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

Robocup 3D

(Tímový projekt)

Dokumentácia k projektu

Odbor: Softvérové inžinierstvo, Informačné systémy
Vedúci tímu: Ing. Ivan Kapustík
Autori: Bc. Rastislav Barlík
Bc. Marian Buchta
Bc. Štefan Dlugolinský
Bc. Michal Kvetan
Bc. Stanislava Leitmanová
Bc. Milan Šillík
Dátum: 20.Máj 2008

Copyright© 2008 Hviezdna jedenástka.

Obsah

1. ÚVOD 1
   - 1.1 Cieľ 1
   - 1.2 Zadanie projektu 1
   - 1.3 Prehľad dokumentu 1
   - 1.4 Referencie 2
   - 1.5 Slovník pojmov 3
2. ANALÝZA 4
   - 2.1 Úvod do simulovaného robotického futbalu 4
   - 2.2 Analýza serveru Robocup Soccer Server 3D 4
     - 2.2.1 3D Server s hráčmi typu “Sphere“ 5
       - 2.2.1.1 Prehľad systému 5
       - 2.2.1.2 Simulácia futbalu 5
       - 2.2.1.3 Perceptory 6
       - 2.2.1.4 Efektory 8
     - 2.2.2 3D Server s hráčmi typu “Humanoid“ 9
       - 2.2.2.1 Štruktúra inštalačného balíčka 9
       - 2.2.2.2 Spustenie simulácie 10
       - 2.2.2.3 Perceptory 11
       - 2.2.2.4 Efektory 13
   - 2.3 Analýza existujúcich riešení 14
     - 2.3.1 Riešenia pre server s hráčmi typu “Sphere“ 14
       - 2.3.1.1 Tím AUTSAJDRY-TNG 14
       - 2.3.1.2 Tím 6th Sense 16
       - 2.3.1.3 Tím Virtual Werder 3D 18
       - 2.3.1.4 Tím ZJUBase 23
       - 2.3.1.5 Tím Mainz Rolling Brains 25
       - 2.3.1.6 FC Portugal 30
     - 2.3.2 Riešenia pre server s hráčmi typu “Humanoid“ 33
       - 2.3.2.1 Tím NimbRo 33
       - 2.3.2.2 Tím Zigorat 35
       - 2.3.2.3 Tím Little Green Bats 41
   - 2.4 Zhodnotenie a výber vhodného typu servera 43
3. ŠPECIFIKÁCIA POŽIADAVIEK 44
   - 3.1 Požiadavky na komunikáciu 44
   - 3.2 Požiadavky na prácu s dátami 44
   - 3.3 Požiadavky na správanie 45
   - 3.4 Požiadavky na zručnosti 45
   - 3.5 Požiadavky na architektúru 45
   - 3.6 Požiadavky na dokumentáciu 46
4. NÁVRH 47
   - 4.1 Hrubý návrh architektúry 47
   - 4.2 Spresnenie hrubého návrhu architektúry hráča 49
5. PROTOTYP 51
   - 5.1 Ciele prototypovania 51
   - 5.2 Implementácia prototypu 51
     - 5.2.1 Výber vývojového prostredia a štruktúra projektu 51
     - 5.2.2 Architektúra hráča 52
     - 5.2.3 Implementácia vlákien 55
     - 5.2.4 Komunikácia so serverom 59
     - 5.2.5 Parser 61
     - 5.2.6 Ovládanie kĺbov, implementácia vstávania 61

© Hviezdna jedenástka, 2008

5.3 Zhodnotenie prototypu 65

6. IMPLEMENTÁCIA 66

6.1 Architektúra hráča 66
6.2 Implementácia modulu komunikácie 67
6.3 Parser správ prichádzajúcich zo servera 71
6.4 Logovanie do súboru 73
6.5 Model hráča 73
6.6 Model sveta hráča 77
6.7 Ovládanie kĺbov hráča 80
6.8 Zručnosti agenta 85
6.9 Učenie sa pohybov pomocou prediktívneho riadenia 88

7. TESTOVANIE 93

8. ZHODNOTENIE 94

Tabuľky

Tab. 1: Kĺbové perceptory 12
Tab. 2: Kĺbové efektory 14
Tab. 3: Prehľad schopností hráča tímu 6th sense 16
Tab. 4: Porovnanie fork() a pthread_create() 56
Tab. 5: Metódy triedy Socke 68
Tab. 6: – Atribúty triedy TCPSocket 69
Tab. 7: – Metódy triedy TCPSocket 69
Tab. 8: – Atribúty triedy UDPSocket 70
Tab. 9: – Metódy triedy UDPSocket 71
Tab. 10: Kĺby v modeli soccerbot056 76
Tab. 11: Trieda PlayerModel 76
Tab. 12: Trieda ForceResistance 76
Tab. 13: Atribúty triedy Joint 80
Tab. 14: Metódy triedy Joint 81
Tab. 15: Parametre metódy moveTo pre kĺb typu Hinge 82
Tab. 16: Atribúty triedy HingeJoint 82
Tab. 17: Makrá na programovanie fázového pohybu 85

Obrázky

Obr. 1: Model potenciálových polí 17
Obr. 2: Architektúra hráča tímu Virtual Werder 3D 19
Obr. 3: Schéma štruktúry bázy znalostí hráča tímu Virtual Werder 3D 20
Obr. 4: Divide and conquer metóda 21
Obr. 5: Zobrazenie Voronoiových buniek v hracom poli 22
Obr. 6: Výsledok evaluačnej funkcie 24
Obr. 7: Architektúra hráča tímu Mainz Rolling Brains 26
Obr. 8: Rozvrhnutie p(ζ) 27
Obr. 9: Jednotky množiny A 28
Obr. 10: Voľba vstupného signálu ζ 28
Obr. 11: Usporiadanie referenčných vektorov 28
Obr. 12: Ukážka priebehu procesu Neural Gas 29
Obr. 13: Rušenie prihrávok tímu CMU United 32
Obr. 14: Tri druhy pohybu 33
Obr. 15: Padanie s vypnutým stabilizačným módom 34
Obr. 16: Padanie so zapnutým stabilizačným módom 34
Obr. 17: Mechanizmus vstávania z ľahu na bruchu 35
Obr. 18: Mechanizmus vstávania z ľahu na chrbte 35
Obr. 19 Hierarchia tried v súbore Object.h 37
Obr. 20 Diagram tried 40
Obr. 21 Architektúra hráča tímu Zigorat 41
Obr. 22 Architektúra hráča tímu Little Green Bats 42
Obr. 23 Hrubý návrh architektúry hráča 47
Obr. 24 Spresnenie hrubého návrhu architektúra hráča. 49
Obr. 25 Architektúra hráča. 53
Obr. 26: Synchronizácia vlákien typu Joinable 57
Obr. 27: Pohyb hráča rukami 57
Obr. 28: Diagram tried prototypu na testovanie komunikácie vo vlákne 58
Obr. 29: Schéma TCP komunikácie medzi klientom a serverom 59
Obr. 30: Schéma UDP komunikácie medzi klientom a serverom (verzia 1). 60
Obr. 31: Schéma UDP komunikácie medzi klientom a serverom (verzia 2). 61
Obr. 32: Prvá fáza vstávania hráča z ľahu na bruchu 62
Obr. 33: Druhá fáza vstávania hráča z ľahu na bruchu 62
Obr. 34: Piata fáza vstávania hráča z ľahu na bruchu 63
Obr. 35: Siedma fáza vstávania hráča z ľahu na bruchu 63
Obr. 36: Posledná – ôsma fáza vstávania hráča z ľahu na bruchu 64
Obr. 37: Výsledná architektúra hráča 66
Obr. 38: Diagram tried pre modul komunikácie 68
Obr. 39: Schéma TCP komunikácie 70

© Hviezdna jedenástka, 2008

Obr. 40: Schéma UDP komunikácie *71*
Obr. 41: Diagram tried modelu hráča *74*
Obr. 42: Model hráča soccerbot056 .... *75*
Obr. 43: Hierarchia základných typov objektov v modeli sveta *78*
Obr. 44: Sférický súradnicový systém *79*
Obr. 45: Karteziánsky súradnicový systém *80*
Obr. 46: Kĺb typu Hinge *81*
Obr. 47: Kĺb typu Universal *83*
Obr. 48: Dva typy pohybu kĺbov *83*
Obr. 49: Hierarchia zručností agenta *86*
Obr. 50: Správanie sa agenta *87*
Obr. 51: Zosnímaný pohyb robota *88*
Obr. 52 Zosnímaný pohyb transformovaný na výchylky *88*
Obr. 53 Vygenerovaný vzor z výchyliek *89*
Obr. 54 Prediktor na vybratie najvhodnejšej akcie *89*
Obr. 55 Model prediktívneho riadenia *90*
Obr. 56 Postup pri učení pomocou prediktívneho riadenia *90*
Obr. 57 Optimalizovaný pohyb robota *91*
Obr. 58 Stabilizovanie gyroskopu po optimalizácii pohybu *91*

© Hviezdna jedenástka, 2008

1. Úvod

1.1 Cieľ

Predkladaný dokument tvorí projektovú dokumentáciu, ktorá je súčasťou riešeného projektu na predmete Tvorba softvérového/informačného systému v tíme. Tento projekt má názov Robocup 3D a venuje sa simulovanému robotickému futbalu.

Cieľom dokumentu je priblížiť priebeh riešenia daného projektu a podrobne opísať jeho jednotlivé etapy.

1.2 Zadanie projektu

Téme RoboCup, presnejšie lige simulovaného robotického futbalu sa naši študenti venujú už osem rokov. Tímy študentov, či už v rámci umelej inteligencie alebo tímového projektu, sa snažia vytvárať a vylepšovať programy, ktoré simulujú správanie sa futbalového hráča. Každý tím sa v rámci obmedzení, určených pravidlami hry futbal a špecifikami simulačného prostredia, snaží vytvoriť čo najlepšieho hráča. Mužstvo, vytvorené z takýchto hráčov, by malo vyhrať nad mužstvom súpera. O súťaži a doterajšej činnosti je dosť popísané aj na stránke STU turnaj v simulovanom robotickom futbale (www.fiit.stuba.sk/robocup).

Simulácia futbalu pôvodne prebiehala iba v dvoch rozmeroch. Pre zvýšenie reálnosti simulácie bolo vytvorené 3D simulačné prostredie, ktoré rozširuje možnosti hry. 3D simulačné prostredie sa pomerne výrazne líši od doposiaľ používaného 2D prostredia, a to jednak spôsobom simulácie, ale hlavne možnosťami ktoré poskytuje hráčom.

Hlavným cieľom projektu bude vytvoriť hráča pre 3D simuláciu, ktorý dokáže plnohodnotne využívať možnosti poskytované simulačným prostredím. Úlohou teda bude prevziať jednoduchého hráča vytvoreného na našej fakulte v minulom roku a doplniť do neho komplexnejšie typy správania a rozhodovania. Keďže sa predpokladá ďalšie rozširovanie hráča v ďalších rokoch, dôležitými požiadavkami sú prehľadnosť a ďalšia rozširovateľnosť, a to na úrovni návrhu aj implementácie. Dôraz pri vytváraní hráča by mal byť kladený najmä na dobre prepracované a odladené nižšie schopnosti hráča, ktoré umožnia hráčovi spracovávať vnemy z prostredia a efektívne konať v prostredí (pohybovať sa, pracovať s loptou). Pri návrhu a implementácii bude tiež možné (a žiadané) čerpať z veľkého množstva prístupov existujúcich v 2D simulácii.

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

1.3 Prehľad dokumentu

Dokument je rozdelený na niekoľko kapitol, ktoré zodpovedajú jednotlivým etapám riešenia projektu.

Kapitola 1 obsahuje úvod. Približujeme v nej, čo je cieľom tohto dokumentu a zadanie riešeného projektu. Taktiež obsahuje referencie a slovník pojmov.

© Hviezdna jedenástka, 2008

V kapitole 2 sa venujeme analýze problémovej oblasti. Analyzujeme v nej dve verzie serveru Soccer Server. Ďalej sa venujeme analýze prístupov a riešení existujúcich tímov, z toho dvoch slovenských a zvyšných svetových.

Kapitola 3 predstavuje špecifikáciu požiadaviek. Identifikujeme a opisujeme v nej požiadavky na vytvárané riešenie.

V kapitole 4 je uvedený hrubý návrh, ktorý zahŕňa návrh architektúry systému a stručný opis jej modulov.

Kapitola 5 obsahuje opis prototypu vytvoreného v zimnom semestri. Sú tu uvedené ciele prototypu, opis implementovaných častí a takisto zhrnutie dosiahnutých výsledkov.

V kapitole 6 je opísaná implementácia hráča a jeho jednotlivých modulov.

Kapitola 7 sa venuje testovaniu hráča.

V kapitole 8 je uvedené zhrnutie dosiahnutých výsledkov.

1.4 Referencie

[1] Oficiálna stránka Robocup (www).
http://www.robocup.org

[2] Robocup Soccer Server User Manual
http://sserver.sourceforge.net/docs/manual.pdf

[3] AUTSAJDRY-TNG Homepage (www).
http://www2.dcs.elf.stuba.sk/TeamProject/2006/team02/public_html/

[4] 6th sense Homepage (www).
http://www2.dcs.elf.stuba.sk/TeamProject/2006/team06/

[5] Virtual Werder 3D Homepage (www).
http://www.virtualwerder.de/

[6] Virtual Werder 3D Team Documentation. University of Bremen: Aug. 30, 2006 [cit. 2007-10-16]. Dostupné na internete:
http://anstoss.informatik.uni-bremen.de/files/doc/vw3d_docu_2006_08.pdf

[7] DURKOT, J. Virtuálna simulácia interakcie ligand – proteín.Univerzita Pavla Jozefa Šafárika v Košiciach: Prírodovedecká fakulta - ústav informatiky: Apr. 29,2005 [cit. 2007-10-16]. Dostupné na internete:
http://s.ics.upjs.sk/~dodo27/diplomka/teoria/dp2005.pdf

[8] ZJUBase Simulation Homepage (www).
http://www.nlict.zju.edu.cn/nlictrobocup/Robocup.htm

[9] ZJUBase 3D. Team Description 2006 [PDF Online]. A Robot Soccer Team of RoboCup 3D simulation: 2006, [cit. 2007-10-14]. Dostupné na internete:
http://www.nlict.zju.edu.cn/nlictrobocup/

[10] Mainz Rolling Brains Homepage (www).
http://www.informatik.uni-mainz.de/RollingBrains/

[11] Mainz Rolling Brains. Team Description Mainz Rolling Brains [PDF Online]. Johannes Gutenberg-University Mainz: 2005, [cit. 2007-10-15]. Dostupné na internete:
http://elara.tk.informatik.tu-darmstadt.de/publications/2005/Flentge05Team.pdf

[12] Fritzke B.: Some Competitive Learning Methods. Institute for Neural Computation,

Ruhr-Universität Bochum: 1997, [cit. 2007-10-18]. Dostupné na internete:
http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/JavaPaper/t.html

[13] FC Portugal Homepage (www).
http://www.ieeta.pt/robocup/index.htm

[14] The CMUnited-97 Robotic Soccer Team: Perception and Multiagent Control. Carnegie Mellon University, Pittsburgh, 1997, [cit. 2007-10-18]. Dostupné na internete:
http://www.cs.cmu.edu/afs/cs/usr/pstone/public/papers/97robot-paper/robot-paper.html

[15] NimbRo Homepage (www).
http://www.nimbro.net/index.html

[16] Zigorat Homepage (www).
http://zigorat3d.googlepages.com/

[17] Little Green Bats Homepage (www).
http://www.littlegreenbats.nl/

[18] ABNF Parser Generator Homepage (www).
http://www.coasttocoastresearch.com/Downloads.htm

[19] IEEE Std 1003.1,2004 Edition
http://www.unix.org/version3/ieee_std.html

[20] Ross Johnson: Pthreads Win32
http://sourceware.org/pthreads-win32/

[21] Blaise Barney: POSIX Threads Programming
https://computing.llnl.gov/tutorials/pthreads/

[22] Chalodhorn, R: POSIX Learning Dynamic Humanoid Motion using Predictive Control in Low Dimensional Subspaces. Dostupné na internete
http://share.allamehelli.ir/Robocup3D/Documents/Humanoid2005.pdf

1.5 Slovník pojmov

RoboCup Turnaj v simulovanom robotickom futbale

Humanoid Objekt, ktorého štruktúra je podobná štruktúre ľudského tela

Soccer Server Systém, ktorý riadi priebeh hry, spúšťa jednotlivých agentov a simuluje pohyby hráčov a lopty

Perceptor Pomocou neho hráč dostáva vnemy z okolitého sveta

Efektor Slúži na vykonanie určitej akcie.

