

RoboCup 3D

Tímový projekt



Autori: Bc. Peter Nosko
Bc. Dušan Rodina
Bc. Daniel Slamka
Bc. Peter Smolinský
Bc. Ondrej Ševce
Bc. Ivan Tomovič

Tím: Agenty 007 (tím č.7)

Študijný odbor: Informačné systémy a Softvérové inžinierstvo

Akademický rok: 2008/2009

Vedúci tímu: Ing. Marián Lekavý

Obsah

1	Úvod.....	4
2	Analýza.....	5
2.1.	Analýza starších tímov	5
2.1.1.	Tím Little Green Bats.....	5
2.1.2.	Tím UIAI.....	7
2.1.3.	Tím Austin Villa.....	8
2.1.4.	Tím Virtual Werder 3D	11
2.2.	Analýza servera SimSpark	16
2.2.1.	Architektúra servera	16
2.2.2.	Práca servera	16
2.2.3.	Spustenie simulácie	16
2.2.4.	Komunikácia medzi serverom a agentom	17
2.2.5.	Perceptory.....	17
2.2.6.	Efektory	20
2.2.7.	Správania agenta	22
2.2.8.	Simulácia.....	22
2.2.9.	Robot NAO	23
2.3.	Neurónové siete a ich využitie	24
2.4.	Zhodnotenie analýzy	25
3	Špecifikácia	26
4	Návrh riešení.....	27
4.1.	Rovnovážny modul	27
4.1.1.	Rovnovážna rovnica	27
4.1.2.	Diferenciálna rovnica	27
4.1.3.	Fyzika pohybov	27
4.1.4.	Zhodnotenie.....	28
4.2.	Prototyp rovnovážneho modulu	28
4.2.1.	Využitie prototypu.....	29
4.3.	Editor pohybov robota.....	29
4.3.1.	Prostredie editora.....	29
4.3.2.	Tvorba pohybov	30
4.3.3.	Náhľad pohybu	31

4.4.	Prototyp editora pohybov	31
4.4.1.	Prostredie programu	31
4.4.2.	Nastavenie editora	32
4.4.3.	Vytváranie akcií	32
4.4.4.	Náhľad vytvorenej akcie	33
4.5.	Riadenie pohybov robota	33
4.5.1.	Triedy	34
4.5.2.	Problémy pri implementácii	37
4.5.3.	Parametre príkazového riadka	37
5	Implementácia	39
5.1.	Editor správania.....	39
5.1.1.	Vytváranie pohybov	41
5.2.	Vykonávanie pohybov.....	46
5.3.	Message Parser	48
5.3.1.	Implementácia spätnej väzby (Server -> Editor).....	48
5.3.2.	Zaznamenávanie správ agentom	48
5.3.3.	Parsovanie správ v editore.....	48
5.3.4.	Vykonanie pohybu v editore	49
6	Plán do budúcnosti	50
6.1.	Vyššia vrstva riadenia	50
7	Testovanie	52
7.1.	Testovanie prevodov medzi rôznymi sústavami:	52
7.2.	Testovanie rotácie vektorov	53
7.3.	Testovanie ťažiska a oporných bodov	53
7.4.	Testovanie gyroskopu z dát posielaných serverom.....	54
7.5.	Testovanie kladného uhla medzi vektormi.....	55
7.6.	Testovanie konvexnej plochy	55
7.7.	Testovanie pohybov	56
8	Zhodnotenie	57
9	Použitá literatúra.....	58
	Príloha A – Prílohy k editoru pohybov	I
	Príloha B – Extended abstract do konferencie IIT SRC	III
	Príloha C – Externé testovanie.....	V
	Príloha D – Elektronické médium	VII

1 Úvod

RoboCup je medzinárodný projekt, ktorého zámerom je propagovať umelú inteligenciu, robotiku a príbuzné oblasti [5]. Hlavným cieľom projektu RoboCup je do roku 2050 vytvoriť tím samostatných robotov, ktorí by dokázali vyhrať nad najúspešnejším tímom sveta. V súčasnosti existujú v RoboCupe tieto druhy líg:

- Liga malých robotov
- Liga stredne veľkých robotov
- Liga štvornohých robotov
- Liga humanoidných robotov
- Simulovaná liga

Predkladaný dokument bol vytvorený v rámci predmetu Tímový projekt a venuje sa problematike simulovaného 3D robotického futbalu s názvom RoboCup 3D. Cieľom dokumentu je priblížiť priebeh riešenia daného projektu a podrobne opísať jeho jednotlivé etapy. Dokument je rozdelený na jednotlivé logicky súvisiace časti.

Analýza je venovaná opisu servera, ako funguje, z čoho sa skladá a taktiež aj opisu hráča. Ďalej sú tu analyzované rôzne tímy, či už úspešné a aj tie menej výrazné. V špecifikácii sú opísané požiadavky, ktoré sme si stanovili splniť do konca letného semestra. V návrhu je opísaná architektúra hráča, rozdelenie do modulov, rôzne podporné prostriedky a diagramy. Na konci dokumentu sa nachádza krátke zhrnutie toho, čo sme doposiaľ stihli spraviť a ako postupujú naše práce na projekte.

2 Analýza

2.1. Analýza starších tímov

2.1.1. Tím Little Green Bats

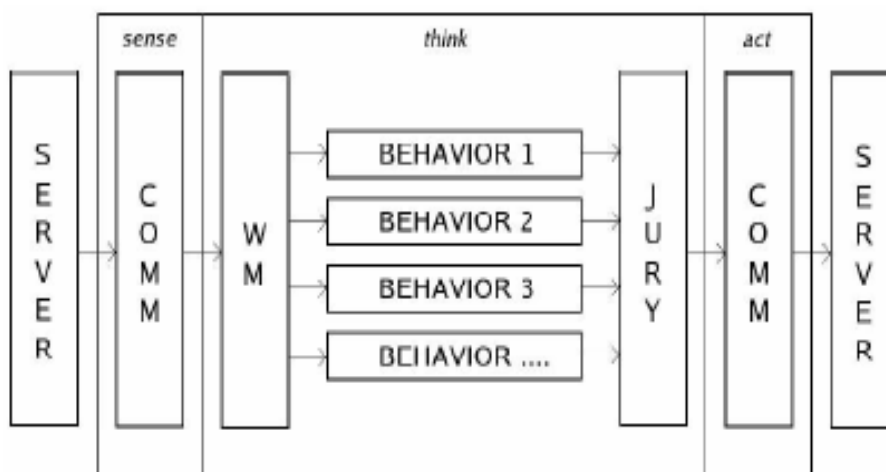
Nakoľko momentálne tím Little Green Bats nemá zverejnený detailný popis architektúry nového hráča, ale iba určité časti, tak v analýze sú vybrané niektoré dôležité myšlienky, s ktorými by sa dalo inšpirovať a umožnili by iný pohľad pri návrhu architektúry hráča.

O tíme

Tento holandský tím je relatívne nováčik v 3D simulovanej robotickej lige. Tím pozostáva zo siedmich študentov študujúcich umelú inteligenciu na Groningenskej univerzite. Pracovať na RoboCup-e 3D začali v roku 2005. V roku 2006 sa zúčastnili majstrovstiev sveta v Brémách, kde sa prebojovali do druhého kola. V roku 2007 sa kvalifikovali aj na majstrovstvá sveta v Atlante, kde skončili druhí. Zúčastnili sa ešte šampionátu Latin American Open 2007 v Mexiku, ktorý vyhrali. Najbližšie sa chystajú na turnaj do Rakúska, ktorý sa začína 29.6.2009 o polnoci a končí 5.7.2009 o 23:59 [4].

Architektúra

Samotný hráč pozostáva z 3 vrstiev. Ďalej sa tu ešte nachádza komunikácia so serverom. Toto je dobre vidieť na Obr. 1.



Obr. 1 - Architektúra podľa LGB [10]

Model sveta

Na obrázku (Obr. 1) je označená ako WM. Ako je vidieť z obrázku, so serverom komunikuje cez komunikačnú vrstvu a uchováva v sebe údaje o svete, teda obsahuje všetky informácie, ktoré hráč dostáva zo servera (pozície).

Správanie

Na obrázku (Obr. 1) je označená ako vrstva BEHAVIOR [6]. Architektúra správania je založená na:

- Implementácia rôznych správání
- Vytvorenie konfiguračného súboru a uloženie správání v určitej hierarchickej štruktúre
- Vytvorenie a spustenie správania
- Subsprávania sú uložené v rozhodovacom strome, kde uzly sú typu or a typu and
- neusporiadaná skupina objektov
- nezávislé správanie nemôže komunikovať s iným nezávislým správaním, ale dostáva príkazy, resp. úlohy od správania z vyššej úrovne a takto sa zadávajú podúlohy na dosiahnutie vyššieho cieľa pomocou nižších subsprávání
- komunikovať sa môže iba s modelom sveta alebo s ďalšou vrstvou
- dáta prenášané medzi vrstvami pozostávajú z aktuálnej pozície., cieľovej pozície, a hodnoty z intervalu $\langle -1, 1 \rangle$:
 - nad 0 - do danej pozície je výhodné sa dostať
 - 0 - nie je dobre sa dostať do tejto pozície
 - pod 0 - má vykonať presný opak tohto správania

Vrstva Jury (porota):

Reprezentuje vrstvu označovanú ako JURY. Táto vrstva reguluje a kombinuje svoj vstup (výstup správania) podľa hernej činnosti. Posudzuje, ktoré správanie by bolo možné a vhodné využiť, prípadne ktoré správania by bolo vhodné skombinovať. Na základe tejto analýzy sa vyberú správania, ktoré sa vykonajú.

Niektoré typy správání [10]:

- ballTowardsGoal – snaha dostať hráča s loptou blízko súperovej brány a následne vystreliť na bránu
- stayInField –stará sa o to, aby hráč nevybehol mimo hraciu plochu. Hráč nemá vybehnúť ani vtedy, keď sa stavia za loptu, ktorú chce prihrať alebo odkopnúť.

- stayInFormation – zabezpečuje pomocou XML konfiguračného súboru, aby hráč zotrval v aktuálnej formácii.
- awayFromTeammates – sa stará o to, aby hráč neostával pri spoluhráčoch, aby spolu pokryli väčšiu časť ihriska.
- Pass – zabezpečuje kopnutie lopty spoluhráčovi, ktorý je vo výhodnejšej pozícii, aby hra tímu bola efektívnejšia.
- Dribble – hráč sa snaží driblovať s loptou, pokiaľ pri ňom nie je súperov hráč. Sila kopnutia lopty je tak limitovaná, aby ju hráč vedel dobehnúť.

Budúca práca tímu Little Green Bats

Tím chce implementovať viacero správanií, aby boli hráči lepší a vedeli vykonávať zložitejšie akcie. Konkrétne sa budú zaoberať zlepšovaním prihrávok. Je možné, že implementujú aj genetický algoritmus, ktorý by mal hráčov naučiť lepšie hodnotiť správania, ktoré sa majú použiť. Budú sa snažiť vylepšiť ešte aj komunikáciu so serverom.

2.1.2. Tím UIAI

Architektúra riešenia

Architektúra riešenia tímu UIAI [8] je rozčlenená do 3 vrstiev – komunikačnej, manažmentu akcií a rozhodovacej. Najnižšia vrstva sa stará o komunikáciu so serverom. Vrstva manažmentu akcií obsahuje model sveta a manažér akcií, ktorý spracováva stavy agenta, prechody a akcie. Najvyššia vrstva (rozhodovacie) zabezpečuje zručnosti strednej a vyššej úrovne.

Podobnú jednoduchú vrstvomú architektúru by sme mohli využiť aj pri našom riešení. Použili by sme hotovú komunikačnú vrstvu, ktorú vytvorili naši kolegovia z fakulty. Pri práci na našom projekte by sme sa viac sústredili na vyššiu vrstvu, ktorú by sme modulárne členili.

Schopnosti agenta

Tím UIAI má na dobrej úrovni spracované zručnosti vstávania robota a chôdze.

Vstávanie

Tím UIAI v svojom agentovi vyriešil vstávanie v takmer všetkých prípadoch. Algoritmus vo svojej dokumentácii presnejšie neopisujú, preto bude nutné ho iba vizuálne sledovať a následne odvodiť od neho vlastné riešenie.

Chôdza

Algoritmus stabilnej chôdze robota tímu UIAI je dobre zdokumentovaný a môžeme sa ním inšpirovať pri implementovaní chôdze v našom riešení. Tento algoritmus by sme mohli ďalej rozvíjať a vylepšovať.

- Pozostáva z troch základných krokov:
- Pomocou členkových kĺbov nakloníme robota do boku
- Robot vykročí opačnou nohou ako je jeho naklonenie
- Robot presunie celé svoje telo dopredu

2.1.3. Tím Austin Villa

Predstavenie tímu

Austin Villa [7] je americký tím Texasu, katedry informatiky na University of Texas at Austin. Tím sa skladá iba z dvoch členov, študenta a profesora. Shivaram Kalyanakrishnan je študentom .2. stupňa štúdia (keďže na webovej stránke má staršie informácie, teraz už asi je študentom doktorandského štúdia). Zaujíma sa hlavne o znalostné štúdie v oblasti učenie sa s odmenou a trestom (angl. reinforcement learning). Simulovaný robotický futbal si vybral ako oblasť pre testovanie svojho výskumu. Druhým členom tímu je docent Peter Stone pôsobiaci v oblasti umelej inteligencie.

História tímu

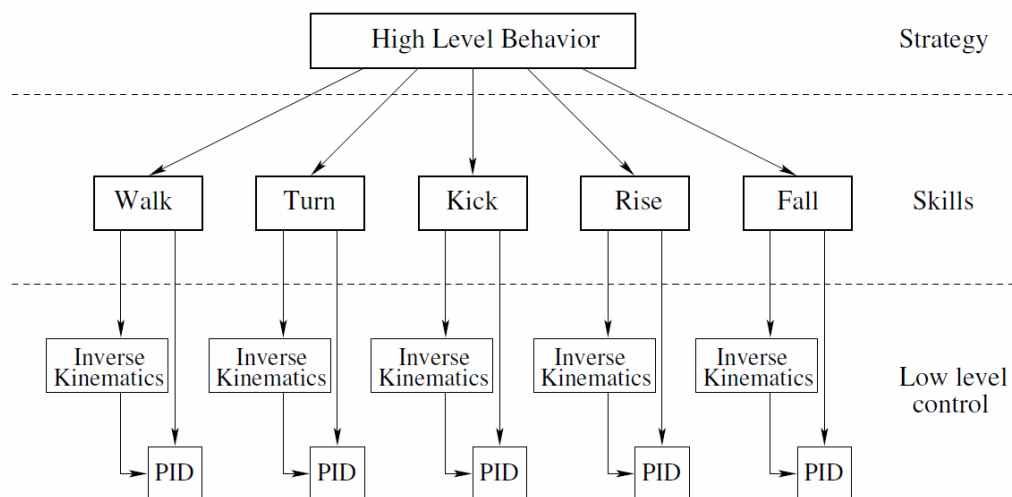
Jedná sa o relatívne mladý tím, ktorý vznikol v roku 2007. Aj z tohto dôvodu tím nedosiahol zatiaľ výraznejší úspech. Tím sa zúčastnil súťaže RoboCup 2007 v Atlante, U.S.A. v júli 2007. Jednalo sa skôr o začiatok simulovaného 3D robotického futbalu na univerzite, kde tím predstavil svoje riešenie a nápady. Tiež sa zúčastnili súťaže RoboCup 2008 v Suzhou, China v júli 2008. O dosť lepšie sa darí tímom z danej univerzity v oblasti simulovaného 2D robotického futbalu, kde tímy dosahujú prvenstvá na súťažiach aj vďaka už dlhšej dobe pôsobenia v danej oblasti.

Architektúra agenta

Tím Austin Villa pracoval iba so starším typom hráča. Napriek tomu si bližšie rozoberieme ich výsledky a zameriame sa na podstatné myšlienky, ktoré môžu platiť aj pre nový model hráča.

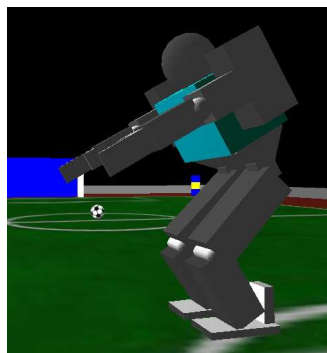
Architektúra ich agenta je zakreslená na obrázku (Obr. 2). Na najnižšej úrovni je agent kontrolovaný pomocou krútiacich momentov (uhl'ových rýchlostí otáčania) jednotlivých

kĺbov. Tím Austin Villa toto implementoval pomocou tzv. PID ovládačov, ktoré berú ako vstup požadovaný uhol natočenia kĺbu a vypočítajú vhodnú uhľovú rýchlosť otáčania. Ďalej používajú inverznú kinematiku pre ovládanie rúk a nôh. Podľa požadovanej pozície a pozície nôh a dlaní, ich inverzná kinematika vypočíta pomocou trigonometrických vzťahov uhly potrebné pre kĺby pozdĺž ruky alebo nohy potrebné pre dosiahnutie daného cieľa, ak je to možné. PID ovládač a inverzná kinematika sú primitívami slúžiacimi pri agentových schopnostiach.



Obr. 2 - Architektúra agenta [7]

Tím zistil, že hráč je oveľa viac stabilnejší pri vykonávaní svojich schopností ako sú chodenie alebo otáčanie sa, ak je mierne naklonený (znázornené na Obr. 3) Preto sa snažili hneď na začiatku ho dostať do takejto pozície a následne ho aj v tejto pozícii udržať. Keďže sa však jedná o staršiu verziu serveru, bude potrebné, aby sme ento poznatok overili aj na novšej. Môžeme však predpokladať, že aj v tomto prípade bude existovať poloha, v ktorej bude hráč stabilnejší pre vykonávanie väčšiny jeho schopností. Tá pravdepodobne bude mierne iná ako pre staršiu verziu hráča.



Obr. 3 - Stabilná poloha hráča pre vykonávanie schopností [7]

Tím sa nepokúšal vyvinúť pokročilé sofistikované stratégie pre tím hráčov hlavne kvôli nedostatku času a tiež kvôli malému množstvu vytvorených schopností hráča. Pri súťaži mali nastaveného brankára na logiku, kedy nehybne stál v bránke a pri strele, ktorá potenciálne môže spôsobiť gól, brankár padol na príslušnú stranu, čím sa pokúsil zachytiť loptu. Ostatní hráči sa vždy pokúšali dosť na pozíciu za loptu a ísť dopredu smerom k oponentovej bránke tlačiac (driblujúc) loptu dopredu. Ideálne mal tím v pláne umožniť agentovi kopnúť loptu do bránky, len sa im nepodarilo spoľahlivo implementovať schopnosť kopania do lopty pred súťažou.

