Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 3, 812 19 Bratislava

RoboCup 3D

Tímový projekt

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

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

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

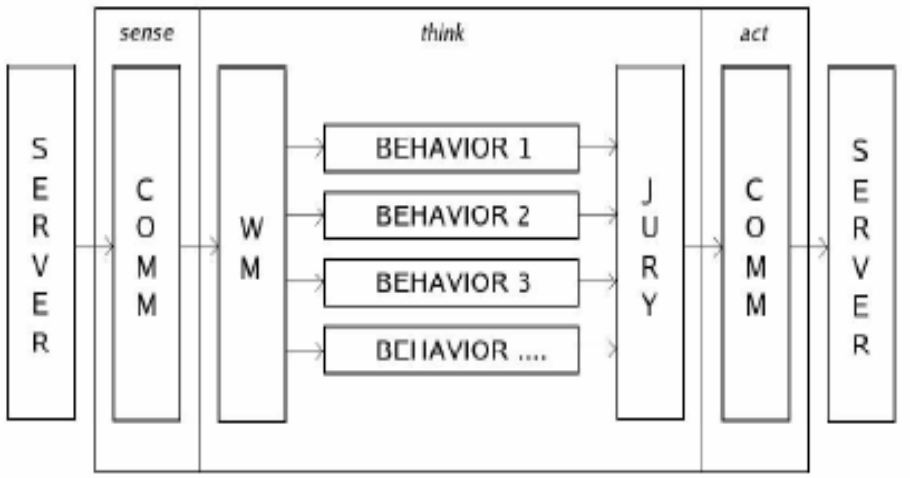
Nakoľko momentálne tím Little Green Bats nemá zverejnený detailný popis architektúry nového hráča, ale iba určité časti, tak v analýze sú vybrané niektoré dôležité myšlienky, s ktorými by sa dalo inšpirovať a umožnili by iný pohľad pri návrhu architektúry hráča.

O tíme

Tento holandský tím je relatívne nováčik v 3D simulovanej robotickej lige. Tím pozostáva zo siedmych 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 správaní
- Vytvorenie konfiguračného súboru a uloženie správaní 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 subsprávaní
- 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 <-1,1> :
    - 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 správaní [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 správaní, 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 vrstvovú 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

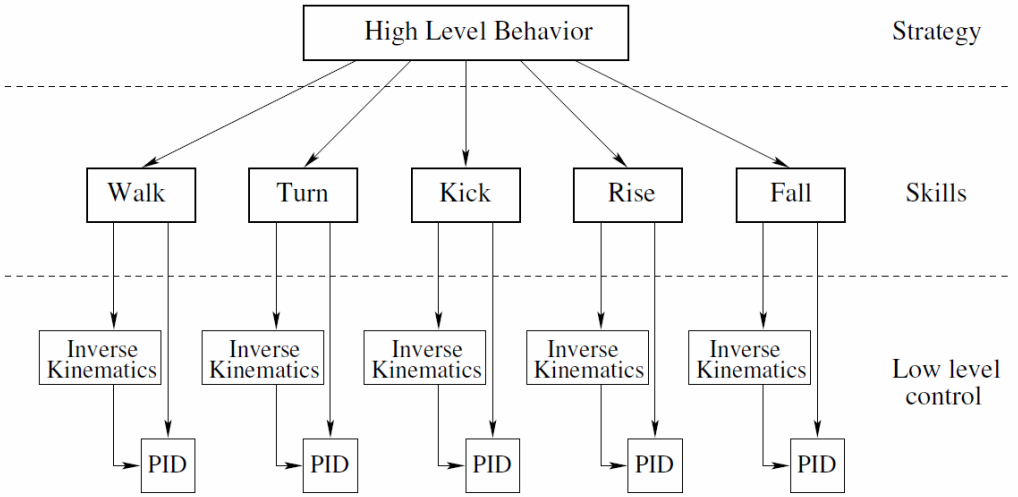
Jedná sa o relatívne mladý tím, ktorý vznikol v roku 2007. Aj z tohto dôvodu tím nedosiahol zatiaľ výraznejší úspech. Tím sa zúčastnil súťaže RoboCup 2007 v Atlante, U.S.A. v júli 2007. Jednalo sa skôr o začiatok simulovaného 3D robotického futbalu na univerzite, kde tím predstavil svoje riešenie a nápady. 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.

