

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

Obsah.....	2
1 Úvod.....	4
1.1 Zadanie.....	4
1.2 Štruktúra dokumentu.....	5
2 Analýza.....	6
2.1 Analýza 3D tímov.....	6
2.1.1 Kouretes.....	6
2.1.2 Nao Team Humboldt.....	8
2.1.3 FC Portugal.....	9
2.1.4 UT Austin Villa.....	10
2.1.5 SEU-3D.....	12
2.1.6 Nexus 3D.....	13
2.1.7 Agenty 007.....	16
2.1.8 Critical Error.....	18
2.1.9 RoboKit.....	20
2.1.10 Hviezdna jedenástka.....	22
2.1.11 RoboKopy.....	25
2.1.12 Neurotics.....	27
2.1.13 Zhodnotenie tímov.....	29
2.2 Analýza servera.....	30
2.2.1 Architektúra servera.....	30
2.2.2 Simulácia na serveri.....	30
2.2.3 Perceptory.....	31
2.2.4 Efektory.....	34
2.2.5 Robot NAO.....	36
2.3 Agent JIM.....	36
2.3.1 Architektúra agenta.....	37
2.3.2 Zhodnotenie.....	39
2.4 Doplnujúce informácie k analýze.....	39
2.4.1 Multiagentové systémy.....	39
2.4.2 Využitie gyroskopu na vylepšenie stability agenta.....	39
3 Špecifikácia požiadaviek.....	40
3.1 Pohyby agenta.....	40
3.1.1 Vstávanie.....	40
3.1.2 Chôdza.....	41
3.1.3 Otáčanie.....	41
3.1.4 Kop do lopty.....	42
3.1.5 Bránenie.....	42
3.2 Vyššia logika.....	43
3.3 Editor pohybov a správania.....	43
3.4 Definovanie vyššej logiky pomocou XABSL.....	44



3.5	Framework pre efektívne testovanie schopností agenta.....	45
3.5.1	Špecifikácia hlavných funkcií	45
4	Návrh prototypu	46
4.1	Hrubý návrh implementácie pohybov	46
4.1.1	Vstávanie	46
4.1.2	Chôdza.....	46
4.1.3	Otáčanie.....	46
4.1.4	Kop do lopty	47
4.1.5	Bránenie.....	47
4.2	Vyššia logika pohybov agenta.....	47
4.2.1	Chôdza.....	47
4.2.2	Vstávanie	48
4.2.3	Kop do lopty	49
4.2.4	Vedenie lopty	49
4.2.5	Zorientovanie sa	50
4.3	Návrh editora správania.....	50
4.3.1	Editor pohybov	51
4.3.2	Editor správania.....	51
4.4	Vyššie správanie v XABSL.....	52
4.5	Návrh testovacieho frameworku	53
4.5.1	Architektúra testovacieho frameworku	53
4.5.2	Opis blokov architektúry	54
4.5.3	Spoločný jazyk pre <i>požadované konanie a záznam zo servera</i>	55
4.5.4	Informácie k implementácii.....	55
5	Použitá literatúra.....	56



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 rozširovateľnosť, 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 na 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 efekторы 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 poslednej časti je uvedený hrubý návrh prototypu. V kapitole 4 sa teda rozširujú špecifikované požiadavky a určujú sa možné cesty implementácie v prototypu. Najprv sa opisujú pohyby a vytváranie vyšších činností pomocou základných pohybov za účelom dosiahnutia nejakého cieľa (dobehtie 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, zadaného v jazyku XABSL. Na záver je navrhnutý testovací framework pre testovanie schopností 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 riešeníach 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 tímu¹.

Základný prístup tímu je taký, že na agenta (robota) pozerajú ako na zoskupenie agentov. 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 zadaných 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 Editor (editor pohybov znázornený na obrázku 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 Language² (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 Agent³ 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 skontretizovanej fyzikálnej časti.

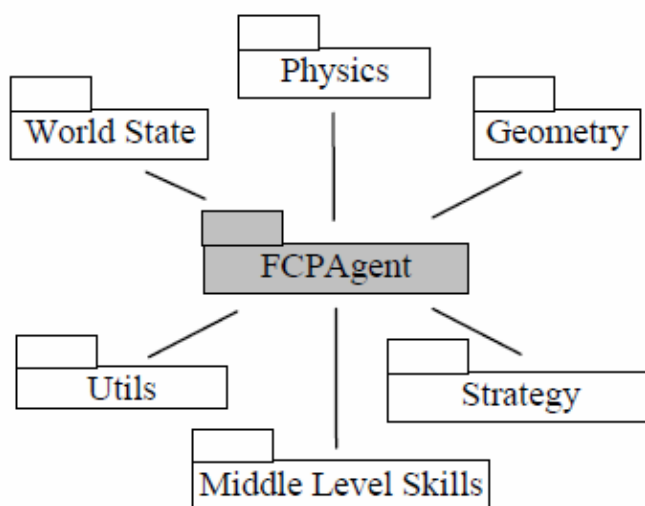
² <http://www.xabsl.de>

³ <http://launchpad.net/simplesocceragent/trunk/0.2/+download/SimpleSoccerAgent.pdf>



2.1.3 FC Portugal

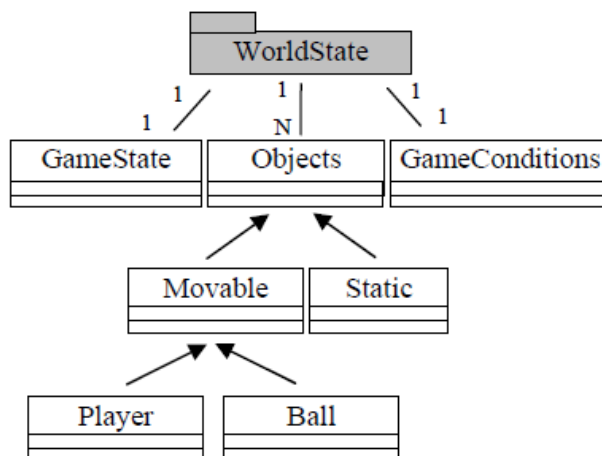
Tím FC Portugal 3D je víťazom súťaže RoboCup World 3D v roku 2006, pričom víťazného agenta aj zverejnil. Zverejnené dokumenty sa zaoberajú predovšetkým stratégiou 2D tímu, ktorý tiež tento tím vyvíjal. Dostupný je však dokument z roku 2004, kde je na dvoch stranách opísaný ich 3D agent. Všetky informácie o tíme sú prevzaté z dokumentácie a publikácií tímu⁴. Obrázok 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. Obrázok 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 priradovanie 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 tímu⁵. Na svojich webových stránkach poskytujú na stiahnutie zdrojové kódy:

- Agentu 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/>



- Agentu 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 protiagentu, vytvára model správania a na základe modelu generuje rady.

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 ku 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 protiagentu (simulovaná liga).