Hráčske schopnosti

Plán tímu pozostával z vytvorenia spoľahlivej sady schopností agenta, ktoré by mohli byť pospájané modulom na vyššej úrovni správania sa agenta. Ich najväčšou obavou bola schopnosť samotného pohybu. Táto problematika bola riešená už aj inými výskumníkmi. Avšak je veľmi ťažké aplikovať spôsob a princípy chôdze jedného robota na iný druh.

Tím experimentoval s rozličnými tradičnými prístupmi ako sú monitorovanie ťažiska, určovanie trajektórií a podobne. Postupom skúška – omyl prišli na to, že dynamický stabilná chôdza (agent je v rovnováhe keď sa pohybuje) je prístupnejšia ako statická (keď robot náhle zastane). Podarilo sa im dosiahnuť celkom rýchlu a stabilnú chôdzu jednoduchým naprogramovaním robota, aby zdvíhal striedavo ľavú resp. pravú nohu a potom ju vrátil do pôvodného stavu o niečo vpredu. Postupovali zadaním cieľovej pozície chodidla a inverzná kinematika dopočítala potrebné uhly otáčania. Takto sa im podarilo zdokonaľiť pohyb až na úroveň akéhosi behu, keď robot istý čas sa ani jedným chodidlom nedotýkal zeme. Podobný postup využili aj pri schopnosti otáčania sa robota. V tomto prípade stačilo mierne natočiť bedrový kĺb a podobne ako pri chôdzi zdvíhať striedavo chodidlá hore a dolu. Obmenou týchto pohybov následne sú chôdza do boku a späť. Driblovanie lopty sa tímu Austin Villa podarilo realizovať iba jednouchou chôdzou smerom k lopte. Zastavenie pohybov realizovali ich postupným spomaľovaním. Medzi ďalšie užitočné schopnosti agenta patrí padanie a vstávanie. Tím naprogramoval padanie pomocou ohnutia kolien, čím robot stratí rovnováhu a spadne. Nesmierne náročnejšie pre nich spraviť vstávanie agenta. Ich postup bol najskôr postaviť agenta „na štyri“.

Videá jednotlivých naučených schopností má tím na svojej webovej stránke ¹.

¹ <http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/>

Plány do budúcnosti

Tím Austin Villa je veľmi rád, že sa môže podieľať na simulovanom robotickom 3D futbale. Napriek priemerným výsledkom na súťaže v roku 2007 považujú svoj pokrok za podstatný z vedeckého hľadiska. Simulácia humanoidného robota otvára celú škálu zaujímavých problémov z oblasti ovládania hráča, optimalizácie, strojového učenia sa a umelej inteligencie. Tím predpokladá pokonanie začiatkových ťažkostí v ďalších rokoch. Pokiaľ doteraz tím riešil a zameriaval sa na funkčnú množinu schopností hráča, od budúcnosti predpokladajú presun na vyššiu úroveň správania. Vývoj humanoidného hráča simulovaného futbalu na rôznych úrovniach predstavuje výzvu pre rôzne vedecké komunity. Tímy z Austin Villa patria k takejto komunite hlavne v oblasti 2D futbalu. Preto sú nadšený aj pre vytvorenie hráča pre 3D futbal. Ich ďalší postup sa bude zameriavať hlavne na schopnosti ako sú kopanie do lopty a na zlepšenie a optimalizáciu už vytvorených schopností. Neskôr sa chcú zamerať na vyššie formy správania sa hráča.

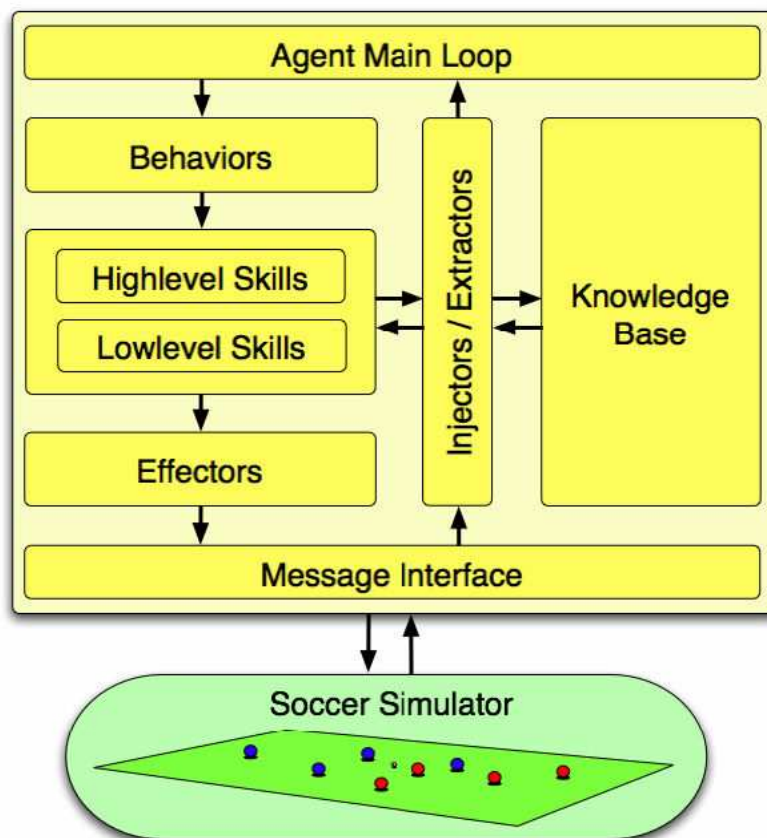
2.1.4. Tím Virtual Werder 3D

Predstavenie tímu

Tím Virtual Werder je nemecký tím pôsobiaci na University of Bremen, ktorý sa v prvých rokoch svojej existencie zameriaval výhradne na 2D ligu, a až od roku 2004 pôsobí v 3D lige humanoidných robotov. Tím sa skladá zo štyroch členov pod vedením Dr. Ubba Vissera, konkrétne sú nimi Cord Niehaus, Arne Stahlbock, Carsten Rachuy a Tobias Warden. Tímu sa, okrem iných domácich úspechov, podarilo dostať do štvrtfinále RoboCup 2007 v Atlante [9].

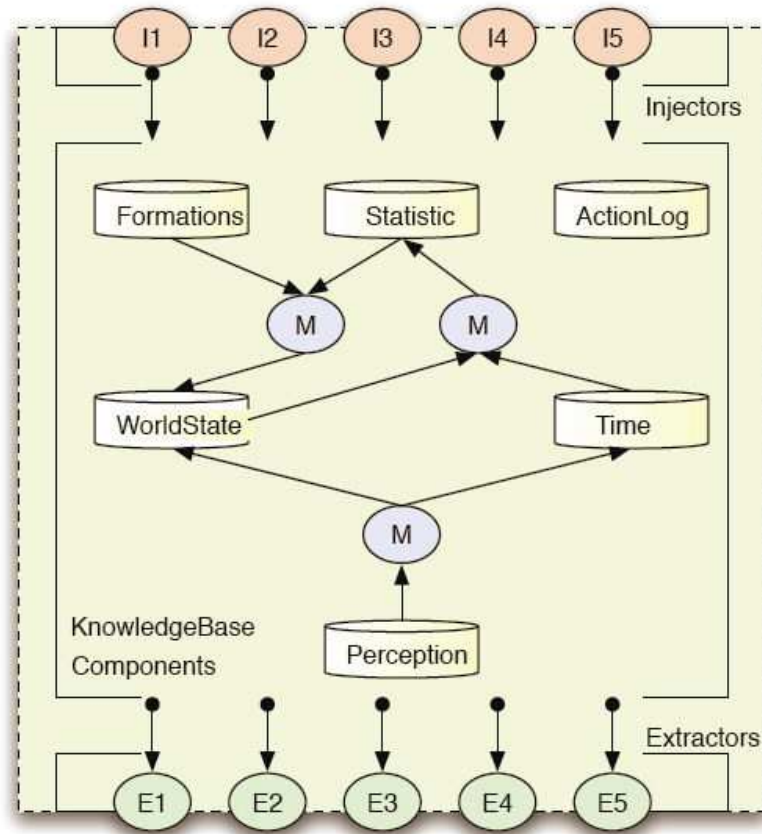
Architektúra hráča

Architektúra pozostáva z dvoch častí. Jedna časť pozostáva z modulov potrebných na vnemy, usudzovanie a akcie, konkrétne agentov hlavný cyklus, jeho správanie, zručnosti, ďalej efekty a komunikačné rozhranie. Tieto časti sú usporiadané do úrovní, kde každý element na jednej úrovni môže používať len elementy z vlastnej, alebo susednej úrovne. Táto architektúra nielenže zobrazuje jasné povinnosti v každej vrstve, ale robí programový kód prehľadnejší a jednoducho rozšíriteľný.



Obr. 4 - Architektúra hráča tímu Virtual Werder 3D [10]

Druhá časť zahŕňa všetky znalosti, ktoré sú potrebné na dedukciu a vykonávanie všetkých výpočtov. Báza znalostí obsahuje relevantné informácie o svete a o samotnom agentovi. Prístup k báze znalostí je možný pre všetky triedy cez definované rozhranie, ktoré podporuje ako nízko-úrovňové tak aj vysoko-úrovňové dotazy pre dáta. Štruktúra bázy znalostí hráča tímu Virtual Werder 3D 2006 je znázornená na obrázku (Obr. 5) [10].



Obr. 5 - Schéma štruktúry bázy znalostí hráča tímu Virtual Werder 3D [10]

Správanie a zručnosti hráča

Správanie hráča tímu Virtual Werder je určené čisto reaktívne za pomoci rozhodovacích stromov. Nepoužívajú sa žiadne neurónové siete ani iné podobné metódy.

Zručnosti hráča sú rozdelené do dvoch častí, nízkoúrovňové zručnosti a vysokoúrovňové zručnosti. Medzi nízkoúrovňové zručnosti patria: Move, Kick, Beam a Say. Tieto prakticky len volajú príslušné funkcie servera. Medzi vysokoúrovňové zručnosti patria: Cover, Score, Intercept, Pass, Reposition a Goalkick [10].

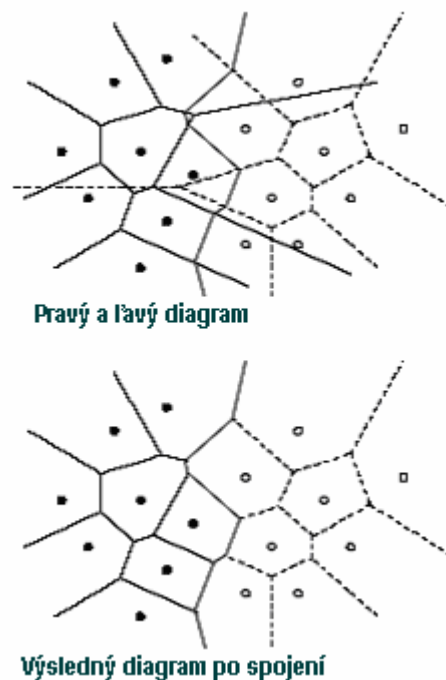
Cover sa používa na „pokrytie“ protihráča, t.j. hráč sa snaží zostať blízko protihráča a pripravovať sa na zachytenie lopty a bráni protihráčovi v streľbe na bránu. *Score* vyberá najslubnejšiu pozíciu v súperovej bránke, za účelom strelenia gólu. *Intercept* vypočítava najlepšiu pozíciu na zachytenie lopty. *Pass* používa agent na kopnutie lopty spoluhráčovi, alebo na iné výhodné miesto. Taktiež poskytuje agentovi schopnosť driblovania s loptou. *Reposition* sa používa na optimalizovanie agentovej pozície v prípade, ak neovláda loptu. Na tento účel používa tím Virtual Werder algoritmus Voronoi segmentácie hracieho pola v závislosti na roli hráča, ktorý je bližšie opísaný nižšie. Pozícia je počítaná tak, aby hráč nebol ďaleko od lopty a zároveň aby nebol blízko pri spoluhráčoch alebo blízko hráčov druhého

tímu. *Goalkick* je špeciálnym kopom, ktorý používa iba brankár pri výkope lopty. Ako úspešné sa u tímu Virtual Werder ukázalo byť použitie metódy „particle filtering“ určenej na predikciu pozícií objektov [10].

Voronoiove diagramy

Voronoiove diagramy sú súhrnom čiar, rozdeľujúcich danú plochu s vopred danými bodmi tak, aby susedné body boli od deliacej čiary rovnako vzdialené. Existuje niekoľko metód, ako vypočítať Voronoiove hrany:

- pretínanie bisekov polrovín: počíta Voronoiovu bunku pretínaním $n-1$ rezov, zložitosť algoritmu je $O(n^2 \log n)$.
- *divide-and-conquer*: rozdelíme body na 2 časti, pre každú z nich vypočítame čiastkový Voronoiov diagram a potom na základe dodatočných Voronoiových hrán, ktoré určíme medzi najbližšími bodmi z oboch oblastí, vytvoríme celistvý Voronoiov diagram. Zložitosť algoritmu je $O(n \log n)$. Na obrázku (Obr. 6) je znázornený princíp tohto algoritmu.



Obr. 6 - Divide and conquer metóda

- *sweep line* (čiaru dosahu): nevýhodou tohto algoritmu je, že nevie predvídať nečakané udalosti, t.j. existenciu nálezísk pod čiarou dosahu. Kľúčom k chodu tohto algoritmu je zistiť všetky nadchádzajúce udalosti – novo sa vyskytujúce náleziska, čo najefektívnejším spôsobom. Sweep line sa posúva smerom nadol, pričom pre všetky

náleziská, ktoré ležia v polrovine nad čiarou, je už zostrojený Voronoioiv diagram ľubovoľným algoritmom - *fortune algoritmus* (odstránenie predvídania, parabolické hrany) a *beach line algoritmus*.

Použitie Voronoiových diagramov

Tím Virtual Werder 3D používa Voronoiove bunky pre reпозиčný systém rozmiestňovania hráčov, ktorí momentálne nemajú loptu. Stavajú sa na pozíciu, kde môžu byť užitoční pre tím. Nachádzajú sa v tzv. Voronoiových regiónoch. Nerozmiestňujú sa okolo celého hracieho poľa, pretože by boli medzi nimi veľké vzdialenosti, boli by ďaleko od miesta akcie a tím by sa stávali nepoužiteľnými. Nepoužívajú celé hracie pole, ale menšie trojuholníkové polia, ktoré sú časťou celého poľa. To, aké sú trojuholníky veľké, závisí od role hráča, aktuálneho miesta lopty a hracieho módu. Použitím tohto mechanizmu nebude hráč nasledovať statickú schému pozície okolí lopty, ale bude sa snažiť byť vždy v okolí akcie [10].

Role definované v tíme Virtual Werder 3D sú nasledovné:

- brankár (v každej formácii je iba jeden)
- obranca
- stredopoliar (na nich je kladený dôraz)
- útočník

Plány do budúcnosti

Tím by sa v budúcnosti chcel venovať dynamickému prispôsobeniu hráča v závislosti od stratégie a rozmiestnenia protihráčov. Popis správania by mohol byť uložený v súbore, aby nebolo potrebné prekompilovanie. Chceli by do agenta zahrnúť možnosť práce s jednotlivými vlastnosťami hráčov – vlastnosti by mohli byť jednoducho kombinované a zamieňané. Chcú využiť optimalizáciu pre lepšie prispôsobenie [10].

Sľubným plánom je vylepšovanie stratégie pomocou techník dolovania dát. Týmto spôsobom je možné analyzovať predchádzajúce zápasy a tiež analyzovať stratégiu protihráča. Môže byť použité napríklad vyhľadávanie vzorov analýzou minulých zápasov.

Ďalším predmetom výskumu je tiež snaha o presnejšie predpovedanie akcií protihráča. Snahou tímu je tiež zaviesť sofistikovanejšie zaznamenávanie histórie hráča pre debugovanie – kreslenie čiar a výpis textu, prehrávanie záznamu a tiež použitie rôznych rýchlostí prehrávania [10].

2.2. Analýza servera SimSpark

Server SimSpark je simulačné prostredie, v ktorom prebieha simulácia robotického 3D futbalu. Kapitola 2.2 je spracovaná zo zdroja [1], ktorý opisuje daný server.

2.2.1. Architektúra servera

Server je postavený na rámci Zeitgeist. Jeho jednotlivé súčasti sú vytvorené v rôznych jazykoch, najmä C++ a Ruby. Server podporuje nahrávanie zásuvných modulov (pluginov) v reálnom čase. Tieto zásuvné moduly musia byť napísané v jazyku Ruby. Vizualizácia scény servera je vykonávaná s využitím knižníc OpenGL a SDL knižnice. Systém je nastaviteľný aj na iné renderovacie knižnice (vd'aka rozhraniu Kerosin).

Dôležitou súčasťou servera SimSpark je vrstva **Oxygen**, ktorá okrem iného obsahuje scénu (reprezentovanú grafom). Ďalej sú v tejto vrstve zapuzdrené transformácie, geometria scény a objektov v nej zahrnutých a kolízie medzi nimi. Vrstva Oxygen takisto udržuje pripojenie s agentmi – robotmi 3D futbalu. Po spustení servera je vo vrstve Oxygen vytvorená modifikovateľná cyklická slučka. Jej modifikovateľnosť spočíva v tom, že je možné pridávať zásuvné moduly, cez ktoré slučka v každom svojom behu prejde a vykoná vybrané ich funkcie.

2.2.2. Práca servera

Server SimSpark pracuje sekvenčne. V každom simulačnom cykle server zozbiera informácie z každého senzoru každého agenta. V cykle takisto vyhodnotí všetky akcie vykonané efektormi jednotlivých agentov. Server renderuje simuláciu za použitia interného alebo externého monitora (podľa nastavení v konfigurácií). SimSpark monitor dostáva od servera informácie o zmenách na scéne a renderuje ich. Formát dát sa nazýva Monitor formát, obsahuje napríklad informácie o aktuálnom móde hry alebo skóre. SimSpark monitor môže aj čítať renderované dáta zo súboru, čím sa dá prehrať záznam (replay) zaznamenané hry. V tomto kontexte sa monitoru hovorí logplayer. Monitor sa v tomto móde spúšťa s prepínačom --logfile, ktorého argument je názov súboru obsahujúceho zaznamenanú hru. Aktuálna verzia 0.6.0 simuluje hru humanoidných robotov (na rozdiel od niektorých starších verzii, ktoré simulovali pohyb sfér).

