a odladiť zložité výpočty. Do prototypu bola pre tento účel implementovaná nová trieda, ktorá bude zo súboru načítavať možné spojenia kĺbov. Implementácia previazania kĺbov zatiaľ nie je dokončená, ale vzhľadom na užitočnosť tejto funkcionality pri tvorbe zložitejších pohybov sme rozhodnutí pokračovať v snahe o jej dokončenie.

Najväčší problém sa vyskytol pri samotnej vizualizácii agenta v editore a pri následnom prehrávaní pohybu. Pri overení sa agent správal neprirodzene a neuskutočňoval pohyby, ktoré boli očakávané. Pre časovú náročnosť riešenia tohto problému sa nám nepodarilo doriešiť ho.

Do budúcna sa pokúsime v osobitnej, novej triede zadefinovať konkrétne typy previazaní. Ak bude toto riešenie úspešné, existuje možnosť implementácie získavania previazaní ďalších kĺbov zo súboru. Štruktúra tohto súboru zatiaľ nie je známa, ale bude musieť obsahovať nie len názvy kĺbov, ale aj rovnice, podľa ktorých sa budú previazané kĺby správať (napríklad ak posunieme bedrový kĺb o 30° hore, kolenný sa posunie o 90° dole).

6 Implementácia

Táto kapitola obsahuje zoznam a opis implementovaných prvkov v projekte v letnom semestri. Rozdelená je na tri základné časti.

V prvej časti sa zaoberáme implementáciou testovacieho frameworku, ako aj písaním testov pre agenta. Opísaný je tu základný princíp práce testovacieho frameworku, ako aj pokrok v rámci písania samotných testov pre agenta.

V druhej časti sú opísane ďalšie pohyby, ktoré sme implementovali pre agenta JIM.

Posledná časť sa zaoberá plánovacím modulom agenta JIM, ako aj samotným správaním. V tejto časti sa zaoberáme sčasti vyššou logikou agenta, pre účely hrania futbalu.

6.1 Testovací framework – tréner

V predchádzajúcej časti bolo spomenuté, že testovací framework vznikol pre potreby aplikácie strojového učenia v simulačnej súťaži Robocup 3D. Od testovacieho frameworku, vychádzajúc aj z informácií z analýzy uskutočnenej v minulom semestri môžeme funkcionalitu zhrnúť do nasledovných bodov:

- vystavanie špecifickej udalosti na ihrisku (+ realizácia)
- spracovávanie priebehu vývoja udalosti
- vyhodnotenie udalostí
    - metrika úspešnosti
    - vzdialený podnet na úpravu rozhodovania agenta adresovaná priamo jemu

6.1.1 Reprezentácia sveta

Pri zmieňovaní o testovacom frameworku sa najprv musíme pozastaviť pri implementovanom monitore.

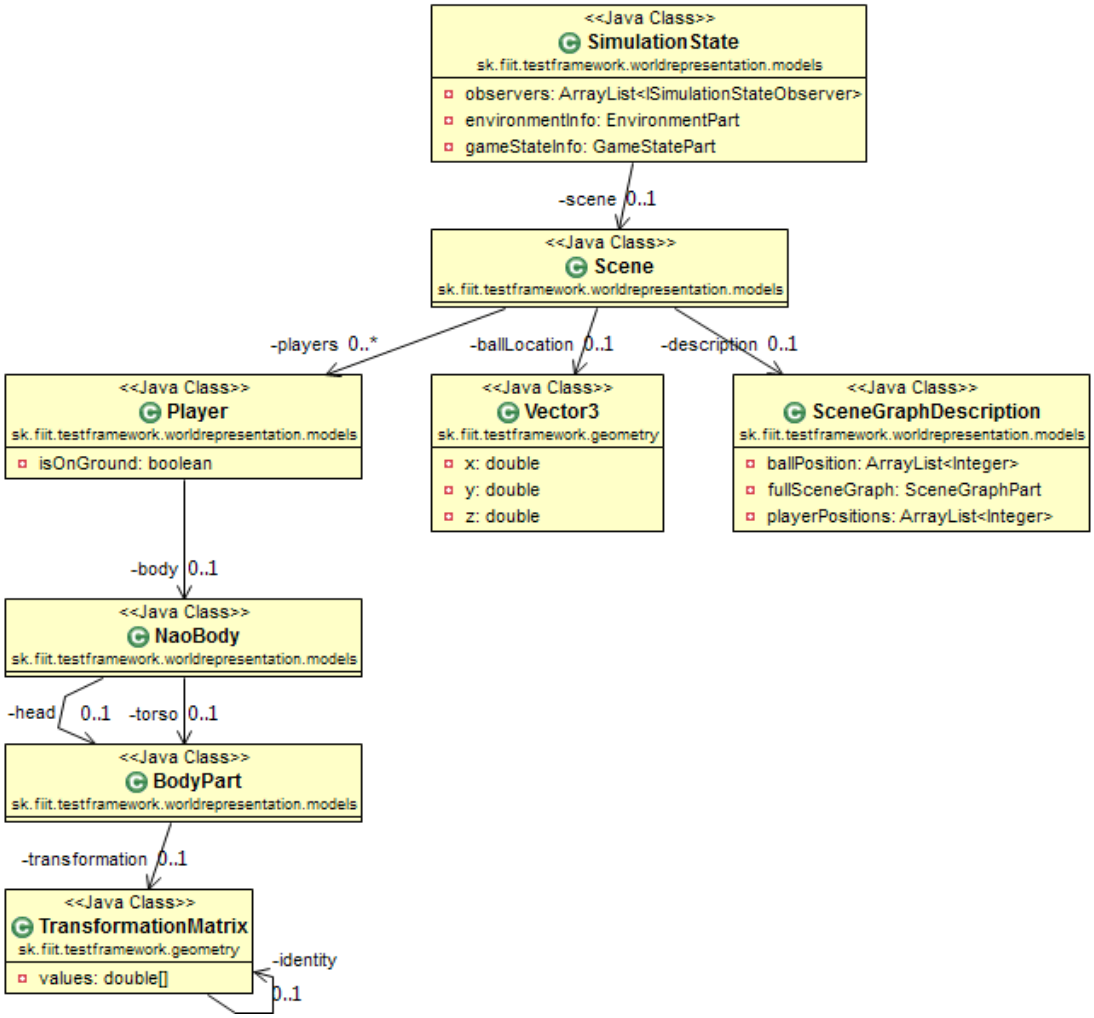
Súčasťou vytvoreného monitora je modul zodpovedný za reprezentáciu stavu simulácie. Tento modul spolupracuje s parsovacím modulom, od ktorého prijíma informácie. Úlohy tohto modulu sú nasledovné:

- Spracovať informácie z parsovacieho modulu
- Zrozumiteľne reprezentovať stav simulácie
- Identifikovať pozície hráčov a lopty

Základnou triedou tohto modulu je trieda *MessageInterpreter*, ktorá interpretuje prijatú správu zo servera spracovanú parsovacím modulom a aktualizuje stav simulácie. Túto úlohu vykonáva jediná verejná metóda *interpret*, ktorá na základe prijatej správy aktualizuje aktuálny stav simulácie.

V prípade, že z monitora príde iba čiastočný graf scény, trieda ho synchronizuje s úplným grafom scény, ktorý má k dispozícii.

Stav simulácie je reprezentovaný triedou *SimulationState*. Pre notifikácie o zmenách v inštancii tejto triedy bol využitý návrhový vzor *Observer*. Celkovú štruktúru modulu pre reprezentáciu sveta zobrazuje obrázok 43.