2.1.5 SEU-3D

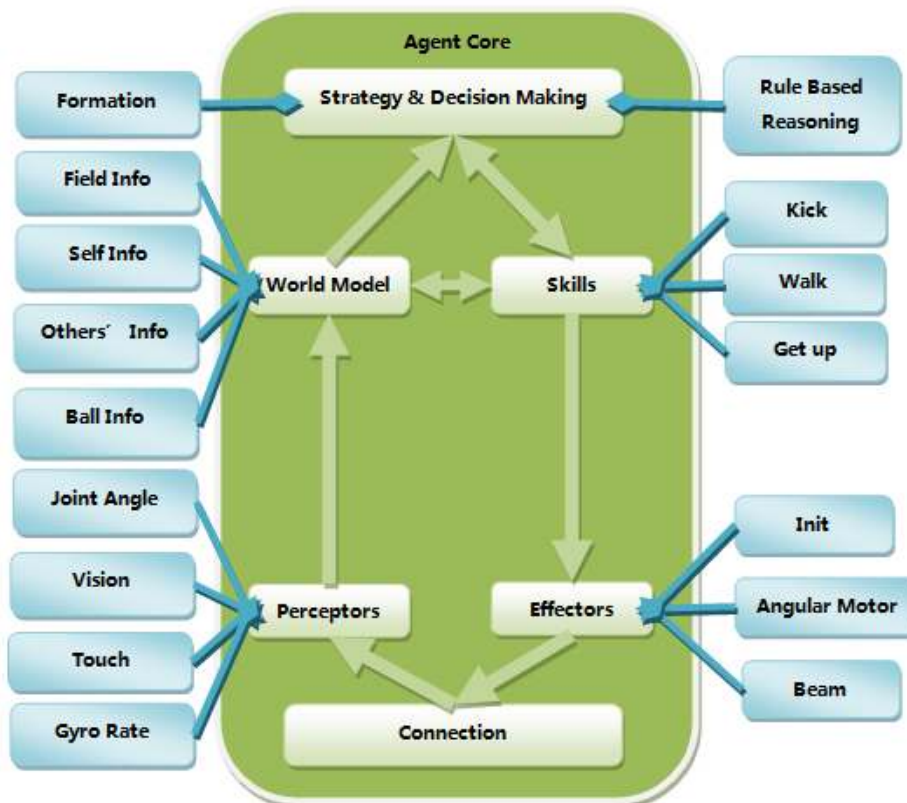
Tím SEU-3D vznikol v roku 2005 na čínskej Southeast University. 3D robotickému futbalu sa tím venuje od roku 2007. V tom istom roku sa umiestnil na treťom mieste vo svetovom pohári tímov robotického futbalu. Tím SEU-3D je zložený z viacerých menších tímov a všetky informácie ohľadom tohto tímu sú prevzaté z tímovej dokumentácie⁶.

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. Strelba 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 obrázku 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-ons. 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 strelbou, logikou a stratégiami. Pri riešení rozhodovania a strelby 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 rýchlejšej a stabilnej humanoidnej chôdze. Všetky informácie o tíme sú čerpané z opisu tímu⁷.

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. Oblíbeným prístupom plánovania trajektórie kĺbov riešiacim problém stability je využitie indikátora stability ZMP (Zero Moment Point).

⁷ <http://nexus.um.ac.ir/Nexus-3d-2007.pdf>



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 skráteneho 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:

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

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 skrátenej Fourierov rad obsahujúci len prvé tri členy trigonometrickej formy Fourierovho radu:

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

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 znižuje 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 skráteneho 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.

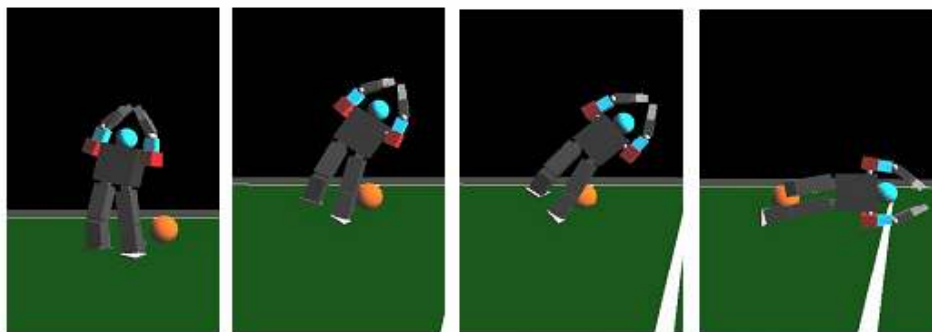
Pohyby niektorých kĺbov boli vo všetkých chôdzach podobné, čo viedlo k odstráneniu ďalších nadbytočných parametrov. Výsledky pokusov ukazujú, že stabilná priama chôdza môže byť vygenerovaná použitím len dvoch kĺbov – bedrového 3 a kolenného. Aj fitness funkcia bola výrazne zjednodušená, aby rýchlejšie chôdze boli ohodnotené lepšie ako stabilné, ale pomalé chôdze. Ďalej bola pri fitness funkcii do úvahy zobrať 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 obrázku 5.



Obr. 5: Brankár v akcii

2.1.7 Agenty 007

Tím Agenty007 je tím z akademického roku 2008/2009. Vyhrál TPCUP v roku 2009. Všetky informácie o tíme vychádzajú z finálnej dokumentácie tímu⁸.

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:

$[X_n, Y_n, Z_n]$ získajNovuPolohuBodu($X_o, Y_o, Z_o, X_p, Y_p, Z_p, \alpha, \beta, \gamma$)

Vstupy:

X_o, Y_o, Z_o – súradnice bodu, cez ktorý prechádzajú osi otáčania

X_p, Y_p, Z_p – počiatočné súradnice bodu

α (os Z), β (os Y), γ (os X) – uhly natočenia bodu okolo daných osí

Výstup:

X_n, Y_n, Z_n – koncové súradnice bodu po aplikovaní natočení okolo jednotlivých osí

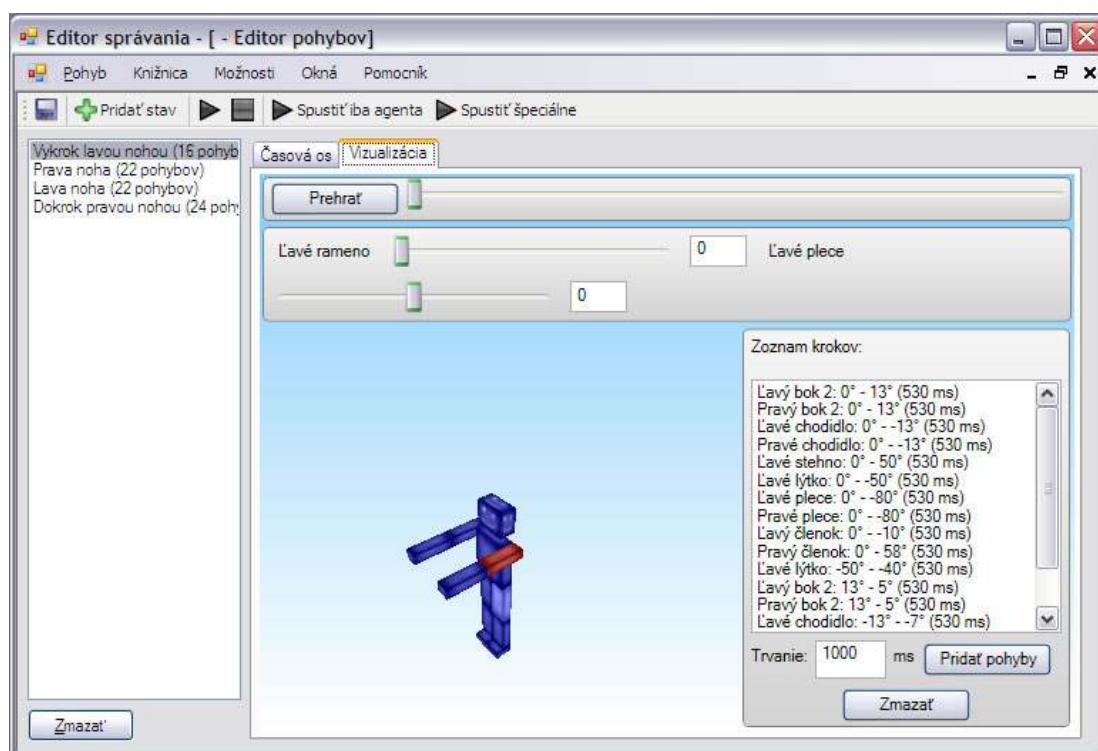
V rovnovážnom module tak sledujú pohyb ťažiska v čase. Samotná implementácia má zopár problémov. Neboli urobené väčšie zmeny na samotných agentoch vzhľadom na tímy z predchádzajúcich rokov. Tím sa najviac sústredil na vytvorenie pomocných nástrojov na spracovanie pohybov agenta, akými sú Editor pohybov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