2.2.3. Spustenie simulácie

Server SimSpark sa spúšťa vykonateľným súborom *sparkserver.exe*. Tento program načíta potrebné zásuvné moduly pre vyhodnocovanie efektorov, perceptorov a realizáciu pohybov

agentov na scéne. Monitor sa spúšťa programom *monitorspark.exe*. Server je nastavený na automatické spúšťanie monitoru, čiže nie je potrebné monitor spúšťať (pokiaľ nechceme, aby bežal na inom počítači). Vzorového agenta pripojíme ku serveru programom *agentspark.exe*. Stlačením písmena "k" v okne monitora sa spustí simulácia (výkop futbalového stretnutia).

2.2.4. Komunikácia medzi serverom a agentom

V komunikácií medzi serverom a agentom sa používajú S - výrazy (S - výraz je buď reťazec, alebo zoznam ďalších S - výrazov). S - výrazy sú používané napríklad v programovacom jazyku Lisp na uloženie kódu aj dát. Správy sú kódované v ASCII (1 znak = 1 byte). Každá správa je prefixovaná svojou dĺžkou (32 bitové bezznamienkové číslo vo formáte Big Endian - najvýznamnejšie bity sú posielané ako prvé).

2.2.5. Perceptory

Perceptory slúžia v RoboCup 3D na vnímanie okolia pre jednotlivých hráčov (agentov). Server pomocou nich posiela každému hráčovi špecifickú správu o jeho pozícií v prostredí, pomocou ktorých sa hráč môže rozhodovať. Perceptory sa môžu rozdeliť do dvoch základných skupín a to základné a špecifické pre futbal.

Základné perceptory

V tejto časti sú popísané základné perceptory, ktoré sa nachádzali na starom type serveru. Popisujú chovanie, ktoré je obvyklé v danom prostredí a nieje spojené priamo s futbalovými schopnosťami hráča.

GyroRate Perceptor

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

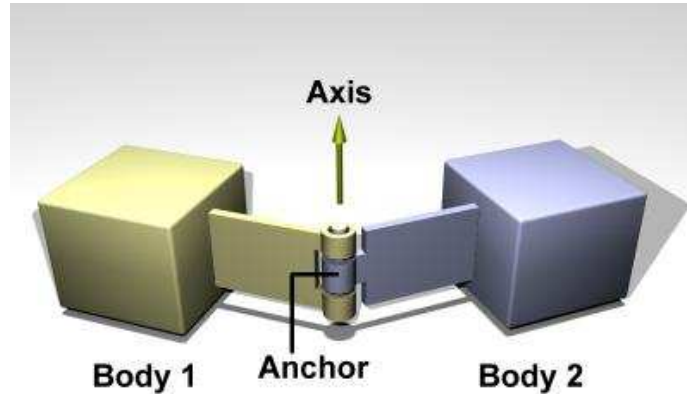
Formát správy pre tento perceptor vyzerá nasledovne:

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

pričom <name> charakterizuje časť tela a hodnoty x, y, z hodnotu o, ktorú je daná časť posunutá.

HingeJoint Perceptor

HingeJoint perceptor určuje, o koľko stupňov sa ohne daný kĺb robota. Kĺb zobrazený na obrázku (Obr. 7) je , kde je vidno práva os ax (Axis).



Obr. 7 - Ukážka jednoduchého kĺbu [1]

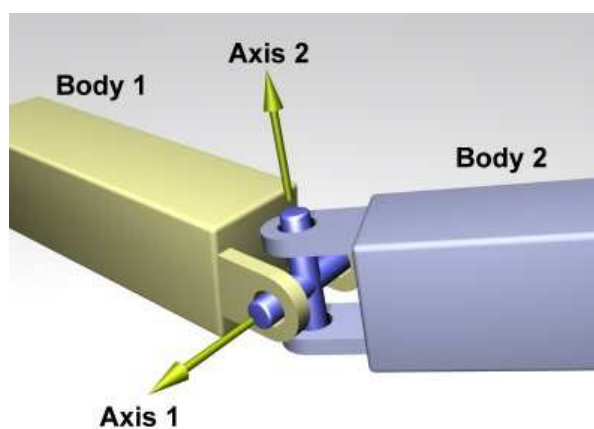
Formát správy pre tento perceptor vyzerá nasledovne:

(HJ (n <name>) (ax <ax>))

pričom <name> charakterizuje názov kĺbu a hodnota ax určuje uhol, o ktorý sa daný kĺb ohol. Hodnota $ax = 0$ označuje, že kĺb je vystretý.

UniversalJoint Perceptor

UniversalJoint perceptor sa už na novom type hráča nevyskytuje. Nahradili ho dva HingeJoint perceptory, pomocou ktorých hráč ohne kĺby do dvoch smerov. Pred tým hráč mohol pohybovať kĺbom pomocou dvoch osí (Obr. 8), čo nový hráč už nepodporuje.



Obr. 8 - Ukážka zložitého kĺbu [1]

Formát správy pre tento perceptor vyzerá nasledovne:

(UJ (n <name>) (ax1 <ax1>) (ax2 <ax2>))

pričom <name> charakterizuje názov kĺbu a hodnoty ax1 a ax2 určujú uhol ohybu.

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, značí kolíziu.

Formát správy pre tento perceptor vyzerá nasledovne:

(TCH n <name> val 0|1)

ForceResistance Perceptor

ForceResistance perceptor slúži na oznámenie pôsobenia sily a jej vektora. Súradnice c určujú bod, na ktorý sila pôsobí a súradnice f zobrazujú práve vektor tejto sily.

Formát správy pre tento perceptor vyzerá nasledovne:

(FRP (n <name>) (c <px> <py> <pz>) (f <fx> <fy> <fz>))

Perceptory špecifické pre futbal

Keďže potrebujeme, aby hráč mal aj futbalové vlastnosti musí mať taktiež aj perceptory, ktoré to umožňujú. Perceptory ďalej opisujem sú spojené priamo s futbalovými schopnosťami hráča.

Vision Perceptor

Na to aby hráč mohol využívať svoje futbalové schopnosti je nutné aby vedel kde sa nachádza a taktiež aby videl ostatných hráčov, loptu a brány. Na to mu slúži práve perceptor Vision, ktorý zachytáva 90° uhol. Na začiatku hry je hráč natočený automaticky na súperovu stranu ihriska, ale musí vedieť aj natočiť sa na opačnú stranu. Keďže formát správy pre tento perceptor je o veľa zložitejší ako predchádzajúce nebudeme ho v tejto práci uvádzať.

GameState Perceptor

Tento perceptor sa využíva hlavne na začiatku, keďže pomocou neho hráč zistí veľkosť ihriska a lopty. Počas hry sa však využíva taktiež, nakoľko hráčovi hovorí aký je čas zápasu a v akom stave sa hra nachádza (napr. stav pred pokutovým kopom).

Formát správy pre tento perceptor vyzerá nasledovne:

(GS (t <time>) (pm <playmode>))

AgentState Perceptor

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

Formát správy pre tento perceptor vyzerá nasledovne:

(AgentState (temp <degree>) (battery <percentile>))

Hear Perceptor

Hear perceptor ako už sám názov napovedá 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.

Formát správy pre tento perceptor vyzerá nasledovne:

(hear <time> 'self'|<direction> <message>)

2.2.6. Efektory

Efektory sa používajú na všetky činnosti, ktoré chceme s našim hráčom vykonať. Pomocou nich posielame serveru správy na zmeny jednotlivých činností, ktoré následne hráč vykoná. Taktiež aj efektory sa delia do dvoch základných skupín a to základná skupina a skupina špecifická pre futbal.

Základné efektory

Tak isto ako základné perceptory aj základné efektory slúžia k určaniu základného chovania, ktoré je obvyklé v danom prostredí a nieje spojené priamo s futbalovými schopnosťami hráča.

Create Effector

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

Formát správy pre tento efektor vyzerá nasledovne: (scene <filename>)

HingeJoint Effector

K pohybu jednotlivými kĺbmi hráča potrebujeme HingeJoint efektor, ktorý nám umožňuje zadať názov kĺbu, s ktorým chceme hýbať a uhol ohybu, o ktorý chceme daným kĺbom ohnúť.

Formát správy pre tento efektor vyzerá nasledovne:

(<name> <ax>)

UniversalJoint Effector

Tak isto ako UniversalJoint perceptor aj UniversalJoint efektor je využívaný iba na starom modeli serveru a slúži na pohyb kĺbu v smere dvoch osí. Toto však nový model hráča už nepodporuje.

Špecifické efektory pre futbal

K ovládaniu špecifických vlastností robota slúžia práve špecifické efektory pre futbal. Tieto efektory ovládajú perceptory, ktoré sú taktiež špecifickými pre futbal.

Init Effector

Ako sme už spomínali Init efektor sa spúšťa zvyčajne následne po Create efektore a priradí hráča k vybranému tímu.

Formát správy pre tento efektor vyzerá nasledovne:

(init (unum <playernumber>)(teamname <yourteamname>))

Beam Effector

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

Formát správy pre tento efektor vyzerá nasledovne:

(beam <x> <y> <rot>)

Say Effector

K odoslaniu správy ostatným hráčom sa využíva Say efektor. Správa sa však neposiela priamo, ale cez server. Kódovanie správy je ASCII.

Formát správy pre tento efektor vyzerá nasledovne:

(say <message>)

2.2.7. Správania agenta

Správanie agenta sa ukladá do tried, ktoré sú odvodené od triedy Behavior. Musia implementovať dve základné metódy. Inicializáciu (Init), ktorá slúži na spustenie hráča a zároveň na uloženie na vhodnú pozíciu na ihrisku. A druhá metóda je myslenie (think), ktorá od serveru získava údaje a následne ich vyhodnocuje. Pomocou vyhodnotenia určí ďalšie správanie hráča.

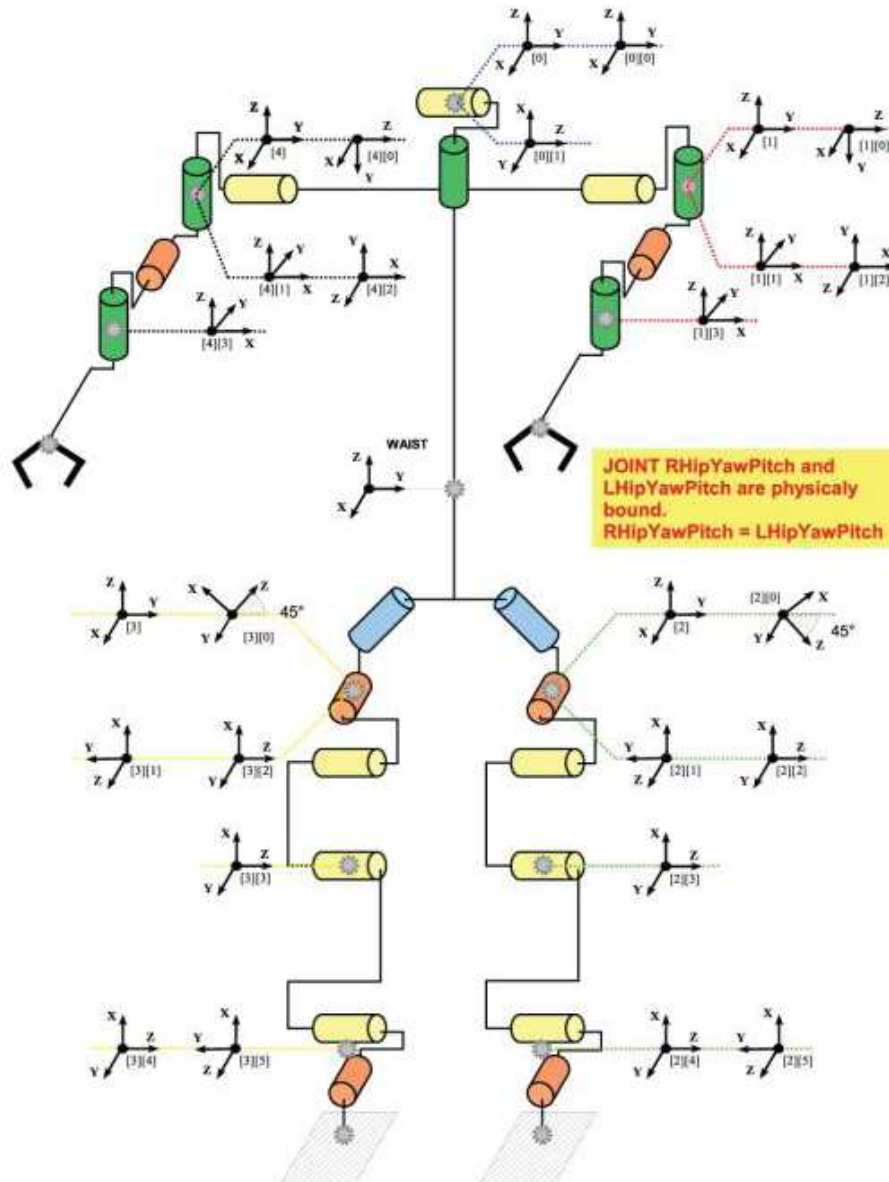
2.2.8. Simulácia

Simulácia RoboCup 3D začína vytvorením hráčov, priradením do tímu a ich umiestnením na ihrisko. V jednotlivých tímoch môže byť najviac 5 hráčov. Každý tento hráč vidí značky, ktoré ohraničujú ihrisko. Tie sa nachádzajú v každom rohu ihriska a taktiež na oboch okrajoch brány. Hráč taktiež vidí relatívnu pozíciu lopty a aj ostatných hráčov.

Treba si uvedomiť, že zo strategického hľadiska je veľmi podstatné umiestnenie hráča na správne miesto pri inicializácii. Keďže hráč vie kde sa nachádza lopta, bránky ostatný hráči je potrebné, aby tieto informácie vedel využiť k streleniu gólu. Na to však treba docieľiť vhodnými pohybmi aby sa dostal najskôr k samotnej lopte a vedel do nej kopnúť. Taktiež je potrebné, aby sa vedel postaviť po prípadnom náraze alebo páde.

2.2.9. Robot NAO

Najnovší model robota pre RoboCup 3D je Robot NAO. Ide o model robota vytvorený podľa reálneho vzoru. Jeho výška je 57 cm a váha 4,5 KG. Na obrázku (Obr. 9) sú vizuálne zobrazené kĺby a osi robota NAO [1].



Obr. 9 - Kĺby a osi robota NAO [1]

2.3. Neurónové siete a ich využitie

Pri používaní neurónových sietí ako prvé vždy riešime, ako správne a vhodne alebo ako vôbec definovať vstupy a výstupy neurónovej siete [2]. V prípade ovládania hráča simulovaného robotického 3D futbalu sa nám hneď naskytuje možnosť využiť vlastnosti samotnej simulácie. Server posiela v pravidelných časových intervaloch informácie o polohe jednotlivých kĺbov agenta a ten naspäť odpovedá zaslaním požadovanej rýchlosti otáčania kĺbov. Preto ako vstupy do neurónovej siete je vhodné použiť priamo hodnoty natočenia kĺbov a ako výstupy uhlové rýchlosti otáčania sa jednotlivých kĺbov. Kĺbov je dvadsaťdva, čo je vzhľadom na iné neurónové siete nízke číslo. To nám poskytne sieť, ktorú je jednoduchšie učiť a aj rýchlejšie reaguje a vypočítava výstupné hodnoty. S počtom neurónov v skrytej vrstve môžeme experimentovať.

Najväčší problém v našom prípade však je, že ako efektívne učiť neurónovú sieť. Asi by nebolo vhodné mať pre všetky pohyby a správanie robota iba jednu sieť aj keď teoreticky by to bolo možné. Reálne však rozložením na viacej jednoduchších problémov, ktoré bude riešiť viac rôznych sietí, získame lepšie riešenie. Preto nám vychádza ako správna voľba použiť pre každý druh primitívneho pohybu (ako sú napr. chôdza, beh, vstávanie, kopanie do lopty atď.) jednu špecializovanú sieť.

Ostáva však správne určiť tréningovú množinu. Vhodné by bolo, kebyže máme dopredu definované okrajové situácie daných pohybov (ako napr. vykročenie pravou nohou pri chôdzi). Potrebujeme jednoznačne povedať, že aké majú byť v nich optimálne zvolené dane parametre (uhlové rýchlosti kĺbov) pre vykonanie a docielenie daného pohybu pri zachovaní stability agenta. Preto je vhodnejšie najskôr použiť inú metódu ako evolučné algoritmy alebo simulovanie pohybov podľa človeka, ľudského tela a až potom na uloženie získaných znalostí použiť neurónovú sieť.

Pri samotnom vytváraní siete si budeme musieť položiť ešte aj otázku, či je vhodné použiť klasickú doprednú sieť alebo rekurentnú. Pri vykonávaní pohybu ako je napr. chôdza je podstatné aby agent vedel určiť aj v ktorej časti pohybu sa nachádza a aká činnosť má potom nasledovať. Napr. pri chôdzi po zdvihnutí nohy ju musí dávať dolu a dopredu. Pre tento účel pamätania si predchádzajúceho stavu, kde si agent pamätá predchádzajúcu sekvenciu pohybov, je vhodnejšia rekurentná sieť. Keďže zmena samotnej siete nijak neovplyvní samotnú architektúru agenta, najlepšie by bolo experimentálne vyskúšať, ktorý typ neurónovej siete dosahuje lepšie výsledky.

Použitie určitého typu aktivačnej funkcie nie je nejakým spôsobom dopredu jasne alebo viazane na určitú časť agenta. Môžeme použiť buď unipolárnu alebo bipolárnu, a buď sigmoidálnu alebo lineárnu aktivačnú funkciu. Podstatné však je správne mapovať daný interval funkcie na interval otočenia kĺbu resp. uhlovej rýchlosti pohybu kĺbom.

Iným prístupom môže byť použitie učenia sa s odmenou a trestom [3] (angl. reinforcement learning). O opodstatní tohto prístupu sa môžeme presvedčiť aj v prípade tímu Austin Villa, ktorý sme bližšie analyzovali. V tomto prípade neurónovej siete dávame priamo možnosť ovládať agenta. Ak agent následne nevykoná správne daný pohyb, zlyhá v ňom (napr. pri chôdzi spadne) tak sieť potrestáme. Naopak ak sieť úspešne riadi agenta, odmeníme ju. Tento prístup sa zdá byť oveľa vhodnejší a preto ešte zvážime jeho použitie pri našom agentovi.