© Hviezdna jedenástka, 2008

2. Analýza

Táto kapitola sa venuje úvodu do robotického futbalu, analýze serveru Soccer Server pre Robocup 3D a analýze riešení niektorých slovenských aj zahraničných tímov.

2.1 Úvod do simulovaného robotického futbalu

RoboCup je medzinárodný projekt, ktorého zámerom je propagovať umelú inteligenciu, robotiku a príbuzné oblasti [1]. 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

V našom projekte sa venujeme simulovanej lige.

2.2 Analýza serveru Robocup Soccer Server 3D

Soccer Server je systém, ktorý umožňuje agentom, pozostávajúcim z programov napísaných v rôznych programovacích jazykoch, hrať proti sebe zápas [2]. Zápas sa uskutočňuje v štýle klient-server, kde server simuluje všetky pohyby hráčov a lopty a každý klient ovláda pohyby a správanie práve jedného hráča - agenta.

Server sa stará takpovediac o platnosť futbalových pravidiel. V rýchlom slede posiela agentom informácie o tom, čo vidia a čo počujú. Zabezpečuje pohyb lopty a kontroluje jednotlivé futbalové situácie. Jednotlivým agentom, ktorí sa pripájajú priamo na tento server, však poskytuje informácie o tom, čo vidia, trochu skreslene vnášaním šumu do posielaných dát.

Na základe toho, ako je vytvorený model sveta a aký je použitý model hráča, môžeme servery rozdeliť na nasledujúce typy:

- 2D server – v tomto type servera je svet reprezentovaný dvojrozmerným modelom. Hráči sú reprezentovaní ako gule a celá simulácia prebieha v dvojrozmernom priestore. Na našej fakulte sa tímy už niekoľko rokov venujú vývoju vlastností hráčov práve pre tento typ servera a je vyvinuté veľké množstvo rôznych vyšších schopností hráčov.
- 3D server s hráčmi typu „Sphere“ – tento typ servera uvažuje s modelom sveta v trojrozmernom priestore. Hráči sú implementovaní ako gule (Sphere) s daným polomerom a hmotnosťou. Vývoju hráča pre tento typ servera sa na našej fakulte venovali doposiaľ tri tímy a vytvorili iba základné schopnosti hráčov.
- 3D server s hráčmi typu „Humanoid“ – tento typ servera je relatívne nový, konkrétne najnovšia verzia 0.5.6 vznikla v júni tohto roku. Na našej fakulte zatiaľ neexistuje tím, ktorý by sa venoval vývoju hráča pre tento typ servera. Aj čo sa týka svetových tímov, je ich veľmi málo v porovnaní s tými, ktoré sa venujú hráčom typu „Sphere“. Tento server má trojrozmerný model sveta a hráč je typu „Humanoid“. To znamená, že model hráča predstavuje zjednodušný model ľudského tela.

Súčasťou systému Soccer Server je monitor – Soccer Monitor, ktorého úlohou je vizualizácia priebehu zápasu.

© Hviezdna jedenástka, 2008

V rámci analýzy sme sa zaoberali dvoma verziami Soccer Servera – 3D server s hráčmi typu „Sphere" (verzia servera 0.5.2) a 3D server s hráčmi typu „Humanoid" (verzia servera 0.5.6).

2.2.1 3D Server s hráčmi typu "Sphere"

2.2.1.1 Prehľad systému

Na začiatok by sme sa mali oboznámiť s komponentmi systému. Simulácia futbalu pozostáva z troch dôležitých častí: zo soccer serveru, monitoru a agentov.

Soccer Server

Na prácu so soccer serverom je potrebné poznať knižnicu SPADES (System for Parallel Agent Discrete Event Simulation). Soccer server je zodpovedný za spustenie procesu agenta. Na rozdiel od 2D soccer servera, 3D soccer server nečaká na pripojenie agenta. Knižnica SPADES používa na konfiguráciu agentov databázu. Ide o súbor `agentdb.xml`, ktorý sa nachádza v adresári `./app/simulator/`.

Agenti sa pripájajú na Commserver SPADES cez UNIX rúry. Commserver SPADES vzápätí komunikuje so soccer serverom. V predvolenom nastavení soccer servera je spúšťaný integrovaný Commserver.

Je možné spustiť viacero Commserverov a distribuovať na ne procesy agentov. Viac o konfigurovaní Commservera je napísane v manuáli SPADES. Ak chceme použiť viacero Commserverov, musíme nakonfigurovať soccer server, aby počkal na pripojenie všetkých Commserverov predtým, než odštartuje simuláciu. Soccer server sa konfiguruje vo svojom spúšťacom skripte `rcssserver3D.rb` v adresári `./app/simulator/`.

Na štart integrovaného servera slúži parameter `Spades.RunIntegratedCommserver`, ktorý je predvolene nastavený na hodnotu `true`. Pri použití viacerých Commserverov sa parametrom `Spades.CommServersWanted` určí počet Commserverov, na ktoré bude soccer server čakať. Ak používame integrovaný Commserver, nastavíme tento parameter na 1.

Monitor

Predvolený monitor `rcssmonitor3D-lite` sa nachádza v adresári `./app/rcssmonitor3d/lite`. Tiež sa používa na prehrávanie záznamov, ktoré automaticky vytvára soccer server. Keď chceme prehrať záznam, použijeme prepínač `--logfile <názov súboru>`. Automaticky generovaný záznam sa ukladá do súboru `monitor.log` a nachádza sa v adresári `Logfiles/`.

Protokol monitora podporuje príkazy na implementáciu trénera. Automaticky sa tak dajú vytvoriť testovacie situácie na ihrisku a vykonať správanie agenta. Knižnicu monitora je možné použiť pri implementácii vlastného monitora a trénera. Knižnica sa nachádza v adresári `./app/rcssmonitor3d/lib/`.

Ako východiskový bod pri implementácii vlastného agenta poslúži program `agenttest` nachádzajúci sa v adresári `./app/agenttest/`. Tento agent má implementované jednoduché správanie behu a kopnutia.

2.2.1.2 Simulácia futbalu

Futbalový tím

Tím obsahuje určitý počet agentov s rovnakými schopnosťami. Programy, z ktorých bude pozostávať tím, si vymieňajú informácie virtuálnym nízkoúrovňovým ovládacím systémom,

© Hviezdna jedenástka, 2008

ktorý je zabudovaný v agentoch. Na programovanie perceptorov a efektorov agentov sa používajú s-výrazy (symbolické výrazy, tak ako ich poznáme napríklad z jazyka LISP).

Prostredie

Veľkosť ihriska má dĺžku <100.0, 110.9> m a šírku <64.0, 75.9> m. Veľkosť brány je na dĺžku 7.32 m, na výšku 1.6 m a na hĺbku 2.0 m. Všetky konštanty na nastavenie prostredia sa nachádzajú v štartovacom skripte servera `rcssserver3D.rb` a je možné ich meniť. Skript je umiestnený v adresári `./app/simulator/`.

Hráči

Hráči sú v tejto verzii reprezentovaní guľami. Priemer hráča je 0.44 m a váha 75 kg.

Pohyb hráča sa ovláda `Drive` efektorom, pomocou ktorého sa hráčovi určí zrýchlenie v ľubovoľnom smere. Je možné aj menšie podskočenie. `Drive` efektor je účinný iba ak sa hráč dotýka hracej plochy ihriska. Ak prestaneme hráča urýchľovať, bude sa zotrvačnosťou ešte chvíľu pohybovať. `Drive` efektor sa môže použiť aj na spomalenie hráča, kedy mu udáme zrýchlenie v opačnom smere. Maximálna rýchlosť hráča, ako aj maximálna výška podskočenia nie je známa a treba ju zistiť.

Po pripojení hráča na server je potrebné urobiť dve veci. Prvou je `Create` efektorom určiť typ modelu hráča, ale v tejto verzii serveru je hráč obmedzený len na model gule. Druhou vecou je určenie čísla hráča a zaradenie do tímu pomocou `Init` efektoru.

2.2.1.3 Perceptory

Vision perceptor

Hráči sú vybavení abstraktnými kamerami, pomocou ktorých ,,vnímajú“ svet. Zorné pole kamery je 360°. `Vision` perceptor sprostredkúva hráčovi zoznam videných objektov, medzi ktoré patria ostatní hráči, lopta a značky na ihrisku. Na ihrisku je spolu osem značiek, z toho štyri v rohoch ihriska a zvyšné tvoria žrde brán.

V zozname videných objektov nesie každá položka údaj o vzdialenosti od hráča, uhol v rovine x-y (uhol 0° vždy ukazuje na súperovu bránu) a uhol v rovine kolmej na rovinu x-y (uhol 0° určuje horizontálny smer). Všetky relatívne vzdialenosti a uhly sa počítajú od stredu hráča, kde je umiestnená jeho kamera.

Kamere je pridaná malá kalibračná chyba. Pre každú súradnicu je to odchýlka z rovnomerného rozdelenia od -0.005 m až 0.005 m. Odchýlka sa vyráta iba raz a počas zápasu sa nemení. Taktiež je vnášaná chyba z normálneho rozdelenia okolo 0.0 aj do vzdialeností a uhlov. Pre vzdialenosti je so sigmou 0.0965, pre smerový uhol je sigma 0.1225 a pre uhol sklonu je sigma 0.1480.

Syntax: (Vision (<Type> (team <teamname>) (id <id>) (pol <distance> <horizontal angle> <latitudal angle>)))

Možné hodnoty sú:

- Flag s <id>: ’1 l’, ’2 l’, ’1 r’, ’2 r’
- Goal s <id>: ’1 l’, ’2 l’, ’2 l’, ’2 r’
- Player s <id>: celé číslo (číslo hráča)

Príklad zoznamu videných objektov:

```
(Vision (Flag (id 1_l) (pol 54.3137 -148.083 -0.152227)) (Flag (id 2_l)(pol
59.4273 141.046 -0.131907)) (Flag (id 1_r) (pol 61.9718 -27.4136 -0.123048))
(Flag (id 2_r) (pol 66.4986 34.3644 -0.108964)) (Goal (id 1_l) (pol 46.1688
```

```
179.18 -0.193898)) (Goal (id 2_l) (pol 46.8624 170.182 -0.189786)) (Goal (id
1_r) (pol 54.9749 0.874504 -0.149385)) (Goal (id 2_r) (pol 55.5585 8.45381 -
0.146933)) (Ball (pol 6.2928 45.0858 -0.94987)) (Player (team robolog) (id 1)
(pol 7.33643 37.5782 5.86774)))
```

Hear perceptor

`Hear` perceptor vráti zachytenú správu odoslanú hráčom. Formát prijatej správy je:

`(hear <time> <direction in degree> <message>)`, kde `<time>` je čas prijatej správy, `<direction in degree>` je smer (uhol v rovine x-y), kde sa nachádza odosielateľ. Ak správu prijal ten istý hráč, ktorý ju odoslal, tak je táto premenná nastavená na hodnotu `self`, `<message>` je text správy. Maximálna dĺžka je `sayMsgSize`.

Parametre servera pre nastavenie `Hear` perceptora:

- `audioCutDist`, predvolene `50.0`
- `hearMax`, predvolene `2`
- `hearInc`, predvolene `1`
- `hearDecay`, predvolene `2`
- `sayMsgSize`, predvolene `512`

Hráč môže zachytiť správu, ak je jeho kapacita počúvania aspoň `hearDecay`. Ak hráč obdrží správu, tak sa mu táto kapacita zníži o hodnotu `hearDecay`. Každým simulačným cyklom sa však kapacita zvyšuje o `hearInc` až po maximálnu hodnotu `hearMax`. Pre každý tím má hráč samostatnú kapacitu počúvania. Je to kvôli tomu, aby sa nepreťažil komunikačný kanál a neznemožnila sa tak komunikácia súperovho tímu.

Príklad prijatej správy:

```
(hear 0.8 -179.99 Test_1)
(hear 0.4 self Test_2)
```

GameState perceptor

Tento perceptor informuje hráča o aktuálnom hernom stave. Prvé, čo hráč dostane od `GameState` perceptora, sú informácie o lopte a rozmeroch ihriska.

Syntax: `(GameState (<Name> <Value>) ...)`

Premenná `<Name>` môže mať nasledujúce hodnoty:

- aktuálny simulačný čas v sekundách (reálne číslo)
- herný mód `playmode` ako reťazec: `BeforeKickOff, KickOff Left, KickOff Right, PlayOn, KickIn Left, KickIn Right, corner kick left, corner kick right, goal kick left, goal kick right, offside left, offside right, GameOver, Goal Left, Goal Right, free kick left, free kick right, unknown.`

Zoznam všetkých možných herných módov sa nachádza v súbore

`./plugin/soccer/soccertypes.h`

Príklad výstupu `GameState` perceptora:

```
(GameState (time 0) (playmode BeforeKickOff))
```

AgentState perceptor

`AgentState` perceptor nás informuje o stave hráča, ako je jeho teplota a stav batérie.

Syntax: `(AgentState (battery <battery level in percent>) (temp <temperature in degree>))`

Príklad výstupu perceptora `AgentState`: `(AgentState (battery 100) (temp 23))`

2.2.1.4 Efektory

Create efektor

Keď sa agent pripojí k simulátoru, tak nemá zadefinovanú žiadnu fyzikálnu reprezentáciu. Jediná vec, ktorú agent má k dispozícii, je `Create` efektor. Pomocou `Create` efektoru vieme získať ďalšie efektory, perceptory a typy robota.

Príklad: `(create)`

Init efektor

`Init` efektor slúži na nastavenie mena tímu a jedinečného čísla hráčovi.

Syntax: `(init (unum <number>) (teamname <string>))`

Príklad: `(init (unum 7) (teamname RoboLog))`

Beam efektor

Používa sa na nastavenie hráča na požadované miesto.