⁸ <http://labss2.fiit.stuba.sk/TeamProject/2008/team07is-si/dokumentacia2.pdf>



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

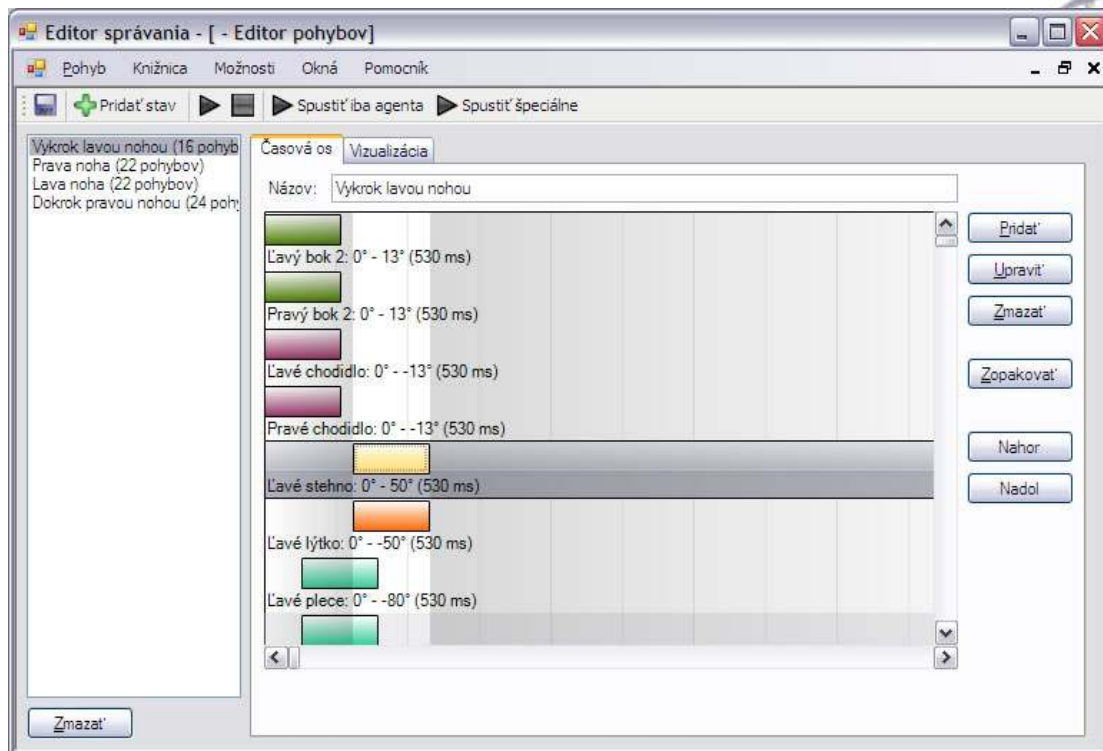
Pohyby v editore sú reprezentované otočením kĺbov, ako je možné vidieť na obrázku 6. Editor umožňuje vytvorenie akéhokoľvek pohybu (vstávanie, beh, chôdza, otočenie,...). Samotné pohyby je možné exportovať do RMO súborov. Tento typ súboru využíva zápis údajov, ktorý je ľahko parsovateľný. Taktiež je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



Obr. 6: Grafický editor pohybov

Ďalšou zaujímavosťou editora je časová os (obrázok 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 funkcionality.

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 tímu⁹.

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

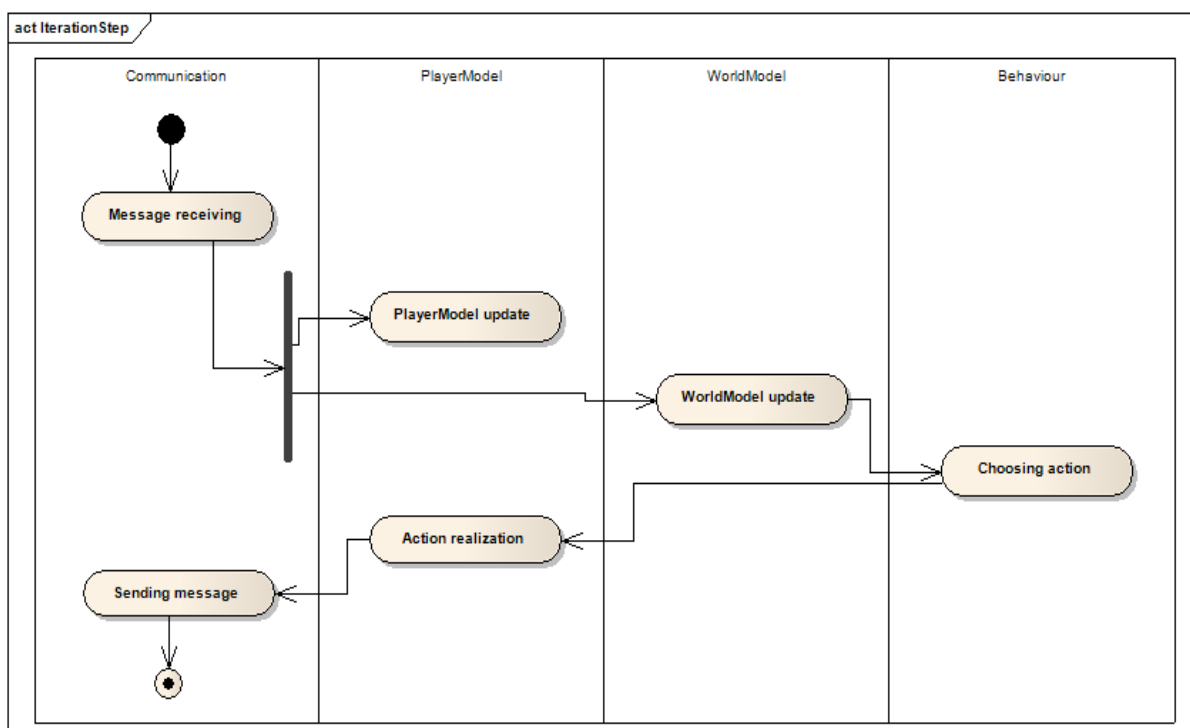
- Pridaním funkcionality do editora pohybov na import a export pohybov vo formáte XML
- Parsovaním (dôvodom bola nekompatibilita pôvodného parsera s novou verziou servera)
- Logovaním agenta
- Pohybmi agenta
- Záznamami o zápase

⁹ <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 obrázku 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 Agentu 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 (prispôbil 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 vyladovaní sa nepodarilo dosiahnuť to, čo očakávali.

Pri implementácii 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 kosru 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

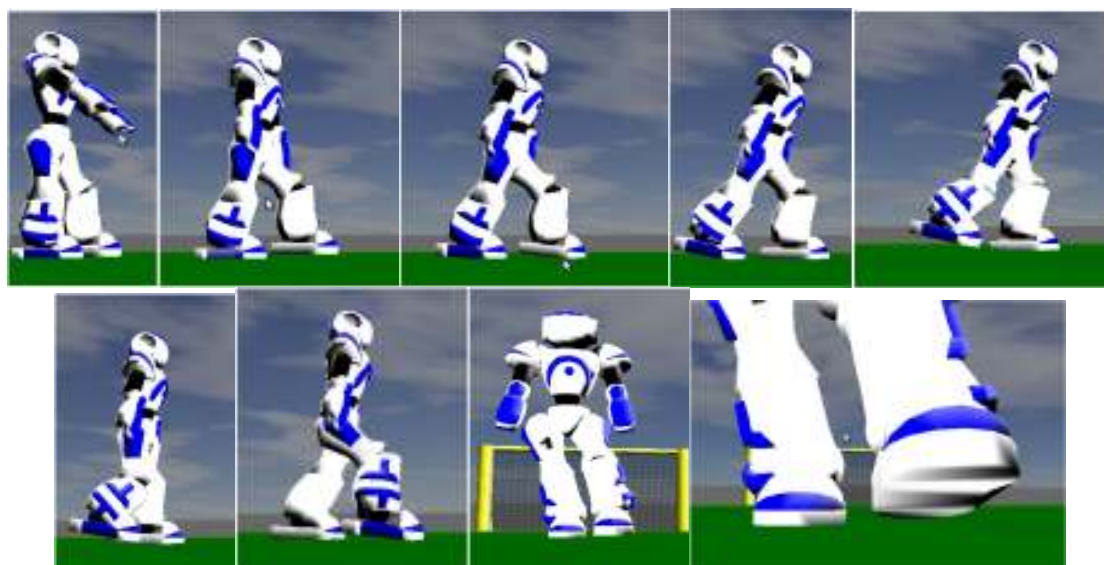
Ako základ si zvolili skorú verziu tímu RoboKopy, kde opravili základné chyby a migrovali na novú verziu servera 0.6.3. Pri tvorbe pohybov bol použitý editor pohybov tímu Agenty007. Vzorom pre jednotlivé pohyby bol človek a jeho pohyby, avšak nie všetky reálne pohyby mohli použiť z dôvodu inej fyziológie 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 navážení 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 neprímerane 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 (obrázok 9).