***Obr. 43**: Modul reprezentácie sveta – diagram tried*

Stav simulácie teda obsahuje okrem informácií o stave hry aj informácie o scéne. Scéna poskytuje informácie o polohe lopty reprezentovanej 3-rozmerným vektorom X, Y, Z a informácie o hráčoch. V súčasnosti sa pri každom hráčovi určuje poloha jeho tela a hlavy, ktoré sú nevyhnutné pre určenie celkovej polohy hráča a zisteniu, či sa daný hráč nachádza na zemi. Poloha hráča je okrem 3-rozmerného vektora X, Y, Z navyše spresnená ďalším 3-rozmerným vektorom, ktorý udáva otočenie hráča v smere osi X, Y a Z.

K výpočtu týchto informácií boli využité maticové operácie tak, aby sme vždy disponovali globálnou polohou hráča a nie iba lokálnou, ktorú nám posiela server. Hráča, ktorého hlava sa nachádza nižšie ako 20 cm považujeme za spadnutého.

6.1.2 Komunikácia trénera a agenta

Jednou z požiadaviek na testovací framework bolo aj to, aby bolo možné testovací framework použiť pre iného agenta ako Jima. Testovací framework sa skladá v podstate z dvoch od seba nezávislých komponentov:

- *Monitor*
- *Trainer*

V prípade `Monitor`u sa jedná o časť frameworku, ktorá je absolútne nezávislá na hráčovi, je to v podstate obdoba *rcssmonitor3d*. Zobrazenie dát zo servera pomocou `rcssmonitor3d` sa deje graficky, pričom pomocou komponentu `Monitor` sa zobrazujú dáta získané zo servera na objektovú reprezentáciu sveta, ktorá je prístupná používateľovi využívajúcemu testovací framework.

Druhou časťou frameworku je *Trainer*, ktorý je vo veľkej miere nezávislý od agenta, ale na to, aby umožňoval interoperabilitu s viacerými agentami je nutné vykonať určité kroky, aby bola zabezpečená komunikácia medzi nimi.

Samotný `Trainer` by sa dal rozdeliť na dve časti a to na nezávislú od agenta a naopak na závislú. Nezávislá časť komunikuje výlučne so serverom: umiesťňovanie hráča na ihrisku (tu však musíme zdôrazniť, že nejde o "beamovanie" hráča, ktoré prebieha zo strany hráča), umiesťňovanie lopty a jej smeru, zmena režimu hry a podobne.

Časť závislá od agenta slúži na explicitnú komunikáciu s hráčom, pričom je zrejmé, že je nutné primárne na strane hráča vytvoriť systém na uskutočnenie tejto komunikácie. Z hľadiska návrhu, špecifikácie a iných dôvodov sa dospelo k záveru, že hráč bude môcť prijímať príkazy z vonku. Ako systém na príjem príkazov sa používa jednoduchý, na UDP založený TFTP server, ktorý umožňuje dvojakú funkcionalitu:

- príjem súborov
- príjem špeciálneho súboru, ktorý reprezentuje skript a jeho následné vykonanie

Na strane testovacieho frameworku sú uvedené metódy implementované v balíčku *sk.fiit.testframework.communication*. Zo zdrojových súborov zároveň je zrejmé, že špeciálny súbor, ktorý reprezentuje skript ma názov "ruby.exec". V podstate to znamená, že kebyže máte k dispozícii TFTP klient a v súborovom systéme súbor s názvom *ruby.exec*, tak jeho odoslaním na štandardný TFTP server spustíte na našom upravenom klientovi ruby skript. Samozrejme, ak budete chcieť

využiť iný jazyk, iný skriptovací systém, prípadne iný typ komunikácie, všetko je možné patrične bez vačších intermodulárnych zásahov pretvoriť, vzhľadom na hierarchickú stavbu frameworku.

Zároveň to však znamená, že musíte upraviť vlastného agenta tak, že doňho pridáte TFTP server, ktorý bude prijímať súbory, resp. vykonávať skript pre vopred uvedený názov.

Jednou z vecí, ktorú pri agentovi JIMovi používame je vzdialená invokácia preplánovania, ktorej je v triede *sk.fiit.testframework.trainer.Trainer* vyčlenená metóda *public void invokeReplan(AgentJim agent)* s nasledovným obsahom:

```
public void invokeReplan(AgentJim agent) throws IOException {
    StringBuilder sb = new StringBuilder();
    sb.append("require \"java\"\n");
    sb.append("include_class \"sk.fiit.jim.init.ScriptBoot\"\n");
    sb.append("ScriptBoot.boot");
    executeRubyScript(agent, sb.toString());
}
```

V argumentoch metódy môžeme vidieť objekt typu *AgentJim*, ktorý zapúzdruje konkrétnu inštanciu agenta, aby bolo nasledovne možné jednoduchšie k agentovi pristupovať. V prípade objektu *AgentJim* je určite vhodné spomenúť nasledujúce fakty:

- agenta dokáže framework spustiť
- na bežiacu inštanciu agenta sa dokáže framework pripojiť

Uvedené dve možnosti sú z toho dôvodu, že agenti sú procesorovo aj pamäťovo náročné entity, pričom primárne pre debugovacie účely je nevyhnutné mať častokrát agenta spusteného explicitne (mimo framework) a iba sa naň pripájať. Pre finálne použitie v testovaní sa táto možnosť bude zrejme využívať menej.

Na to aby bolo možné agenta zo strany frameworku spúšťať bolo ešte potrebné vytvoriť vlastnú triedu v agentovi, ktorá obsahuje spúšťací mechanizmus na to, aby sa jednoznačne špecifikovalo nasledovné:

- port TFTP servera (framework bude k nemu pristupovať)
- číslo dresu a tím (tréner keď komunikuje so serverom a posiela príkazy, tak identifikuje hráča práve pomocou uvedených informácií)

Po implementácií uvedených konceptov by mal byť framework pripravený komunikovať aj s iným typom agenta.

6.1.3 Vystavanie špecifickej udalosti na ihrisku

Vytvorenie konkrétnej udalosti na ihrisku sa z hľadiska servera Robocup 3D uskutočňuje cez pripojenie na protokol *Monitor / Trainer*, ďalej len MT. V defaultnom nastavení je to komunikácia

prostredníctvom TCP-IP na porte č. `3200`. Komunikácia prebieha pomocou odosielania správ, ktoré sú v testovacom frameworku implementované ako verejné vnútorné statické triedy abstraktnej triedy *TrainerCommand*. Najdôležitejšie "podtriedy" sú

- *TrainerCommand.Agent*
- *TrainerCommand.Ball*
- *TrainerCommand.PlayMode*

Uvedené triedy nie sú určené výlučne pre koncového používateľa, ktorému postačuje pracovať s triedou *Trainer. Trainer* slúži ako high-level API pre používateľa, ktorému poskytuje konkrétne metódy na uskutočnenie určitých udalostí na serveri. To dosahuje komunikáciou s protokolom *Monitor / Trainer*, resp. iba samotným odosielaním údajov[15].

Nasledujúci príkaz slúži na umiestnenie lopty blízko stredu, s náhodnou rýchlosťou v rozmedzí <0,1> na osiach *y*, *z*.

```
Trainer trainer = new Trainer(address);
trainer.setBall(new Point3D(0, 0, 0.1),
                new Point3D(1, rnd.nextDouble(), rnd.nextDouble()));
```

6.1.4 Spracovanie a vyhodnotenie udalosti

Na spracovanie priebehu vývoja udalostí slúži abstraktná trieda *TestCase*, ktorú je nutné implementovať, ak chceme vykonať konkrétny test. Nakoľko vývoj udalostí je vývoj závislý od času, vhodným riešením je spustenie testov v samostatných vláknach. Uvedený princíp má naviac výhodu jednoduchej paralelizácie (napr. testovanie na viacerých serveroch a pod.)