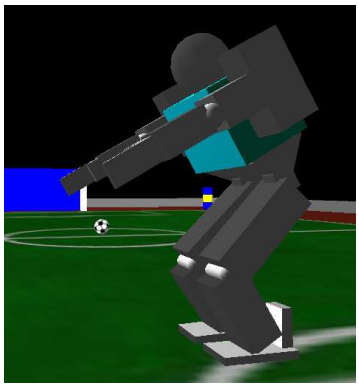
Architektúra ich agenta je zakreslená na obrázku (Obr. 2). Na najnižšej úrovni je agent kontrolovaný pomocou krútiacich momentov (uhľových rýchlostí otáčania) jednotlivých

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 schopnosti. 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 dosať 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 zdokonaliť 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 začiatočných 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.

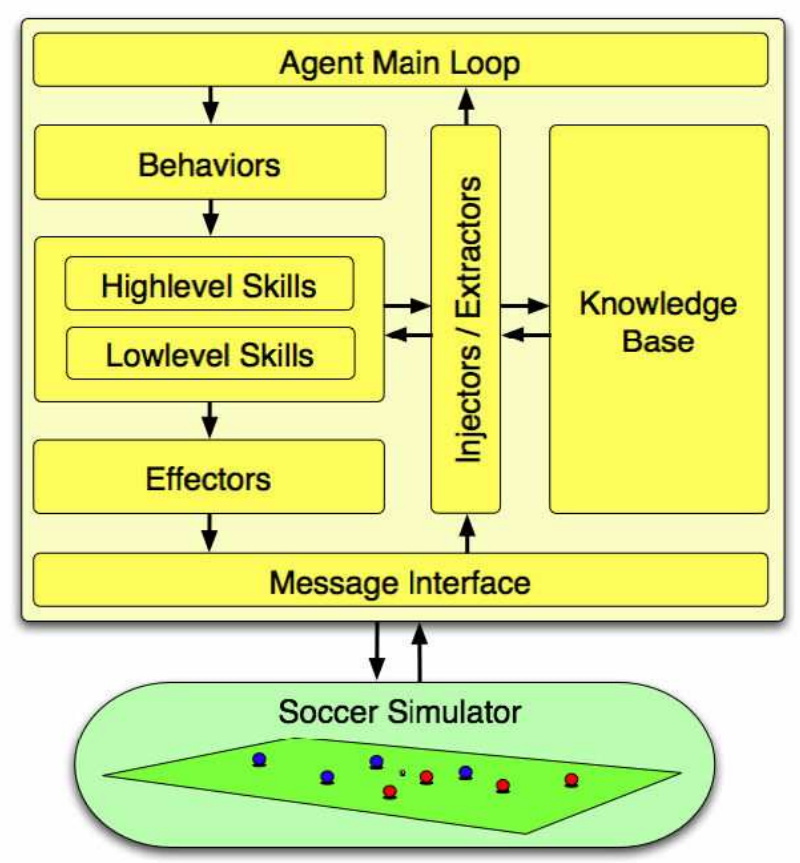
2.1.4. Tím Virtual Werder 3D

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 štvrťfinále RoboCup 2007 v Atlante [9].