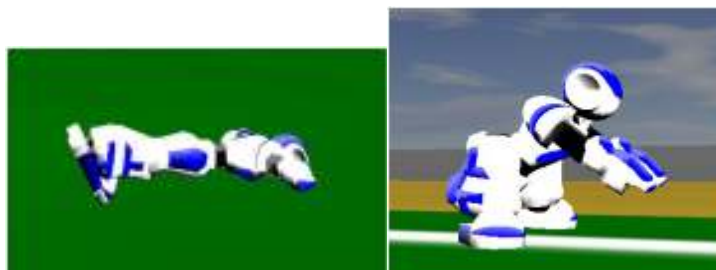


Obr. 9: Vykročenie a dokrok robota

Zdvíhanie robota zo zeme je znázornené na obrázku 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 rozťahne nohy, pokrčí kolena 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 zotrúva, aby nepodľahol oscilácii a nepadol. 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 rozťahne 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, nespádnutie (obrázok 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 z finálnej dokumentácie tímu¹⁰.

¹⁰ <http://labss2.fkit.stuba.sk/TeamProject/2007/team11is-si/>

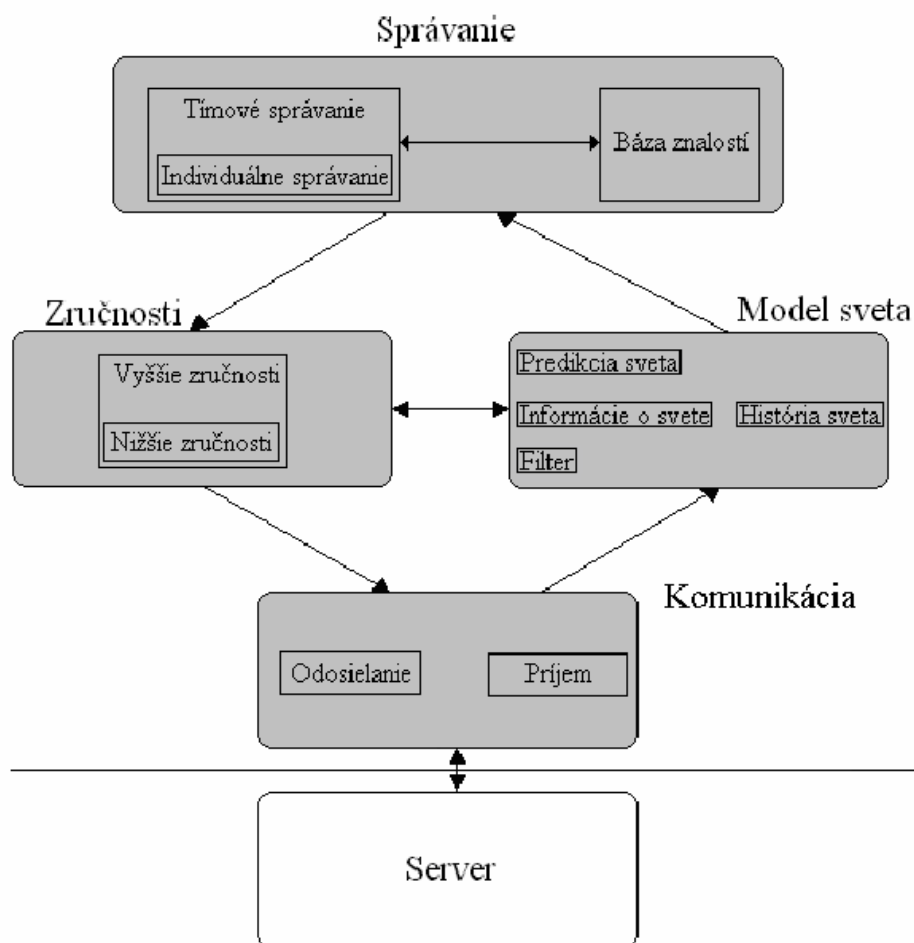


V analýze sa tento tím venoval najmä výberu servera, pre ktorý budú vytvárať svojho agenta (2D, 3D sphere, 3D humanoid). Množinu stanovených požiadaviek tvorili komunikácia agenta so serverom (spoľahlivý TCP verzus rýchly UDP protokol), práca s údajmi (vhodne uložené, odstraňovanie nepresností), správanie (tímové správanie, nekonfliktnosť výpočtov správania a pohybu – vlákna – rýchlosť verzus správnosť), architektúra (s ohľadom na ďalšie rozširovanie, moduly). Architektúra agenta, znázornená na obrázku 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 prílišnej previazanosti kódu pôvodne uvažovaného agenta. Cieľom prototypovania tohto tímu teda bolo najmä vytvorenie základnej štruktúry agenta a overenie ovládania pohybov kĺbov. Agent 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 funkcionality vpisované do vopred pripravených callback funkcií (definovať, čo sa má s prijatými vyparovanými dátami urobiť).



Obr. 12: Navrhnutá architektúra agenta tímu Hviezdna jedenástka

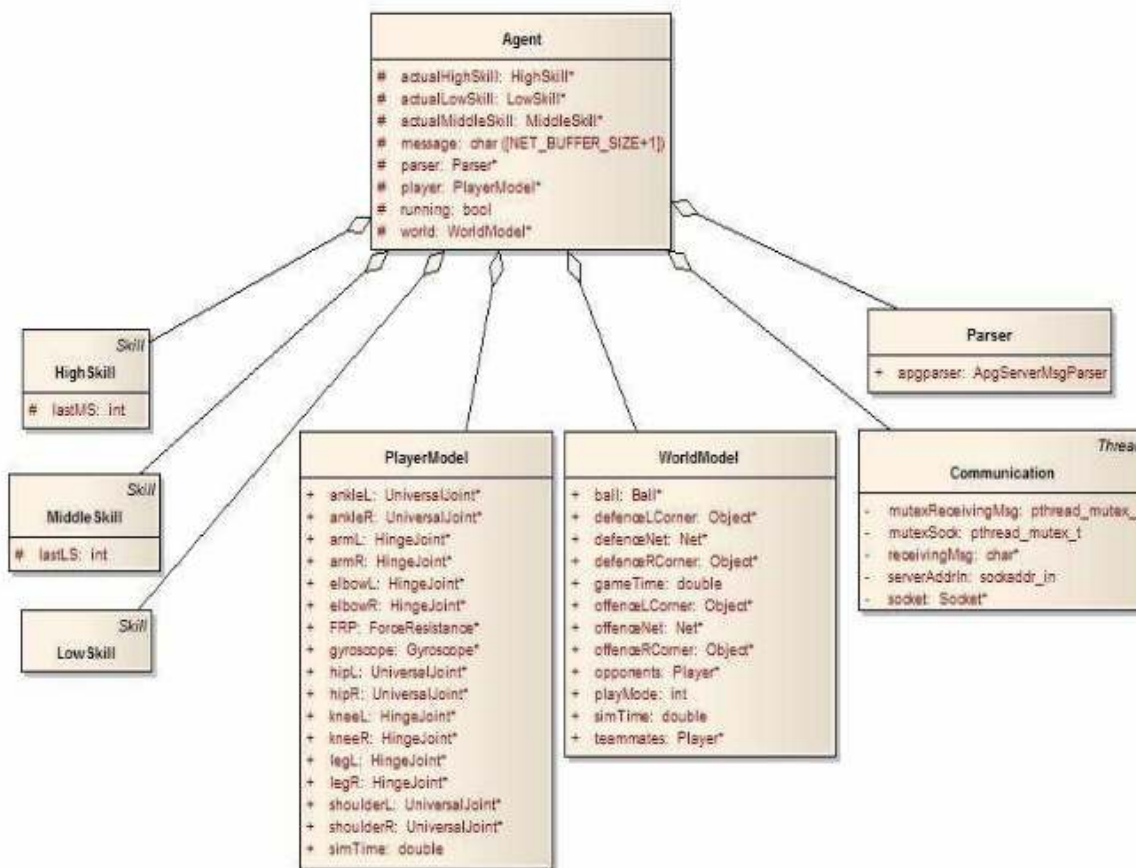
Na komunikáciu boli implementované triedy TCPSocket a UDPSocket, aby agent mohol pomocou daných protokolov so serverom komunikovať (v samostatnom vlákne). Bola vytvorená aj spoločná trieda Socket, od ktorej obe spomínané triedy dedia. Komunikácia prebieha pomocou správ v určenom formáte – prvé štyri 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 obrázku 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 z finálnej dokumentácie tímu¹¹.