2.4. Zhodnotenie analýzy

Analyzovali sme hlavné parametre servera a nového hráča. Ako základ analýzy sme využili už existujúce riešenia tímov, ktoré sa pravidelne zúčastňujú na turnajoch robocup 3D, ako aj doterajší výskum na našej fakulte. A keďže nový server aj s novým hráčom vyšli len prednedávnom, tak sme nenašli taký počet materiálov a v dostatočnej kvalite, ako sme očakávali. Naštudovali sme si aj smerovanie, ktorými sa jednotlivé tímy uberali a aké metódy pri vývoji používali. Jedným z poznatkov tejto analýzy je fakt, že použitie neurónovej siete, aj keď sa to môže javiť ako dobrý nápad, je nevhodné, lebo nemáme zaručený výsledok. Uvažovať by sa ešte dalo o použití evolučných algoritmov na pre určité pohyby hráča.

Pri analýze serveru sme sa museli najskôr oboznámiť, z čoho sa skladá, ako funguje, ako sa spúšťa simulácia a ako prebieha komunikácia medzi hráčom a serverom. Veľkou pomocou bolo pre náš tím poskytnutie servera od minulého tímu. Naša analýza servera sa zameriavala hlavne na to, čo ktorý efektor/perceptor robí, na čo slúži a ako sa ovláda.

Pri analýze hráča sme použili ako základ starý typ hráča. Zo neho sme si osvojili niektoré dobré myšlienky, ktoré by mohli fungovať aj na model nového hráča. Napríklad na základe analýzy tímu Little Green bats sme sa rozhodli použiť súbory, v ktorých by boli uložené určité dopredu nadefinované pohyby hráčov kĺbov. Ďalším vhodným nápadom sa nám pozdáva myšlienka, že hráč má pri vykonávanej činnosti určitú najstabilnejšiu polohu. Napríklad na starom modeli hráča bol robot pri chôdzi najstabilnejší práve vtedy, keď bol pokrčený v kolenách a trošku predklonený. Dá sa predpokladať s vysokou pravdepodobnosťou, že takáto poloha bude aj v novom modeli hráča.

3 Špecifikácia

Jednoduchá a rýchla tvorba pohybov

Na základe predošlej analýzy sme sa rozhodli vytvoriť podporný nástroj, ktorý by na základe našich vstupných údajov vygeneroval súbor, ktorý by si vedel hráč načítať a vykonávať nadefinované činnosti. Takýto prístup k tvorbe pohybu má výhodu v tom, že za relatívne veľmi krátku dobu môžeme vyrobiť veľké množstvo pohybov a to aj bez znalostí z programovania.

Hráč by mal udržať rovnováhu aj pri neočakávaných udalostiach

O udržanie by sa staral rovnovážny modul, ktorý by korigoval pohyb hráča a zabezpečil by, aby pri neočakávanej situácii nepadol na zem. Takouto neočakávanou situáciou môže byť prípad, keď sa hráč zrazí s iným hráčom. Nedá sa naivne očakávať, že hráč udrží rovnováhu pri každej situácii, to nedokáže ani človek.

Naučiť hráča vstať

Nevyhnutná súčasť výbavy hráča, pokiaľ ho chceme použiť do reálneho zápasu, je schopnosť vstať. Hráč bude schopný sa postaviť z akejkoľvek polohy na vodorovnej podložke. Bude schopný zistiť, v akej je polohe, dostať sa do východiskovej polohy pre postavenie, a postaviť sa.

Naučiť hráča zmeniť smer pohybu

Hráč bude reagovať na podnety ako je pohyb lopty alebo súperovho hráča. Analyzovaním dát z perceptorov bude v prípade potreby vykonaná zmena smeru pohybu hráča. Hráč bude reagovať na podmety v reálnom čase.

Naučiť hráča kopnúť do lopty

Hráč bude schopný lokalizovať loptu, pristúpiť ku nej do polohy, v ktorej je možné do nej kopnúť, a kopnúť do nej. Náročnejšiu časť tejto úlohy je kopnutie spôsobom, ktorý ovplyvní smer pohybu lopty. Toto bude riešené v prípade, že hráč bude schopný vykonať všetky predchádzajúce pohyby na dostatočnej úrovni.

4 Návrh riešení

4.1. Rovnovážny modul

Pri riešení rôznych pohybov agenta a situácií, do ktorých sa počas hry môže dostať, sme často riešili problém, ako udržať agentovu rovnováhu. Je dôležité, aby agent nespadol pri jemných kolíziách alebo nie celkom vydarených pohyboch. Na tento účel sme v architektúre agenta navrhli rovnovážny modul, ktorého úlohou je udržovať agenta v polohe čo najmenej vedúcej do pádu.

Pre konkrétne podmienky RoboCup3D sme zvažovali niekoľko alternatív, ako by mohol rovnovážny modul fungovať:

Zabezpečenie rovnováhy pomocou rovnovážnej rovnice n -tého rádu

Vyhodnocovanie pohybu agenta v čase pomocou diferenciálnej rovnice

Úprava pohybov agenta podľa vzorcov fyziky (rýchlosť, hybnosť, zrýchlenie, ...)

4.1.1. Rovnovážna rovnica

Podľa dokumentácie ku robotovi poznáme váhu každej jeho časti, jej umiestnenie vzhľadom na časť tela na ktorú je pripojená (Translation) a takisto vzhľadom na torzo robota (Anchor) [1].

Z týchto údajov by sa pomocou rovnice dalo vypočítať, akými silami pôsobia jednotlivé končatiny a iné časti robota na jeho ťažisko. Bude potrebné zohľadniť aj výšku vzhľadom na povrch, na ktorom sa robot nachádza (čím vyššie je umiestnená nejaká časť tela, ktorá je vychýlená, tým viac vplýva na stabilitu robota).

4.1.2. Diferenciálna rovnica

Pri diferenciálnej rovnici prichádza do úvahy aj čas. V prípade RoboCup3D ide o diskretný čas, keďže server komunikuje s klientom v niekoľko mili-sekundových intervaloch. Vzhľadom na náročnosť použitia diferenciálnych rovníc zatiaľ ich použitie nezvažujeme.

4.1.3. Fyzika pohybov

Pohyb robota je možné opísať pomocou fyzikálnych zákonov, keďže server by sa mal riadiť fyzikou reálneho sveta. Do úvahy prichádzajú vzorce pre rýchlosť, zrýchlenie, hybnosť alebo silu. Tieto vzorce by mal mať robot implementované na niekoľkých úrovniach, aby vedel vypočítať celkovú silu na neho pôsobiacu (ako súčet vektorov čiastkových síl zachytených

perceptormi) alebo svoju celkovú hybnosť, ktorá by nemala prekročiť určitú hranicu (aby sa robot nestal neovládateľný)

4.1.4. Zhodnotenie

Použitie vyššie uvedených prístupov ku riadeniu rovnováhy robota si vyžiada hlbšiu analýzu. Podľa doterajších experimentov s agentom môžeme povedať, že aj implementácia jednoduchých pravidiel do robota môže vzhľadom na množstvo rôznych vstupov predstavovať komplikované riešenie.

4.2. Prototyp rovnovážneho modulu

Ako jeden z cieľov nášho úsilia sme stanovili vypracovanie rovnovážneho modulu agenta, ktorý bude schopný určiť aktuálne ťažisko robota, a prípadne predchádzať pádom. Ako prvý krok k tomuto cieľu sme rozobrali štruktúru agenta, zistili, z akých častí je zložený a aké sú polohy týchto častí po inicializácii robota, teda keď sa pripojí k serveru. Tieto hodnoty sú pre náš rovnovážny model kľúčové, pretože výpočty aktuálnych polôh jednotlivých častí robota sú voči nim relatívne. Pri odvodzovaní vzorcov pre výpočet posunutia bodov v 3D priestore sme dbali na jednoduchosť vzorcov a minimálnu výpočtovú náročnosť.

V prvej verzii sme počítali novú polohu bodu pomocou sústavy dvoch rovníc kružnice. Tento výpočet síce dával správne výsledky, ale nevedeli sme osamostatniť neznámu tak, aby nebola v kvadratickom člene. Preto sme vymysleli iný spôsob počítania polôh, ktorý využíva iba goniometrické funkcie. Pomocou týchto funkcií vieme implementovať funkciu, ktorá vyzerá nasledovne:

$$\begin{aligned} [X_n, Y_n, Z_n] \text{ získajNovuPolohuBodu}(X_{os}, Y_{os}, Z_{os}, \\ X_p, Y_p, Z_p, \\ \text{alfa, beta, gama}) \end{aligned}$$

Vstupy:

X_{os}, Y_{os}, Z_{os} – súradnice bodu, cez ktorý prechádzajú osi otáčania

X_p, Y_p, Z_p – počiatočné súradnice bodu

Alfa (os Z), beta(os Y), gama(os X) – uhly natočenia bodu okolo daných osí

Výstup:

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

Vyššie uvedená funkcia je len prototyp, reálna implementácia funkcie nebude v každom kroku preberať počiatočnú polohu bodu, keďže tá je nemenná. V dokumentácií neuvádzame presné vzorce, pretože sú ešte predmetom vývoja, a nie sú zatiaľ dostatočne overené v praxi.

4.2.1. Využitie prototypu

Pomocou vyššie uvedenej funkcie budeme vedieť vypočítať polohu ľubovoľného bodu na spojniciach kĺbov robota, a to v ľubovoľnom čase. Toto využijeme na výpočet polohy ťažiska jednotlivých častí robota (rameno, predlaktie, trup, ...). Spolu s informáciou o hmotnosti danej časti vieme vypočítať polohu celkového ťažiska robota.

V rovnovážnom moduly budeme sledovať pohyb ťažiska v čase. Ak bude jeho pohyb zrýchľovať nad určitú hranicu, môže to znamenať, že sa robot dostáva do nestabilnej polohy. V takom prípade sú možné 2 alternatívy:

1. Robot sa pokúsi eliminovať pád - Toto je možné riešiť zrýchleným pohybom častí robota, ktorými sme schopní vytvoriť silu pôsobiacu smerom opačným k smeru pádu
2. Robot informuje spoluhráčov - Ak robot vyhodnotí svoj pád ako neodvratný, informuje svojich spoluhráčov efektorom SAY o tomto stave. Tí môžu dočasne vytvoriť stratégiu, kde nebudú počítať s padnutým spoluhráčom.

Vytvorený prototyp budeme ďalej upravovať a rozširovať. Jeho predvedenie v tejto fáze nie je celkom možné, keďže zatiaľ žiadnym spôsobom neovplyvňuje správanie robota. Do budúca je naším cieľom spojiť funkcionality editora pohybov a rovnovážneho modulu tak, aby nespôsobovali zmätocný pohyb robota, ale pomáhali mu v stabilizácii navrhnutých pohybov.

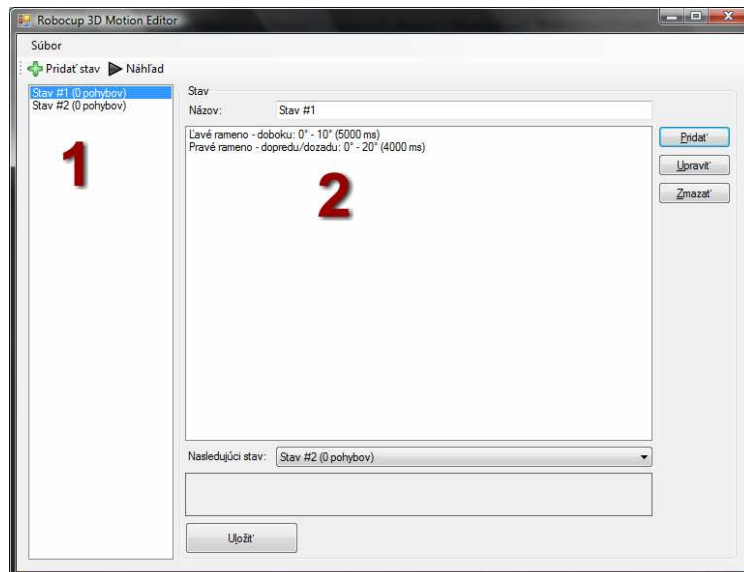
4.3. *Editor pohybov robota*

Dôležitým nástrojom, ktorý využijeme pri práci na projekte je Editor pohybov robota. Jeho vytvorením získame „laboratórium“, v ktorom budeme môcť experimentovať s pohybmi a jednoducho skúšať rôzne postupy, bez nutnosti zasahovania do kódu agenta. Agent bude schopný čítať súbory vytvorené týmto editorom a vykonávať inštrukcie uvedené v ňom.

4.3.1. Prostredie editora

Hlavné okno editora pozostáva z menu, panelu nástrojov a zápisu akcie robota. V ľavej časti sa nachádza zoznam stavov (Obr. 10– bod 1) a napravo sú podrobnosti vybraného stavu.

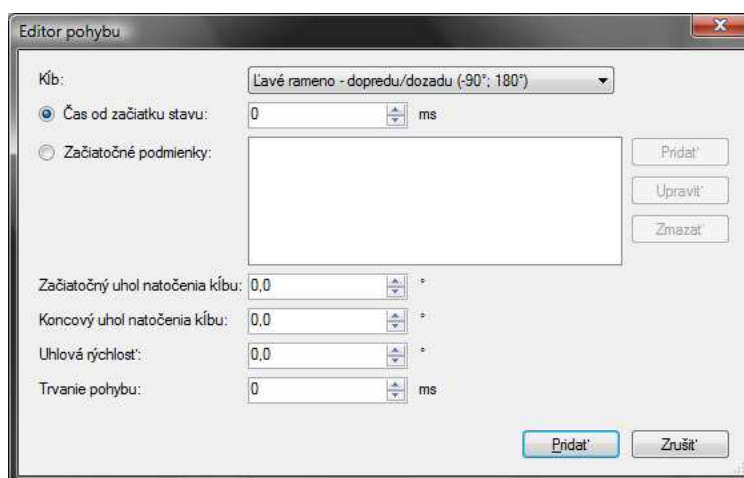
Každý stav má svoj názov, zoznam čiastkových pohybov (Obr. 10– bod 2) a definovaný nasledujúci stav.



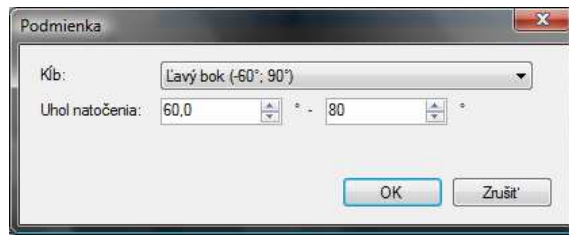
Obr. 10 - Pôvodné hlavné okno editora pohybov

4.3.2. Tvorba pohybov

Pohyby sa vytvárajú prostredníctvom dialógového okna Editor pohybu. V tom vyberieme kĺb robota, ktorým chceme pohnúť. Načasovanie pohnutia je možné nastaviť 2 spôsobmi – buď *explicitne* zadaním presného času, ktorý má uplynúť od začiatku stavu alebo *implicitne* zadaním začiatočných podmienok pre daný pohyb (pozície kĺbov robota). Pohyb sa ďalej definuje zadaním začiatočného, koncového uhla natočenia kĺbu a uhlovej rýchlosti (resp. trvania pohybu). Uhlová rýchlosť alebo trvanie pohybu sa dopočítajú, podľa toho, ktorý z týchto parametrov je zadaný.



Obr. 11 - Editor čiastkového pohybu



Obr. 12 - Definovanie začiatočnej podmienky pohybu

4.3.3. Náhľad pohybu

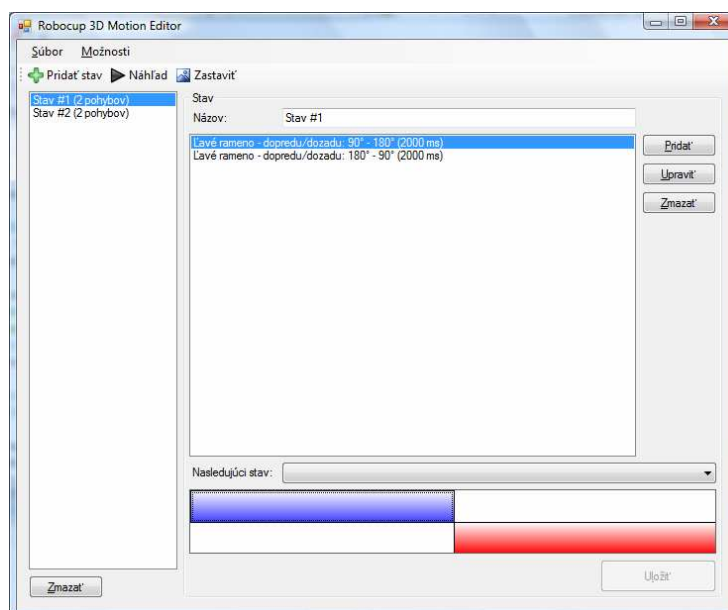
Pohyb si môžeme po vytvorení ihneď prezrieť. Kliknutím na tlačidlo Náhľad sa vygeneruje súbor s pohybom, ktorý sa zašle agentovi, ktorý je pripojený k serveru. Agent súbor s pohybom načíta a inštrukcie zadané v ňom vykoná.

4.4. Prototyp editora pohybov

Motiváciou pre vytvorenia editora pohybov robota bolo vytvorenie podmienok pre jednoduchšie experimentovanie s pohybovými možnosťami robota bez nutnosti programovania každého pohybu. Nástroj by mal zjednodušiť a zrýchliť ďalšiu prácu na vývoji pohybov robota.

4.4.1. Prostredie programu

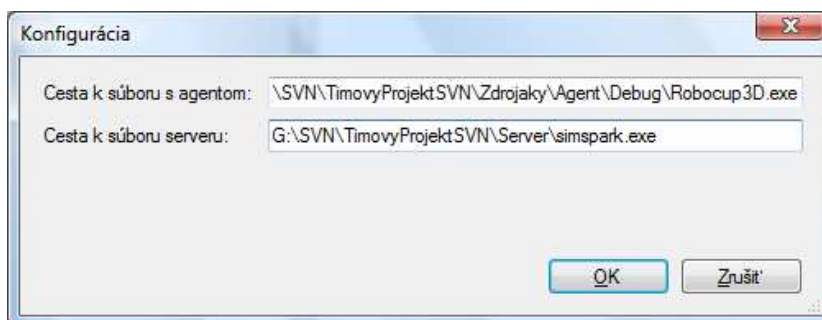
Prostredie programu (Obr. 13) je prehľadne rozčlenené na menu, hlavný panel (na ktorom sa nachádzajú aj tlačidlá Náhľad a Zastaviť), panel stavov a časovú os, na ktorej je znázornené vykonávanie jednotlivých pohybov.