Architektúra hráča

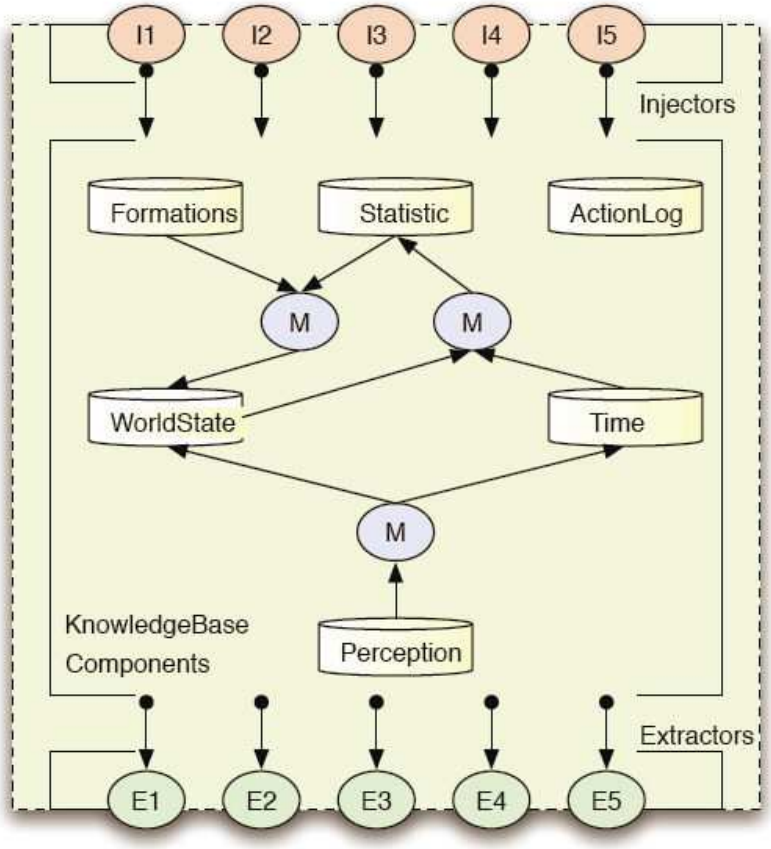
Architektúra pozostáva z dvoch častí. Jedna časť pozostáva z modulov potrebných na vnemy, usudzovanie a akcie, konkrétne agentov hlavný cyklus, jeho správanie, zručnosti, ďalej efektory a komunikačné rozhranie. Tieto časti sú usporiadané do úrovní, kde každý element na jednej úrovni môže používať len 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 časti, 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].

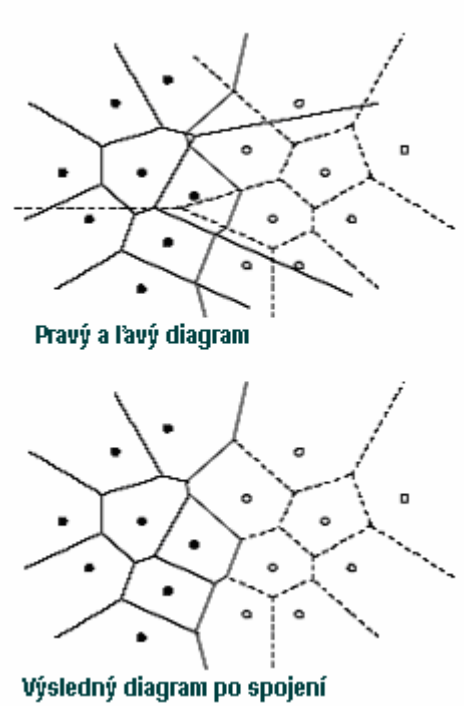
Cover sa používa na „pokrytie“ protihráča, t.j. hráč sa snaží zostať blízko protihráča a pripravovať sa na zachytenie lopty a bráni protihráčovi v streľbe na bránu. *Score* vyberá najsľubnejšiu pozíciu v súperovej bránke, za účelom strelenia gólu. *Intercept* vypočítavá najlepšiu pozíciu na zachytenie 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* (čiara 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 Voronoiov diagram

ľubovoľným algoritmom - *fortune algoritmus* (odstránenie predvídania, parabolické hrany) a *beach line algoritmus.*

Použitie Voronoiových diagramov

Tím Virtual Werder 3D používa Voronoiove bunky pre repozičný 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 tým by sa stávali nepoužiteľnými. Nepoužívajú celé hracie pole, ale menšie trojuholníkové polia, ktoré sú časťou celého poľa. To, aké sú trojuholníky veľké, záleží 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 (vďaka 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) zaznamenanej 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.