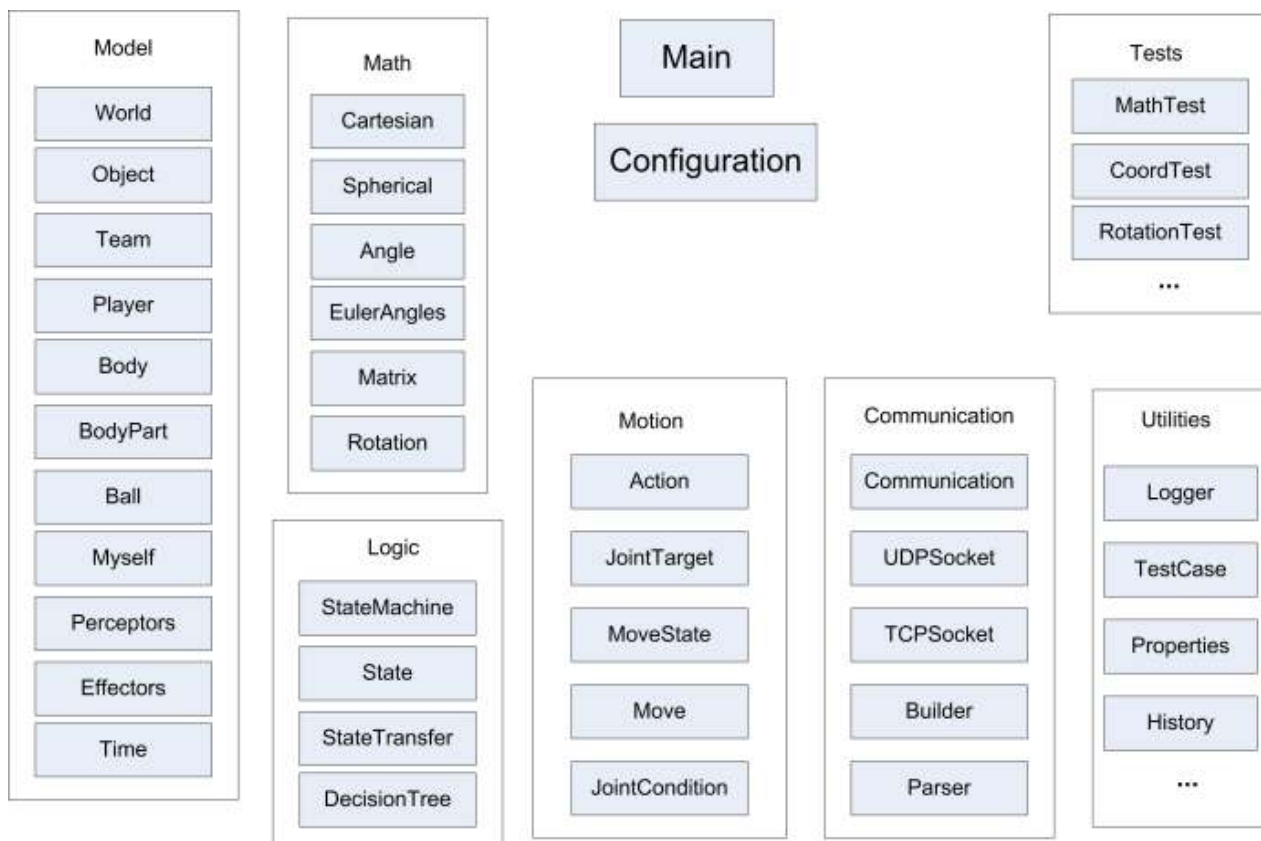
Samotný projekt bol rozdelený do niekoľkých modulov (obrázok 14):

- 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

¹¹ <http://labss2.fiiit.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, súradnic ťažiska a častí tela a jednoduchú predikciu pohybu na niekoľko simulačných krokov
- Históriu kinematických údajov pre každý dynamický objekt
- Správanie pomocou stavového automatu
- Predpoklady pre pohyby – rozšírenie formátu RMO (editor tímu Agenty 007), načítanie a procedurálne vykonávanie predpokladov
- 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 rozširiteľná 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 oľhľadom tímu sú prevzaté z tímovej dokumentácie¹².

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 diskretných 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.

¹² <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 $<0;1>$ pre násobenou 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 soкетов 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 upravujú 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 funkcionálnu 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 tréňovaním 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 frameworku 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. Agentu 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 [1].

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 *simsark.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 beznamienkové 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

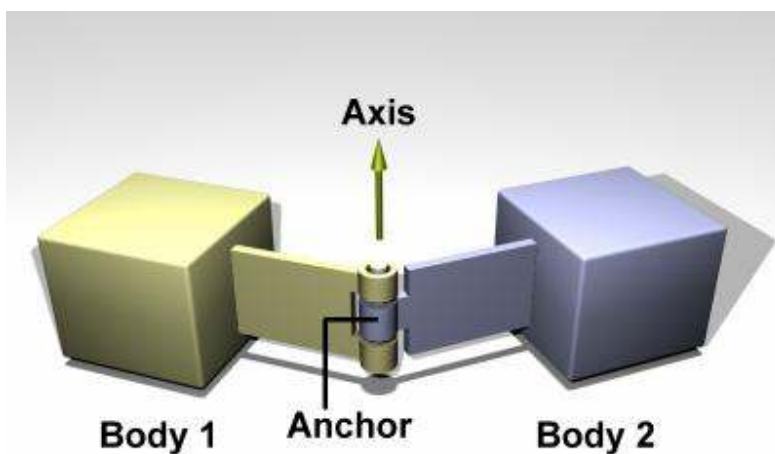


sústave. Robot NAO má tento perceptor umiestnený v hornej časti tela. Formát správy pre tento perceptor vyzerá nasledovne:

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 obrázku 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 vyzera 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 neprebíha 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 efekторы ovládajú perceptory, ktoré sú špecifické pre simulovaný futbal. Tieto efekторы 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.