Obr. 13 - Hlavné okno editora pohybov

4.4.2. Nastavenie editora

Ku konfigurácii editora sa dostaneme kliknutím na položku Konfigurácia v menu Možnosti. V dialógovom okne Konfigurácia (Obr. 14) potom musíme pre zabezpečenie správneho fungovania funkcie Náhľad pohybu nakonfigurovať cesty k súboru s agentom a k súboru serveru (simspark.exe). Musia byť zadané absolútne cesty k spustiteľným súborom.



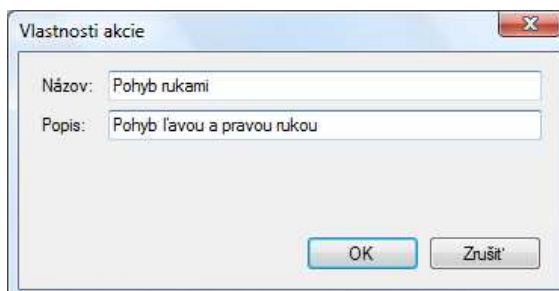
Obr. 14 - Konfigurácia editora

4.4.3. Vytváranie akcií

Akcia predstavuje súhrn viacerých za sebou nasledujúcich stavov. Stavov pozostávajú z viacerých elementárnych pohybov, ktoré sa môžu vykonávať aj paralelne avšak nemôžu si odporovať.

Akciu vytvárame pridávaním stavov (pomocou tlačidla Pridať stav), prípadne odstraňovaním chybných, či neželaných stavov (pomocou tlačidla Zmazať pod panelom stavov). Stavov pridáme názov, ktorý ho najlepšie vystihuje a postupným pridávaním, upravovaním, alebo aj mazaním elementárnych pohybov vytvoríme celkovú pohybovú sekvenciu stavu.

Akciu nakoniec pomenujeme vhodným názvom (v menu Súbor vyberieme položku Vlastnosti akcie). Pre lepšie pochopenie danej akcie, ju môžeme podrobnejšie opísať do textového poľa popis (Obr. 15).



Obr. 15 - Pomenovanie a popis vytváratej akcie

4.4.4. Náhľad vytvorenej akcie

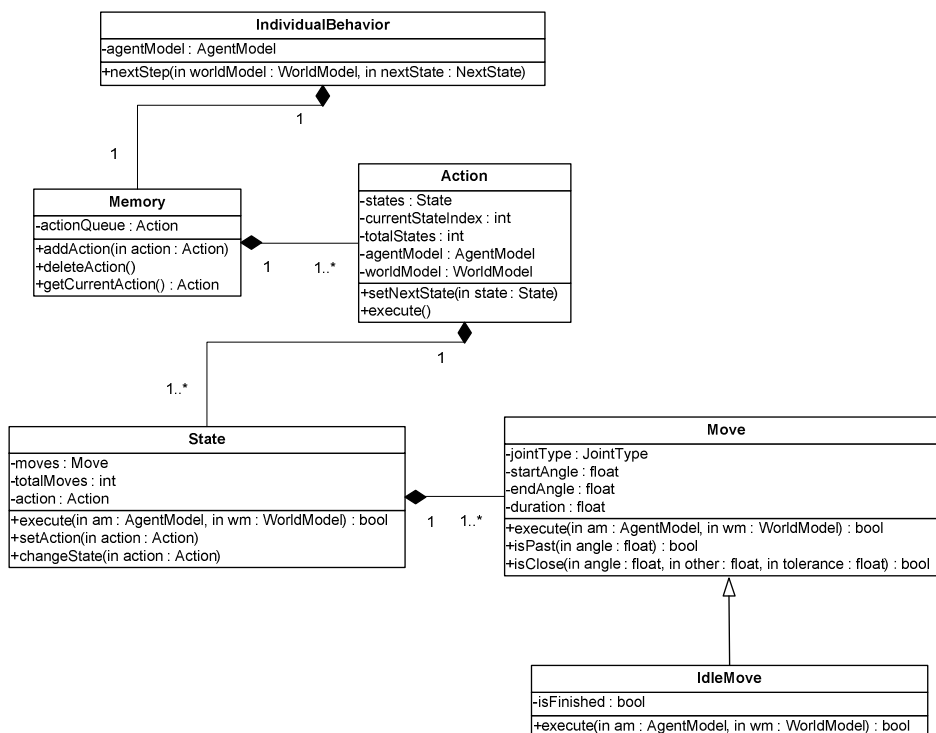
Aplikácia vie zabezpečiť predvedenie zadaných akcie robota. Po kliknutí na tlačidlo Náhľad sa vykonaná niekoľko krokov potrebných na zobrazenie pohybu robota, ktorý sme navrhli prostredníctvom editora:

1. Vytvorí sa dočasný súbor s aktuálnou akciou
2. Spustí sa server Robocupu
3. Spustí sa agent s parametrom, v ktorom definuje cestu k súboru akcie robota, ktorá sa má vykonať

Po dokončení vykonania akcie je možné program servera aj agenta jednoducho ukončiť kliknutím na tlačidlo Zastaviť.

4.5. Riadenie pohybov robota

V pôvodnej verzii agenta je pridávanie nových akcií pomerne zložitý úkon, vyžadujúci veľa opakujúceho sa kódu. Z tohto dôvodu sme pristúpili k redizajnu architektúry správania sa robota. V novej architektúre by mali byť jednotlivé akcie inštanciami triedy, ktorá akciu opisuje. Toto do veľkej miery zjednoduší implementáciu nového správania, ale aj umožní pomerne jednoduché ukladanie a načítanie pohybov zo súboru.



Obr. 16 - Diagram tried týkajúcich sa pohybov robota

Diagram tried znázorňuje implementáciu samotného agenta. Je úzko spätý aj s formátom súborov pre uloženie správania sa a hlavne primitívnych akcií agenta. Príkladom primitívnej akcie môže byť napríklad chôdza, alebo vstávanie.

Každá akcia by mala mať vstupné podmienky. Keby neboli podmienky splnené (napr. agent je padnutý na zemi), danú akciu (napr. chôdzu) nie je možné vykonať. Samotné vykonávanie jednotlivých akcií bude riadiť pamäť a vyššia logika správania. Chôdzu ako takú môžeme rozdeliť na pohyby pravou a ľavou nohou, následne sa opakujúcich. Rovnako pohyb jednou nohou môžeme rozdeliť na časť, keď robot nohu na začiatku zdvíha, a keď ju potom dáva dole. Tieto časti daného pohybu budeme považovať za stavy. Nejedná sa o stav, kedy by bol agent v statickej polohe, ale skôr kedy je v stave vykonávania určitej činnosti. Napr. stavom bude zdvihnutie pravej nohy. Vtedy hýbe viacerými kĺbmi, preto stav obsahuje „mikropohyby“ pre každý potrebný kĺb. Na začiatku danej akcie sa zvolí začiatkový stav. Pri vykonávaní stavu sa na základe aktuálnej pozície kĺbov, cieľovej pozície a požadovanej doby otáčania vypočíta uhlová rýchlosť, ktorou bude agent nimi hýbať. Opakovanie pohybu bude zakomponované buď do akcie samotnej, alebo do vyšších tried.

4.5.1. Triedy

Memory

Touto triedou je reprezentovaná vnútorná pamäť robota. Slúži na uchovávanie akcií v poradí v akom sa majú vykonať (pomocou prioritného radu), ako aj základnú manipuláciu s akciami.

Polia:

Názov poľa	Typ	Popis
actionQueue	Action**	Rad s akciami, zoradený v chronologickej postupnosti

Metódy:

Názov metódy	Návratový Typ	Popis
addAction	void	Zaradí novú akciu na koniec radu
deleteAction	void	Odstráni akciu, ktorá je prvá v rade
getCurrentAction	Action*	Vráti prvú akciu v rade

Action

Táto trieda predstavuje v danej hierarchii akciu, ktorá pozostáva z jednotlivých stavov. Táto trieda bude pravdepodobne podliehať rozsiahlejším zmenám, hlavne čo sa týka prepínaniu stavov.

Polia:

Názov poľa	Typ	Popis
states	State**	Stavy v poradí, v akom idú za sebou
currentState	Int	Index aktuálneho stavu
totalStates	Int	Celkový počet stavov

Metódy:

Názov metódy	Návratový Typ	Popis
execute	void	Vykoná akciu, resp. pokračuje v jej vykonávaní
setNextState	void	Prepne sa do nasledujúceho stavu

State

Každý stav uchováva informácie o pohyboch, ktoré sa majú v rámci daného stavu vykonať. Stav je zodpovedný za správne ukončenie vykonávania všetkých pohybov, z ktorých sa skladá, ako aj za následný prechod do nasledujúceho stavu.

Polia:

Názov poľa	Typ	Popis
moves	Move**	Pohyby, ktoré sa v rámci stavu simultánne vykonávajú
action	Action*	Akcia, ku ktorej stav prislúcha
totalMoves	Int	Celkový počet pohybov

Metódy:

Názov metódy	Návratový Typ	Popis
execute	void	Vykoná akciu, resp. pokračuje v jej vykonávaní
changeState	void	Vyvolá zmenu stavu na akcii
setAction	void	Nastaví privátne pole action

Move

Táto trieda slúži na reprezentáciu jedného z pohybov, ktoré sa budú vykonávať počas jedného stavu. Pohyb má začiatkové natočenie, v ktorom sa musí kĺb (približne) nachádzať pred začatím vykonávania pohybu. Pole endAngle určuje konečné natočenie kĺbu. Dosiahnutie

tohto stavu sa zisťuje metódou `isPast`, ktorá vráti `true` hneď ako kĺb prekročí konečné natočenie. Pole `duration` určuje dĺžku trvania celého pohybu a na jeho základe sa vypočíta zmena natočenia kĺbu v každom cykle. Taktiež je prítomné pole `startTime`, ktoré reprezentuje čas, kedy sa spustí vykonávanie pohybu.

Polia:

Názov poľa	Typ	Popis
<code>jointType</code>	<code>JointType</code>	Kĺb, ktorý je potrebné natočiť
<code>startAngle</code>	<code>float</code>	Uhol, v ktorom má byť kĺb natočený pri začiatku pohybu
<code>endAngle</code>	<code>float</code>	Konečný uhol, do ktorého chceme kĺb dostať
<code>duration</code>	<code>float</code>	Počet cyklov, ktorý má pohyb trvať
<code>startTime</code>	<code>float</code>	Čas od začiatku vykonávania stavu, kedy sa má začať vykonávať pohyb

Metódy:

Názov metódy	Návratový Typ	Popis
<code>execute</code>	<code>bool</code>	Vykoná pohyb, resp. pokračuje v jeho vykonávaní
<code>isPast</code>	<code>bool</code>	Vráti <code>true</code> , ak je kĺb za konečným natočením
<code>isClose</code>	<code>bool</code>	Vráti <code>true</code> , ak sa skutočná a konečná hodnota natočenia líši o hodnotu menšiu ako je tolerancia

IdleMove

Špecializácia triedy `Move`, ktorá implementuje koncové natočenie kĺbu. Poslúži hlavne pri implementácii koncových stavov akcie.

Polia:

Názov poľa	Typ	Popis
<code>isFinished</code>	<code>bool</code>	<code>True</code> , ak je kĺb natočený do koncovej pozície

Metódy:

Názov metódy	Návratový Typ	Popis
<code>execute</code>	<code>bool</code>	Vykoná pohyb do koncového natočenia

4.5.2. Problémy pri implementácii

Pri implementácii sa vyskytlo viacero problémov, medzi inými:

- Dodržiavanie podmienok s počiatočným natočením kĺbov
- Vykonávanie pohybu skutočne až po uplynutí požadovaného času pred jeho začatím
- Zisťovanie, kedy je pohyb skutočne ukončený
- Výpočet zmeny natočenia kĺbu v aktuálnom cykle
- Zaistenie, že pohyby jedného kĺbu vykonávané sekvenčne sa vykonajú podľa požiadaviek

Riešenie problémov:

Počiatočné natočenie v tejto fázy vývoja nie je fixné a je možné ho zanedbať. V budúcnosti bude tomuto problému venované úsilie.

Čo sa týka druhého problému, tento bol vyriešený tak, že pred spustením pohybu v metóde `execute()` v triede `State` sa kontroluje rozdiel medzi súčasným stavom hodín a časom zahájenia vykonávania stavu, resp. či tento rozdiel je väčší ako pole `startTime` v aktuálnom pohybe.

Momentálne je pohyb považovaný za ukončený, keď sa kĺb dostane do natočenia za uhol `endAngle`, s prihliadnutím na predchádzajúcu zmenu natočenia, a súčasne uplynula doba väčšia ako je trvanie pohybu.

Po viacerých neúspešných pokusoch o počítanie inkrementu v každom cykle vykonávania pohybu sa momentálne počíta zmena natočenia iba raz, a to na začiatku vykonávania pohybu. Vo veľmi blízkej budúcnosti sa plánuje so zmenou, ktorá by umožňovala funkčné prepočítavanie prírastku uhla pre každý cyklus.

Poslený zo spomenutých problémov, teda riešenie vykonávania sekvenčných pohybov v rámci stavu je vo fázi riešenia, nakoľko pôvodný dizajn s takýmto riešením nepočítal. Riešenie tohto problému je závislé na riešení vyššie opísaného problému s inkrementáciou natočenia v každom cykle, ako aj problémom so zistením dokončenia pohybu.

4.5.3. Parametre príkazového riadka

Kvôli prepojeniu editora pohybov a samotného agenta bolo potrebné pridať do implementácie agenta ošetrovanie argumentov programu. Vzhľadom na to, že v budúcnosti bude zrejme potrebné pridávať a modifikovať existujúce argumenty, je vhodné vytvoriť úložisko prepínačov a argumentov. Pretože aj triedy na spodných úrovniach môžu potrebovať prístup k týmto argumentom, malo by byť možné manipulovať s týmto úložiskom bez toho, aby sa

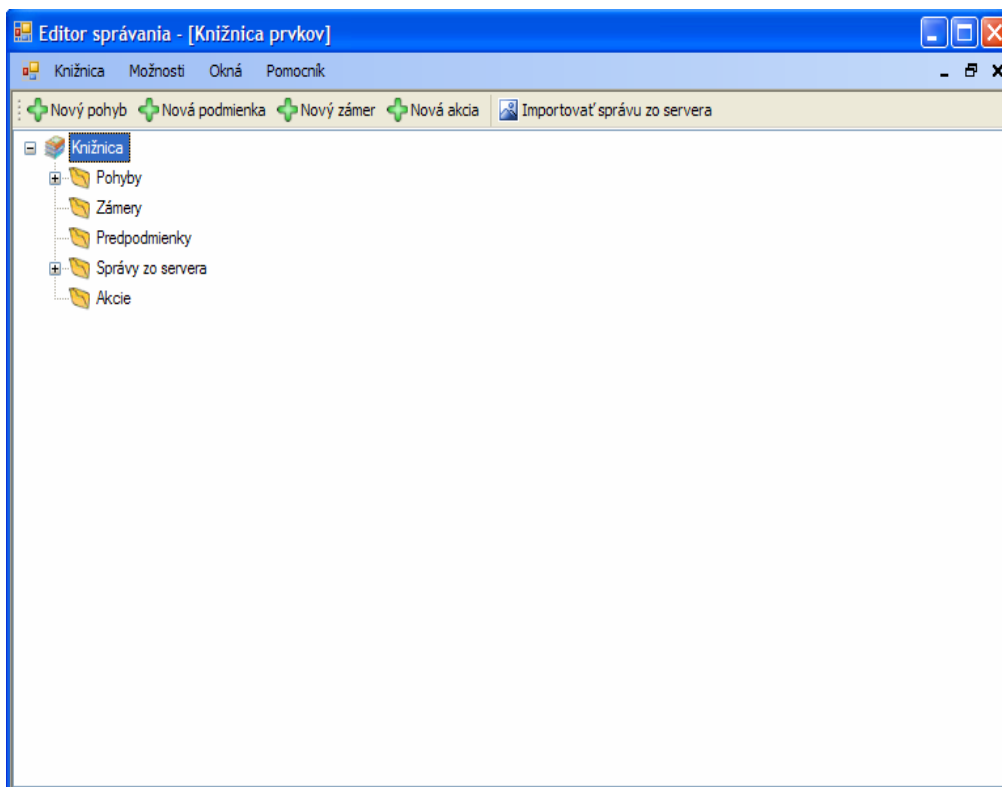
musela referencia na neho šíriť po celej hierarchii volaní, aby sa predišlo veľkým zmenám pri pridaní ďalšieho argumentu, alebo triedy, ktorá chce k argumentom pristupovať. Ako vhodné riešenie sa naskytlo použitie návrhového vzoru Singleton na implementáciu repozitára ConfigRepository. Vďaka tomuto je možné pomocou statickej metódy instance() získať referenciu na unikátnu inštanciu repozitára a čítať z neho argumenty, prípadne pridávať do neho nové. Repozitár je implementovaný ako wrapper na kolekciu objektov hašovacia mapa, kde kľúčom k argumentu(om) je enumerácia Option, ktorá musí byť rozdielna pre každý prepínač.

5 Implementácia

V tejto kapitole sa nachádza opis a vylepšenie niektorých častí prototypu. Hlavnou snahou bolo vytvoriť editor pohybov, ktorý by umožňoval jednoduchú a rýchlu tvorbu pohybov. Takto vytvorené pohyby sa dajú následne vhodne využívať pri riešení vyššej logiky správania sa robota.