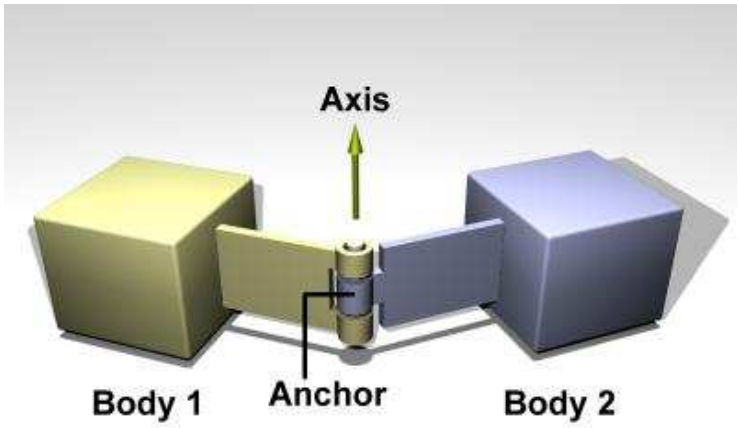
Formát správy pre tento perceptor vyzerá následovne:

(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]

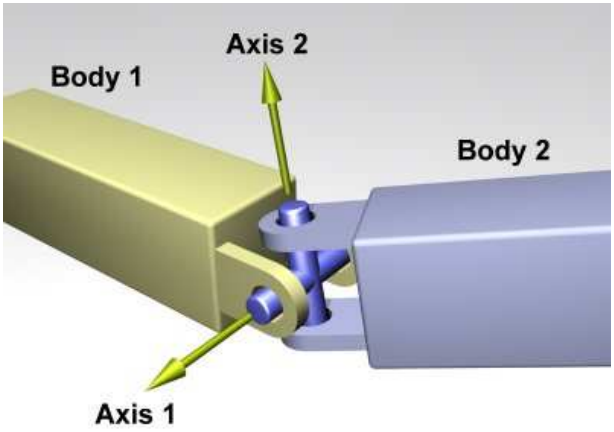
Formát správy pre tento perceptor vyzerá následovne:

(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 následovne:

(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 následovne:

(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 následovne:

(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 následovne:

(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 následovne:

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

Hear Perceptor

Hear perceptor ako už sám názov napovedá slúži na komunikáciu medzi hráčmi. Táto komunikácia však neprebieha priamo, ale len cez server. Hráč taktiež nemôže počuť všetko, ale len do vzdialenosti, ktorú určuje server.

Formát správy pre tento perceptor vyzerá následovne:

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

2.2.6. Efektory

Efektory sa požívajú 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 určeniu základného chovaniu, 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 následovne: (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 následovne:

(<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 následovne:

(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 inicializácií.

Formát správy pre tento efektor vyzerá následovne:

(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 následovne:

(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.