Trieda, ktorá implementuje *TestCase* musí z jej definície zároveň implementovať:

- *public abstract void init()*
- *public abstract TestCaseResult evaluate(SimulationState ss);*
- *public abstract boolean isStopCriterionMet(SimulationState ss);*

Konkrétna realizácia tvorby testu viac spadá pod oblasť používateľskej príručky, kde je vytvorenie testu a jeho spustenie názorne ukázané.

[15] Nakoľko správania sa servera bola problematické pri *neustále* pripojenom sockete, bola komunikácia *zatiaľ* vytvorená tak, že vždy pri odosielaní príkazu sa pripojí socket na port, odošle údaje a odpojí sa. Z definície protokolu vyplýva, že uvedený spôsob je menej efektívny (napr. automaticky sa pripojením na port MT odošlú zo strany servera informácie o celej scéne).

6.1.5 Inštalácia simulačného servera na školský server

Pre potreby výpočtovo aj pamäťovo náročného testovania sme si zadovážili od školy virtuálny server s pomerne sľubnými parametrami. V prípade, že by aj z vašej strany bolo na mieste inštalovať server, táto príručka by vám mala pomôcť zamedziť viacerým problematickým miestam, ktoré pri jeho spustení vznikli.

Príručka sa týka konkrétne serverovej 64-bitovej verzie *Ubuntu 10.10 Maverick*. Je veľmi pravdepodobné, že by fungovala správne aj na klasickej desktopovej verzii.

Po nainštalovaní všetkých podstatných balíčkov, prichádza na rad inštalácia Robocup Simulation Server 3D, ktorá až na samotné spustenie aplikácie *rcssserver3d* je bezproblémová.

Z oficiálne zverejného návodu sa dozvedáme, že pridáme repozitár, aktualizujeme balíčkovací systém a nainštalujeme rcssserver3d, konkrétne:

```
$ sudo apt-add-repository ppa:gnurubuntu/rubuntu
$ sudo apt-get update
$ sudo apt-get install rcssserver3d
```

V prípade, že spustenie servera pomocou *rcssserver3d* vám skončí chybou, dôvodom je knižnica *libruby*. Knižnicu v staršej verzii treba stiahnuť a nainštalovať pomocou *dpkg*, prípadne rozbaliť a oklamať systém, že ju používa. Potrebná knižnica je v balíčku:

`libruby1.8_1.8.6.111-2ubuntu1.3_amd64.deb`[16]

Knižnicu nainštalujeme príkazom, pričom po tejto malej úprave by mal server spoľahlivo fungovať (aspoň tak to bolo v našom prípade).

```
$ sudo dpkg -i libruby1.8\1.8.6.111-2ubuntu1.3\amd64.deb
```

6.2 Dodatočné pohyby agenta

V letnom semestri sme doimplementovali niektoré zostávajúce pohyby z pôvodne plánovaných, ostatné sme sa po starostlivom zvážení rozhodli neimplementovať z rôznych dôvodov (zložitá implementácia so slabým výsledkom, dlhé trvanie alebo celková nadbytočnosť). Okrem nových pohybov sme vykonali drobné úpravy pohybov zo zimného semestra na základe nových informácií o funkcii atribútu fázy *isFinal* a s ním súvisiaceho <*finalize*>.

[16] Túto knižnicu je možné stiahnuť na http://packages.ubuntu.com/hu/hardy/amd64/libruby1.8/download, alebo z nášho repozitára.

6.2.1 Základné pohyby

V letnom semestri sme tak implementovali:

- Vstávanie
- Chôdzu
- Otáčanie
- Bránenie
- Kop do lopty
- Pomocné pohyby

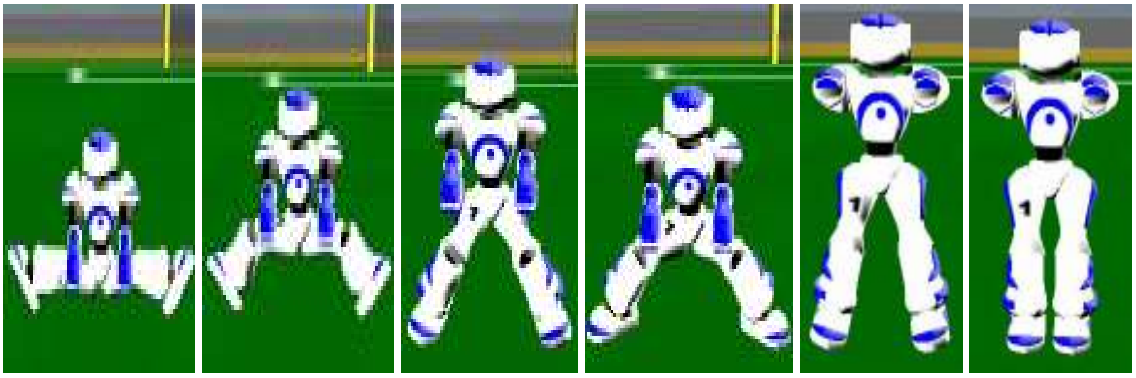
6.2.1.1 Vstávanie

Pri implementácii druhého plánovaného spôsobu bránenia – bránenie sadnutím rozkročmo – sa objavila otázka, ako sa z takejto polohy najlepšie postaviť. Usúdili sme, že najlepším riešením bude vytvoriť pre tento účel špeciálne vstávanie.

Vstávanie zo sedu rozkročmo

Tento pohyb vznikol úpravou samotného sadnutia rozkročmo a jeho hlavnou myšlienkou bolo dostať obe nohy do základnej polohy bez použitia rúk len jednoduchým prinožením, pričom najväčším problémom bolo udržanie rovnováhy. Priebeh pohybu je na Obr. 44 a môžeme ho zhrnúť do nasledovných krokov:

- prinožovanie do polostoja
- následné roznoženie, aby pokleslo ťažisko a zachovala sa rovnováha
- pokračovanie prinožovania až do úplného postavenia sa



***Obr. 44**: Vstávanie zo sedu rozkročmo*

Overenie

Pohyb dosiahol úspešnosť 100% v prípade, že hráč po dosadnutí na zem ostal vo vzpriamenom sede. V prípade, že je hráč po dosadnutí v miernom predklone navážený na ruky toto vstávanie nefunguje.

Tento problém sa nám nepodarilo odstrániť, avšak v predklonenom sede hráč končí len výnimočne. Celé vstávanie zo sedu rozkročmo trvá 1950 ms, pričom posledných 1000 ms je pomalá, dokončovacia, vystieracia fáza.