Syntax: `(beam <x> <y> <z>)`

Príklad: `(beam -6.6 0 0)`

Drive efektor

Slúži na pohyb hráča. Súradnica x smeruje k súperovej bráne a súradnica z smeruje nahor. Každej súradnici môžeme priradiť maximálnu hodnotu 100. Ak chceme hráča maximálne urýchliť, použijeme `Drive` efektor iba raz. Potom sa v každom simulačnom kroku vypočíta sila podľa nastavenia efektora, pokiaľ ho znovu nezmeníme.

Syntax: `(drive <x> <y> <z>)`

Príklad: `(drive 20.0 50.0 0.0)`

Kick efektor

`Kick` efektor je určený na odkopnutie lopty od hráča v určitom smere. Lopta musí byť od hráča v menšej vzdialenosti ako je 0.04 m, aby mohol byť efektor použitý. Sklon kopu je obmedzený na maximálnych 50° od zeme. Sila kopu môže byť z intervalu 0 až 100.

`Kick` efektor pridá lopte silu a moment sily. Vykoná sa za 10 simulačných krokov. Pri kopnutí nemôžeme nastaviť smer kopu v rovine ihriska. Ak chceme loptu odkopnúť v požadovanom smere, musíme najprv v tomto smere nastaviť aj hráča, tak aby bol na priamke s loptou.

Tak ako `Drive` efektor, aj `Kick` efektor je možné použiť iba pri styku hráča s hracou plochou. `Kick` efektor vnáša chybu do smeru odkopu lopty. Teda smer v rovine x-y je vychýlený náhodnou chybou normálneho rozdelenia okolo 0.0 so sigmou rovnou 0.02. Chyba zarátavaná do sklonu je z normálneho rozdelenia okolo 0.0 a sigmou 0.9 (pri uhloch 0° a 50°) a sigmou 4.5 (v strede intervalu uhla sklonu). Chyba sily je tiež z normálneho rozdelenia so sigmou 0.4.

Syntax: `(kick <angle> <power>)`

Príklad: `(kick 20.0 80.0)`

Say efektor

`Say` efektor sa používa na komunikáciu medzi hráčmi. Hráči si môžu posielať správy s maximálnou veľkosťou `sayMsgSize` (512 znakov). Dovolené znaky v správe sú všetky okrem

© Hviezdna jedenástka, 2008

bielych znakov a zátvoriek. Vyslanú správu jedného hráča môže druhý hráč prijať, ak je od neho vo vzdialenosti maximálne `audioCutDist` metrov (stanovené na 50).

Syntax: `(say <message>)`

Príklad: `(say player10_Pass)`

2.2.2 3D Server s hráčmi typu "Humanoid"

2.2.2.1 Štruktúra inštalačného balíčka

./app/

Rôzne aplikácie. Sú tu väčšinou testovacie programy rôznych subsystémov simulátora.

./app/simulator/

Nachádza sa tu simulátor futbalu.

./app/rcssmonitor3d/

Jednoduchý monitor na zobrazovanie agentov pripojených na server simulátora.

./app/agentspark/

Vzorový humanoidný agent, ktorý stojí a máva rukami. So serverom komunikuje pomocou protokolu TCP.

./lib/

Hlavné časti simulátora a knižnice. Simulátor sa skladá z troch častí, `Salt`, `Zeitgeist` a `Oxygen`.

./lib/salt/

Knižnica s kolekciou základných tried. Obsahuje hlavne matematické a systémové funkcie ako triedy `3D Vector`, `Matrix`, pomôcky na vstup a výstup alebo na nahrávanie a prístup k zdieľaným knižniciam.

./lib/zeitgeist/

Knižnica `Zeitgeist` umožňuje dve hlavné funkcie. Prvá je mechanizmus na prácu s objektmi v C++. Okrem tohto mechanizmu knižnica umožňuje hierarchizáciu objektov. Je to v podstate niečo ako systém súborov, kde adresáre a súbory sú inštancie C++ tried. Tieto dve funkcie sú navzájom prepojené. Objekty v hierarchii sú identifikované jedinečným menom. Knižnica `Zeitgeist` ďalej poskytuje tri veľmi dôležité služby: `LogServer`, `FileServer` a `ScriptServer`.

./lib/oxygen/

`Oxygen` tvorí jadro simulácie. Je implementovaný nad `Zeitgeist` rozhraním a poskytuje viacero služieb klientskej aplikácii. Stará sa o fyzikálnu stránku (`PhysicsServer`), geometrickú stránku (`PhysicsServer`) a o stránku agenta (`AgentAspect`, `ControlAspect`). Ďalšou dôležitou časťou je vykonávanie simulácie scény (`SceneServer`).

./lib/kerosin/

`Kerosin` sa stará o vizualizačnú časť (`ImageServer`, `FontServer`, `OpenGLServer`, `TextureServer`, `MaterialServer`). Triedy v tejto knižnici poskytujú aj základné rozhranie na prijímanie vstupov pri interaktívnej simulácii (`InputServer`) a na prehrávanie zvukov (`SoundServer`).

© Hviezdna jedenástka, 2008

./plugins/

Adresár obsahujúci prídavné moduly na rozšírenie simulátora.

2.2.2.2 *Spustenie simulácie*

Simulácia futbalu sa spúšťa príkazom

```
./app/simspark/simspark
```

Tento príkaz spustí shellovský skript `simspark`, ktorý skompiluje zdrojové kódy v adresári

```
./app/simspark/
```

Skompilované zdrojové kódy programu `simspark` sa uložia do adresára

```
./app/simspark/.lib/
```

odkiaľ shell skript program `simspark` spustí.

SimSpark

Zdrojový kód triedy `SimSpark`, ktorá wrapuje triedu `Spark`, sa nachádza v súbore `main.cpp` v adresári `./app/simspark/main.cpp`. Inštancia wrapovacej triedy `SimSpark` je inicializovaná parametrami príkazového riadku.

```
SimSpark.init(<parametre príkazového riadku>)
```
[1]

Po inicializácii sa pomocou simulačného servera[2] spustí simulácia:

```
SimSpark.GetSimulationServer()->Run(<parametre príkazového riadku>)
```

Inicializácia triedy Spark

Zdrojové kódy triedy `Spark` sa nachádzajú v adresári `./lib/spark/spark.h`

Pri inicializácii v metóde `init(…)` sa spustí

- Logovací server
- Skriptovací server

Skriptovací server spustí ruby skript `./usr/local/share/rcssserver3d/spark.rb`. Po jeho vykonaní sa skontroluje, či bol vytvorený `SceneServer` a `SimulationServer`. Ak áno, tak sa uložia smerníky na ich inštancie do atribútov `mSceneServer` a `mSimulationServer`.

Skript spark.rb

V skripte `spark.rb` sa definujú základné parametre pre komunikáciu hráča so serverom, parametre monitoru servera a monitoru klienta.

Parametre AgentControl:

```
$agentStep = 0.02
$agentType = 'tcp'
$agentPort = 3100
```

Parametre Monitor Control:

```
$renderStep = 0.04
$monitorInterval = 2;
$monitorStep = 0.04
$serverType = 'tcp'
$serverPort = 3200
```

Parametre SparkMonitorClient:

---

[1] metóda SimSpark::init(<par. prík. riadku>) je metódou rodičovskej triedy Spark (./lib/spark/spark.h)

[2] trieda `oxygen::SimulationServer`)

```
$monitorServer = '127.0.0.1'
$monitorPort = 3200
$monitorType = 'tcp'
```

Ďalej sa v skripte nastavuje cesta k scéne a k hlavnému serveru:

```
$scenePath = '/usr/scene/'
$serverPath = '/sys/server/'
```

Vytvorí sa server na fyzikálnu simuláciu

```
new('oxygen/PhysicsServer', $serverPath+'physics')
```

server scény

```
sceneServer = new('oxygen/SceneServer', $serverPath+'scene')
sceneServer.createScene($scenePath)
```

a server pre geometriu

```
geometryServer = new('oxygen/GeometryServer', $serverPath+'geometry')
importBundle 'voidmeshimporter'
geometryServer.initMeshImporter("VoidMeshImporter");
importBundle 'objimporter'
geometryServer.initMeshImporter("ObjImporter");
```

Inicializuje sa importér scény

```
importBundle 'rubysceneimporter'
sceneServer.initSceneImporter("RubySceneImporter");
```

a vytvorí sa model sveta

```
world = new('oxygen/World', $scenePath+'world')
world.setGravity(0.0, 0.0, -9.81)
world.setCFM(0.001)
world.setAutoDisableFlag(true)
world.setContactSurfaceLayer(0.001)
new('oxygen/Space', $scenePath+'space')
```

Okrem týchto objektov skript inicializuje ešte objekty `monitorServer`, `gameControlServer`, `simulationServer` a načíta perceptory a efektory agenta.

Podrobný postup pre inštaláciu servera a vývojového prostredia a taktiež podrobnosti týkajúce sa ovládania servera a simulácie sú popísané v používateľskej príručke.

Agent

Agent komunikuje so serverom prostredníctvom siete. Môže sa použiť TCP alebo UDP protokol. Typ protokolu ako aj porty, na ktorých prebieha komunikácia hráča so serverom, sa definujú v ruby skripte `spark.rb`.

2.2.2.3 Perceptory

TimePerceptor

Udáva čas od začiatku simulácie

Syntax: `(TIME (now <x>))`, kde `<x>` je čas od začatia simulácie v sekundách

Príklad: `(TIME (NOW 40.62))`

GameStatePerceptor a VisionPerceptor

Sú rovnaké ako pri verzii servera typu "Sphere".

GyroRatePerceptor(torso)

Vektor rotácie agenta [lokálny], podáva informácie o zmene orientácie trupu. Perceptor poskytuje tri hodnoty zodpovedajúce zmene uhlov okolo x, y, respektíve z osi.

Syntax: `(GYR (name torso) (rt <x> <y> <z>))`, kde `<x> <y> <z>` sú zmeny uhlov jednotlivých osí.

Príklad: `(GYR (name torso) (rt 0.01 0.00 0.00))`

UniversalJointPerceptor

Perceptor univerzálneho kĺbu, poskytuje informácie o aktuálnych uhloch osí kĺbu.

Syntax: `(UJ (n <name>) (ax1 <x>) (ax2 <y>))`, kde `<name>` je názov kĺbu, `<x>` je uhol prvej osi, `<y>` je uhol druhej osi

Príklad: `(UJ (n laj1_2) (ax1 0.05) (ax2 0.00))`

HingePerceptor

Perceptor pántového kĺbu, poskytuje informácie o aktuálnom uhle kĺbu.

Syntax: `(HJ (n <name>) (ax <x>))`, kde `<name>` je názov kĺbu, `<x>` je uhol osi

Príklad: `(HJ (n laj3) (ax 0.05))`

Dostupné kĺbové perceptory:

V Tab. 1 je uvedený prehľad kĺbových perceptorov.

typ	názov	umiestnenie
UniversalJointPerceptor	laj1_2	ľavé rameno
UniversalJointPerceptor	raj1_2	pravé rameno
HingePerceptor	laj3	ľavé nadlaktie
HingePerceptor	raj3	pravé nadlaktie
HingePerceptor	laj4	ľavé predlaktie
HingePerceptor	raj4	pravé predlaktie
HingePerceptor	llj1	ľavý bok
HingePerceptor	rlj1	pravý bok
UniversalJointPerceptor	llj2_3	ľavé stehno
UniversalJointPerceptor	rlj2_3	pravé stehno
HingePerceptor	llj4	ľavá holenná kosť
HingePerceptor	rlj4	pravá holenná kosť
UniversalJointPerceptor	llj5_6	ľavé chodidlo
UniversalJointPerceptor	Rlj5_6	pravé chodidlo

Tab. 1: Kĺbové perceptory

ForceResistancePerceptor

Perceptor odporu sily, poskytuje informácie o sile aplikovanej na chodidlá v čase nastania kolízie. Taktiež poskytuje bod (v lokálnych súradniciach), kde je koncentrácia sily najvyššia.

Pri kolízií dvoch objektov, pri ktorej sa obidva objekty dotýkajú celým povrchom, existuje veľa kolíznych bodov. Keďže nie je praktické poskytovať kompletné informácie o takejto kolízií, `ForceResistancePerceptor` poskytuje informácie len o kontaktnom bode (vážený priemer

© Hviezdna jedenástka, 2008

všetkých bodov, na ktoré pôsobí sila) a vektore sily (celková sila pôsobiaca na kontaktné body). Výstup je teda len približná aproximácia aplikovanej sily.

ForceResistancePerceptor (lf) a ForceResistancePerceptor (rf)

Perceptory odporu ľavého/pravého chodidla

Syntax: `(FRP (n <name>) (c <px> <py> <pz>) (f <fx> <fy> <fz>))`, kde `<name>` je `lf` (ľavé chodidlo) | `rf` (pravé chodidlo), `<px> <py> <pz>` sú lokálne koordináty bodu koncentrácie sily, `<fx> <fy> <fz>` sú komponenty celkového vektoru sily

Príklad: `(FRP (n lf) (c 0.03 -0.13 0.00) (f 0.20 0.10 3.22))`

2.2.2.4 Efektory

InitEffector a BeamEffector

Sú rovnaké ako pri verzii servera typu "Sphere"

UniversalJointEffector

Efektor nastavuje uhol univerzálneho kĺbu.

Syntax: `(<name> <ax1> <ax2>)`, kde `<name>` je názov efektoru, `<ax1>` je uhlová rýchlosť prvej osi, `<ax2>` je uhlová rýchlosť druhej osi.

Príklad: `(lae4 0.0 22.0)`

HingeEffector

Efektor nastavuje uhol pántového kĺbu

Syntax: `(<name> <axis>)`, kde `<name>` je názov efektoru, `<axis>` je uhlová rýchlosť kĺbu

Príklad: `(lae3 22.0)`

Pohyb hráča zabezpečujú kĺby, ktoré sú ovládané nasledovnými efektormi (Tab. 2)

typ	názov	umiestnenie	min. hodnota	max. hodnota
UniversalJointEffector	lae1_2	ľavé rameno	-90/-10	180
UniversalJointEffector	rae1_2	pravé rameno	-90/-180	180/10
HingeEffector	lae3	ľavé nadlaktie	-135	135
HingeEffector	rae3	pravé nadlaktie	-135	135
HingeEffector	lae4	ľavé predlaktie	-10	130
HingeEffector	rae4	pravé predlaktie	-10	130
HingeEffector	lle1	ľavý bok	-60	90
HingeEffector	rle1	pravý bok	-90	60
UniversalJointEffector	lle2_3	ľavé stehno	-45	120/75
UniversalJointEffector	rle2_3	pravé stehno	-45/-75	120/45
HingeEffector	lle4	ľavá holenná kosť	-160	10
HingeEffector	rle4	pravá holenná kosť	-160	10
UniversalJointEffector	lle5_6	ľavé chodidlo	-90/-45	90/45
UniversalJointEffector	rle5_6	pravé chodidlo	-90/-45	90/45

© Hviezdna jedenástka, 2008

Tab. 2: Kĺbové efektory

2.3 Analýza existujúcich riešení

Cieľom analýzy existujúcich riešení je zistiť, aké prístupy k riešeniu využívajú iné tímy, preskúmať použité algoritmy a postupy. Táto kapitola je rozdelená na dve časti.

V prvej sa venujeme analýze súčasných riešení, ktoré boli vyvinuté pre server s hráčmi typu "Sphere". Podrobnejšie sa venujeme analýze dvoch tímov, ktoré sa tomuto projektu venovali na našej fakulte v minulých rokoch. Ďalej prikladáme analýzu riešenia vybraných zahraničných tímov. Pri nich sa sústreďujeme predovšetkým na niektoré zaujímavé postupy a použité algoritmy.