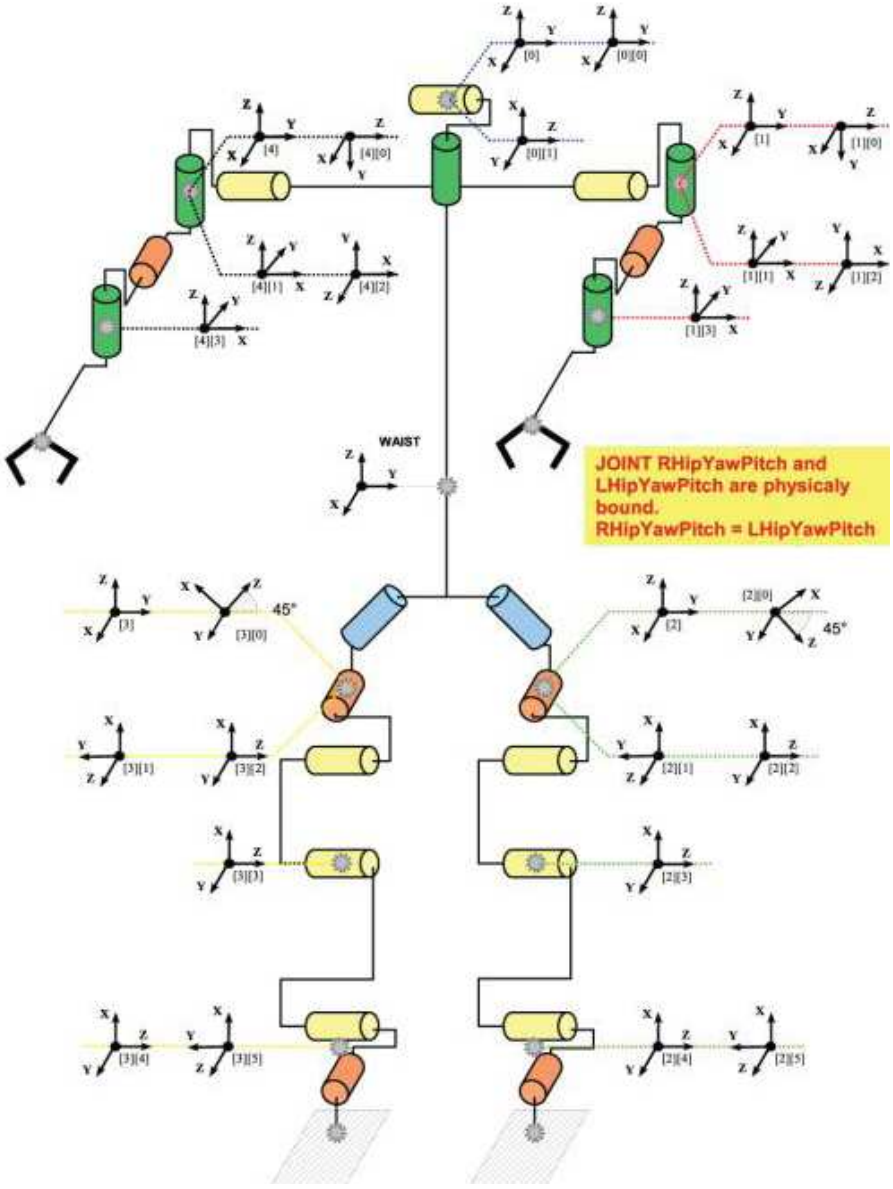
2.2.8. Simulácia

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 docieliť 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 sieti 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 trénovaciu 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 byt v nich optimálne zvolene 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 znalosti 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 byt 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 sieti 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 byt 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 sieti, 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 hráčových kĺbov. Ďalším vhodným nápadom sa nám pozdáva myšlienka, že hráč má pri vykonávanej činnosti určitú najstabilnejšiu polohu. Napríklad na starom modely 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 modely 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 situácií nespadol 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 situácií, 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 loptu. Toto bude riešené v prípade, že hráš 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 diskrétny 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 inicializácií 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 verzií 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:

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

Vstupy:

Xos,Yos, Zos – súradnice bodu, cez ktorý prechádzajú osi otáčania

Xp, Yp, Zp – počiatočné súradnice bodu

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

Výstup:

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

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 budúcna je naším cieľom spojiť funkcionalitu editora pohybov a rovnovážneho modulu tak, aby nespôsobovali zmätočný pohyb robota, ale pomáhali mu v stabilizácií navrhnutých pohybov.