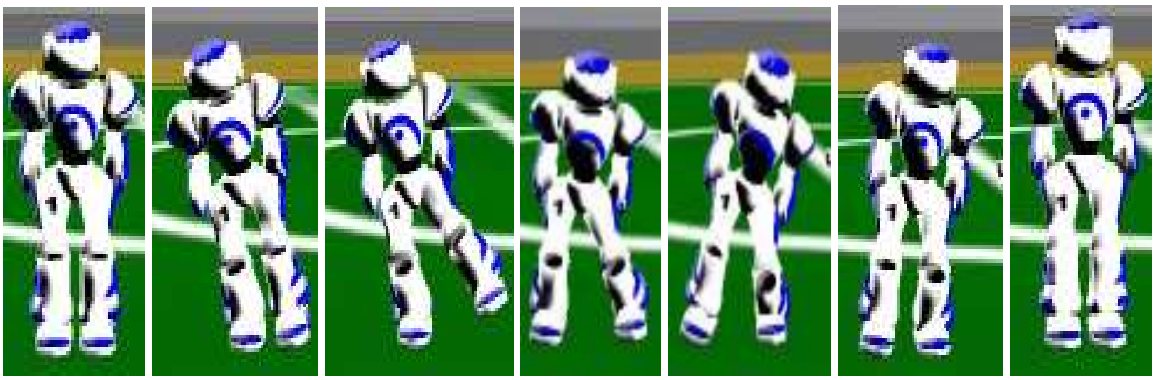
6.2.1.2 Chôdza

Rozhodli sme sa neimplementovať chôdzu šikmo, humanoidnú chôdzu a beh, takže nám ostala posledná chôdza – chôdza do strany, ktorá je obzvlášť dôležitá pre brankára. Pri implementácii a ladení chôdze do strán nám vznikla aj druhá verzia chôdze dozadu.

Chôdza do strany

Je zimplementovaná pre obe strany, pre každú dvomi spôsobmi, ktoré sa od seba odlišujú len tým, že jeden sa stáča mierne dozadu a druhý dopredu, teda robot v oboch prípadoch ide nie priamo doboku, ale v oblúku. Využiteľná je hlavne verzia so stáčaním dozadu, ktorá sa približuje reálnemu pohybu brankára. Úkrok doľava je zobrazený na Obr. 45 a pozostáva z nasledovných fáz:

- pokrčenie nôh a prenesenie váhy na pravú nohu
- zdvihnutie ľavej nohy a jej natiahnutie čo najviac do strany
- dostúpenie na ľavú nohu a prenos váhy na ňu
- prisunutie pravej nohy



***Obr. 45**: Úkrok doľava*

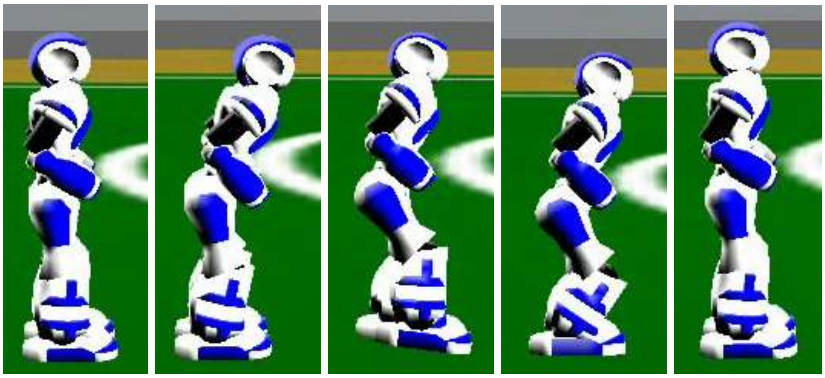
Overenie

Robot ani počas niekoľkominútového testovania pohybu v slučke nespadol ani nejavil známky nestability. Jedným úkrokom sa posunie približne o dĺžku chodidla do strany a úkrok trvá 5500 ms. Tento pohyb je extrémne pomalý, ale pri pokusoch o zrýchlenie strácal stabilitu. Najväčší problém robí neúplné prenesenie váhy na nohu, ktorou sa robí úkrok. Tento problém sa nám nepodarilo odstrániť, čo je jedným z dôvodov, prečo je úkrok taký krátky a pomalý. Napriek tomu je tento pohyb stabilný a v kontexte výkonu ostatných pohybov stále použiteľný.

Chôdza dozadu 2

Vznikla z úkrokov a je pomalšia (drobnejšia) ako chôdza dozadu zo zimného semestra. Výhodou tejto verzie chôdze dozadu je, že robot pri nej nepredkopáva ako pri zimnej verzii, vďaka čomu sa hodí na presnejší pohyb v okolí lopty. Robot sa pri tejto chôdzi odráža zo špičiek. Jeden krok v tomto type chôdze prebieha nasledovne a je zobrazený na Obr. 46:

- robot pokrčí nohy
- odrazí sa zo špičky, a tým sa posunie mierne šikmo dozadu
- dostúpi a vystrie nohy



***Obr. 46**: Krok dozadu v chôdzi dozadu 2*

Overenie

Pohyb je oproti ostatným našim pohybom menej stabilný, robot sa pri ňom výrazne „pohupuje“ a stáva sa, že spadne kvôli vplyvom chýb zo servera. Jeden krok trvá 2200 ms a približné posunutie dozadu je o štvrť chodidla. Kvôli týmto vlastnostiam nie je uvedený pohyb vôbec vhodný na prekonávanie väčších vzdialeností. Napriek veľkému priestoru pre vylepšenia sme zrýchľovaniu a vylepšovaniu tohto pohybu nevenovali veľa času kvôli jeho relatívne nízkej priorite a existencii druhého, rýchlejšieho kráčania dozadu. Výhodou tohto pohybu je pomerne veľká presnosť, keďže robot zachováva veľmi spoľahlivo priamy smer, kým nespadne.

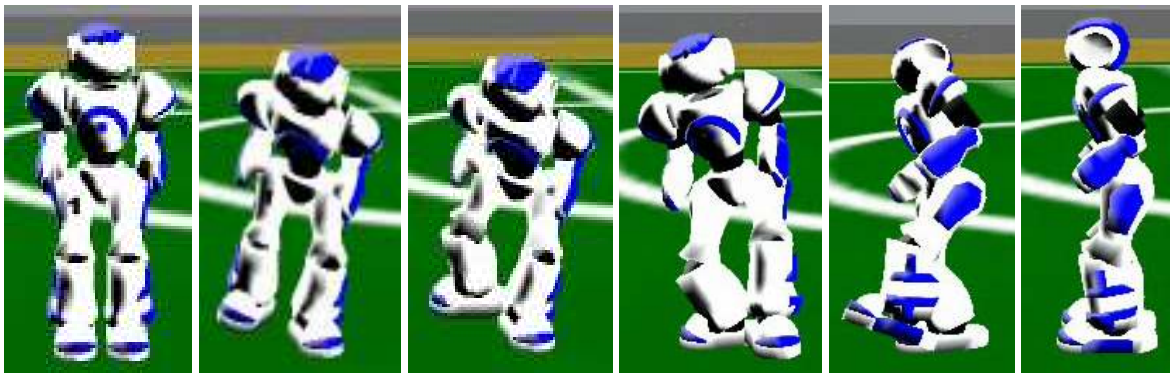
6.2.1.3 Otáčanie

Keďže v zimnom semestri sme pracovali len na jednom spôsobe otáčania (otočenie o 90°), a ani to sa nám nepodarilo vytvoriť k našej spokojnosti, v letnom semestri sme venovali otáčaniu zvýšenú pozornosť. Nanovo sme vytvorili otočenie o 90° do oboch strán, keďže verzia zo zimného semestra nebola použiteľná ako vhodný základ pre ďalšiu prácu. Ďalej sme vytvorili dve menšie otáčania. Otočenie o 135°, 180° a vojenské otočenie sme sa rozhodli nakoniec nevytvárať, keďže aj otočenie o 90° naraz trvá pomerne dlho a kvôli fyzike servera bolo pomerne zložité dosiahnuť stabilitu tohto pohybu.