Druhá časť je venovaná riešeniam pre server s hráčmi typu "Humanoid". Keďže server pre tento typ hráčov bol vyvinutý len niekoľko mesiacov pred začatím nášho projektu, neexistujú žiadne tímy z našej fakulty, ktoré by sa tomuto typu hráča venovali. Počet svetových tímov, ktoré používajú túto novú verziu servera, je oveľa menší než tých, ktoré používajú predchádzajúcu verziu. Opisujeme prístup niektorých vybraných svetových tímov, pričom jednému z nich (tímu Zigorat) sa venujeme podrobnejšie.

2.3.1 Riešenia pre server s hráčmi typu "Sphere"

2.3.1.1 Tím AUTSAJDRY-TNG

Tím AUTSAJDRY-TNG [3] sa venoval RoboCup-u 3D na našej fakulte v predchádzajúcom akademickom roku. Bol jedeným z prvých tímov, ktoré sa venovali 3D robotickému futbalu u nás. Tento tím implementoval kompletný fyzikálno-matematický model sveta pre hráča, čím vytvoril dobré základy pre implementovanie vyšších schopností hráča.

Model sveta

Model sveta obsahuje informácie o polohe ostatných hráčov a lopty. Pre odstránenie šumu, ktorý vnáša server do vysielaných hodnôt, použili Kalmanov filter. Aby mohli tento filter použiť, museli implementovať kompletný fyzikálno-matematický model sveta. Kalmanov filter sa používa na počítanie vlastnej polohy a rýchlosti. Bol zámer realizovať jeho použitie aj na výpočet pozície a rýchlosti lopty, ten sa však z časových dôvodov nepodarilo uskutočniť.

Model sveta implementovali ako jednu triedu `WorldModel`. Obsahuje všetky objekty nachádzajúce sa na ihrisku a dve triedy, ktoré vyjadrujú parametre a stav hry.

Hráč je simulovaný pevnou guľou s polomerom R a hmotnosťou m. Ťažisko hráča je v strede gule. Pohyb hráča je valivý pohyb bez kĺzania. Pri pohybe uvažujeme nasledovné sily:

- Hnacia sila – F (Drive force)
- Krútiaci moment – T (Torque)
- Gravitačná sila – W (Weight)
- Normálová sila - (Normal force)
- Odpor vzduchu
    - lineárny - D_L(Linear Drag)
    - rotačný - D_A(Angular Drag)

Predikcia pohybu

Pre predikciu pohybu bol implementovaný matematický aparát, ktorý bol vytvorený použitím knižnice `Salt`.

Predikcia pohybu zahŕňa:

- Určenie pozície lopty v čase
- Určenie pozície lopty v čase, ak predpokladáme pôsobenie sily na loptu
- Určenie stavu hráča, ak predpokladáme pôsobenie sily
- Určenie miest, kam môže hráč dobehnúť za nejaký čas
- Určenie poradia dobehnutia hráčov na nejaké miesto

Odchytávanie lopty

Hráč je schopný určiť, kedy a kde sa lopta zastaví. Komplexnejšie odchytávanie lopty nebolo z časových dôvodov implementované.

Kopnutie lopty na stanovené miesto

Bolo implementované iba jednoduché kopanie lopty po zemi.

Beh hráča na určené miesto

Implementované boli nasledujúce typy behu:

- beh na miesto so zastavením v cieli
- beh na miesto bez zastavenia
- beh k lopte

Vyhýbanie sa hráčom

Pri návrhu uvažovali dva spôsoby vyhýbania sa:

- Reaktívny prístup - hráč v každom cykle vyhodnocuje situáciu a reaguje na ňu . Je problematické odhadnúť čas potrebný na presun, ak treba obchádzať hráča.
- Pohyb po krivke – hráč na začiatku pohybu určí riadiace body vhodnej krivky a potom sa pohybuje po tejto krivke. Ľahšie sa dá odhadnúť potrebný čas na presun hráča.

Rozhodli sa implementovať vyhýbanie sa pohybom po krivke. Vyhýbanie sa po krivke je najlepšie využiť pre pohyb na krátke vzdialenosti, ako napríklad predbiehanie hráča. Preto si myslia, že v hráčovi by bolo dobré doplniť aj reaktívne vyhýbanie sa, ktoré by sa dalo použiť v prípade, že by sa v okolí hráča nachádzalo viacero prekážok.

Jednoduchá logika hráča

- hneď na začiatku sa hráči premiestnia (beam) na svoje miesto vo formácií
- určí sa najbližší hráč k lopte, ktorý spraví výkop
- výkop spraví kopnutím lopty pár metrov pred seba
- počas hry hráč určí zodpovednosť za loptu (najbližší spoluhráč)
- ak má hráč zodpovednosť za loptu, beží k nej
- ak nemá zodpovednosť, tak beží vo formácií
- hráč beží za loptou na miesto jej zastavenia
- smer natočenia okolo lopty volí podľa najlepšej možnosti na prihrávku
- určenie najlepšej prihrávky spočíva vo výbere hráča najviac vpredu, ktorý nie je ďalej ako určitá hranica – teda netestuje sa vhodnosť prihrávky
- ak nemá komu prihrať, tak dribluje v smere kladnej x-ovej osi

Podpora ladenia

Podpora ladenia hráča pozostáva z troch hlavných častí a to ukladanie informácií, obnovenie stavu hráča a vizuálny monitor. Informácie o stave hráča sa ukladajú do výstupného súboru vo formáte XML. Tento súbor je následne možné nahrať a zobraziť v monitore.

2.3.1.2 Tím 6th Sense

Tento tím, rovnako ako predchádzajúci, pôsobil na našej fakulte minulý akademický rok. Podrobný popis ich práce je uvedený v [4].

Tím vychádzal z hráča predchádzajúceho tímu HazardTeam. Model hráča je určený pre server verzie 0.5.2. Tím implementoval hráčovi tieto základné schopnosti (Tab. 3)

Názov schopnosti	Popis	Vrstva
runToPlace(x, y)	Beh na miesto.	1
runToObject(obj)	Beh k objektu obj.	1
runAroundObject(obj, dir)	Beh k objektu obj. Pri dobehnutí k objektu má mať spojnica ťažísk objektu a hráča smerový uhol dir.	1
runToEvent(x, y, t)	Dobehnutie na miesto (x,y) v čase t.	1
kickToPlace(dist)	Kopnutie na vzdialenosť danú parametrom dist. Kope sa smerom, v ktorom je hráč natočený.	1
kickToEvent(dist, t)	Kopnutie na vzdialenosť danú parametrom dist. Kope sa smerom v ktorom je hráč natočený. Lopta má doraziť na cieľové miesto v čase t.	1
kickToPlaceWithVelocity(dist, v)	Kopnutie na vzdialenosť danú parametrom dist. Kope sa smerom v ktorom je hráč natočený. Lopta má doraziť na cieľové miesto s rýchlosťou v.	1
lookAtPlace(x,y)	Hráč natočí kameru tak, aby zadané miesto bolo v strede jeho zorného poľa.	1
lookAtObject(obj)	Hráč natočí kameru tak, aby zadaný objekt bol v strede jeho zorného poľa.	1
scan()	Hráč otáča kameru tak, aby postupne zaznamenal celé svoje okolie.	1
extendedKickToPlace(x, y)	runAroundObject + kickToPlace	2
extendedKickToEvent(x, y, t)	runAroundObject + kickToEvent	2
extendedKickToPlaceWithVelocity(x, y, v)	runAroundObject + kickToPlace	2
dribbleToPlace(x, y)	Beh s loptou na dané miesto. Hráč musí mať pred začiatkom tohto správania loptu vo svojej blízkosti.	2
shoot()	Strela na bránu. Hráč musí mať pred začiatkom tohto správania loptu vo svojej blízkosti.	3
pass(obj)	Prihrávka objektu obj. Hráč musí mať pred začiatkom tohto správania loptu vo svojej blízkosti.	3
passToRun(obj)	Prihrávka do pohybu objektu obj. Hráč musí mať pred začiatkom tohto správania loptu vo svojej blízkosti.	
interceptBall()	Prerušenie pohybu lopty.	3
extendedInterceptBall(dir)	Prerušenie pohybu lopty a nastavenie sa do polohy, pri ktorej má spojnica lopty a hráča uhol dir.	3
kickIn()	Vkopnutie lopty do hry.	3
holdPosition()	Hráč zostáva stáť na mieste.	3
holdFormation()	Hráč sa presunie na svoje miesto vo formácii.	3

Tab. 3: Prehľad schopností hráča tímu 6th sense

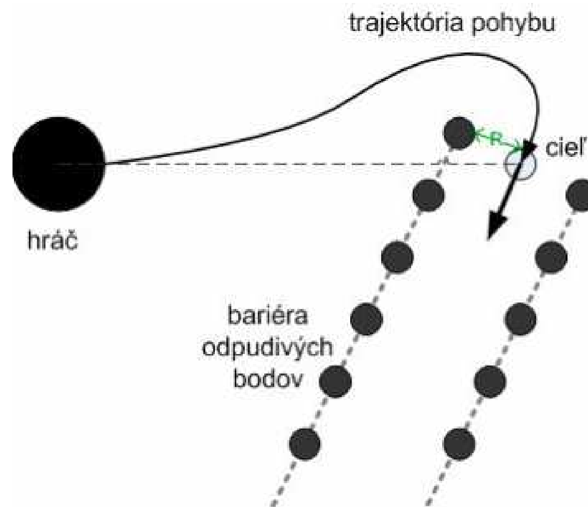
Implementované základné schopnosti boli použité pri modulárnej implementácii vyššieho správania a rozhodovania hráča. Jednotlivé moduly správania rozdelili do vrstiev tak, že moduly vyšších vrstiev využívajú moduly nižších.

Hráčov model sveta (Information storage)

Tím 6th sense vytvoril hráčov model sveta, `Information storage` (IS). Vzorom im pri tom bol prístup tímu Brainstormers 2D, ktorí pri základných činnostiach hráča upustili od neurónových sietí a spoľahli sa na matematický výpočet prihliadajúc na fyzikálne zákony. Hráč má k dispozícii svet v dvoch súradnicových sústavách, v ortogonálnej a polárnej. V ortogonálnej sústave určuje hráč svoju polohu a polohy ostatných objektov. IS tvoria dva moduly, `WorldModel` (WM) a `Prediktor`. Vo WM sa pre každý objekt vytvára história jeho stavov. Údaje v histórii slúžia potom prediktoru na výpočty. Tím vytvoril základný prediktor, z ktorého odvodili ďalšie dva prediktory, pre hráčov a loptu. Prediktor plní dve základné úlohy: predikcia aktuálnej polohy, výpočet rýchlosti a zrýchlenia a predikcia budúcej polohy.

Behanie hráča

Na dosiahnutie toho, aby hráč dobehol na miesto z požadovaného smeru použili model potenciálových polí (Obr. 1).



Obr. 1: Model potenciálových polí

Bariéra odpudivých bodov sa generuje na základe požadovaného smeru v cieľovom bode. Odpudivé body bariéry sú rovnomerne rozostavené rovnobežne s požadovaným vektorom rýchlosti v cieľovom bode.

Formácie

Modul na formácie hráčov bol prebratý z 2D futbalu, z tímu Deravá kopačka a bol prerobený na podmienky 3D futbalu. Modul obsahuje 27 formácií a počas simulácie sa pomocou váhovania a ratingu jednotlivých formácií vyberá najvhodnejšia pozícia hráča.

Dribling

Samotné driblovanie spočíva v postupnom nakopávaní lopty tak, aby sa lopta dostala na cieľové miesto za najkratší možný čas. Využívajú sa tu potenciálové polia a radiálny kop lopty od hráča.

Rozhodovacie moduly

Princíp rozhodovania spočíva v hierarchickom usporiadaní rozhodovacích modulov do akéhosi stromu podľa vrstiev. Na najvyššej úrovni je jediný hlavný modul, ktorý vyberie a spustí modul vo vrstve pod ním, teda jeden z modulov herných módov. Každý modul herného módu má zasa v nižšej vrstve akcie, ktoré môže vykonávať. Samotné vykonanie akcie je zapuzdrené v rozhodovacích moduloch na najnižšej úrovni. Na vykonanie akcie používajú moduly schopnosti hráča.

Brankár

Brankár sa snaží vykrývať strelecký uhol protihráča na spojnici stredu bránkovej čiary a aktuálnej polohy lopty. Od pevne danej vzdialenosti vyráža proti lopte v snahe ju chytiť. Po chytení lopty ju odkopáva smerom od brány pod uhlom 45 stupňov. Modul brankára nerieši situáciu, kedy súper strieľa z diaľky na bránu.

Zhrnutie

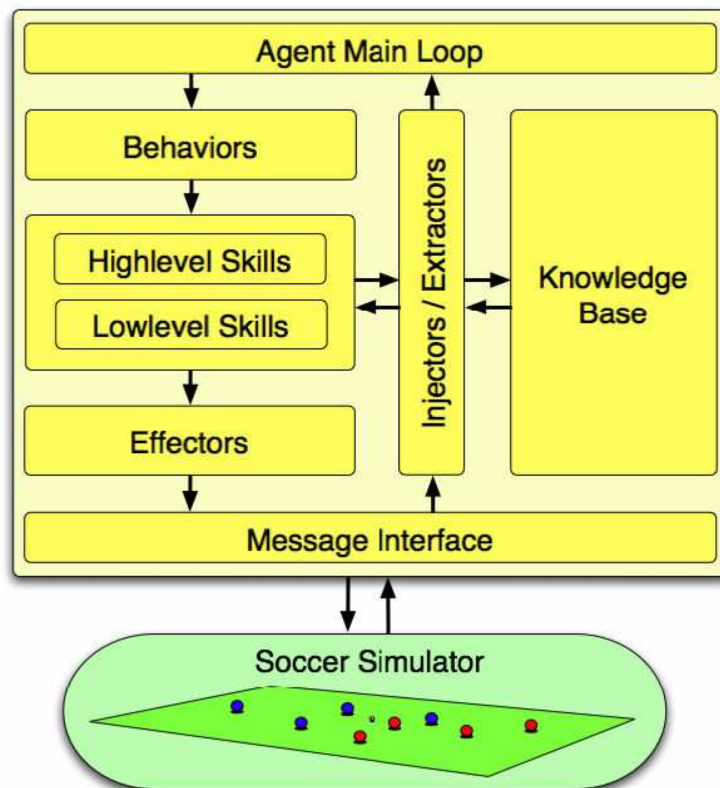
Strategické moduly sú vytvorené iba na úrovni prototypu a nie sú dostatočne odladené. Vzájomná integrácia modulov taktiež nie je dokonalá, kvalite hry vytvoreného hráča by zrejme prospelo aj experimentálne doladenie konštánt, ktorých je v kóde veľké množstvo. Veľmi dobre je navrhnutá architektúra hráča, ktorá počíta s modulárnym rozširovaním hráčových schopností.