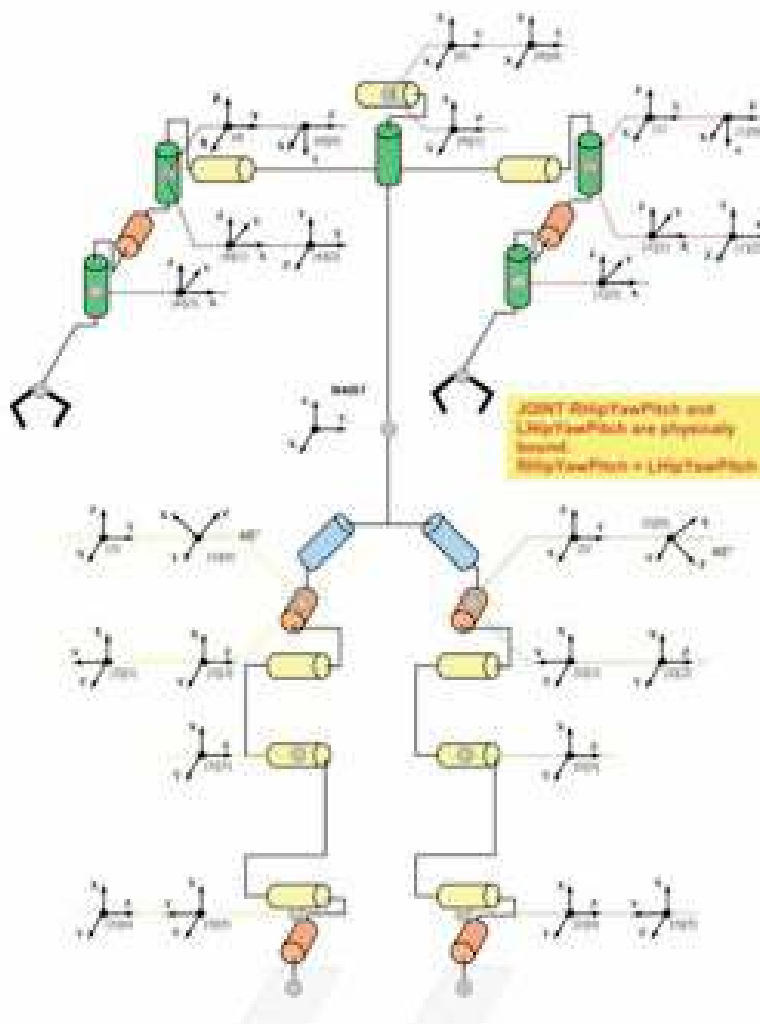
Beam Effector. Tento efektor musí byť zavolaný ešte pred začiatkom hry a určuje umiestnenie hráča na hraciu plochu po jeho inicializácii.

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 obrázku 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 dispozícii 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:

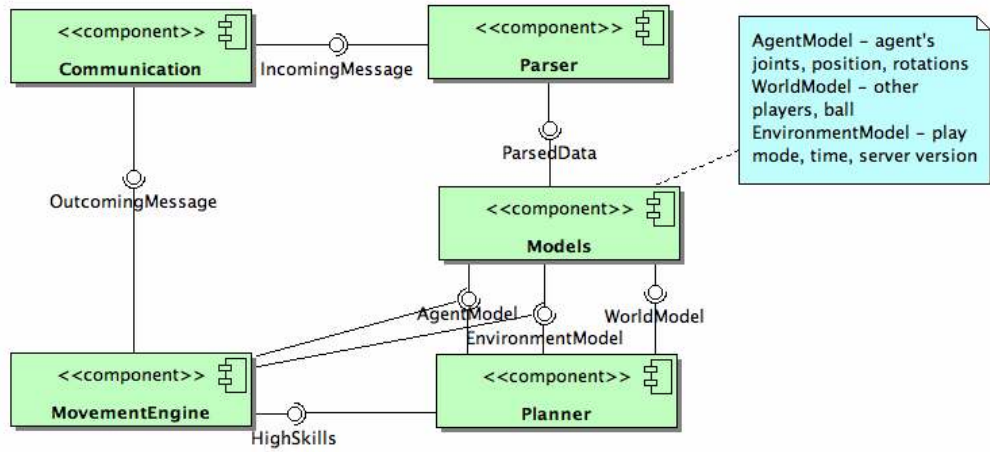
- Štruktúrovaný, vhodne organizovaný
- Obsahujúci moderné techniky - anotácie, testy
- Dokumentovaný na úrovni súborov, nie konkrétnych metód

2.3.1 Architektúra agenta

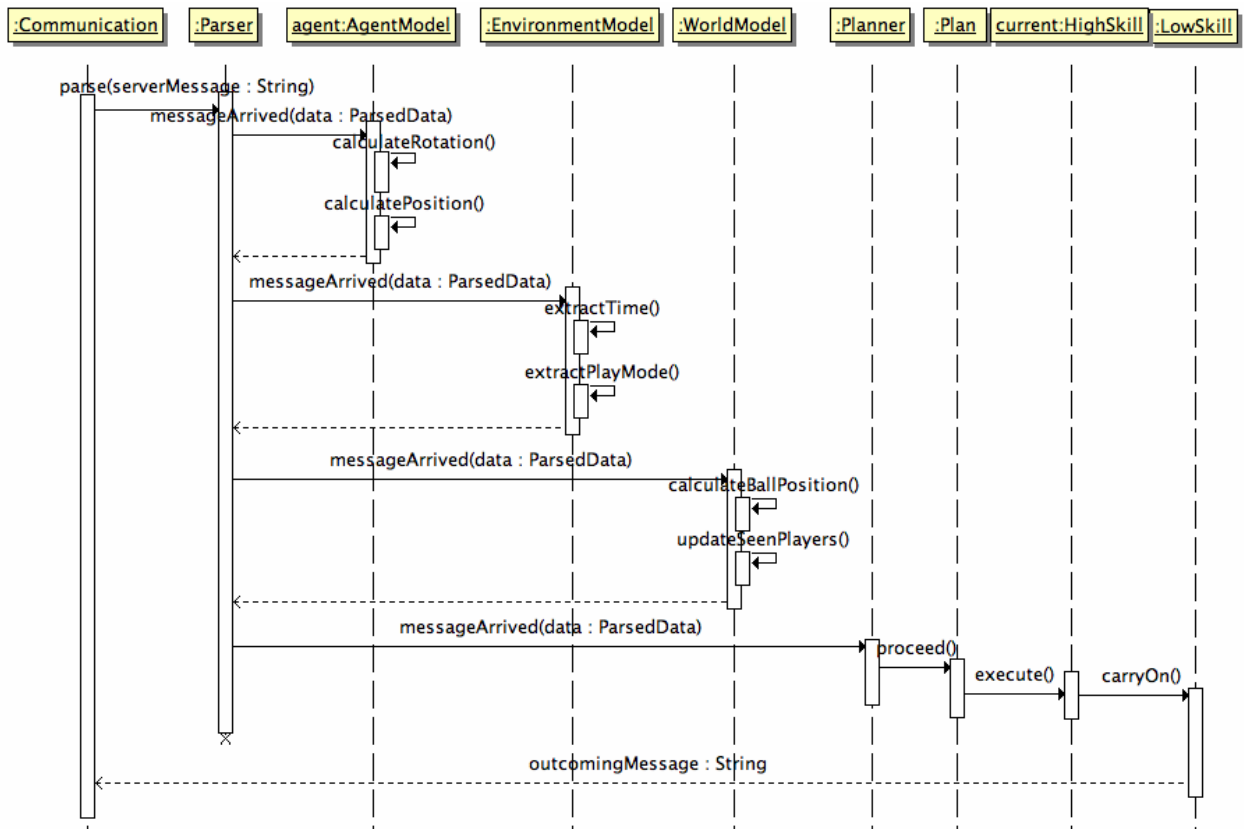
V nasledujúcej časti na základe informácií z repozitára prehľadne zhodnotíme agenta.

Komponenty agenta. Agentu 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 obrázok 17, kde sú názorne vyjadrené vzťahy medzi komponentmi.

Hlavný cyklus komunikácie. V predchádzajúcej časti bolo ukázané, z akých komponentov sa skladá systém. Obrázok 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 Doplnujúce 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

Potrebuje 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 zadaným prostredím, keďže neobsahujú žiadnu formu vyššej inteligencie, ktorá by im dovolila prispôbovať 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. Potrebuje sa 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 oblastí. 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 funkcionality, 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 „trávníku“. 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 kráčať a nepadol. 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* - predstavuje 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 polohy 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 zadané.