Otočenie o 90°

Keďže pôvodný pokus vytvoriť tento pohyb na základe pozorovania jeho ľudskej verzie zlyhal, rozhodli sme sa vytvoriť jeho výrazne robotickú verziu, ktorá je prispôsobená vlastnostiam a možnostiam fyziky servera. Pohyb je zimplementovaný do oboch smerov a jeho „pravá“ verzia pozostáva z nasledovných krokov (Obr. 47):

- naváženie sa ľavú nohu tak, že hráč stojí len na nej
- zdvihnutie a pokrčenie pravej nohy
- pohyb v bedrovom kĺbe pravej aj ľavej nohy tak, že zvierajú pravý uhol
- položenie pravej nohy a preváženie na ňu
- zdvihnutie, pokrčenie, prinoženie a vystretie ľavej nohy
- dokončenie pohybu celkovým narovnaním robota



***Obr. 47**: Otočenie o 90° doprava*

Overenie

Pohyb je 100% stabilný, robot ani v jednom z 30 testovacích pokusov nespadol. Výsledný uhol otočenia je mierne väčší ako želaných 90°, avšak táto odchýlka je len niekoľkostupňová, čo je pri takom veľkom otočení zanedbateľné. Trvanie pohybu je 4800 ms, čo je pomerne veľa, ale vyššiu rýchlosť sa nám nepodarilo dosiahnuť bez výrazného zhoršenia stability. V porovnaní so zimnou verziou pohybu sa nám podarilo dosiahnuť determinizmus, čo bolo naším cieľom, preto výsledok hodnotíme pozitívne.

Plynulé drobné otáčanie - 5°

Plynulé drobné otáčanie je cyklický pohyb, ktorý by mal hráč využívať v situáciách, keď sa potrebuje pomerne presne, po malých úsekoch, natočiť (typicky napríklad k lopte).

Otáčanie o cca 5° je jemnejšia verzia otáčania, ktorá si vyžaduje pomerne krátke pohyby kĺbov agenta. Pozostáva zo štyroch fáz. V prvej sa agent mierne prikrčí a zloží ruky s cieľom zvýšenia stability. V ďalšej fáze sa naváži pootočením bedrových a členkových kĺbov na jednu zo strán. Tretia

fáza pozostáva z mierneho nadvihnutia a predsunutia jednej z nôh a štvrtá slúži na jej položenie s miernym vychýlením. Výsledkom je natočenie agenta o malý úsek. Finalize fáza slúži na postavenie agenta do východzej polohy.

Overenie

Agenta sme nechali otáčať sa počas jednej minúty. Dokázal sa otočiť približne o 130°. Pri takomto rozsahu otáčania nedokázal ustáť otáčanie na jednom mieste, pohyb tvoril skôr akúsi špirálu. Napriek tomu považujeme pohyb za dostatočný, nakoľko väčší rozsah otáčania sa bude realizovať buď otočením o 90° alebo plynulým otáčaním o 20°. Počas testovania agent ani raz nespadol, hoci náhodne prejavoval známky miernej nestability (ktorá však vďaka neveľkému rozsahu pohybov nespôsobila pád).

Plynulé drobné otáčanie - 20°

Plynulé drobné otáčanie je cyklický pohyb, ktorý by mal hráč využívať v situáciách, keď sa potrebuje pomerne presne, po väčších úsekoch, natočiť (typicky napríklad k lopte).

Otáčanie o cca 20° je verzia otáčania po väčších úsekoch, ktorá je vďaka rozsiahlejším pohybom kĺbov menej stabilná ako plynulé otáčanie o 5°. Pozostáva zo štyroch fáz, ktoré vychádzajú z pohybu plynulého otáčania o 5°. V prvej sa agent mierne prikrčí a zloží ruky s cieľom zvýšenia stability. V ďalšej fáze sa podobne ako pri jemnejšom otáčaní naváži agent pootočením bedrových a členkových kĺbov na jednu zo strán. Tretia fáza pozostáva z výraznejšieho nadvihnutia a predsunutia jednej z nôh a štvrtá slúži na jej položenie s vychýlením. Výsledkom je natočenie agenta o úsek zodpovedajúci približne 20°. Finalize fáza slúži na postavenie agenta do východzej polohy.

Overenie

Agenta sme podobne ako pri plynulom otáčaní o 5° nechali vykonávať pohyb po dobu jednej minúty. Za tento čas sa otočil približne o 400° a rovnako ako pri plynulom otáčaní o 5°, aj tento pohyb spôsobil pohyb agenta „do špirály“. V tomto prípade bol tento efekt výraznejší, nakoľko prikročenie môže aj vinou vnášania rôznych chýb serverom spôsobiť menší či väčší posun. Počas testovania bolo nutné predĺžiť trvanie niektorých fáz s cieľom dosiahnuť takú stabilitu, aby počas testovania nenastal ani jeden pád agenta.

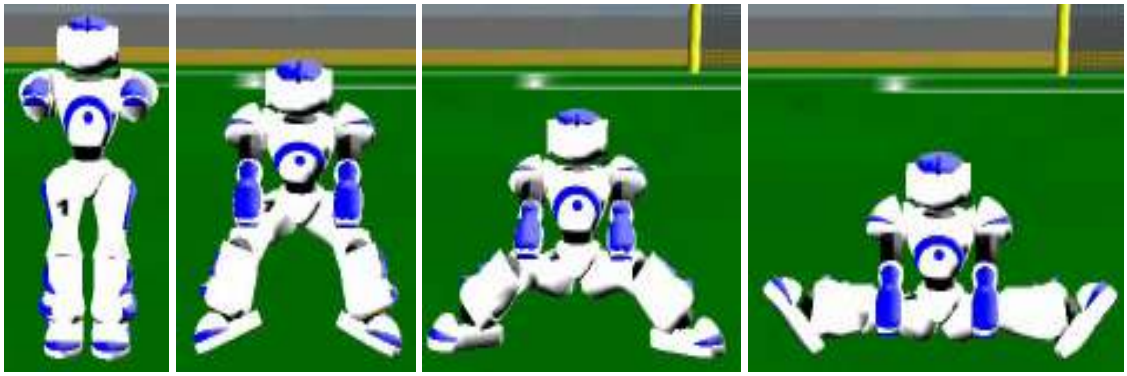
6.2.1.4 Bránenie

V letnom semestri sme vytvorili druhý z uvažovaných spôsobov bránenia – bránenie sadnutím rozkročmo.

Bránenie sadnutím rozkročmo

Tento pohyb je vhodný pre zabránenie gólu strelou, ktorá ide priamo na brankára. Je vcelku jednoduchý, avšak problémy v jeho implementácii spôsobovala potreba udržať rovnováhu pri výrazne dynamickom pohybe nôh. Pohyb je zobrazený na Obr. 48 a pozostáva z nasledovných krokov:

- nastavenie rúk šikmo dole
- postupné rozkračovanie a vytáčanie nôh smerom von v bedrovom kĺbe, a zároveň skŕčanie kolien a nastavovanie členkov
- poloha v širokom stoji rozkročmo s chodidlami plochou na zemi
- vystieranie kolien a dosadnutie



***Obr. 48**: Bránenie sadnutím rozkročmo*

Overenie

Pohyb je stabilný, robot sa ani v jednom z 30 pokusov neprevrátil ani na tvár, ani na chrbát. Základná koncová poloha, pre ktorú existuje aj špeciálne vyvinuté vstávanie, je vzpriamený sed rozkročmo bez opierania sa o ruky. Výnimočne sa stane, že robot skončí v miernom predklone alebo nedosadne vycentrovane, teda sa posunie prienik zvislej osi jeho tela s ihriskom mierne do strany. Tieto odchýlky sú spôsobené chybami zo strany servera. Trvanie pohybu je 1350 ms.