2.3.1.3 Tím Virtual Werder 3D

Virtual Werder [5] vznikol v Nemecku v roku 1999 ako 2D tím simulovaného robotického futbalu. Následne od roku 2004 začal pôsobiť aj v 3D simulovanej lige na univerzite Bremen. Zúčastnil sa na medzinárodnom turnaji v 3D simulovanej lige (Lisabon 2004, Osaka 2005, Bremen 2006, 2. miesto v German Open 2007, štvrťfinále v Atlante 2007). Najväčším úspechom bolo dosiahnutie štvrťfinálového kola v Bremene 2006, kde 19 zápasov po sebe nedostali gól. Momentálne sa tím skladá z troch inžinierov, jedného PhD študenta a jedného študenta bakalárskeho štúdia.

Celková architektúra hráča

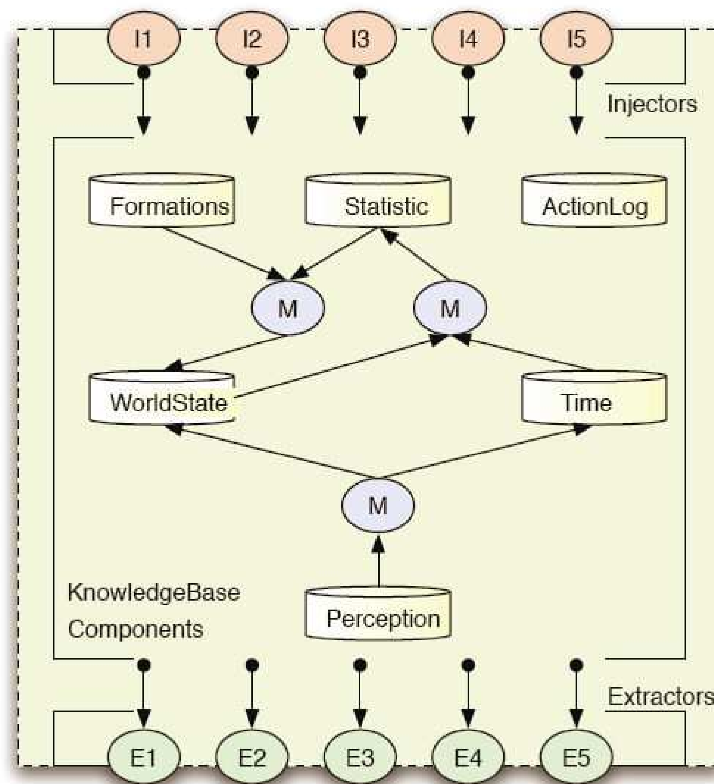
Na Obr. 2 je znázornená celková architektúra agenta, resp. hráča tímu Virtual Werder 3D 2006 [6].



Obr. 2: Architektúra hráča tímu Virtual Werder 3D

Architektúra je postavená z dvoch častí. Prvá časť obsahuje všetko, čo je potrebné pre vnem – usudzovanie – akciu, konkrétne agentov hlavný cyklus, správanie, zručnosti, efektory a komunikačné rozhranie. Tieto časti sú usporiadané do úrovní, kde každý element na jednej úrovni môže používať elementy v jeho a priamo podradenej úrovni. Táto architektúra nielenže zobrazuje jasné povinnosti v každej vrstve, ale robí programový kód prehľadnejší a ľahší na rozšírenie.

Druhá časť zahŕňa všetky znalosti, ktoré sú potrebné na dedukciu a vykonávanie všetkých výpočtov. Báza znalostí obsahuje všetky informácie o svete, ako aj všetky informácie o samom agentovi. Prístup k báze znalostí je možný pre všetky triedy cez definované rozhranie, ktoré podporuje ako nízkoúrovňové tak aj vysokoúrovňové dotazy pre dáta. Štruktúra bázy znalostí hráča tímu Virtual Werder 3D 2006 je znázornená na Obr. 3.



Obr. 3: Schéma štruktúry bázy znalostí hráča tímu Virtual Werder 3D

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.

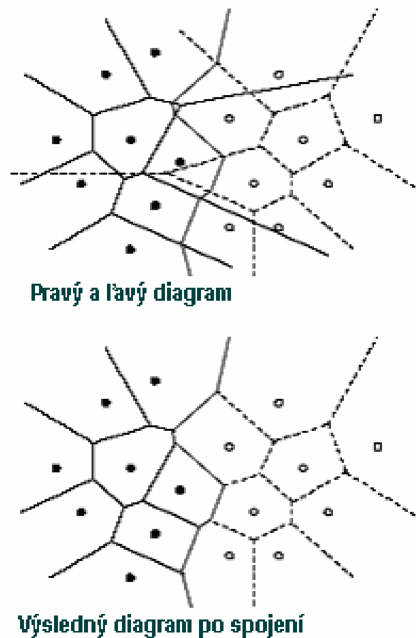
Zručnosti hráča sú rozdelené do dvoch časti, nízkoúrovňové zručnosti a vysokoúrovňové zručnosti. Medzi nízkoúrovňové zručnosti patria: Move, Kick, Beam a Say. Tieto prakticky len volajú príslušné funkcie servera. Medzi vysokoúrovňové zručnosti patria: Cover, Score, Intercept, Pass, Reposition a Goalkick.

Cover sa používa na pokrytie protihráča, t.j. hráč sa snaží zostať blízko protihráča a pripravovať sa na zachytenie lopty. *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. 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.

Voronoiove diagramy

Voronoiove diagramy sú súhrnom čiar, rozdeľujúcich danú plochu s vopred danými bodmi tak, aby susedné body boli od deliacej čiary rovnako vzdialené [7]. Existuje niekoľko metód, ako vypočítať Voronoiove hrany:

- pretínanie bisekov polrovín: počíta Voronoiovu bunku pretínaním n-1 bisekov, zložitosť algoritmu je O(n2log 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 Obr. 4 je znázornený princíp tohto algoritmu.



Obr. 4: 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.

Role definované v tíme Virtual Werder 3D sú nasledovné:

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

- útočník

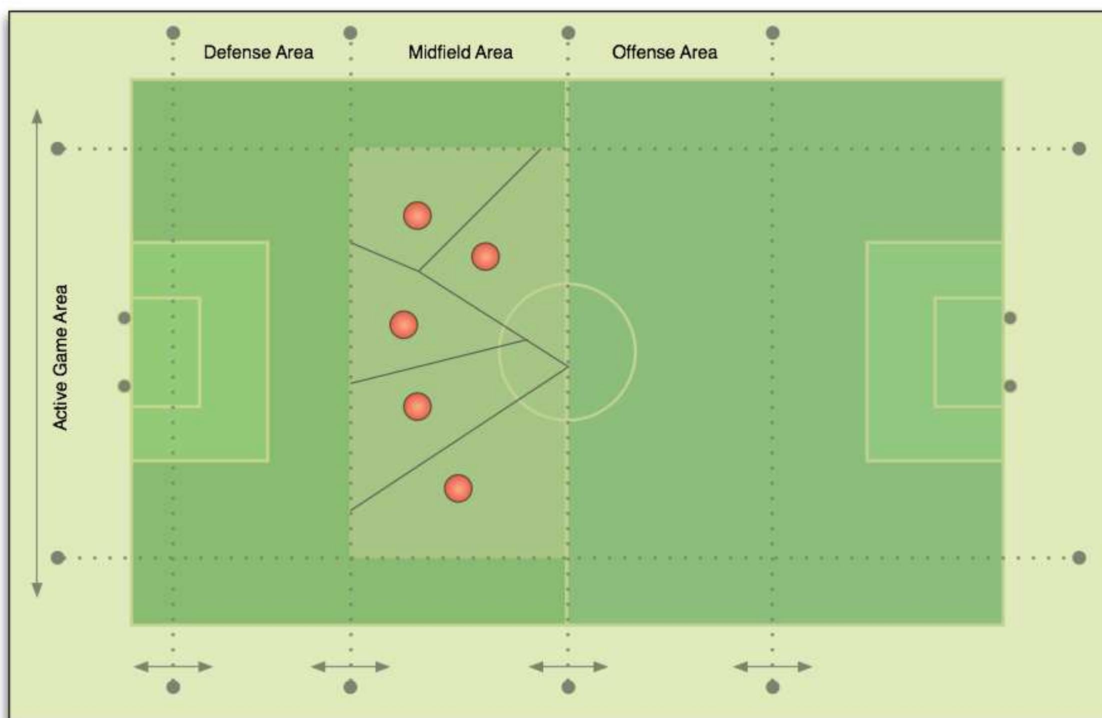
Ak útočníci majú loptu, stredopoliari a obrancovia sa trošku vysúvajú dopredu, aby im mohli pomáhať. Ak sa útočí na obrancov, tak stredopoliari a útočníci sa sťahujú, aby pomohli obrancom.

Popis zmeny miesta (reposition)

Reposition je zodpovedný za výpočet „dobrej“ pozície pre agenta, ktorý nejde za loptou. Dobrá pozícia je taká, keď agent nie je ďaleko od lopty, nie je blízko k tímovému hráčovi a nie je až tak blízko pri protihráčovi. Potom môže dostať loptu a má čas na premýšľanie nasledujúceho ťahu, kým k nemu nepríde protihráč.

Premiestňovanie hráčov je na základe Voronoiových diagramov. Každému hráčovi je pridelený jeden polygón v diagrame, v ktorom sa snaží ísť do jeho stredu. Z Obr. 5 vyplýva, že hráči potom nie sú tesne pri sebe.

Nepoužíva sa celá hracia plocha. Tá je rozdelená na 3 časti: obranná, stredná a útočná.



Obr. 5: Zobrazenie Voronoiových buniek v hracom poli

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.

Ďalším 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.

© Hviezdna jedenástka, 2008

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.

2.3.1.4 Tím ZJUBase

ZJUBase [8] je čínsky tím pochádzajúci zo Zhejiang University. Tím sa venoval pôvodne 2D futbalu, neskôr od roku 2004 tiež 3D futbalu. ZJUBase patrí medzi popredné svetové tímy, na svetových šampionátoch sa umiestnil na 3. mieste v roku 2005 a 2006. Tím má na svojej webovej stránke zverejnené zdrojové kódy, tie sú však bez dokumentácie. Pri zdrojových kódoch sa však nachádza krátky dokument popisujúci najnovšie použité techniky tímu [9].

Tím sa zameral na analýzu skutočného futbalu a uplatnenie jeho techník v robotickom futbalu. V dokumente sa ďalej píše o postupe v tomto type hráčových techník, hlavne o metóde vyhodnotenia prihrávky a o stratégií dynamických pozícií. Tieto dve problémy zodpovedajú dvom spôsobom rozhodovania hráča, rozhodnutie ak je hráč pri lopte, respektíve nie je. V dokumente je tiež opísaná metóda Bayesovského odhadu pre sebalokalizáciu hráča.

Rozhodovanie sa s loptou

Hráč, ktorý má pod kontrolou loptu, má v zásade tri možnosti:

- vystreliť na bránu
- driblovanie do určenej pozície
- nahrávka spoluhráčovi

Tieto tri možnosti však môžeme redukovať na prihrávku: strelu na bránku je možné zredukovať na prihrávku do pozície za bránou a driblovanie môžeme chápať ako prihrávku samému sebe.

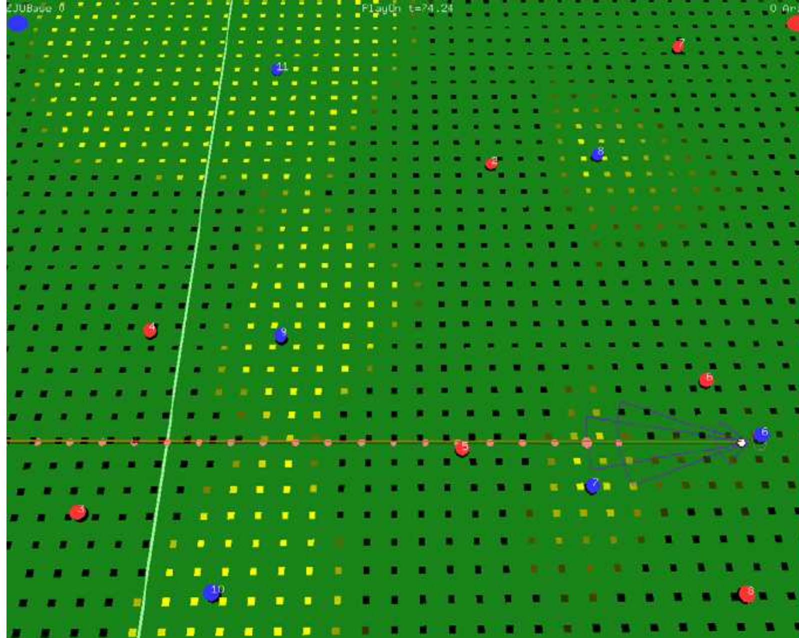
Ak sa hráč rozhodne prihrať loptu, ešte je potrebné vyriešiť dva problémy:

- kam nahrať loptu
- ako nahrať loptu na zvolené miesto

Druhý problém je základná zručnosť hráča, takže je potrebné sa zamerať na prvý problém. Tím ZJUBase riešil tento problém nasledovne: Máme zoznam pozícií v poli ako kandidátov, je potrebné, aby sme všetky vyhodnotili. Po vyhodnotení hráč bude vedieť, čo má spraviť ako nasledujúce. Tím navrhol nasledovnú evaluačnú funkciu:

$$V(P) = \kappa_1 e^{-\alpha_1 d_1} - \kappa_2 e^{-\alpha_2 d_2} + \frac{\kappa_3}{|P - P_{goal}| + 1}$$

Pričom $V(P)$ je vyhodnotenie prihrávkovej pozície P. d_1 a d_2 sú vzdialenosti medzi hráčom a najbližším hráčom nášho, respektíve protihráčovho tímu. P_{goal} predstavuje pozíciu oponentovej brány. K_1, K_2 a K_3 sú parametre, ktoré sú nastavené manuálne alebo automaticky pomocou strojového učenia. Tím si vybral funkciu e^x pretože ak hráč stojí príliš ďaleko od lopty, jeho vplyv musí byť zanedbateľný a funkcia e^x spĺňa tieto vlastnosti.



Obr. 6: Výsledok evaluačnej funkcie

Na Obr. 6 vidíme efekt vyhodnotenia vzorca. Farebné malé body na zelenej ploche ilustrujú body ako vhodné plochy na prihrávku. Farba bodu znázorňuje jeho numerické vyhodnotenie. Svetlejšie farby znamenajú vyššie hodnoty. Hráči sú modré body, protihráči červené.

Rozhodovanie sa bez lopty

Väčšinu času počas hry nie je hráč pri lopte. V týchto chvíľach musí hráč ostať alebo sa presunúť na dobrú pozíciu, na prihrávku alebo obranu. Problémom je, ktorú strategickú pozíciu si má hráč vybrať a ako sa tam dostať. Vyhodnotenie strategickej pozície záleží na pozícií lopty a na distribúcií ostatných hráčov. Tím v dokumente popisuje svoju metódu závislú na pozícií lopty.

Používajú prístup "NURB krivky" na popísanie strategických pohybov hráčov v určitých momentoch. V stratégií definujú parametrickú funkciu s použitím pozície lopty na vypočítanie hráčovej pozície. Matematicky má funkcia tvar:

$$Q(u) = \frac{\sum B_i \omega_i N_{i,k}(u)}{\sum \omega_i N_{i,k}(u)}$$

, kde B_i sú projekcie štvor-dimenzionálnych kontrolných bodov a w_i sú ich váhy. $N_{i,k}(u)$ je definované takto:

$$N_{i,0}(u) = \begin{cases} 1 & t_i \leqslant u < t_{i+1} \\ 0 & \end{cases}$$

$$N_{i,k}(u) = \frac{(u - t_i)N_{i,k-1}(u)}{t_{i+k} - t_i} + \frac{(t_{i+k+1} - u)N_{i+1,k-1}(u)}{t_{i+k+1} - t_{i+1}}$$

V tejto rovnici je *Ti* notácia pre *i-ty* uzol vo vektore uzlov a *k* je stupeň krivky. Vo funkcii sú tieto parametre konečné. Tieto tri rovnice sú založené na NURB krivke. Tím použil tieto krivky z dôvodu schopnosti kontrolovať ich hladkosť.