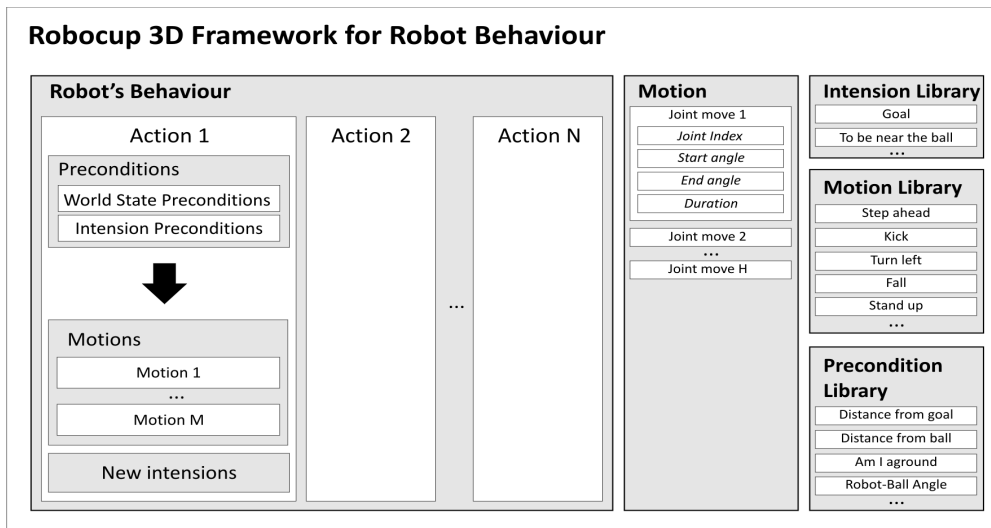
5.1. Editor správania

Editor správania (Obr. 17) sa dá prepojiť s agentom a so serverom a následne umožňuje simulovať vytvorené pohyby. Prepojenie znamená, že editor vyskladá zadané parametre a s tie pošle agentovi pri vytváraní nového procesu. Takéto pohyby netreba pracne programovať, ale postačuje len experimentálnym spôsobom zlepšovať.



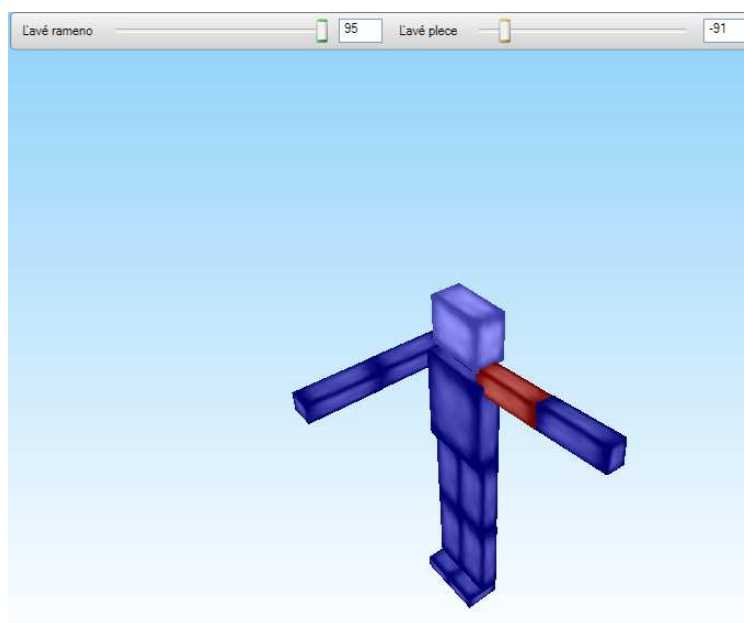
Obr. 17 - Editor správania

Pri vytváraní editora sme navrhli základný framework jeho správania, z ktorého sme následne vychádzali (Obr. 18).



Obr. 18 - Framework Editor správania

Pohyby v editore sa dajú vytvárať dvomi spôsobmi pričom oba spôsoby sa dajú ľubovoľne kombinovať. Jeden spôsob umožňuje používateľovi nastaviť kĺb robota, ktorý plánujeme po stanovenú dobu natočiť o zvolený uhol. Tento spôsob sme používali na vytvorenie prvých pohybov. Následne sme vytvorili pre budúcich používateľov druhý spôsob vytvárania alebo editácie pohybov, ktorý umožňuje priamo na modeli robota nastaviť kĺby a natočenia (Obr. 19). Tento model robota si používateľ môže prispôbiť tak, že je možné nahráť z konfiguračného súboru model robota podľa predefinovaných parametrov. Momentálne sú prístupné dve verzie a to nami vytvorený model robota a model robota Nao.



Obr. 19 - Vizualná editácia pomocou modelu robota

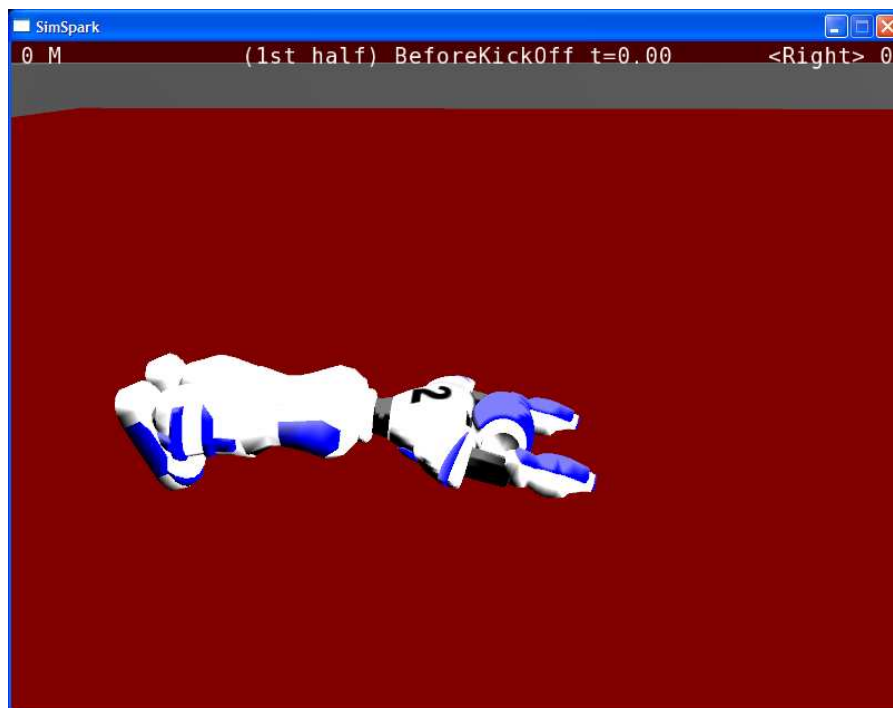
Editor správania nám umožňoval vytvoriť základnú sadu pohybov, ktorá nám potvrdila, že takáto cesta môže byť prospešná aj pre nasledujúce tímy, nakoľko tieto pohyby sú funkčné a aj ich prípadná editácia je relatívne rýchla a jednoduchá. To umožňuje budúcim tímom samotné pohyby ešte viac vylepšiť a zefektívniť ich vykonávanie.

5.1.1. Vytváranie pohybov

Na to, aby sme overili funkčnosť editora sme vytvorili viacero pohybov. Samotné vytváranie pohybov sa zlepšovalo s postupným pridávaním viacerých vylepšení editora.

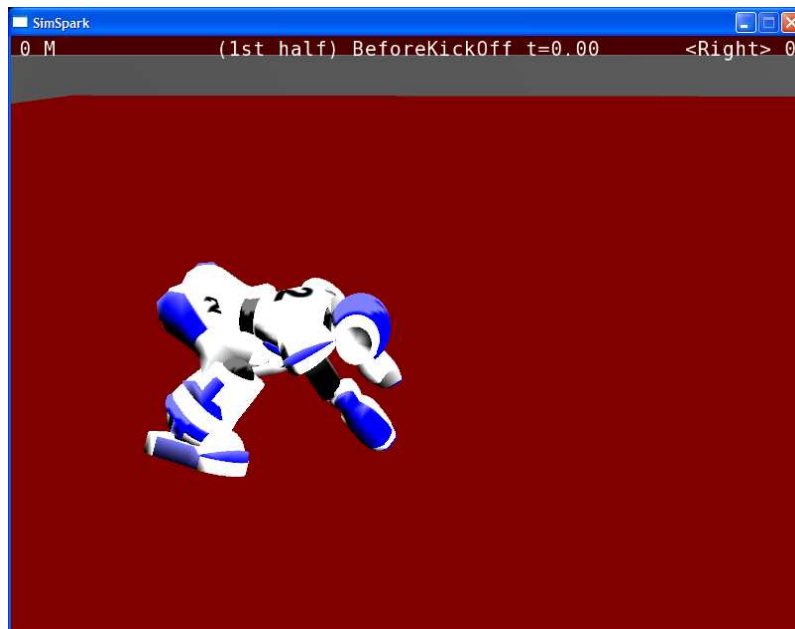
Pri prvých pohyboch vznikol problém, že hráč po zadefinovaní natočení kĺbov neudržal stabilitu. Preto sme sa zamerali na to, aby sa hráč v prípade pádu dokázal postaviť. Toto sa podarilo vyriešiť a ukážka pohybov vstávania sa nachádza na nasledujúcich obrázkoch.

Na to aby sme mohli vytvoriť pohyb vstávanie bolo potrebné hráča najskôr zhodiť na zem (Obr. 20).



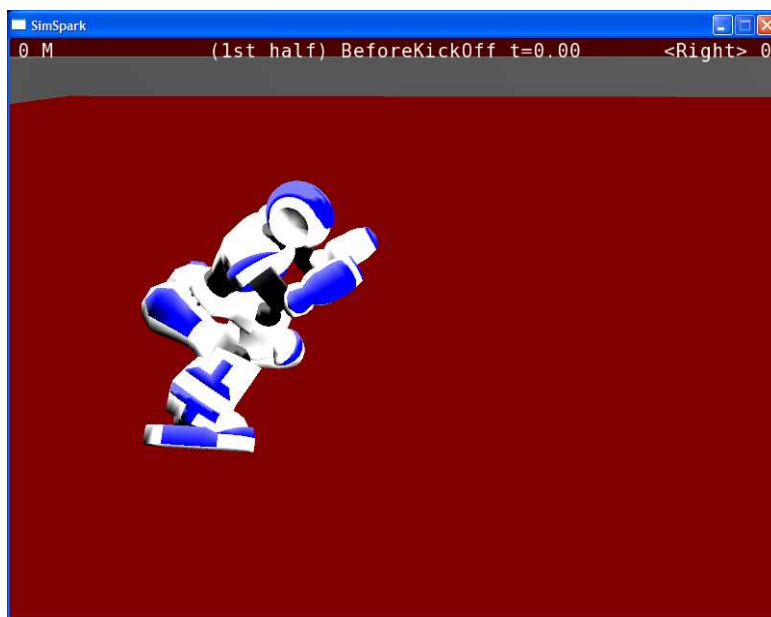
Obr. 20 - Hráč ležiaci na zemi

Následne sme vytvorili viacerými natočeniami kĺbov prvú časť pohybu ,ktorá umožní hráčovi dostať sa na nohy a zároveň sa pri tomto pohybe podopierať rukami tak, aby znova nepadol (Obr. 21).



Obr. 21 - Hráč pomaly vstáva

Táto poloha bola pri vstávaní kritická. Po jej úspešnom absolvovaní bola pravdepodobnosť, že hráč znovu spadne skoro nulová. Z tohto postoja následne bolo potrebné hráča ešte lepšie stabilizovať a to pohybom ako keby si hráč sadal na stoličku (Obr. 22).



Obr. 22 - Hráč je polosede

Poslednou fázou pohybu vstávania je vzpriamenie sa robota (Obr. 23). Takto stojaci robot môže znovu vykonávať činnosť, ktorú vykonával pred pádom. Môže však začať vykonávať aj inú činnosť podľa potreby a vývoja hry.



Obr. 23 - Hráč stojí vzpriamene

Na to však, aby robot mohol niečo urobiť nepostačuje len vstať zo zeme, ale je potrebné, aby sa vedel pohybovať po ihrisku. Preto sme museli vytvoriť pohyb chôdze. Prvý pohyb chôdze, ktorý sme vytvorili bol síce stabilný, ale neposkytoval rýchlosť a tým pádom nebol až tak veľmi efektívny. Preto sme potrebovali tento pohyb zrýchliť.

To však vyžadovalo vytvoriť úplne nový pohyb, ktorý však nie je až tak veľmi stabilný ako jeho predchodca. Pri samotnej hre je preto tieto pohyby vhodné meniť podľa potreby. Rýchlejší pohyb by sa dal používať v prípade útoku a pomalší pohyb v prípade obrany. To však nemusí byť pravidlo. Ukážka rýchlejšieho pohybu hráča na ihrisku je zobrazená na nasledujúcich obrázkoch.

Pohyb, ktorý sme vytvorili ako prvý bol jeden krok ľavou nohou, ktorý sa skladal z vykročenia (Obr. 24) a následného dokročenia ľavej nohy (Obr. 25).

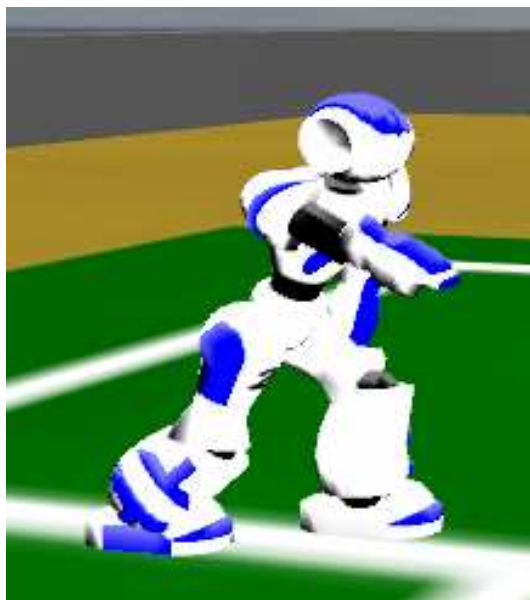


Obr. 24 - Výkrok ľavou nohou



Obr. 25 - Dokročenie ľavou nohou

Tento pohyb sme následne rozšírili aj o pohyb pravej nohy tak, aby hráč spravil oba kroky za sebou. To bolo trochu náročnejšie nakoľko hráč strácal stabilitu. Z toho dôvodu bolo nutné ho mierne nakloniť dopredu (Obr. 26).



Obr. 26 - Naklonenie sa dopredu

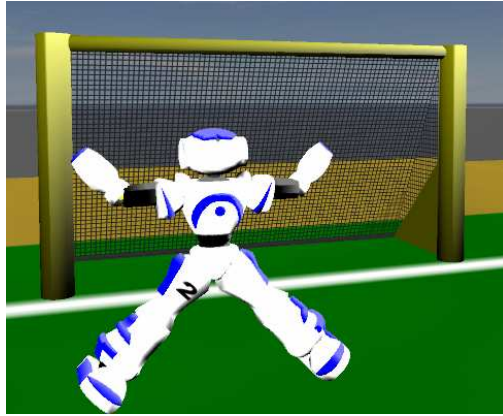
Po tomto naklonení už musel hráč len dokročiť pravou nohou a tým vytvoril jeden krok ľavou a jeden krok pravou nohou (Obr. 27).



Obr. 27 - Dokročenie pravou nohou

Takýto dvojkrok sa dá pomocou editora jednoducho zduplicovať a tým pádom bola takto vytvorená chôdza hráča, ktorá mu umožňuje prejsť cez celé ihrisko.

Editor umožňuje vytvoriť množstvo pohybov a preto sme vytvorili zopár pohybov aj pre brankára, ktoré sú však iba na základnej úrovni a pre samotnú hru ich je potrebné ešte zefektívniť. Jedným z takýchto pohybov je aj rozkročenie tak, aby brankár zabral čo najviac miesta z bránky (Obr. 28).



Obr. 28 - Postoj brankára

Medzi základné pohyby, ktoré bolo potrebné vytvoriť patria aj pohyby otáčania sa. Na to aby sa hráč mohol pohybovať po ihrisku správnym smerom je potrebné ho natočiť tak, aby bol tvárou k bodu kam sa má dostať. K tomu nám slúžia vytvorené pohyby otočenia sa vpravo a vľavo. Tieto pohyby sa podobne ako pohyb chôdze dá viackrát zopakovať a tým vytvoriť otočenie hráča okolo svojej osi.

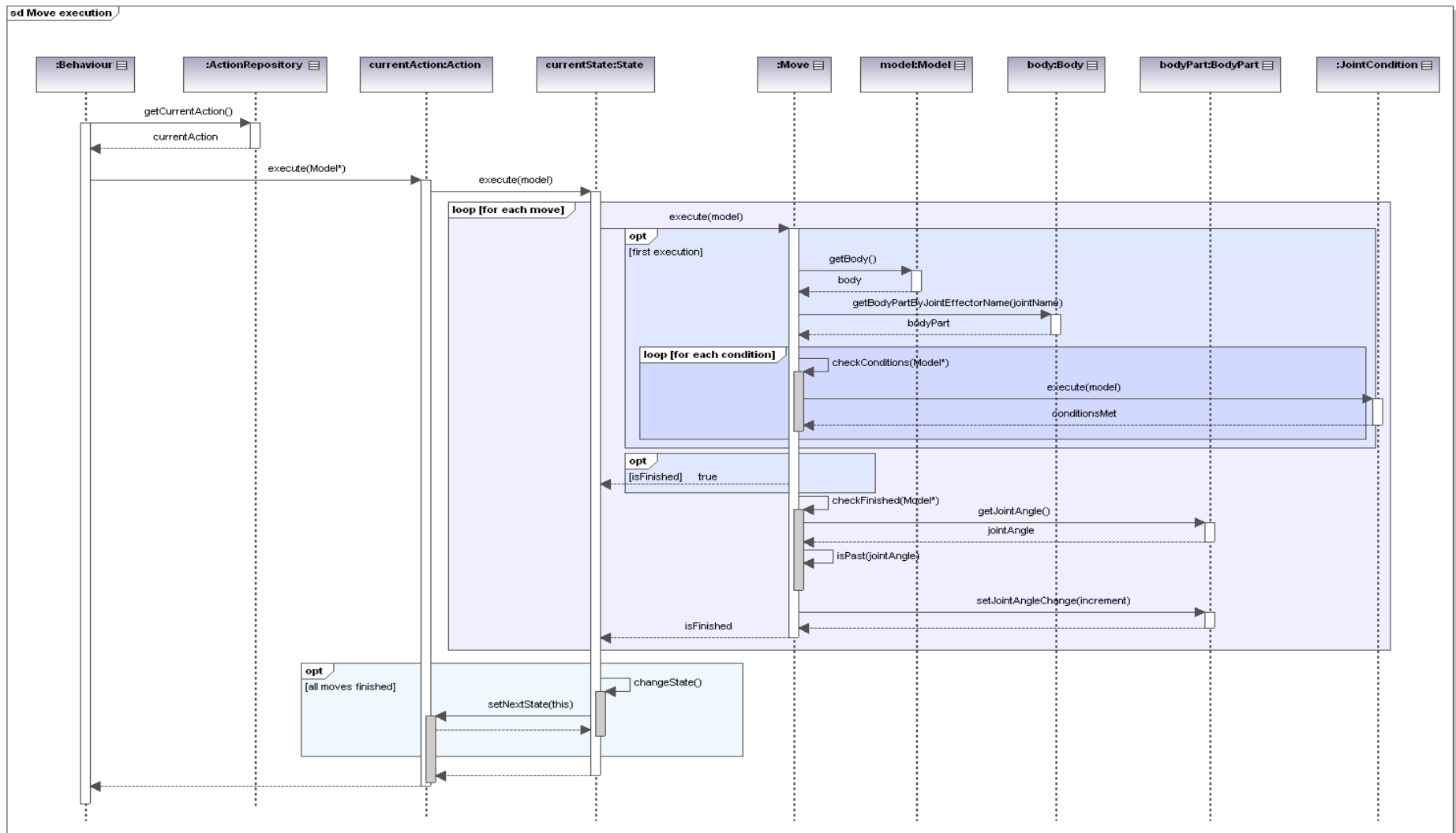
Pri vytváraní pohybov nastávali aj nepredvídateľné situácie. Jedna takáto situácia nastala pri vytváraní pohybu vstávania. Hráč nezostal stabilný a prevalil sa dopredu. Následne vykonal ďalšie natočenia kĺbov a na koniec zostal v polohe stojky na hlave (Obr. 29).