6.2.1.5 Kop do lopty

Rozhodli sme sa implementovať ďalší kop – kop do lopty hranou chodidla

Kop do lopty priamy hranou chodidla

Tento kop dopĺňa našu množinu kopov do lopty – priamy kop a kop do boku hranou chodidla. Tu ide vlastne o kombináciu týchto dvoch pohybov – styčnou plochou hráča s loptou je hrana chodidla a lopta smeruje priamo.

- V prvej fáze sa agent dostane do stabilizovanej polohy, ktorá mu umožňuje v nasledovnej fáze stáť na jednej nohe. Táto poloha spočíva v pokrčení kolien, nastavení torza a miernom upažení rúk.
- Druhá fáza slúži na naváženie sa agenta na jednu stranu (na ľavú alebo na pravú, podľa toho, ktorou nohou agent kope).
- Tretia fáza slúži na zdvihnutie a vytočenie nohy tak, aby bolo možné kopnúť hranou chodidla.
- Štvrtú fázu tvorí samotný kop.
- V piatej, poslednej, fáze sa agent dostane opäť do stabilizovanej polohy.

Overenie

Tento kop je určite slabší oproti našim ďalším dvom kopom. Spôsobené je to najmä tým, že uhol vytočenia neumožní dostatočný náklon, a preto je tento pohyb vhodný skôr ako prihrávka, než ako výkop na bránu. Zároveň však vďaka kopacej ploche má tento kop potenciál byť presnejším.

Vďaka neveľkej sile kopu je tento zároveň pomerne stabilný, tridsať vykonaných pokusov zvládol agent bez pádu. Pri tomto množstve pohybov sa však agent otočil o vyše 90°, čo je spôsobené drobným posunom pri došľapovaní po kope. Pri zachovaní tejto stability trvá pohyb 2900 ms.

6.2.1.6 Pomocný pohyb - Pohyb hlavy

Pôvodne sme neuvažovali o využití pohybov hlavy, keďže sme mali informácie z oficiálnych zdrojov, že vizuálny senzor robota sa nachádza v jeho trupe a teda pri získavaní prehľadu o situácii v prostredí nám otáčanie hlavy nepomôže. V polke letného semestra sme sa však dozvedeli, že informácie v oficiálnych zdrojoch boli zastarané a kamera je už niekoľko verzií servera v hlave robota, preto sme sa rozhodli vytvoriť pohyby hlavy, ktoré využijeme pri orientácii.

Otočenie o 120°

Zvažovali sme niekoľko alternatív, ako najlepšie využiť otáčanie hlavy pri orientácii a nakoniec sme sa rozhodli pre 2 oddelené pohyby hlavy – otočenie o plný rozsah 120° doprava s návratom do priamej polohy a rovnaký pohyb hlavy doľava. Táto voľba bola založená na potrebe, aby bol hráč schopný rozhliadnuť sa do oboch smerov, aby vždy hlava nakoniec skončila v priamej polohe, a aby sme minimalizovali časové straty, teda ak získame potrebné informácie z otočenia do prvej strany (náhodne alebo heuristicky si vyberieme, ktorá to bude), už sa nebudeme rozhliadať aj do druhej strany.

Pohyb je jednoduchý, preto neuvádzame obrázky ani kroky, ktoré sú jasné z charakteristiky. Trvanie pohybu je 1000 ms (plné otočenie na jednu stranu a vrátenie späť, obe po 500 ms).

6.3 Plánovanie agenta

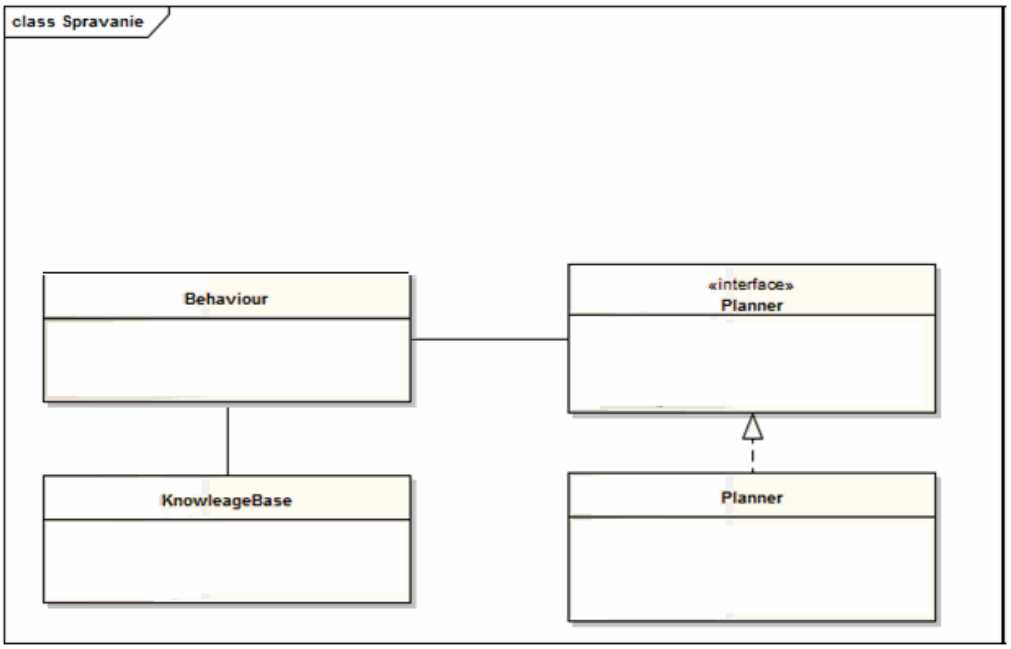
Pre potrebi plánovania vyššej logiky agenta, potrebovali sme sa zahľadieť do plánovacieho modulu agenta JIM. Tento modul možno charakterizovať dvomi submodulmi: komponent správania a komponent plánovania. Tieto dva komponenty tesne súvisia zo sebou.

6.3.1 Návrh plánovania agenta

Ak sa pozeráme na komponent správania agenta, musíme zvážiť, že tento komponent zaručuje správanie sa agenta v čase, teda zaručuje vyhodnocovanie aktuálnej situácije a na základe nich následne plánuje akcie. Z toho hľadiska, môžeme teda usúdiť, že modul správania volá jednotlivé metódy tried v module plánovania.

Správanie je navrhnuté na základe jednoduchého produkčného systému. Akcie, ktoré agent má vykonať, sa naplánujú na základe bázy znalostí a vložia sa do plánovača. V plánovači sa následne tieto akcie budú usporiaduvávať alebo po prípade zrušovať.

Pre implementáciu plánovacieho modulu sme si zvolili prístup, pri ktorom bude treba použiť produkčný systém. Všetky pravidlá, podľa ktorých sa bude agent správať vudú tak zapísané v textovom súbore. Následne sa bude vyhodnocovať dopyt z množiny pravidiel, pomocou funkcie v triede *KnowledgeBase()*, ktorá by mala vrátiť akciu, ktorú agent má vykonať. Následne istá akcia bude naplánovaná a zaradená do plánovača. Na Obr. 49 sa náchádza zjednodušené zobrazenie navrhnutej architektúry tried plánovacieho modulu agenta JIM.



***Obr. 49**: Zjednodušená architektúra plánovacieho modulu (modulu správania agenta)*

Rozhodli sme sa pre takýto prístup pomocou produkčného systému preto, lebo je pomocou neho možná jednoduchá úprava pravidiel. V tom prípade bude možné definovať pravidlá aj pre vyššie, tímové správanie agentov. Tiež jeden z dôvodov, prečo sme sa rozhodli implementovať takýto prístup pri správaní agenta je aj skutočnosť, že už sčasti máme naimplementovaný produkčný systém.