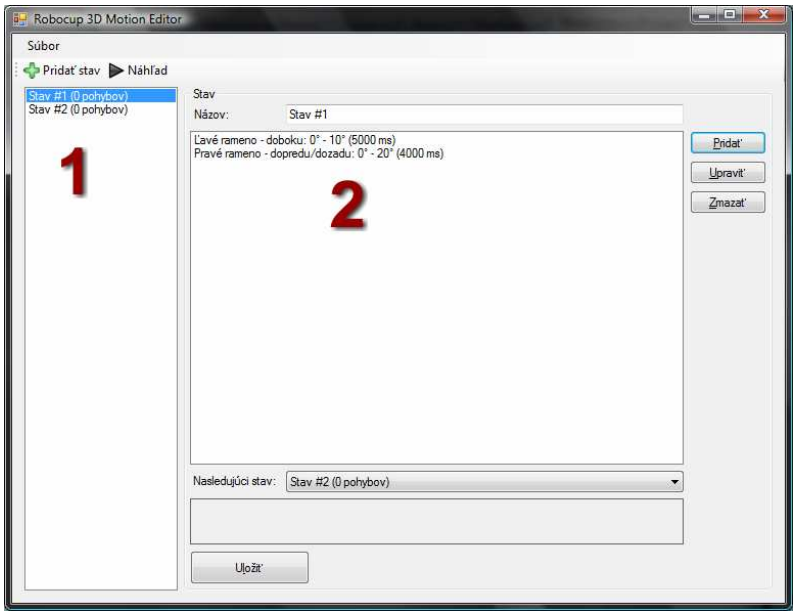
4.3. Editor pohybov robota

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.

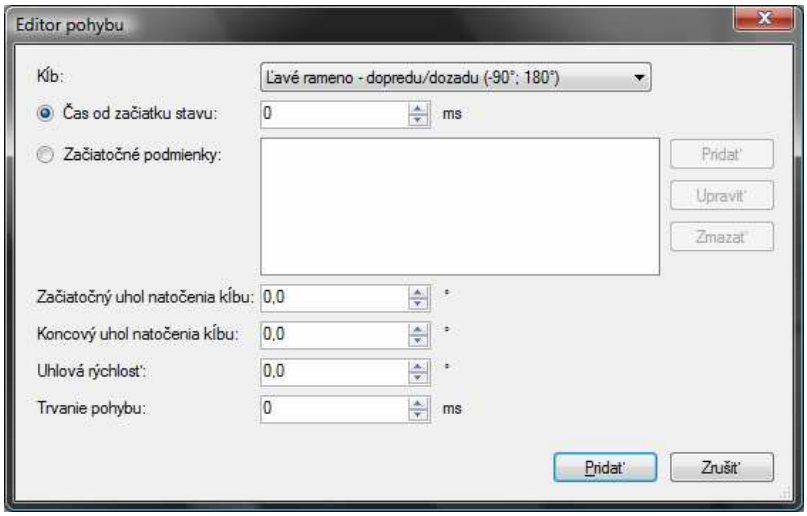
Každý stav má svoj názov, zoznam čiastkových pohybov (Obr. 10– bod 2) a definovaný nasledujúci stav.



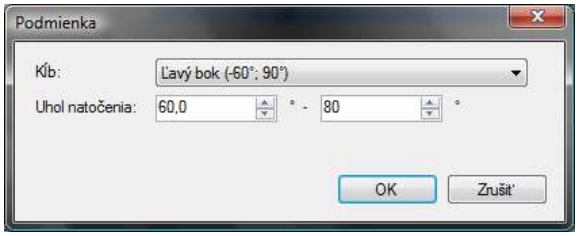
Obr. 10 - Pôvodné hlavné okno editora pohybov