Sebalokalizácia

Na začiatku simulácie musia hráči lokalizovať svoju polohu pomocou zrakových senzorov, ktoré rozlišujú relatívne pozície rohov a bránok. Tím očakáva šum, ktorý je známy a ktorý využíva na lepšiu lokalizáciu. Používa pri tom Bayesov odhad.

Off-line debuger

Off-line debugovací systém je dôležitá časť robotického výskumu. Tak ako simulátor vyžaduje monitor na vizualizáciu stavu simulácie, debugovací systém si žiada tiež určitú vizualizáciu dát. Grafická vizualizácia môže významne pomôcť porozumieť rôznym situáciám, najmä v 3D simulácií.

Záver

Tím sa zaoberal problémom rozhodovania hráča. V nedávnych experimentoch však zistili, že vzorec na ohodnotenie prihrávky v istých situáciách nemôže fungovať správne. Tím stále pracuje na spôsobe, ako vytvoriť iné riešenie, ktoré by zahrňovalo tiež pozície iných agentov.

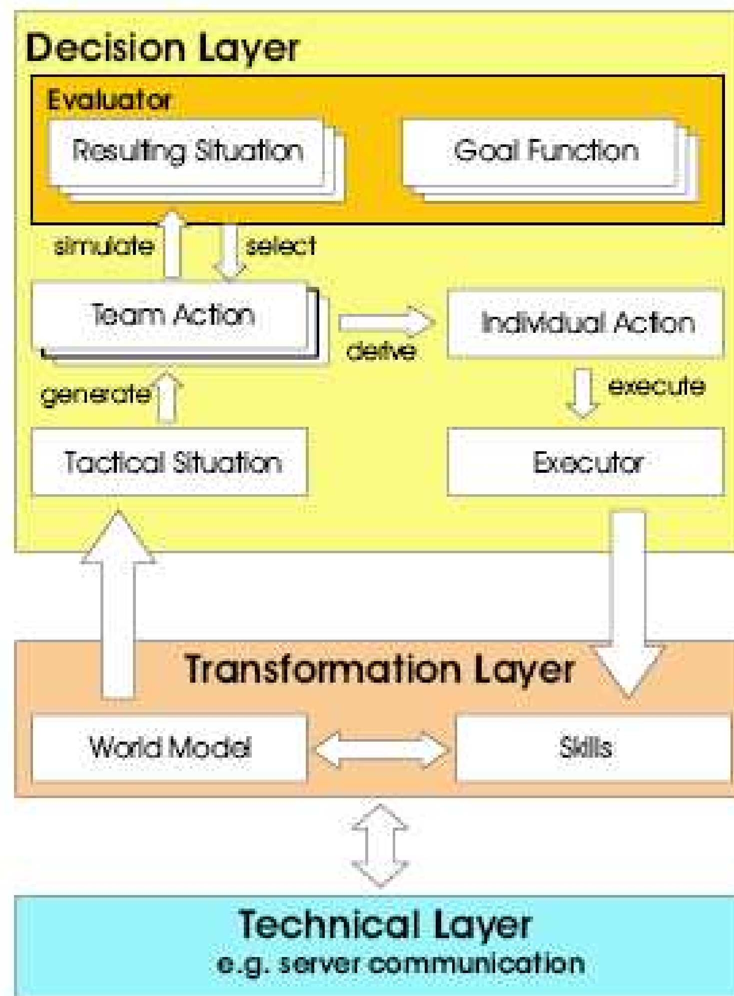
2.3.1.5 Tím Mainz Rolling Brains

Tím Mainz Rolling Brains [10][11] vznikol v Nemecku (Mainz). Od roku 1998 súťažil v 2D ligách simulovaného robotického futbalu. Od roku 2004 sa zúčastňuje aj 3D simulovaných robotických líg.

Architektúra hráča

V roku 2004, keď začali súťažiť v 3D lige, ich hráč vychádzal z hráča používaného v 2D lige. Avšak po zistení viacerých chýb a odlišností sa rozhodli zmeniť štruktúru celého hráča. Odlišnosti boli napr.: pri kopnutí lopty, v 2D sa dalo kopať do strán, v 3D len pred seba. V 3D si bolo treba vybrať spoluhráča na nahrávanie niekoľko cyklov dopredu. Architektúra nového hráča vychádza z troch vrstiev: technickej, transformačnej a rozhodovacej (Obr. 7).

© Hviezdna jedenástka, 2008



Obr. 7: Architektúra hráča tímu Mainz Rolling Brains

Technická vrstva

Zabezpečuje komunikáciu medzi agentom (hráčom) a serverom. Na komunikáciu hráča so serverom sa využíva knižnica SPADES. Komunikáciu prebrali z 2D agenta, t.j. agent čaká na správu od servera a aktualizuje svoj model sveta. Na prijatú správu od servera agent reaguje vybratím určitej akcie, ktorú pošle serveru a opäť čaká na ďalšiu správu.

Transformačná vrstva

Predstavuje nižšiu úroveň hráča. Táto vrstva je rozdelená do dvoch častí a to *model sveta* a *zručnosti*. Model sveta opisuje stav celého sveta pozostávajúceho z veľkého množstva pozícií, vektorov a rýchlostí všetkých hráčov a lopty. Zručnosti hráča predstavujú: utekanie za loptou, zastavenie na mieste, kopnutie do lopty, atď.

Rozhodovacia vrstva

Predstavuje zase vyššiu úroveň hráča a je rozdelená do piatich častí: *taktická situácia*, *tímová akcia*, *individuálna akcia*, *vyhodnocovateľ* a *vykonávateľ*. Taktická situácia získava informácie z modelu sveta. Tieto informácie sa ďalej použijú na opísanie celého tímu na abstraktnej úrovni, napr.: tím má loptu a je na svojej polovici územia alebo súper má loptu a je na našej polovici územia, atď. Tieto taktické situácie tvoria dosť malú množinu situácií. Tímová akcia predstavuje

jednu akciu celého tímu, ktorá pozostáva z 11 individuálnych akcií t.j. 10 individuálnych akcií hráčov a jednej individuálnej akcie brankára. Podľa toho, aká tímová akcia sa má vykonať, budú jednotliví hráči vykonávať svoje individuálne akcie. Typickým príkladom individuálnej akcie je: hráč má loptu a je pred ním súperov hráč, tak loptu nahrá niekomu, alebo hráč má loptu, pred ním nie je súperov hráč, iba brankár, tak vystrelí na bránu, atď. Po určení individuálnej akcie hráča je tu vykonávateľ, ktorý má za úlohu danú akciu vykonať na základe hráčových zručností (2. vrstva). Ešte sa tu nachádza vyhodnocovateľ, ktorý na základe rozostavenia hráčov a pozície lopty vyberie najvhodnejšiu tímovú akciu. Na rozostavenie hráčov sa využíva *REINFORCE* algoritmus kombinovaný s *GROWING NEURAL GAS*.

Algoritmus Neural Gas

Algoritmus Neural Gas bol vytvorený dvoma autormi Martinetzom a Schultenom roku 1991. Tento algoritmus, pre každý vstupný signál ζ, usporadúva jednotky siete podľa vzdialenosti ich referenčných vektorov k ζ. Nasleduje presný postup tohto algoritmu [12].

Inicializácia

Najprv musíme vytvoriť rozvrhnutie $p(\zeta)$, pre ktoré vytvárame model Neural Gas. Názorná ukážka je na Obr. 8.

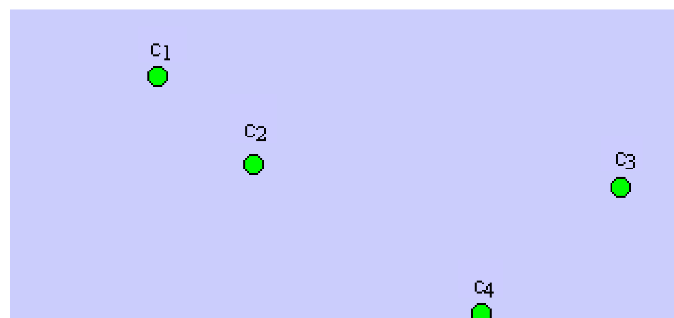
Obr. 8: Rozvrhnutie $p(\zeta)$

Ďalej potrebujeme inicializovať nasledovné parametre:

- Parametre λ_i, λ_f sa používajú na nastavenie miery, akou miera učenia konverguje. λ_i sa zvyčajne nastavuje na hodnotu 30 a λ_f na hodnotu 0.01.
- Parametre e_i, e_f sú inicializačná a finálna hodnota miery učenia. e_i sa zvyčajne nastavuje na hodnotu 0.3 a e_f na hodnotu 0.05.
- Parameter t_{max} predstavuje čas, pokiaľ bude proces pokračovať.

Krok 1

Vytvoríme si množinu A obsahujúcu N rôznych jednotiek s referenčnými vektormi patriacimi do $p(\zeta)$. $A = \{c_1, c_2, \dots , c_N\}$. Jednotky množiny A sú znázornené na Obr. 9.

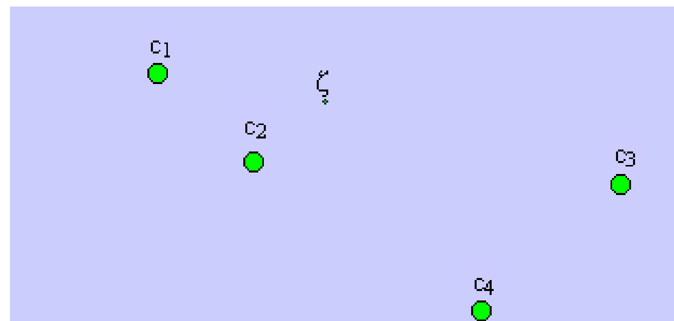


Obr. 9: Jednotky množiny A

Taktiež nastavíme časový parameter t na 0.

Krok 2

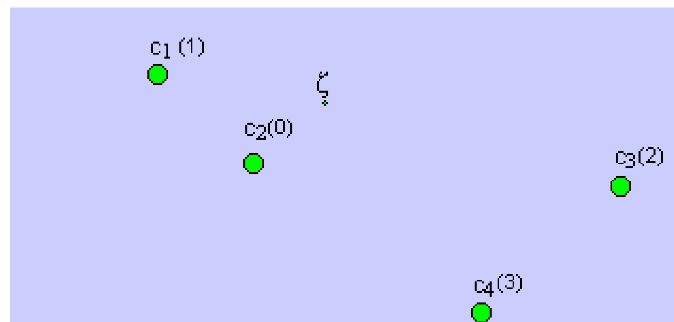
Náhodne si zvolíme vstupný signál ζ, tak aby patril do p(ζ). Ukážka na Obr. 10.



Obr. 10: Voľba vstupného signálu ζ

Krok 3

Usporiadame prvky z množiny A podľa vzdialeností ich referenčných vektorov k ζ. Každý prvok množiny A bude mať priradené určité k z množiny {0,1, … , N - 1}, ktoré vyjadruje jeho pozíciu v usporiadanej postupnosti. Usporiadanie je na Obr. 11.



Obr. 11: Usporiadanie referenčných vektorov

Krok 4

Podľa konvencií budem k označovať $k_i(A, \zeta)$.

Upravím referenčné vektory podľa vzťahu $w_i = w_i + e(t) * h_\lambda(k_i(A, \zeta)) * (\zeta - w_i)$, kde

- $\lambda(t) = \lambda_i(\lambda_f / \lambda_i)^{t / tmax}$,
- $e(t) = e_i(e_f / e_i)^{t / tmax}$,
- $h_\lambda(k_i(A, \zeta)) = \exp(- k / \lambda(t))$.

Krok 5

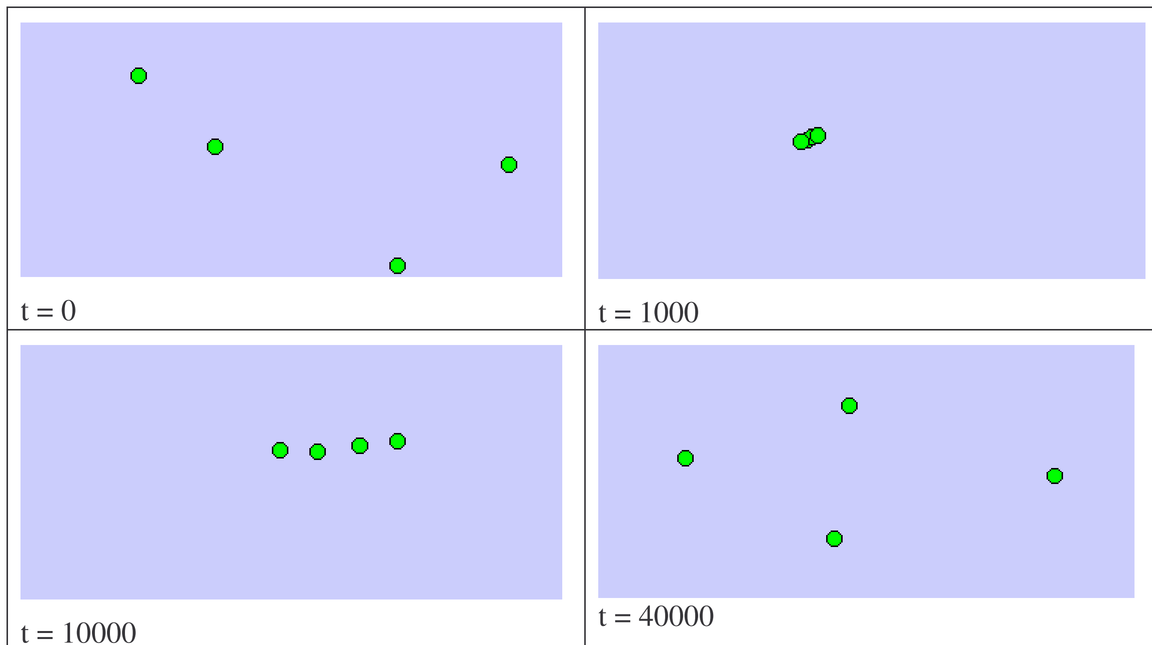
Zvýšim parameter t o 1.

Krok 6

Ak $t < t_{max}$, tak pokračujem krokom 2.

Ukážky priebehu algoritmu

Pre $\lambda_i = 30$, $\lambda_f = 0.01$, $e_i = 0.3$, $e_f = 0.05$ a $t_{max} = 40000$ sú na Obr. 12.



Obr. 12: Ukážka priebehu procesu Neural Gas

Koordinované rozhodovanie

Tento nemecký tím riešil problém, ako zabezpečiť tímové správanie bez explicitnej komunikácie v tíme. Tento problém sa riešil na základe takého prístupu, že sa vyberie vhodná tímová akcia. Výber tejto akcie je podriadený hernej situácii na ihrisku. Keďže tímová akcia pozostáva z individuálnych akcií jednotlivých hráčov, tak výber tímovej akcie koordinuje aj individuálne akcie hráčov. Následne budú rozpísané rôzne typy akcií a situácií, ktoré tento tím vo svojom hráčovi rozoznával.