Nevýhoda tohto prístupu môže byť pomalšie vyhodnocovanie pravidiel. Na rozdiel od toho je možné navrhnúť spôsob, pri ktorom všetky pravidlá budú písané rovno v kóde agenta v implementačnom jazyku. Tento spôsob sme však zatiaľ usúdili ako nevhodný pre nás.

6.3.2 Pravidlá správania agenta

Na Obr. 50 je znázornený rozhodovací strom agenta JIM. V produkčnom systéme sme sa rozhodli použiť na začiatok jednoduché pravidlá. Pravidlá budú zapísané v tvare *ak* podmienok. Jednotlivé pravidlá sme navrhli nasledovne:

1. Ak nevidím loptu => otočím hlavu
2. Ak vidím loptu a mám otočenú hlavu => otočím sa smerom k lopte
3. Ak vidim loptu => bežím k lopte
4. Ak mám loptu => kopnem k bráne súpera
5. Ak spadnem => vstanem
6. Ak som brankár => som len pri bráne
7. …

Uvedené pravidlá sú iba tie najzákladnejšie. Nezahŕňajú stratégiu hrania. Sú navrhnuté ako začiatočné, avšak keďže sme navrhli použitie produkčného systému, bude ich možné jednoducho meniť alebo po prípade pridavať nové, čo zaručí možnosť strategického plánovania hry. Potrebné je rozdeliť dva základné typy agenta: bránkar a hráč. Tieto dva typy by sa nemali správať rovnako. Preto je na Obr. 51 znázornený rozhodovací strom brankára.

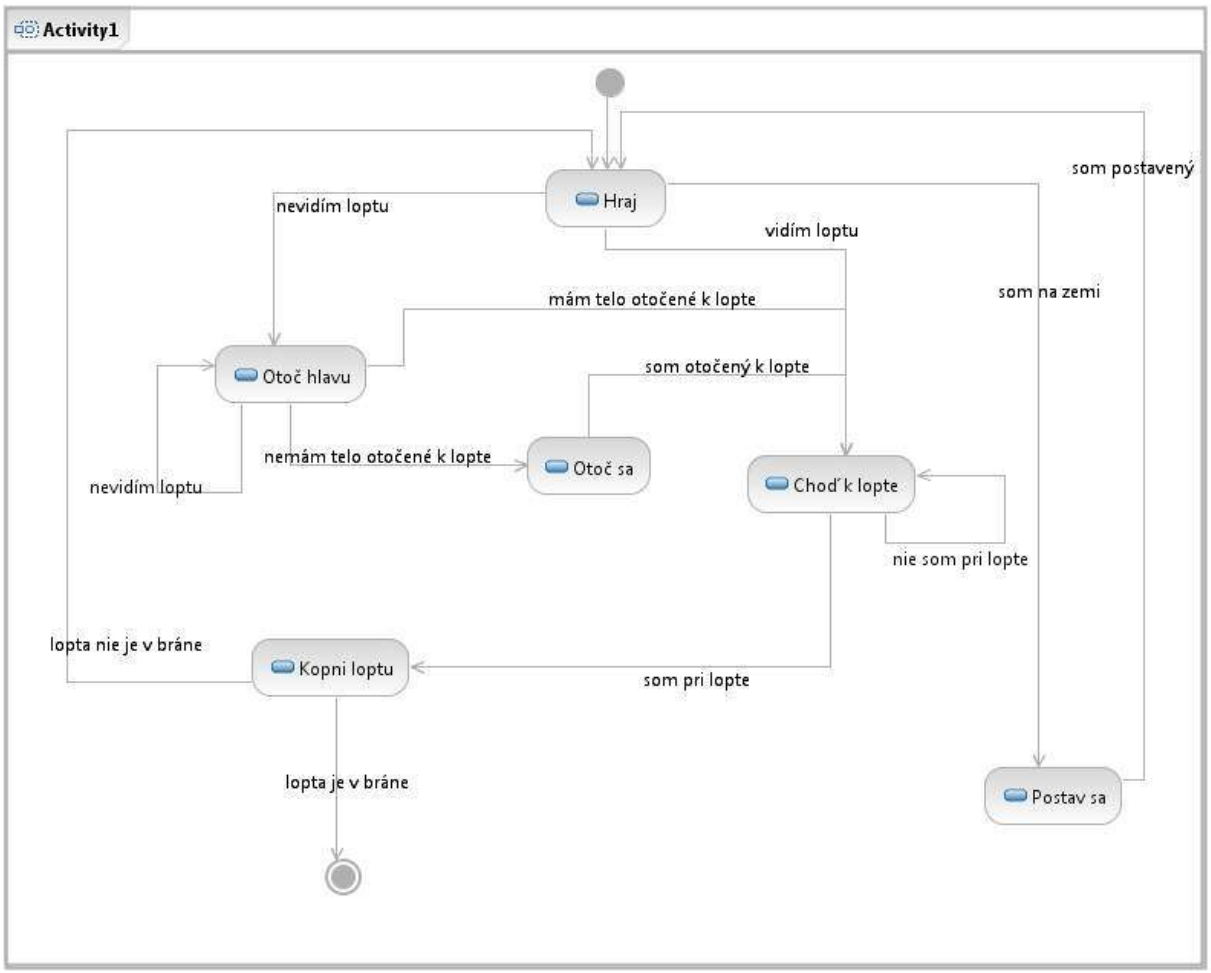
Keďže sme potrebovali zabežpečiť správanie agenta, vznikla požiadavka na implementáciu metódy, ktorá bude zisťovať polohu lopty a po prípade predpokladať jej polohu v určitom čase. Agent JIM má na tento účel vytvorenú triedu *DynamicObject*, ktorej inštanciou je aj samotný objekt *lopta*. Keďže vieme zistiť polohu lopty z tohto objektu, následne sme využitím relatívnej polohy lopty získanej zo správy servera vypočítali ansolútnu polohu lopty. Táto poloha je zachovaná spolu s časom videnia.

Ak zistíme ďalšiu polohu lopty po nasledovnom cykle (keď nasledovný krát príde správa zo servera), vieme z novšej absolútnej polohy zistiť pravdepodobnú polohu lopty. Implementácia predikcie polohy lopti je zatial iba vo fáze návrhu.

S častí produkčného systému je zatiaľ implementovaná časť detekcie lopty a nasledného príchodu k nej. Implementácia tohto modulu je zatial rozpracovaná a v najbližšom období sa budeme zaoberať práve dokončovaním implementácie produkčného systému, a následne aplikácie testovacieho frameworku na plánovací modul (teda spolupráce testovacieho frameworku a agenta JIM – posielanie pokynov agentovi a následné vyhodnotenie vykonania pohybov).