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

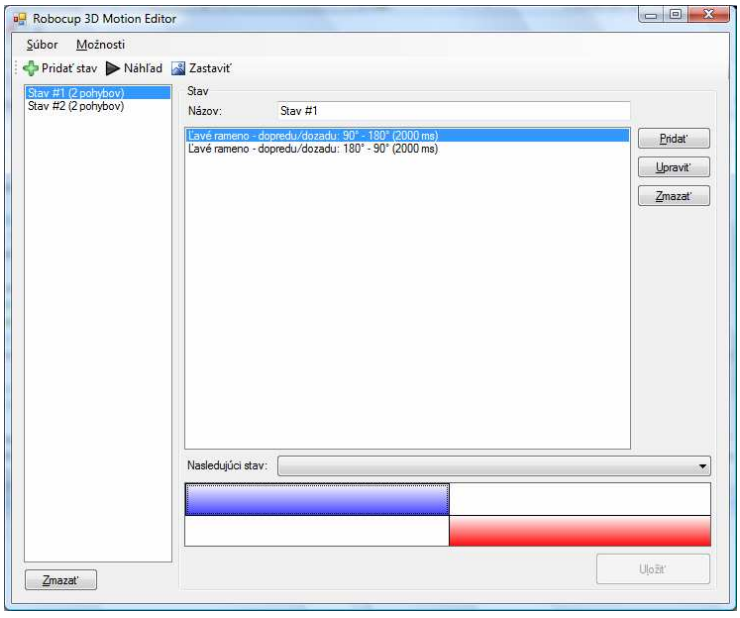
Pohyb si môžeme po vytvorení ihneď prezrieť. Kliknutím na tlačidlo Náhľad sa vygeneruje súbor s pohybom, ktorý sa zašle agentovi, ktorý je pripojený k serveru. Agent súbor s pohybom načíta a inštrukcie zadané v ňom vykoná.

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

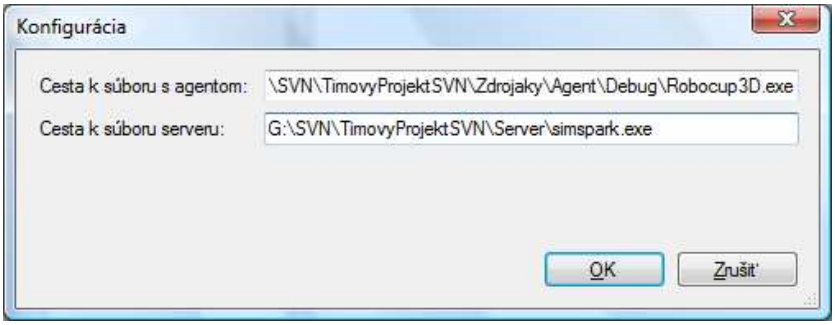
Prostredie programu (Obr. 13) je prehľadne rozčlenené na menu, hlavný panel (na ktorom sa nachádzajú aj tlačidla Náhľad a Zastaviť), panel stavov a časovú os, na ktorej je znázornené vykonávanie jednotlivých pohybov.



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.



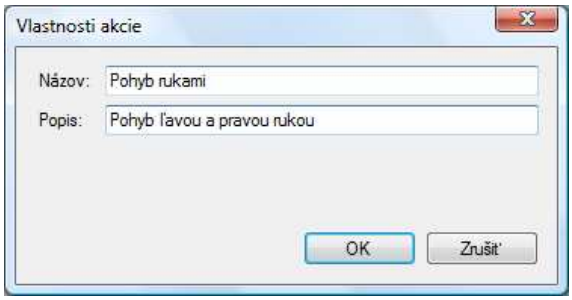
Obr. 14 - Konfigurácia editora

4.4.3. Vytváranie akcií

Akcia predstavuje súhrn viacerých za sebou nasledujúcich stavov. Stavy 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). Stavu pridelíme 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 vytváranej akcie