Akcie

Akcie zabezpečujú koordinované tímové správanie. Hlavné akcie predstavujú *tímové akcie*, ktoré tvorí malá skupina vopred preddefinovaných akcií. Ako už bolo spomínané, tak tímové akcie pozostávajú z *individuálnych akcií*. Vykonávanie individuálnych akcií je následne plnené na základe zručností jednotlivých hráčov.

Situácie

Ideálne v každom kroku by všetci hráči mali participovať na spoločných tímových akciách a plniť svoje akcie, ktoré z toho vyplývajú. Avšak toto je len modelový prípad. Na zvýšenie pravdepodobnosti, že všetci hráči hrajú tak, ako by mali, sa používa štruktúrovaný a systematický rozhodovací proces. Využívajú sa na to tri typy situácií.

Situácia sveta – predstavuje opis celého sveta (pozície, rýchlosti hráčov a lopty).

Taktická situácia – opisuje situáciu celého tímu na abstraktnej úrovni. Taktické situácie sú tvorené určitou skupinou situácií.

Výsledková situácia – situácia získaná z výsledkov (efektov) tímových akcií. Poskytuje všetky potrebné informácie, ktoré sa použijú na výber vhodnej tímovej akcie. Môže byť tiež chápaná ako situácia, ktorá vznikne predvídaním.

© Hviezdna jedenástka, 2008

Výber akcie

Na ohodnotenie tímovej akcie, ktorá sa má vykonať, sa používajú tzv.: cieľové funkcie. Tieto funkcie ohodnocujú tímové akcie z rôznych hľadísk a priraďujú im hodnoty od 0 po 1, kde 0 znamená, že akcia z daného hľadiska nie je vhodná, 1 značí, že tímová akcia z toho hľadiska je najlepšou voľbou. Na základe výsledkov týchto funkcií sa vyberie tá, ktorá dostala najväčšie ohodnotenie a tá sa aj vykoná.

Záver

V budúcnosti by tento tím chcel zdokonaliť tímové (koordinované) rozhodovanie, vylepšiť a doplniť niektoré základné zručnosti. Plánujú sa venovať aj rozvíjaniu robotov strednej veľkosti.

2.3.1.6 FC Portugal

FC Portugal [13] je veľmi úspešný portugalský tím, ktorý od roku 1999 súťaží v 2D futbale kde získal aj niekoľko svetových či európskych prvenstiev. Od roku 2006 sa venuje aj 3D futbalu, kde sa stali takisto majstrami sveta. Úspechy:

Robocup 2D

- ME 2000 – 1. miesto
- MS 2000 – 1. miesto
- MS 2001 – 3. miesto
- MS 2002 – 5. miesto

Robocup 3D

- MS 2006 – 1. miesto

Web stránka je už niekoľko rokov neaktualizovaná, avšak všetky podstatné informácie sa tam nachádzajú, vrátane zdrojových kódov 2D tímov, a základného popisu stratégií, taktiky a schopností hráčov.

Charakteristika

- Zamerali sa na podobnosť s reálnym futbalom, majú prepracovanú stratégiu, taktiku, správanie hráča s loptou, bez lopty, trénera a pod.
- používajú veľmi pružný systém hry - rôzne rozostavenia hráčov podľa hernej situácie, podľa toho, kto má práve loptu, či súper alebo vlastný tím, a pod.

V častiach nižšie sa zameriam na niektoré najdôležitejšie vlastnosti tohto tímu.

Situation Based Strategic Positioning

Situation Based Strategic Positioning (SBSP) je vlastne rozoznávanie aktuálnej situácie, tím FC Portugal zaviedol členenie na 2 základné typy:

- Strategická situácia – nie je to žiadna kritická situácia, hráč nemusí akútne vykonať žiadnu činnosť, len držať pozíciu, prípadne hľadať lepšiu
- aktívna situácia – práca s loptou, prípadne získanie lopty, ak je nablízku, prípadne nejaká aktívna činnosť

Napríklad, ak hráč nemá loptu, a nemá ani šancu ju v najbližšej dobe získať, nehrozí žiadne momentálne riziko, situácia je strategická a SBSP zabezpečí, aby držal pozíciu, alebo sa presunul na také miesto na ihrisku, ktoré maximalizuje jeho užitočnosť pre tím.

V prípade, že má loptu, alebo má možnosť sa v najbližšej dobe dostať k lopte, použije svoje schopnosti na to, aby získal loptu a dopravil ju do súperovej brány. SBSP zároveň odhaduje aj rozhodnutia svojich spoluhráčov.

Dynamic Positioning and Role Exchange

DPRE je schopnosť hráčov si v prípade potreby vymeniť role v tíme, ak to povedie k väčšej užitočnosti. Napríklad ak sa hráč príliš vzdialil od svojej strategickej pozície, a nablízko je ďalší podobný hráč, je zbytočné, aby obidvaja plytvali energiou a vracali sa ďaleko na svoje miesta, keď si ich môžu vymeniť a skrátiť tak dráhu a čas presunu, a tým pádom ušetriť energiu

Marking techniques, predvídanie a rušenie prihrávok

Ďalšou veľkou výhodou oproti súperom je tzv. Marking techniques, tj. predvídanie prihrávok a následné snaženie sa o získanie lopty. V momente prihrávky si hráč vypočíta dráhu lopty, zistí svoju pozíciu a ak má šancu vzhľadom na svoju rýchlosť dostať sa na priamku pohybu lopty ešte pred tým, ako do daného bodu dôjde lopta, tak sa pohne vypočítaným smerom a pokúsi sa prihrávku prerušiť.

Jedným z prvých tímov, ktoré prišli s taktikou rušenia prihrávok, je práve tím FC Portugal, kedy im táto stratégia pomohla suverénnym spôsobom vyhrať majstrovstvá sveta, pričom neinkasovali ani jediný gól – dokázali totiž včas predvídať a teda aj prerušiť prihrávky súpera, a tým pádom zničiť všetky jeho akcie už v zárodku. Dnes už má túto stratégiu každý dobrý tím, a väčšinou ide vždy o to isté – vypočítanie dráhy lopty a následný pohyb hráča tak, aby ju bol schopný prerušiť.

V nasledujúcej časti bude podrobnejší opis tejto techniky, ktorú použil tím FC Portugal, a taktiež porovnanie s ďalším prístupom k predvídaniu a rušeniu prihrávok od tímu CMUUnited-97 [14].

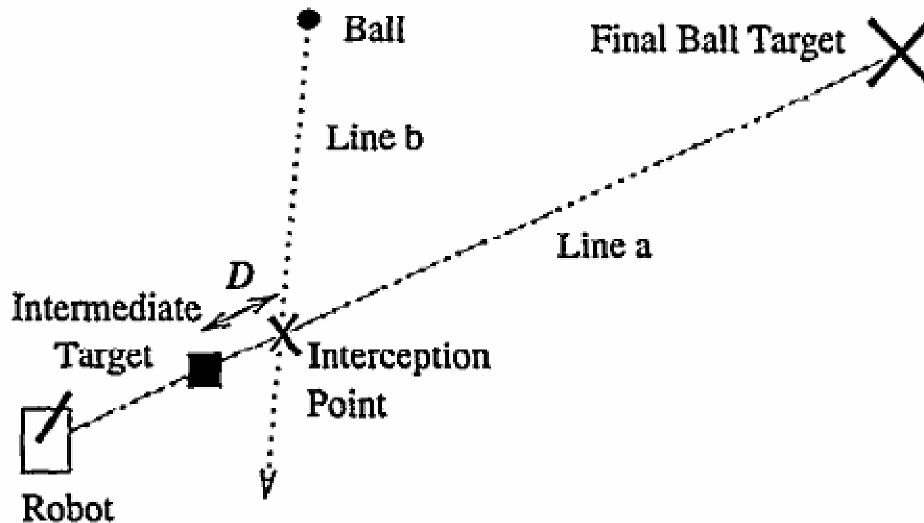
Stratégia tímu FC Portugal

Stratégia tohto tímu je veľmi jednoduchá a zároveň efektívna. Snažia sa vypočítať dráhu lopty, následne zistiť hráča, ktorý je k nej najbližšie, a ak má šancu ju prerušiť, počká na vhodný okamih a potom vyrazí dopredu. Tu je presnejší algoritmus, ako to funguje:

1. ak vedie čiara prihrávky vedľa miesta, ktoré je blízko strategickej pozície, ktorú drží môj hráč, tak sa zapne mód prerušenia prihrávky
2. v ňom sa vypočíta optimálne - najvhodnejšie miesto prerušenia
3. vypočíta sa, aký bude čas cesty lopty k danému miestu – T1, ďalej koľko nášmu hráčovi bude trvať dostať sa na to miesto – T2. Ak T2 < T1 (tj. hráč má šancu stihnúť to), odovzdá sa pokyn hráčovi, ktorý je najbližšie, aby sa tam v správny moment dostavil a prerušil prihrávku.

Stratégia tímu CMUnited-97

Tu je riešené prerušenie prihrávky podobne, len s miernou úpravou – nevypočítava sa iba bezmyšlienkovité prerušenie lopty, ako pri FC Portugal, ale sa počíta aj s tým, že agent by mal prihrávku prerušiť vtedy, ak chce loptu niekam dostať – do nejakej cieľovej pozície. Ľahšie to bude pochopiť z Obr. 13:



Obr. 13: Rušenie prihrávok tímu CMU United

Algoritmus prerušenia je nasledovný:

1. robot vypočíta dráhu lopty – priamka `b`
2. robot vypočíta dráhu priamky, ktorá vedie medzi ním a cieľom cesty – priamka `a`
3. následne vypočíta ich preťatie, a čas potrebný k presunutiu sa z aktuálnej pozície do bodu prerušenia – čas `T`
4. následne si vypočíta, kedy sa lopta dostane do bodu prerušenia, a dovtedy udržiava svoju pozíciu od bodu preťatia konštantnú – t.j. buď stojí na mieste, alebo cúva, alebo ide vpred, aby tá vzdialenosť bola stále približne rovnaká
5. keď nastane čas `T`, vtedy vyštartuje vpred na miesto preťatia a preruší prihrávku
6. keďže má v momente prerušenia aj chcený smer (k vopred určenému cieľu), hneď bez rozmýšľania či dodatočných výpočtov môže aj kopnúť loptu ku cieľu, prihrať loptu cieľu, či driblovať s loptou smerom do cieľovej pozície bez zmeny smeru - je to teda celkom rýchle prevedenie, v momente prerušenia prihrávky má na súpera určitý náskok, keďže smer aj rýchlosť už má určenú a danú a stačí ju len dodržiavať

Stratégia

Tím FC Portugal dôsledne dodržuje stratégiu hry – podobne ako "živí" futbalisti. Dodržujú sa formácie, pričom tieto sa môžu meniť variabilne počas hry v závislosti na stave hry a štýle hry protivníka. Inak sa hrá keď tím prehráva, a inak keď vysoko vyhráva.

Sú implementované 3 základné stratégie správania pre každého hráča:

- strategické správanie – hráč nie je v aktívnej situácii, drží či upravuje pozíciu
- správanie pri držaní lopty – hráč sa rozhoduje ako a komu prihrať, ako vystreliť, držať loptu, driblovať, čo s loptou robiť v danej situácii
- správanie vedúce k získaniu lopty – prerušovanie prihrávok, beh ku lopte, ...(napr. spomínané marking techniques)

2.3.2 Riešenia pre server s hráčmi typu "Humanoid"

2.3.2.1 Tím NimbRo

Tím Nimbro [15] pôsobí na Univerzite vo Freiburgu a venuje sa lige s humanoidnými robotmi.

Základné charakteristiky hráčov tohto tímu sú:

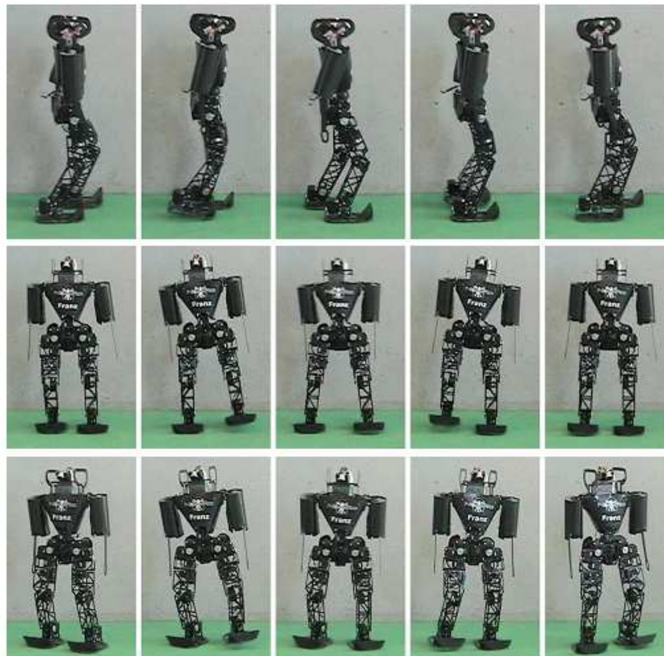
- hierarchia delená na štyri časti – samotný kĺb, určitá časť robota, celý robot, a tím
- pohyb sa robí vo vrstvách:
  - na najnižšej vrstve sa neustále monitorujú ciele, aktuálna pozícia, rýchlosť, ...
  - na ďalšej vrstve sa generujú cieľové pozície pre jednotlivé kĺby tela – s frekvenciou 83.3kHz, pričom sa dáva pozor, aby zmeny uhlov neboli príliš veľké, t.j. aby boli plynulé

Chôdza

Na samotné kráčanie sa používa prenášanie váhy z jednej nohy na druhú, skracovanie (ohýbanie) nohy, ktorá nie je nutná na podopieranie, a následne samotný pohyb nohy v danom smere. Toto všetko sa robí vo frekvencii 3.5Hz, t.j. skoro 4x za sekundu. Na stabilizáciu sa používajú gyroskopy v robotoch.

Počas chôdze je možné kedykoľvek zmeniť smer, čo je výhodné napr. kvôli malým odchýlkam a pod., t.j. celkový vektor smeru pohybu (vx, vy, vo) je možné "za jazdy" meniť.

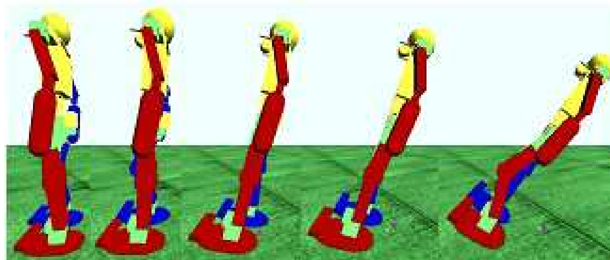
Maximálna rýchlosť je pomerne vysoká, 25cm/s, avšak logicky pri priblížení sa ku prekážke, či lopte klesá. Na Obr. 14 sú zobrazené tri druhy pohybu : 1. priamy pohyb vpred, 2. pohyb vpred kývaním a prenášaním ťažiska z ľavej na pravú nohu a naopak, 3. otáčanie na mieste.