Obr. 29 - Stojka na hlave

5.2. *Vykonávanie pohybov*

Postup vykonávania je zobrazený na sekvenčnom diagrame na nasledujúcej strane (Obr. 30). Pred samotným spustením pohybu je potrebné nastaviť cesty k serveru a k agentovi.



Obr. 30 - Sekvenčný diagram vykonania pohybu

Po nastavení ciest na agenta a server je možné cez editor spustiť vykonanie samotného pohybu. Tento pohyb sa dá spustiť v normálnom režime, ale aj špeciálne. V tomto špeciálnom režime je možné pohyby robota spomaliť alebo zrýchliť. Taktiež je možné oneskoriť čas spúšťania samotného pohybu o hodnotu, ktorá je potrebná aby sa používateľ dostal do prostredia servera a mohol si pozrieť ako sa vykoná celý vytvorený pohyb.

5.3. Message Parser

Pre lepšie pochopenie práce servera sme vytvorili mechanizmus, ktorým je možné ľubovoľný pohyb robota, vykonaný na servery SimSpark, zaznamenať, preniesť do editora pohybov a tam ho prehrať. Takto zaznamenané pohyby sa nachádzajú v Knižnici správ zo servera.

5.3.1. Implementácia spätnej väzby (Server -> Editor)

Následne bolo potrebné vytvoriť nasledovnú funkcionálnosť:

- Zaznamenávanie správ zo servera, ktoré prijme agent,
- Parsovanie správ na strane editora správania a vloženie rozpoznaných pohybov do pripravených dátových štruktúr,
- Vykonanie a vykreslenie pohybu uloženého v daných štruktúrach.

5.3.2. Zaznamenávanie správ agentom

Agentovi sme implementovali možnosť prijímať argumenty z príkazového riadku. Pre účely zaznamenávania serverových správ bol zavedený argument `-message`, ktorý prijíma ako parameter názov súboru, do ktorého budú logované správy. V prípade, že je daný argument vynechaný, logovanie serverových správ bude vypnuté. Treba zdôrazniť, že ide o úplne samostatné logovanie, ktoré je nezávislé od debugovacích, chybových a iných výpisov.

Zaznamenávanie správ na strane agenta je implementované v triede `Communication` v metóde `receive(bool *run, char *message)`, ktorej parameter `message` obsahuje parsovanú správu.

5.3.3. Parsovanie správ v editore

Parsovanie správ v editore správania je implementované v triede `ServerMessageParser` v metóde `ParseMotions()`, ktorá vracia zoznam pohybov a zmeny gyroskopu (akcelerátora).

V súbore sú zaznamenané aj statické polohy kĺbov, aj keď s nimi nebolo v danej iterácii pohybované. Takéto záznamy sú rozpoznané a nevytvárajú sa na základe nich pohyby,

keďže reálne žiadny pohyb nereprezentujú. Medzi pohyby sú vložené iba záznamy, ktoré reálne afektujú pohyb hráča. Naparsované znaky získame spočítaním aktuálnej pozície a maximálnej dĺžky s-výrazu pre daný kĺb. Odpočítame konštantu, aby sme v prípade, že sme zasiahli pri parsovaní do následovného kĺbu, tento kĺb nepreskočili, ale správne naparsovali v ďalšej iterácii cyklu. Pri parsovaní súboru sa načítavajú aj hodnoty gyroskopu.

5.3.4. Vykonalie pohybu v editore

Na vykonanie zaznamenaného pohybu v editore je použitý engine, ktorý sa používa aj na vykonávanie modelovaných pohybov. Toto je zabezpečené použitím rovnakého rozhrania (dátovej štruktúry JointMotion) pri modelovaní nových pohybov, aj pri prehrávaní zaznamenaných pohybov.

6 Plán do budúcnosti

Cieľom je vytvoriť rámec na ovládanie hráča na vyššej úrovni tak, aby bol schopný aktívne sa zúčastniť na turnaji RoboCup na FIIT. Niektoré časti sme nestihli implementovať počas semestra, ale plánujeme to dokončiť tak, aby sme sa mohli zúčastniť turnaja.

6.1. Vyššia vrstva riadenia

Základným predpokladom pre vyššiu vrstvu je knižnica už vytvorených stabilných pohybov. Tie budú vyššou vrstvou kombinované a volané vo vhodnom poradí. Celé vykonávanie bude riadené pravidlami. Pravidlá sa budú vyberať sekvenčne. Pravidlo obsahuje svoj názov, podmienku zloženú z viacerých častí zapísanú v postfixovom tvare (z dôvodu jednoduchšieho vykonania) a výslednú akciu. Výsledná akcia určí nasledujúcu činnosť, ktorou je najčastejšie voľba nasledujúceho pohybu. V prvej fáze potrebnými pohybmi sú:

- Chôdza,
- Otočenie sa o určitý malý uhol, vhodné sú aj otočenia o rôzne iné uhly, vhodné je mať samostatne otočenia doľava aj doprava,
- Kopnutie do lopty priamo dopredu,
- Otočenia sa pri lopte, slúži na správne nasmerovanie pred kopnutím lopty, môže byť aj rovnaké ako bežné otočenie,
- Vstávanie zo zeme,
- Pohyb na vrátenie hráča no nulovej pozície (ruky môžu byť vedľa torza), tento pohyb musí byť priamo integrovaný v hráčovi a nie v externom súbore a musí byť schopný dostať agenta do nulovej pozície z hocijakej inej, neuvažujeme pozíciu hráča vzhľadom na ihrisko, ale iba vzhľadom na natočenia kĺbov.

Vyššia logika bude zapísaná a načítaná, rovnako ako samotné pohyby, z konfiguračných súborov. Editovanie súborov sa bude uskutočňovať pomocou editora. Vyššia logika nebude vyhodnocovaná pri každom cykle (trvajúcim 20 ms), ale bude volaná v určitých intervaloch a stavoch pohybov, kedy je možné meniť pohyby a rozhodovanie. Volaná je z nižšej úrovne. Jedná sa o sekvenčný výber pravidla a následné uskutočnenie vybraného pohybu nižšou vrstvou. Po vykonaní pohybu sa proces opakuje. Podmienky v pravidlách sú logickým výrazom zostrojeným z nasledujúcich predikátov (funkcií):

- IsAgentFallenOnTheGround – zistí, či je hráč padnutý na zem.
- IsBallInDirectView – zistí, či hráč vidí loptu priamo pred sebou (s odchýlkou 5°).
- IsBallOnTheLeft – zistí, či agent vidí loptu naľavo.
- IsBallOnTheRight – zistí, či agent vidí loptu napravo.
- CanKickTheBall – zistí, či je lopta v dosahu a hráč je schopný do nej kopnúť.
- IsGoalpostInDirectView – zistí, či sa bránka nachádza priamo pred hráčom.
- IsGoalpostOnTheLeft – určuje, či hráč vidí bránku naľavo.
- IsGoalpostOnTheRight – určí, či hráč vidí bránku napravo.

Pričom musia platiť nasledujúce vlastnosti:

- Môže platiť iba jeden predikát z IsBallInDirectView, IsBallOnTheLeft a IsBallOnTheRight.
- Môže platiť iba jeden predikát z IsGoalpostInDirectView, IsGoalpostOnTheLeft a IsGoalpostOnTheRight.
- Predikáty IsGoalpostInDirectView, IsGoalpostOnTheLeft a IsGoalpostOnTheRight uvažujú stred bránky.
- Predikáty IsGoalpostInDirectView, IsGoalpostOnTheLeft a IsGoalpostOnTheRight uvažujú súperovu bránku.
- Predikáty IsBallInDirectView, IsBallOnTheLeft, IsBallOnTheRight, IsGoalpostInDirectView, IsGoalpostOnTheLeft a IsGoalpostOnTheRight neuvažujú Z-ovú súradnicu predstavujúcu výšku, pracujú v dvojrozmernej sústave pri pohľade zhora na ihrisko.

7 Testovanie

Testovanie prebiehalo počas celej implementácie. Pri testovaní sme výsledné hodnoty porovnávali s hodnotami, ktoré sme vypočítali. V agentovi je implementovaný aj framework pre test, ktorý tiež vyhodnocuje výsledky a to tak, že porovná výsledky z rôznych metód a keď sa líšia iba o určitú malú odchýlku, tak sa výsledky vyhodnotia ako TRUE. Tento test sa spúšťa pomocou prepínača `-test`.

7.1. Testovanie prevodov medzi rôznymi sústavami:

Ukážka výsledkov testov na prevody medzi rôznymi sústavami.

Výsledok testu:

```
----- Test run results -----
-----
Test: QuaternionTest

TestQuaternionFromAxis          Passed
TestQuaternionRotation          Passed

Result:                          Passed
-----

Test: MathHelperTest

TestAngleBetween                Passed
TestGetRotationFromVectors      Passed
TestVectorsToRotation           Passed
TestRotationToQuaternion        Passed
TestAngleConverions             Passed
TestRotationToQuaternionToRotation Passed
TestQuaternionToRotation        Passed

Result:                          Passed
-----

Test: Vector3DTest

TestVectorProduct               Passed
TestVectorSum                   Passed

Result:                          Passed
-----

Test: Point3DTest

TestMatrixRotation              Passed

Result:                          Passed
-----

Test run result:                 Passed
```

7.2. Testovanie rotácie vektorov

Pre výpočty ťažiska a údajov z kamery hráča potrebujeme mať funkciu, ktorá je schopná počítať v karteziánskych súradniciach rotáciu bodu okolo vektoru o zadaný uhol. Daný výpočet uskutočňujeme pomocou Rodrigézovej metódy. Testujeme danú funkciu na rotáciu bodu. S uhlami pracujeme v stupňoch.

Pri testoch považujeme za správny výsledok, ktorý sa v ani jednej súradnici neodlišuje o viac ako jeden stupeň. Testované vstupné a očakávané výstupne hodnoty zapíšeme do tabuľky.

Tab. 1 - VSTS Unit Testing

Vektor okolo, ktorého sa rotuje	Bod, ktorý rotujeme	Uhol, o ktorý daný bod rotujeme	Očakávaný bod
1; 1; 0	1; 0; 0	180	0; 1; 0
12; 12; 0	56; 0; 0	90	28; 28; -39,59
1; 1; 1	5; 6; 7	270	5,42; 7,15; 5,42
84; 35,657; 23	23; 45; 67	196	68,37; 11,77; -47,21
44; -96; 0	4; 0; 4	143	-4,13; -3,72; -1
1; 1; 0	0; 0; 0	0	0; 0; 0
1; 1; 0	1; 0; 0	0	1; 0; 0
1; 1; 0	0; 0; 0	96	0; 0; 0
1004; 354; -569	5849,32; 55; 74	360	5849,32; 55; 74
46; 83; 4	1; 23; 0	89	8,84; 18,11; 11,11

Testovaná funkcia pre všetky vstupné hodnoty vrátila očakávané hodnoty (v rámci určenej odchýlky).

7.3. Testovanie ťažiska a oporných bodov

V implementácií samotného hráča ako aj v editore pri jeho vizualizácii počítame ťažisko. Výpočet spočíva v zostavení modelu hráča z jeho častí tela a ich následných rotáciách podľa aktuálnych natočení kĺbov, ktoré nám server posiela. Model hráča sme zostavili na základe údajov z oficiálneho manuálu a konfiguračných súborov serveru. Navyše počítame oporné body, ktorými sú časti tela umiestnené v danom okamihu najnižšie – na zemi.

Presné údaje o polohe častí tela, ťažiska ani oporných bodoch určiť nevieme. Preto pristupujeme k intuitívnemu testovaniu – necháme hráča vykonať určité pohyby a pomocou

logovanie sledujeme správanie sa ťažiska, častí tela a oporných bodov. Pre daný test sme zvolili pohyby vstávania a stojky na hlave.

Pri danom teste treba brať do úvahy hlavne rozloženie hmotností častí tela. Výrazne najťažšie je torzo a potom tiež dosť ťažké sú nohy. Dané hodnoty sú zamýšľané pre zjednodušenie stabilizácie hráča. Dôsledkom však je aj fakt, že ťažisko v praxi väčšinu času sa bude nachádzať medzi nohami a torzom, nie je ho možné pomocou rúk a hlavy jednoducho zmeniť. Daný fakt potvrdili aj naše pokusy. Správanie sa ťažiska a oporných bodov zodpovedá očakávaným hodnotám pri daných pohyboch. Napr. jasne je vidieť pri skrčení hráča, priblíženie jeho ťažiska smerom do stredu a dopredu, kde smerujú nohy. Podobne pri kľaknutí vidieť zmenu oporných bodov. Test považujeme za úspešný.

7.4. Testovanie gyroskopu z dát posielaných serverom

Hráč RoboCupu 3D má v torze perceptor nazvaný gyroskop. Jedná sa, hlavne pri výpočte ťažiska a určení polohy, o veľmi dôležitý údaj. Bohužiaľ perceptor gyroskopu nie je gyroskopom v zmysle fyziky a nevracia absolútne natočenia voči pôvodnej začiatočnej pozícii. Údaje so servera predstavujú (ako na základe experimentov predpokladáme) uhlovú rýchlosť v danom momente v stupňoch za jednu sekundu. Z tohto dôvodu máme implementovanú funkciu, ktorá pripočíta zmenu rotácie hráča k aktuálnej hodnote gyroskopu. Testujeme výslednú hodnotu gyroskopu (absolútneho natočenia tela hráča) po vykonaní určitých pohybov.

Keďže nevieme vyčítať presné hodnoty gyroskopu v danom momente a navyše ani nevieme daný test automatizovať, presnosť výpočtu gyroskopu testujeme vizuálne. Pridali sme kód do implementácie agenta, ktorý zabezpečuje pravidelné logovanie hodnoty vypočítaného gyroskopu. Následne sme pomocou nášho editora spustili hráča s príkazmi pre vykonanie pohybu – stojky na hlave. Jedná sa o pohyb, kde hráč výrazne mení svoju polohu relatívne na začiatočnú pozíciu. Preto sme očakávali, že aj nami vypočítaná hodnota sa bude adekvátne meniť.

Test sme vykonali niekoľkokrát. Narazili sme na problémy s nedostatočným výkonom počítača resp. vysokými hardvérovými nárokmi na beh serveru aj hráča súčasne. Tiež sme riešili problém logovania do konzoly. Keďže logovanie je výrazne pomalé, extrémne spomaľuje hráča a ten následne stráca správy so servera. Preto sme ponechali iba logovanie do súboru. Po daných úpravách vypočítavané hodnoty gyroskopu približne zodpovedajú natočeniu hráča ako ho vidíme na monitore serveru. Možné nepresnosti sú hlavne dôsledkom strácania sa komunikácie so servera z dôvodu nedostatočného výkonu počítača. Tiež určitý

negatívny vplyv na nepresnosť hodnôt gyroskopu môže mať šum, ktorý server úmyselne vnáša do posielaných údajov. Bohužiaľ aj naša metóda výpočtu gyroskopu nie je úplne presná, keďže sa jedná o predikciu. Výsledok testu však hodnotím skôr pozitívne.

7.5. Testovanie kladného uhla medzi vektormi

Počas výpočtu plochy stability potrebujeme počítať kladný (v smere hodinových ručičiek) uhol od prvého vektora k druhému. Na splnenie daného cieľa máme vytvorenú funkciu, ktorú testujeme. Funkcia sa používa pri určení nasledujúceho bodu, ktorý pridáme do konvexnej plochy stability. Pridávame ten bod, ktorý vidno pod čím menším uhol od vektoru k predchádzajúcemu bodu. S uhlami pracujeme v radiánoch.

Pri testoch považujeme za správny výsledok, ktorý sa neodlišuje od očakávaného o viac ako tisícinu radiánu. Testované vstupné a očakávané výstupné hodnoty zapíšeme do tabuľky.

Tab. 2 - Výpočet kladného uhla medzi vektormi

Prvý vstupný vektor	Druhý vstupný vektor	Očakávaný uhol
0; 20	1; 1	PI / 4
30; 30	1; 0	PI / 4
23; 23	1; -1	PI / 2
0; -1	0; 1	PI
-1; 0	-45; 45	PI / 4
-1; 1	-1; 0	7 * PI / 4
-12; 54	35; 0,33	1,78003
45; 96,31	91,32; 4	1,08992
-36; 3	-1; 0	6,20004
-4; -446	-62; -99	0,55053

Testovaná funkcia pre všetky vstupné hodnoty vrátila očakávané hodnoty (v rámci určenej odchýlky).

7.6. Testovanie konvexnej plochy

Plocha stability je vytvorená z bodov častí tela hráča, ktoré sa dotýkajú zeme. Dané body určíme ako tie najnižšie položené. Tie následne obklopíme konvexnou plochou pozostávajúcou iba z niektorých daných bodov. Vypočítaná plocha predstavuje plochu

stability – pokiaľ sa ťažisko nachádza nad ňou, hráč je stabilný. Testujeme funkciu, ktorá zo zadaných bodov vyberie ich podmnožinu zodpovedajúcu konvexnej ploche.