4.4.4. Náhľad vytvorenej akcie

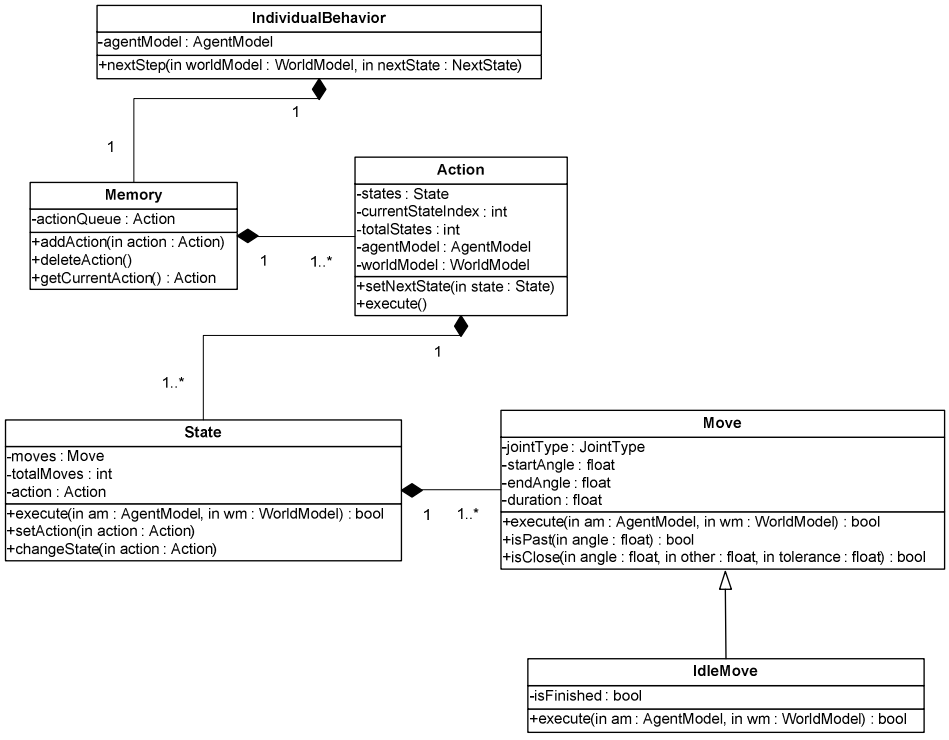
Aplikácia vie zabezpečiť predvedenie zadefinovanej akcie robota. Po kliknutí na tlačidlo Náhľad sa vykonaná niekoľko krokov potrebných na zobrazenie pohybu robota, ktorý sme navrhli prostredníctvo editora:

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 začiatočný 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 hierarchií 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 začiatočné 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
jointType	JointType	Kĺb, ktorý je potrebné natočiť
startAngle	float	Uhol, v ktorom má byť kĺb natočený pri začiatku pohybu
endAngle	float	Konečný uloh, do ktorého chceme kĺb dostať
duration	float	Počet cyklov, ktorý má pohyb trvať
startTime	float	Čas od začiatku vykonávania stavu, kedy sa má začať vykonávať pohyb

Metódy:

Názov metódy	Návratový Typ	Popis
execute	bool	Vykoná pohyb, resp. pokračuje v jeho vykonávaní
isPast	bool	Vráti true, ak je kĺb za konečným natočením
isClose	bool	Vráti true, 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
isFinished	bool	True, ak je kĺb natočený do koncovej pozície

Metódy:

Názov metódy	Návratový Typ	Popis
execute	bool	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 músí byť rozdielna pre každý prepínač.

5 Zhodnotenie

Naša doterajšia práca počas tohto semestra 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 a zistili sme, že najlepšie asi bude pri chôdzi úplne presunúť váhu na opornú nohu. 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.

Naše stretnutia mimo oficiálnych stretnutí boli typu brainstorming, kde každý predložil svoj nápad, čo si doma naštudoval a pripravil a z týchto nápadov vzišli aj rôzne ďalšie zaujímavé nápady. Niektoré sa po krátkej diskusii zavrhli, ale na niektorých stále pracujeme a zdokonaľujeme ich. Pre názornosť uvediem jeden z kategórie dobrých nápadov a to je vytvorenie programu, ktorý by generoval súbor s inštrukciami (pre pohyb hráča) na základe našich vstupov, ktorý by si hráč vedel načítať. Táto metóda tvorby pohybov je výrazne rýchlejšia, ako pracné programovanie. Taktiež sa dajú rýchlo editovať.

Naša práca pokračuje aj na programátorskej úrovni, priamo v zdrojových kódoch hráča, kde sme si vyskúšali a naprogramovali základné pohyby hráča, ako je mávanie rukou a chôdza. Toto nám pomohlo k lepšiemu pochopeniu fungovania ako hráča, tak aj fyziky servera.

6 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 Brement, 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 **AC**tion).

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 následovne:

- **[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** – začiatočné 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