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 oboch 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 zadaných 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 prototypu 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 prototypu sa budeme zaoberať oboma spôsobmi pohybu.

4.1.3 Otáčanie

Existujú XML súbory pre otáčanie do oboch strán, preto v prototypu budeme tieto pohyby testovať a vylepšovať. Z vlastných navrhovaných spôsobov otáčania v prototypu 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 prototypy 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.

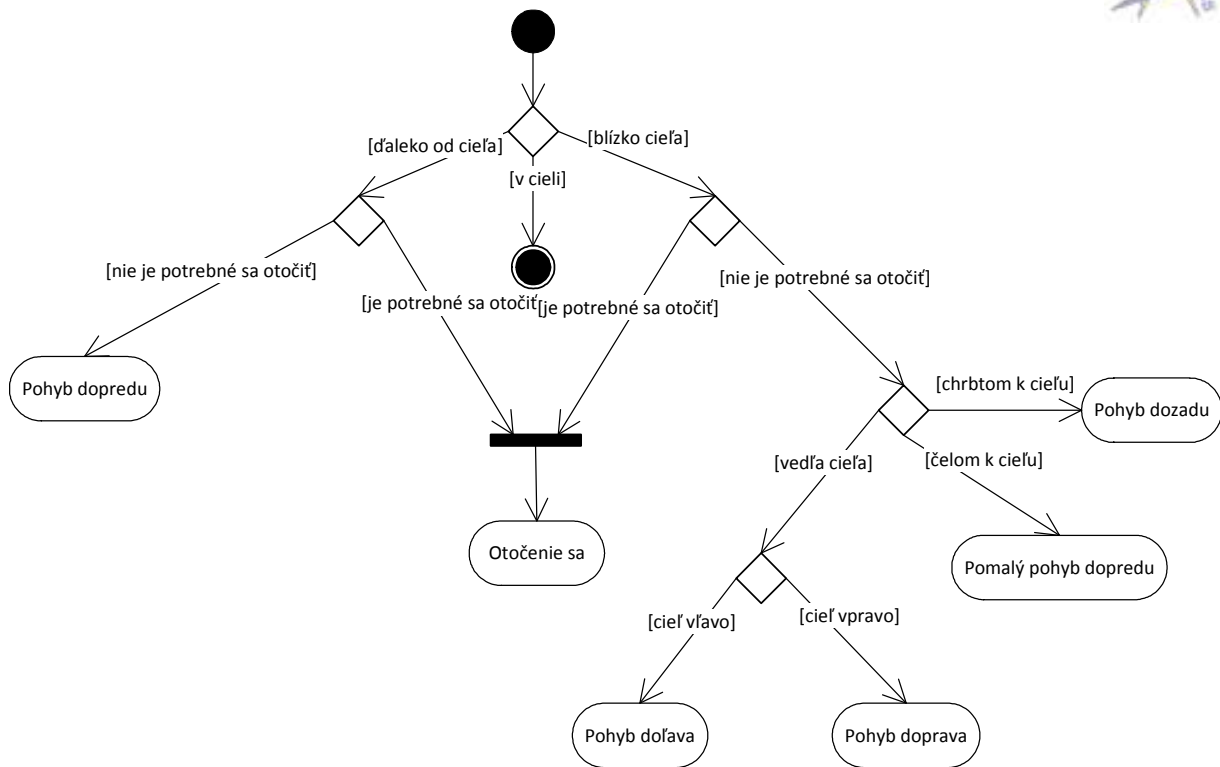
4.2.1 Chôdza

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

Tabuľka 1: Vstupné parametre chôdze

Parameter	Opis
Cieľ	Bod, kam agent smeruje

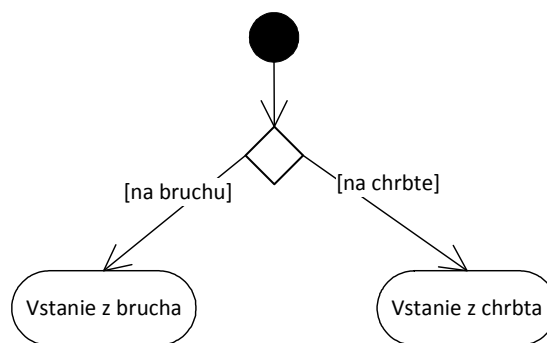
Návrh chôdze vychádza z požiadavky minimalizovať jej trvanie. Z tohto dôvodu je výber akcií závislý od vzdialenosti agenta od cieľa. Ak je táto vzdialenosť dostatočne malá, agent bude môcť využiť chôdzu do strán, či dozadu. Inak sa agent otočí na požadovaný uhol a pôjde rovno. Postup je znázornený na obrázku 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 obrázok 20.



Obr. 20: Vstávanie agenta

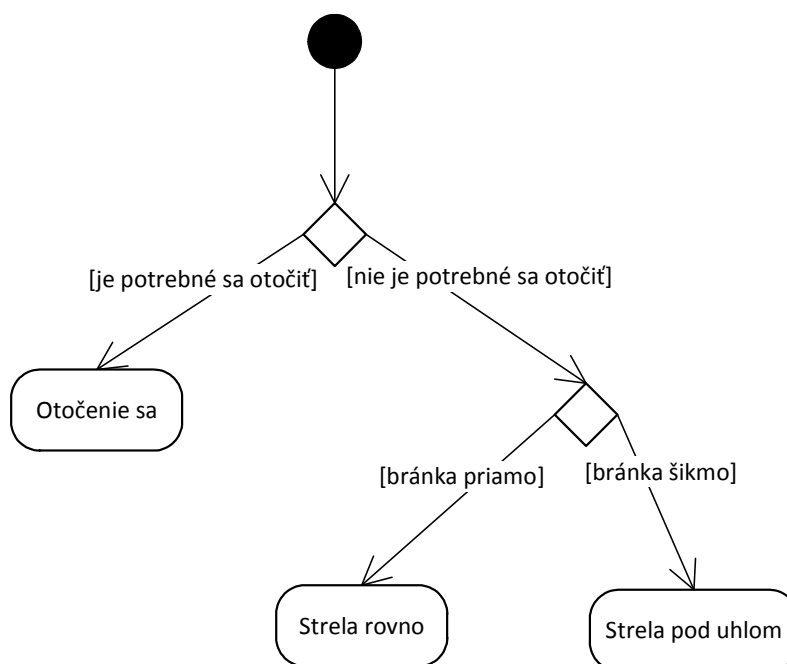


4.2.3 Kop do lopty

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 obrázku 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)



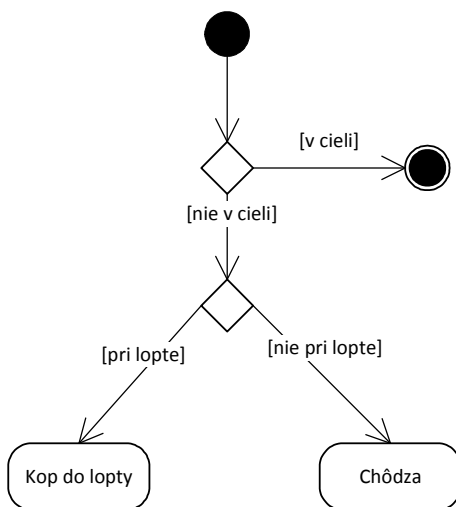
Obr. 21: Kop do lopty

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 obrázku 22.