6.3.3 Rozhodovanie agenta

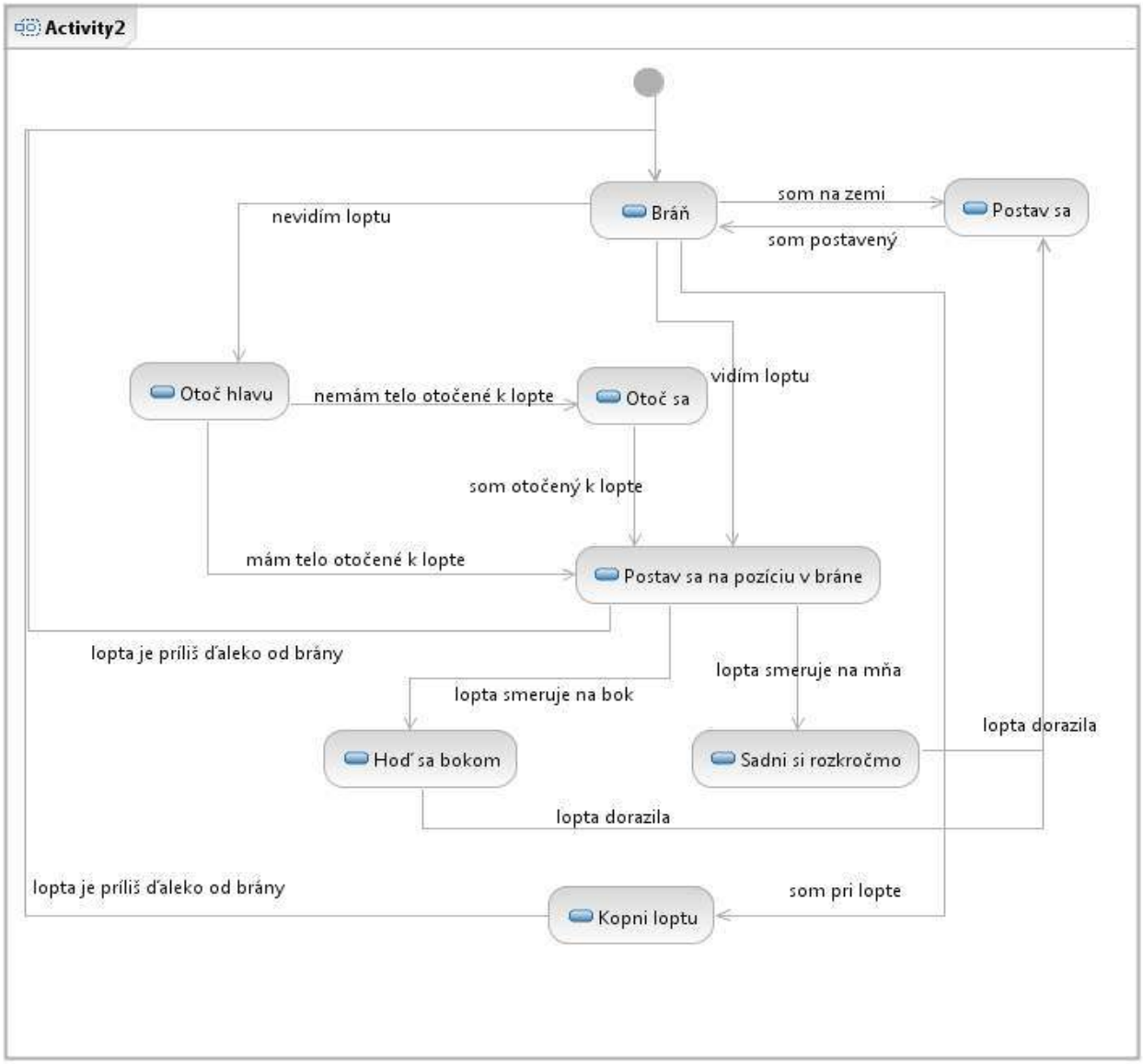
Na Obr. 50 je znázornený rozhodovací strom agenta vo funkcii hráča. Potrebujeme si všimnúť, že v strome existuje akcia *„Postav sa“*. Táto akcia sa v podstate nevykonáva iba v pôvodnom stave, ak je agent na zemi, ale pri každéj jednej hrane v strome. Teda ak agent v určitom momente spadne na zem, vyhodnotí sa postavenie hlavy vzhľadom na zem a následne sa vykoná akcia *„Postav sa“*. Po postavení agent znovu hľada loptu, teda vracá sa na začiatok rozhodovacieho stromu.



***Obr. 50**: Rozhodovací strom agenta JIM – hráč*

Rozhodovanie zachytené rozhodovacím stromom na Obr. 50 je zatiaľ iba názorné. Teda toto rozhodovanie sme sa rozhodli implementovať, keďže sme presvedčený, že sa pri implementácii týchto základných rozhodovacích pravidel, agent môže postaviť do zápasu a samostatne hrať. Následne sa tento rozhodovací strom bude rozširovať a bude komplikovanejší, pričom budeme zvažovať aj iné objekty, ako sú spoluhráči, súperi a pod.

Takýmto spôsobom, teda využitím produkčného systému na tvorbu pravidiel a plánovania akcií, ako už bolo povedané, je možné následne strom rozšíriť aj o ďalšie prvky, ktoré sa už týkajú samotnej stratégie agentov v tíme. Toto je však práca pre budúce generácie, keďže sme časovo ohraničený.



***Obr. 51**: Rozhodovací strom agenta JIM – bránkar*

Obr. 51 znázorňuje jemne pozmenený rozhodovací strom obsahujúci základné príncípy bránenia. Podľa tohto systému rozhodovania sa bude správať iba jeden agent, ktorého na začiatku zápasa vyhlásime za bránkara. Na rozhodnutie, ktorý agent bude bránkar sme sa rozhodli použiť jednoduché pravidlo „prvý príhlásený“. Teda prvý agent, ktorý sa prihlási do zápasu, bude vyhlásený za bránkara.

V tejto časti implementácie je funkcia bránkara zatial iba čiastočne implementovaná. Práve pre potreby bránenia bolo potrebné urobiť zmeny v *DynamicObject* triede, a implementovať dodatočný modul, ktorý bude zodpovedný na predikciu polohy lopty.

Pre účely hrania futbalu sa v najbližšom času budeme zaoberať a implementovať ďaľšie rozšírenia plánovacieho modulu.

7 Záver a námety na prácu do budúcnosti

Táto kapitola sa zaoberá námetmy na budúcu prácu. Chceme najmä poukázať, na čom je ešte potrebné pracovať, aby sa zdokonalil agent JIM a taktiež aj testovací framework.

8 Použitá literatúra

[1] Kouretes 2009: *Nao Standard Platform League Team Description Paper*. CD Proceedings of the 13th RoboCup International Symposium, June 2009, Graz, Austria.

[2] Todd Hester, Michael Quinlan, and Peter Stone: *Generalized Model Learning for Reinforcement Learning on a Humanoid Robot*. Proceedings of IEEE International Conference on Robotics and Automation, May 2010, Anchorage, AK.

[3] Amin Milani Fard, M. Hossein Ansari, Amir Farzad, Amin Zamiri, Ehsan Saboori, Mahmoud Naghibzadeh: *Nexus 3D 2007 Soccer Simulation Team Description*. RoboCup, July 2007, Atlanta, US.

[4] Agenty 007: *Dokumentácia k projektu – Finálna verzia*. Máj 2009, Bratislava, Slovensko. http://labss2.fiit.stuba.sk/TeamProject/2008/team07is-si/dokumentacia3.pdf

[5] Hviezdna jedenástka: *Dokumentácia k projektu.* Máj 2008, Bratislava, Slovensko. http://labss2.fiit.stuba.sk/TeamProject/2007/team11is-si/download/tp_dokumentacia_final.pdf

[6] RoboKopy: *Tímový projekt: dokumentácia final*. Máj 2010, Bratislava, Slovensko. http://labss2.fiit.stuba.sk/TeamProject/2009/team15is-si/documents/TIMOVY%20PROJEKT_dokumentacia_final.pdf

[7] Boedecker, J.: *SimSpark User's Manual (version 1.1)*, 2008