Pri testoch považujeme za správny výsledok podmnožinu zadanej množiny, ktorá má rovnaký počet bodov ako nami očakávaná a obsahuje rovnaké body ako nami očakávaná množina, pričom na ich poradí nezáleží. Testované vstupné a očakávané výstupne množiny bodov zapíšeme do tabuľky.

Tab. 3 - Výpočet konvexnej plochy

Vstupná množina bodov	Výstupná množina bodov predstavujúca konvexnú plochu
[-3; -2], [-3; 0], [-3; 2], [-2; 0], [-1; 3], [0; -3], [0; -1], [0; 0], [1; 0], [1; 2], [2; -1], [3; 2], [3; 1], [3; 0], [3; -1]	[-3; -2], [-3; 0], [-3; 2], [-1; 3], [3; 2], [3; 1], [3; 0], [3; -1], [0; -3]
[1; 1], [0; 0], [0; 0,5], [-1; 1]	[-1; 1], [1; 1], [0; 0]
[24; 94], [34.565; -354], [567.3; 34], [-12; 1], [93; 23], [-59; -44], [9; 9], [-34; 56], [95; 1254]	[-59; -44], [95; 1254], [567.3; 34], [34.565; -354]

Testovaná funkcia pre všetky vstupné hodnoty vrátila očakávané hodnoty.

7.7. Testovanie pohybov

Pohyby vytvorené v editore boli testované spôsobom, že sa vykonali agentom a začiatkový a konečný stav sa logoval. Potom sa tento výstup porovnal s očakávanými hodnotami.

Ukážka z logovaného výstupu:

Move:

```

JointName:    lae1
CurrentTime:  1843
CurrentAngle: 0
StartTime:    0
EndTime:      1000
StartAngle:   0
EndAngle:     -90
    
```


8 Zhodnotenie

Zo začiatku projektu naša práca spočívala hlavne v analýze existujúcich tímov. Zamerali sme sa hlavne na úspešnejšie tímy, ktoré už dosiahli výraznejších úspechov na súťažiach, ale aj na tímy, ktoré sa výraznejšie nepresadili a od nich sme si vzali ponaučenie, že takýmto smerom to asi nepôjde. Taktiež sme si pozreli aj videá na internete a skúmali sme, ako sa jednotlivý hráči pohybujú.

Veľkou pomocou pre nás bolo získanie serveru a hráča od minuloročného tímu, ktoré nám ušetrilo mnoho času. Ten je pri takomto projekte veľmi dôležité správne rozvrhnúť. K tomu nám pomáha viacero softvérových nástrojov.

Zo začiatku naše úsilie smerovalo hlavne k oboznámeniu sa ako so serverom, tak aj s hráčom a jeho ovládaním. K úspešnému zvládnutiu tohto cieľa nám výrazne dopomohla prezentácia, ktorú sme mali počas tímového stretnutia a ktorú viedli naši starší kolegovia. Tu sme získali širší pohľad na danú problematiku a bez ďalšieho zdržania sme sa mohli naplno venovať ďalšej analýze a hľadať riešenia na problémy.

Následne sme vytvorili ako prototyp Editor správania, ktorý sme naďalej zlepšovali a vyvíjali. Pomocou tohto editora sa jednoduchšie vytvárajú samotné pohyby, ale okrem toho sme do neho v letnom semestri pridali aj ďalšie funkcionality.

Tento projekt sme sa snažili vypracovať tak, aby na neho mohli nadviazať aj budúce tímy, ktoré sa budú tejto problematike na FIIT zaoberať. Veríme, že naša práca pomôže pri práci ďalším tímom ako aj iným študentom, ktorí túto tému budú riešiť na iných projektoch.

9 Použitá literatúra

- [1] Boedecker, Joschka, *SimSpark User's Manual* (verzia 1.1), 2008, posledný prístup 8.11.2008 <<http://labss2.fiit.stuba.sk/TeamProject/2007/team11is-si/download.html>>
- [2] Čerňanský, Michal, Ing. PhD., Inžiniersky seminár z neurónových sietí, 2008, posledný prístup 11.11.2008, <<http://www2.fiit.stuba.sk/~cernans/nn/>>
- [3] Kvasnička, Vladimír, prof., Ing., DrSc., Doktorandský seminár z umelej inteligencie, 2008, posledný prístup 11.11.2008, <<http://www2.fiit.stuba.sk/~kvasnicka/NeuralNetworks/>>
- [4] Little Green BATS, RoboCup Soccer Simulation 3D, 2008, posledný prístup 11.11.2008, <<http://www.littlegreenbats.nl/>>
- [5] RoboCup Official Site, 2008, posledný prístup 12.11.2008, <<http://www.robocup.org/>>
- [6] SourceForge.net, Little Green BATS, 2007, posledný prístup 11.11.2008 <<http://sourceforge.net/projects/littlegreenbats>>
- [7] Stone Peter, Kalyanakrishnan Shivaram, RoboCup 2007, Atlanta, U.S.A., 2007, posledný prístup 10.11.2008, <http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/>
- [8] Team UIAI, UI-AI3D 2007 Team Description, 2007, posledný prístup 12.11.2008, www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/uiai2007TDP.pdf
- [9] Team Virtual Werder 3D, RoboCup 2007 Team Description Paper. Universität Bremeht, 2007, posledný prístup 10.11.2008, <http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/rc07_vw3d_tdp.pdf>
- [10] Tím Hviezdna 11, Robocup3D, Dokumentácia k projektu. FIIT STU, 2007, posledný prístup 8.10.2008, <<http://labss2.fiit.stuba.sk/TeamProject/2007/team11is-si/>>

Príloha A – Prílohy k editoru pohybov

Súbor pohybu

Vytvorený hráč Robocupu vie spracovávať súbor s definovanou akciou. Akcia definovaná v súbore pozostáva z viacerých stavov a tie obsahujú čiastkové pohyby. Paralelne sa môže vykonávať viacero pohybov (takých ktoré si logicky neodporujú).

Štandardná prípona súboru s akciou je .rac (**R**obot **A**Ction).

Popis štruktúry súboru

Súbor akcie robota je jednoduchý textový súbor. Údaje sú v súbore organizované po riadkoch. Formát súboru rozlišuje dva typy riadkov – skupina údajov a údaj. Syntax riadku s údajom je nasledovná: vlastnosť = hodnota.

Každý súbor musí obsahovať presne jednu skupinu údajov – [**ACTION**], v ktorej musí byť vnorená minimálne jedna skupina údajov [**STATE**] a v nej zas vnorená minimálne jedna skupina [**MOTION**].

Celá štruktúra súboru je veľmi jednoduchá a vyzerá nasledovne:

- [**ACTION**]
 - **NAME** – názov akcie definovanej v súbore
 - **DESCRIPTION** – popis akcie definovanej v súbore
 - [**STATE**] – stav akcie
 - **INDEX** – index stavu
 - **NAME** – názov stavu
 - [**MOTION**] – elementárny pohyb robota
 - **INDEX** – index pohybu
 - **JOINT** – číselný kód kĺbu
 - **TIME** – čas (v ms) od začiatku vykonávania stavu, od ktorého sa začne vykonávať daný pohyb
 - **DUR** – trvanie pohybu (v ms)
 - **STARTPOS** – začiatkové natočenie kĺbu
 - **ENDPOS** – koncové natočenie kĺbu

Príklad súboru s pohybom

```
[ACTION]
NAME=Pohyb rukami
DESCRIPTION=Pohyb ľavou a pravou rukou
[STATE]
INDEX=0
NAME=Stav #1
[MOTION]
INDEX=1
JOINT=6
TIME=0
DUR=2000
STARTPOS=90
ENDPOS=180
[MOTION]
INDEX=2
JOINT=6
TIME=2000
DUR=2000
STARTPOS=180
ENDPOS=90
[STATE]
INDEX=1
NAME=Stav #2
[MOTION]
INDEX=1
JOINT=2
TIME=0
DUR=1000
STARTPOS=90
ENDPOS=180
[MOTION]
INDEX=2
JOINT=2
TIME=1000
DUR=1000
STARTPOS=180
ENDPOS=90
```

RoboCup 3D Framework for Robot Behaviour

Peter NOSKO, Dušan RODINA, Daniel SLAMKA, Peter SMOLINSKÝ, Ondrej ŠEVCE, Ivan TOMOVIČ*

*Slovak University of Technology
Faculty of Informatics and Information Technologies
Ilkovičova 3, 842 16 Bratislava, Slovakia
tim007tp2008@gmail.com*

The aim of the RoboCup project is to create a team of robots, which could play football so well, that they would beat the best human football team [2]. A minor step towards achievement of this ambitious goal is to design a usable, extensible and robust framework, which can be used for modelling of robot's behaviour. The main goal of our work is to develop such framework. The framework allows defining the behaviour of a robot with a set of actions. These actions are composed of robot's limbs motions. The main advantage of our solution is that it can be used by future RoboCup developers to help develop more complex solutions in the future.

In the original version of the robot, adding a new action to the robot behaviour, which is generally one of the first tasks of every new RoboCup team, was a difficult, tedious task susceptible to many errors and involving too much of duplicate code. For these reasons we decided to create the framework, whose main purpose is to make creation and modification of various actions of the robot easier. This is achieved by providing a mechanism to treat all such actions similarly.

In the framework (Figure 1) we implemented, every action, the robot might perform (whether it is walking, passing a ball, dribbling, or shooting at the goal) is represented by an instance of a class that describes the action. The main advantage of this concept is that a user of the framework can easily define an action in the interactive editor, which is developed in parallel with the framework, and later use it in the robot simulation even with very little knowledge about how the limb movements are executed in the SimSpark simulator [1].

The behaviour of the robot can be represented as a sequence of actions that the robot has to perform. Each action has to meet its required preconditions in order to be scheduled and executed. An action itself can be further decomposed into separate motions of limbs, i.e. rotations of hinge joints. Let such motion be defined as the following quadruple: (J, T, D, A) , where J is the identifier of a joint which is to be bent; T is the time elapsed since the execution of the action before the move is started; D represents the duration of the entire move and A stands for the angle of rotation. With motions defined this way, it is possible to represent an arbitrary action as a set of groups of motions, ordered according to their start time. A motion group can contain motions that are to be synchronized with respect to each other.

* Master degree study programme in field: Software Engineering/Information Systems Supervisor: Ing. Marián Lekavý, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava.

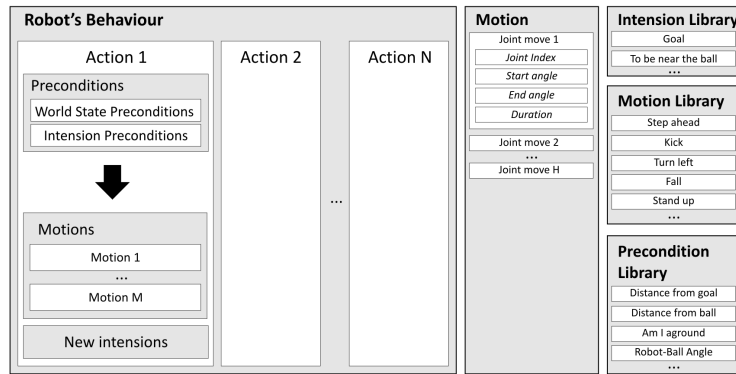


Figure 1. RoboCup 3D Framework for Robot Behaviour

Yet another of the most important features is that the actions that compose behaviours can be stored into a file and loaded by the robot when the need arises. The type of behaviour that the robot will conform to will then be selected by the higher logic on the basis of the robot's intentions, represented by predefined constants, and the fulfilment of behaviours'/actions' preconditions.

One of the problems, robot developers often have to cope with, is how to maintain the stability of the robot. It would be very frustrating if the robot kept falling down only because of small collisions or not-so-precise movements. That is why we propose a robot stability module, whose task is to balance the robot at the event that would otherwise lead to its fall. The idea is that the module will keep track of the momentum and the relative position against the stabilization points (usually the feet of the robot) of the robot's centre of mass, and via preventive movements and action correction attempt to regain stability.

Acknowledgement: This work was partially supported by the Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava. We would like to thank Lukáš Dvonč for his valuable consultations regarding the robot stability physics.

References

- [1] Boedecker, J., *SimSpark User's Manual* (version 1.1), 2008, [Online; accessed November 8th, 2008]. Available at: <http://labss2.fiit.stuba.sk/TeamProject/2007/team11is-si/download.html>
- [2] The RoboCup Federation, Robocup, [Online; accessed February 20th, 2009], Available at: <http://www.robocup.org/>

Príloha C – Externé testovanie

Dotazník pre externé testovanie

Meno testera: Marcel Kanta

Externé testovanie pre tím: Agenty 007

Pozn.: Odpovedajte na otázku zakrúžkovaním jedného z čísel 1 až 6. Číslo 1 až 6 predstavujú hodnotenie pričom číslo jeden predstavuje úplnú spokojnosť s testovaným produktom a číslo šesť predstavuje úplnú nespokojnosť s produktom. Číslo tri a štyri predstavujú hodnotenie pri neutrálnom postoji, pričom číslo tri je skôr pozitívne a číslo štyri skôr negatívne hodnotenie.

Otázky:

Otázka: Ako ste boli spokojný s prostredím Editoru správania?

Odpoveď: 2

Bol som spokojný, editor dokáže vo vizuálnom prostredí spraviť také veci, ktoré sa doteraz dali iba zdĺhavým programovaním. Hodnotenie 2, pretože sa dá čo rozširovať, ale v zásade aj táto verzia splnila požiadavky.

Otázka: Ako hodnotíte ovládanie Editoru správania?

Odpoveď: 3

Ovládanie je intuitívne, na prostredie si netreba zvykať a dá sa pochopiť aj bez čítania používateľského manuálu. Jedine som nepochopil, načo slúži tlačidlo synchronizácia, a pohybovanie s vizualizáciou hráča je netriviálne. Keď som chcel, aby sa hráč celý(nie kĺb) otočil presne tak, ako som chcel, nevedel som, ako pohnúť myšou, aby to išlo, hráč sa otáča zrejme okolo jednej nohy, mal by sa skôr okolo stredu tela.

Otázka: Ako hodnotíte efektívnosť tvorby pohybov v Editoru správania?

Odpoveď: 2

Každý pohyb sa dá spraviť presne tak, ako používateľ chce, dá sa nastaviť čas aj uhlovú rýchlosť. Pokročilému používateľovi by sa zišlo pridať také pohyby, ktoré nebudú mať konštantnú rýchlosť, ale ich rýchlosť bude záležať od vzorca, napr. Rovnomerné zrýchľovanie iba jedného kĺbu, alebo aj celej frekvencie pohybu.

Otázka: Ako hodnotíte tvorbu pohybov pomocou nastavení kĺbov na časovej osi?

Odpoveď: 3

Proti nastavovaniu kĺbov na časovej osi nemám nič, len tvorba v grafickom režime je omnoho lepšia.

Otázka: Ako hodnotíte tvorbu pohybov pomocou grafického modelu?

Odpoveď: 1

Tvorba je v tomto režime jednoduchá a efektívna.

Otázka: Ako hodnotíte kvalitu vytvorených pohybov?

Odpoveď: 5 (v0.67) / 1 (v7.0)

V čase skúšania som mal staršiu verziu editora aj hráča a hráč často padal, aj keď nemal. S novšou verziou sa padanie stratilo a pohyby sú presne, ako majú byť.

Otázka: Ako hodnotíte možnosti nastaviť rôzne pomocné výpisy Agenta?

Odpoveď: 3

Pomocné výpisy rozširujú funkčnosť, ale napr. výpisy na konzolu sa nedajú čítať, lebo idú príliš rýchlo a hráč vtedy začne padať.

Otázka: Ako hodnotíte možnosti zníženia nárokov na procesor pri vykonávaní pohybov?

Odpoveď: nehodnotím

Jediné nájdené možnosti boli vypnutie logovania a výpisov.

Otázka: Ako hodnotíte používateľskú a systémovú príručku k samotnému produktu?

Odpoveď: 2

Myslím, že je kompletná, len som nenašiel, čo znamená synchronizácia.

Otázka: Ako sa spolupracovalo pri testovaní s tímom Agenty 007?

Odpoveď: 1

Po oznámení nahlásených chýb sa chyby opravili veľmi rýchlo a na každú otázku týkajúcu sa editora som dostal jasnú odpoveď.

Iné poznámky:

Nevedel som zistiť, ako vytvoriť log file, ktorý by akceptoval váš editor.

Pri vymazaní pohybu vo vizuálnom editore sa automaticky neaktualizuje konečný stav, v akom bude robot po skončení sekvencie pohybov(stačilo by pohnúť kĺb o opačný vymazaný pohyb, a pri prehrávaní pohybu by sa nezmenilo nič).

Divné otáčanie hráča v priestore vizuálneho editora, otáča sa okolo päty.

Nie je možnosť parametrizovania agenta priamo z editora (vo verzií 0.7 už je).

Program s prázdny súborom pohybov padne(vo verzií 0.7 už nepadne).

V Bratislave dňa: 11.5.2009

Podpis testera:

Príloha D – Elektronické médium

Na priloženom médiu sa nachádza produkt a dokumentácia k nemu. Zároveň sa tam nachádzajú aj iné materiály, ktoré sme pri tvorbe produktu vytvorili. DVD médium má nasledujúcu štruktúru:

\

Dokumenty (obsahuje dokumentáciu, riadenie, príručky k projektu)

Dokumenty\IIT SRC (obsahuje Abstract, Extended Abstract a poster k IIT SRC)

Dokumenty\Iné materiály (obsahuje doplnkové materiály)

Dokumenty\ISI TP CUP (obsahuje prihlášku a priebežnú správu pre súťaž ISI TP CUP)

Dokumenty\Návody (obsahuje návody k nastaveniu niektorých funkcií)

Dokumenty\Ponuka tímu (obsahuje ponuku tímu a prezentáciu tímu)

Dokumenty\Poster a logo tímu (obsahuje poster a logo tímu)

Dokumenty\Posudky (obsahuje dokumentácie Tímu č.1 a naše posudky na Tím č.1)

Dokumenty\Testovanie (obsahuje dotazník pre externé testovanie a interné testovanie)

Dokumenty\Zápisnice (obsahuje zápisnice tímu)

Produkt (obsahuje výsledný produkt)

Sources\Agent (obsahuje zdrojové kódy hráča)

Sources\Builds (obsahuje vytvorené verzie produktu počas semestra)

Sources\Library (obsahuje knižnicu pohybov a správ zo servera)

Sources\RobotBehaviourEditor (obsahuje zdrojové kódy Editora správania)

Sources\Server (obsahuje zdrojové kódy servera Simspark)

Web stránka (obsahuje webovú stránku tímu)