Tabuľka 3: Vstupné parametre vedenia lopty

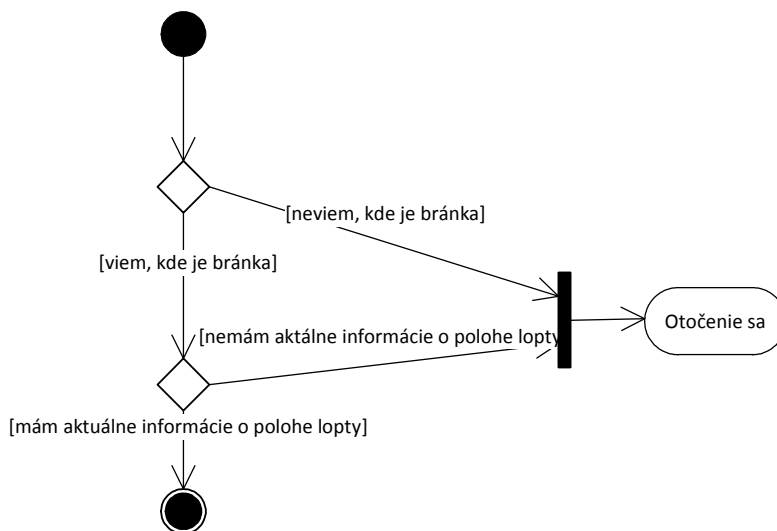
Parameter	Opis
Cieľ	Bod, kam agent s loptou smeruje



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 obrázku 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 funkcionálnosť, 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 štruktúru 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.



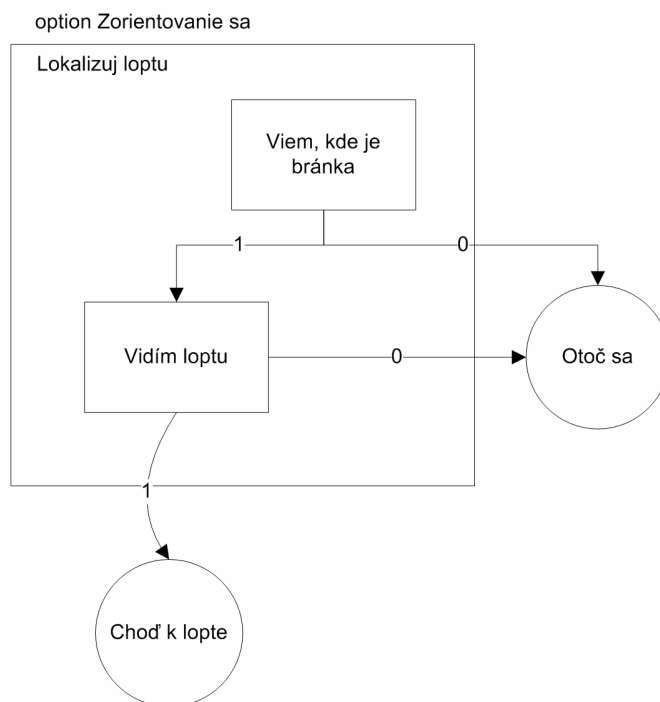
4.4 Vyššie správanie v XABSL

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

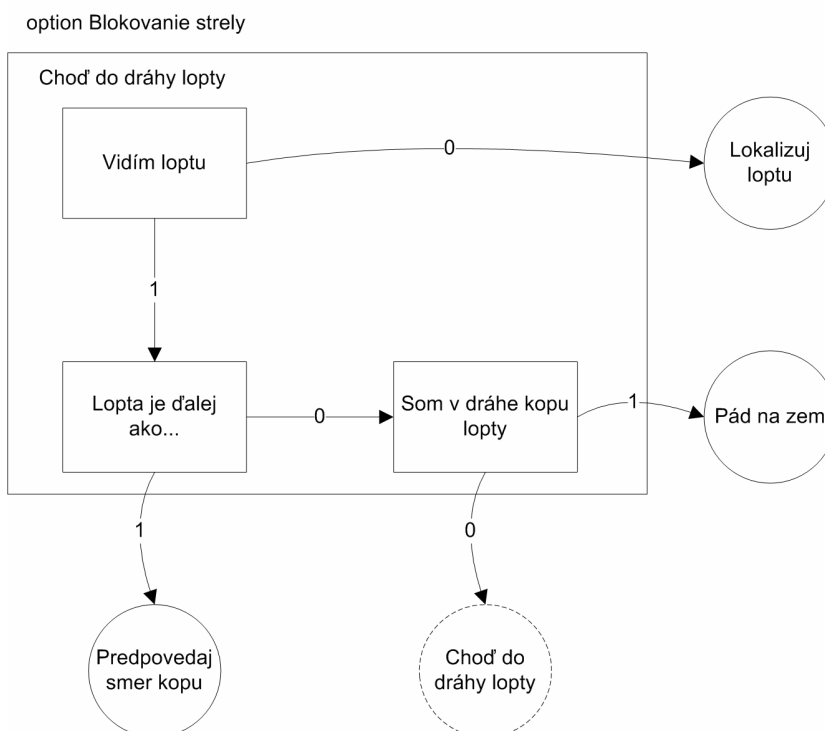
Agent v XABSL pozostáva z menších celkov – tzv. *options*, čo sú vlastne moduly správania, z ktorých každý je konečným automatom. V našom prípade pôjde o:

- 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 obrázkoch 24 a 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)

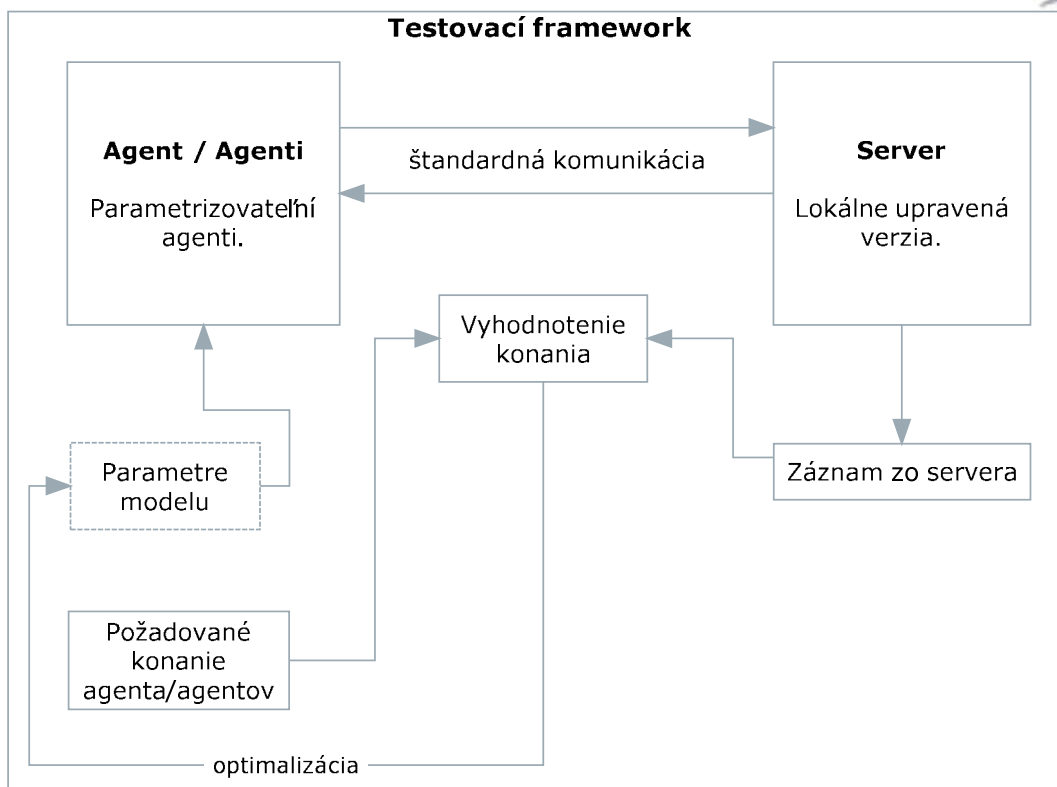
4.5 Návrh testovacieho frameworku

Na úspešné splnenie požiadaviek uvedených v špecifikácii 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 docieľiť efektívnejšie testovanie agenta.

4.5.1 Architektúra testovacieho frameworku

Nakoľko požadovaná funkcionálna 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 obrázku 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 obrázku 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 kľbov, 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 Použitá literatúra

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