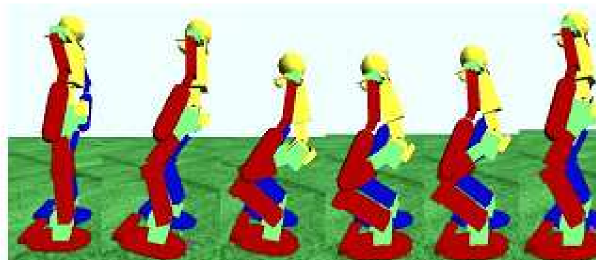
Obr. 14: Tri druhy pohybu

Prevencia pádu, a vstávanie zo zeme

Keďže kontakt medzi robotmi je úplne bežná a nevyhnutná vec, je nutné nejakým spôsobom eliminovať pády – keď už nie všetky, aspoň tie, ktoré sa eliminovať dajú. Počas chôdze sa monitoruje aktuálna fáza kroku a cieľová pozícia. V prípade, že odchýlky sú príliš veľké od vopred vypočítaných, je zrejmé, že nejaká sila narušila pohyb. Vtedy sa zapne stabilizačný mód, ktorý spomalí chôdzu, a zároveň sa snaží čo najviac pokrčiť kolená. Na obrázku nižšie je ukázaný rozdiel medzi padaním, keď je stabilizačný mód vypnutý (Obr. 15) a zapnutý (Obr. 16).



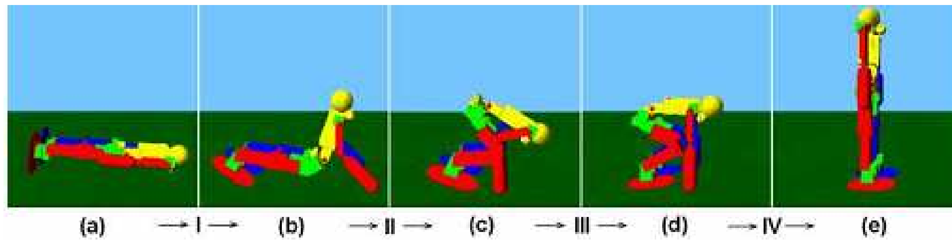
Obr. 15: Padanie s vypnutým stabilizačným módom



Obr. 16: Padanie so zapnutým stabilizačným módom

Ak je však náraz príliš silný, pádu sa vyhnúť nedá. Keď teda robot leží na zemi, musí aj nejakým spôsobom vstať. Mechanizmus vstávania je nasledovný. V prípade pádu robot prostredníctvom senzorov zistí, či leží na bruchu alebo na chrbte a podľa toho aplikuje vstávajúci mechanizmus. Postup vstávania z ľahu na bruchu je nasledovný (Obr. 17):

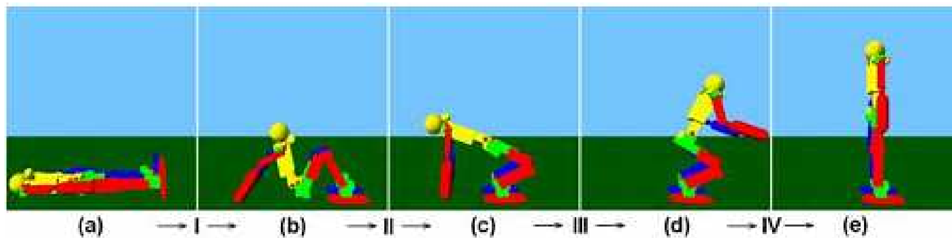
1. robot leží na zemi
2. nasleduje zdvihnutie trupu a umiestnenie predlaktí popod plecia
3. následne začne dvíhanie trupu prostredníctvom ohýbania kolenných kĺbov a lakťových kĺbov v správnom smere (viď obrázok)
4. ako sa robot skrčuje, a vyrovnáva ruky, postupne mení polohu a ťažisko, až “dopadne” na plochu nôh – chodidlo
5. a nasleduje vyrovnanie tela



Obr. 17: Mechanizmus vstávania z ľahu na bruchu

Postup vstávania z ľahu na chrbte (Obr. 18):

1. robot leží na zemi
2. zdvihnutie trupu hore a následne posúvanie rúk dozadu, aby sa dosiahla stabilná poloha sedu
3. vyrovnávanie panvového kĺbu a následný vznik "mostíka"
4. presunutie ťažiska švihom do pokrčeného stoja
5. vyrovnanie



Obr. 18: Mechanizmus vstávania z ľahu na chrbte

2.3.2.2 Tím Zigorat

Tím Zigorat [16] predstavuje výskumnú skupinu, ktorá sa venuje oblasti robotiky a umelej inteligencie. Jej hlavným cieľom je pomôcť k spolupráci medzi univerzitami. Z tohto dôvodu sa snažia sprístupňovať výsledky svojej práce, hlavne zdrojové kódy a tutoriály, aj iným tímom.

Analýze hráča tohto tímu sme sa venovali podrobnejšie, pretože je vysoká pravdepodobnosť, že sa rozhodneme použiť časť ich zdrojových kódov a tým vlastne pokračovať v ich práci.

Zdrojové kódy základného hráča tímu Zigorat sú vytvorené za účelom šetrenia času pri programovaní základných funkcií hráča. Kód je veľmi dobre zdokumentovaný. Nachádza sa v ňom krátka funkcia na demonštráciu otáčania a chôdze hráča. Podporuje vygenerovanie dokumentácie v doxygen. Niektoré `.cpp` súbory obsahujú metódu `main`, pomocou ktorej si môžeme overiť funkčnosť metód vytvorených inštancií tried.

Stručná analýza súborov (súbor – charakteristika):

- `ActHandler.h` – posielanie správ a príkazov na server
- `Agent.h` – modul obsahujúci nízkoúrovňové zručnosti agenta. Obsahuje príklad chôdze
- `BasicAgent.h` – rodičovská trieda modulu Agent, obsahuje základné metódy pre otáčanie kĺbov
- `Connection.h` – komunikácia so serverom, posielanie, prijímanie sprav, manipulácia so sieťou
- `Formation.h` – obsahuje informácie o agentovi a formáciách
- `Geometry.h` – obsahuje triedy pre počítanie geometrie
- `Logger.h` – vypisuje logy hráčov, modelu sveta, správy spojenia a normálne informácie

- `Main.cpp` - hlavný bod programu.
- `Object.h` – obsahuje triedy reprezentujúce objekty v RoboCup-e
- `Parse.h` – Parsovanie serverových sprav
- `SoccerTypes.h` – obsahuje množstvo výpočtových typov a deklarácii príkazov pre SoccerCommand
- `Start.cpp` - skript na vytvorenie a naštartovanie tímu
- `WorldModel.h` – pamäť hráča vonkajšieho sveta

Detailnejšia analýza súborov:

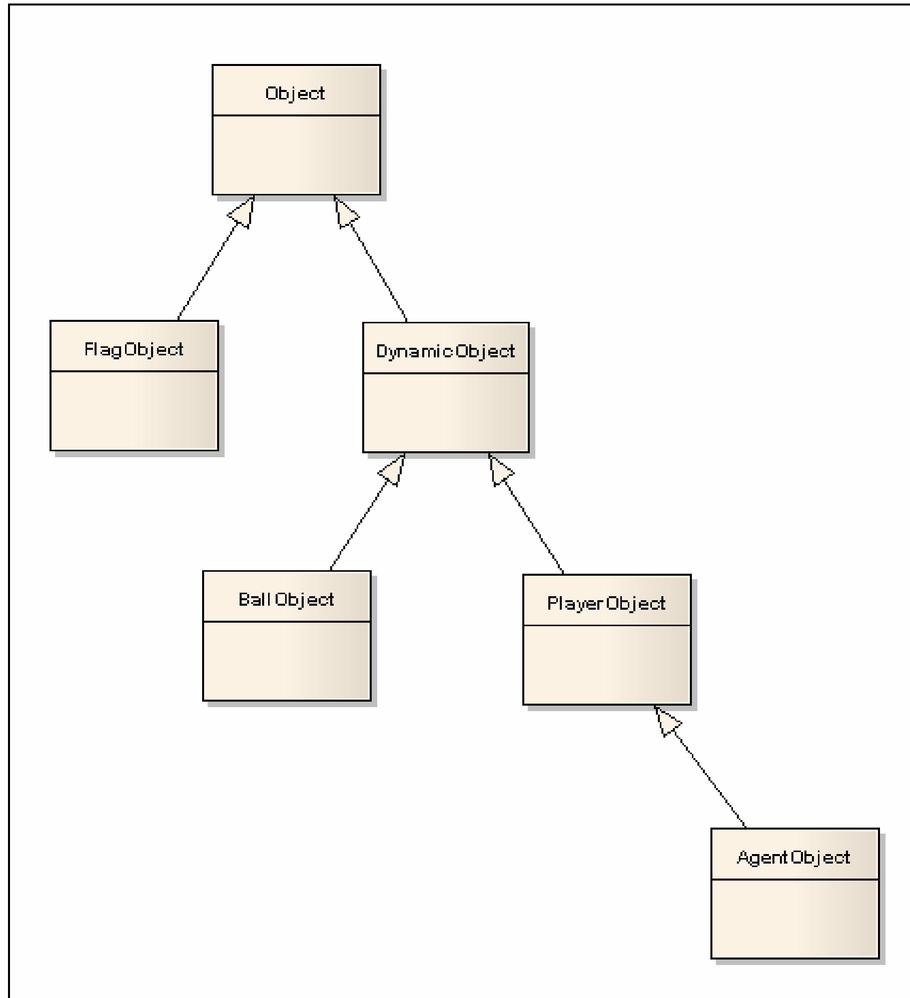
Parse.h, Parse.cpp – sa zaoberá parsovaním správ pre serverové správy. Skladá sa z triedy `Parse`. Obsahuje statické metódy, ktoré sa delia na 2 časti:

- metódy vracajúce prvý výskyt celých a reálnych čísel zo správy
- metódy, ktoré sa posunú na zvolené miesto v správe

Logger.h, Logger.cpp – Obsahuje jednu triedu `Logger`. Zabezpečuje logovanie rôznych druhov správ, ktoré sú ukladané do súborov. Medzi ne patria logovanie údajov o agentovi (`BasicActions.log`), modelu sveta (`WM.log`), normálnych informácii (`Actions.log`) a správami, ktoré sa posielajú na server (`Communications.log`).

Object.h, Object.cpp – Obsahuje hierarchiu tried, ktoré reprezentujú objekty v RoboCup-e. Ich hierarchia je zobrazená na Obr. 19. Každá trieda obsahuje metódy typu get a set.

- Základom je trieda *Object*, ktorá je dostupná pre všetky objekty. Obsahuje metódy pre zistenie a nastavenie pozície objektu a jeho posledne časovo dostupné miesto.
- Trieda *DynamicObject* je zdedená z triedy *Object*. Obsahuje informácie, ktoré sú dostupné pre pohybujúce sa objekty (lopta a hráč).
- Trieda *BallObject* je zdedená z triedy *DynamicObject*. Obsahuje informácie o lopte (hmotnosť a polomer lopty).
- Trieda *PlayerObject* je opäť zdedená z triedy *DynamicObject*. Obsahuje informácie, ktoré sú charakteristické pre hráča. Medzi ne zaraďujeme číslo hráča v tíme, tím, v ktorom sa nachádza a smer, ku ktorému sa hráč pohybuje.
- Trieda *AgentObject* je zdedená z vyššie spomínanej triedy *PlayerObject*. Obsahuje doplňujúce informácie o agentovi, ako objem zostávajúcej energie v batérii, teplota agenta, hmotnosť a maximálnu rýchlosť, ktorou sa môže pohybovať.
- Prvou triedou, ktorá nepatrí do vyššie opisovanej hierarchie tried, je *Joint*. Obsahuje informácie o kĺboch (číslo , resp. ID kĺbu, uhol a rýchlosť pohybu). Jej charakter je čisto informatívny.
- Druhou triedou, ktorá nepatrí do vyššie opisovanej hierarchie tried, je *TSenzor*. Reprezentuje senzor v simulácii. Obsahuje meno a hodnotu senzora.



Obr. 19 Hierarchia tried v súbore Object.h

Agent.h, Agent.cpp – modul, v ktorom sa definujú nízkoúrovňové zručnosti agenta. Obsahuje jednu triedu *Agent*, ktorá je odvodená od triedy `BasicAgent`. Trieda je len konceptom, ktorú je potrebné rozpracovať. Spúšťa hlavnú slučku agenta, v ktorej dostane zo servera správu, rozhodne sa, čo spraví (metóda *makeDesicion*) a pošle rozhodnutie serveru, resp. komunikačnej vrstve. Na ukážku funkčnosti obsahuje ucelenú časť, ktorá umožňuje hráčovi chodiť.

Connection.h, Connection.cpp – Obsahuje triedy pre sieťovú manipuláciu.

- Trieda *TDataPorter* slúži na otváranie, odmietnutie, prijímanie a posielanie dát cez soket.
- Trieda *TConnection* obsahuje rutiny spojení. Spúšťa počúvanie a akceptovanie spojení, posielanie a prijímanie dát, vytvára existujúce pripojenie na serveri. Podporuje komunikáciu cez TCP/IP a UDP/IP.
- Trieda *TRobocupConnection* je zdedená od triedy *TConnection*. Dopĺňa metódy pre posielanie a prijímanie dát na simulačný server.

SoccerTypes.h, SoccerTypes.cpp – Obsahuje množstvo konštánt, definícií, výpočtových typov a deklarácie tried pre príkazy servera:

- Definície (maximálny počet hráčov, pre obe strany, dĺžka správy zo servera, dĺžka say správy, časti tela a iné)

- Konštanty pre neznáme hodnoty
- Výpočtové typy (*ObjectT* definuje všetky objekty poskytované simulačným serverom, *JointT* združuje všetky časti tela (kĺbov) humanoida, *PlayModeT* definuje všetky hracie módy servera, *PlayerT* združuje všetky role hráčov v poli (brankár, ľavý obranca, stredný útočník a pod), *PlayerSetT* obsahuje všetkých obrancov, stredopoliarov, útočníkov a všetkých hráčov, *FormationT* nastavuje rôzne typy formácií (inicializačná, útočná – 443, 334, 343), senzory nôh, typy správ a iné.
- Trieda *SoccerCommand* – obsahuje všetky príkazy, ktoré sa posielajú na server, pričom formát správ záleží na implementácii servera. *SoccerTypes* – používa statické metódy využívajúce vyššie definované konštanty a výpočtové typy. Metódy, ktoré sa vysporiadajú s rôznymi objektmi, hracích módov, strán hracieho poľa, senzory a kĺbov

BasicAgent.cpp - Táto trieda obsahuje v podstate základné “low level skills“ agenta. Začína od tých úplne najjednoduchších, ako je otočenie jednotlivých kĺbov, kolenný, členkový, lakťový, kĺby pliec, atď... Prostredníctvom týchto elementárnych pohybov kĺbov vie vykonať zložitejšie operácie, ako otočenie sa požadovaným smerom či chôdza.

Geometry.cpp - Obsahuje deklarácie konštánt a metód a niekoľko tried:

- VECPOSITION- obsahuje x,y,z súradnice, preťažené matematické operátory - základné metódy na zisťovanie smeru, pozície, vzdialenosti od cieľa, pričom toto všetko môže byť reprezentované v polárnom alebo karteziánskom súradnicovom systéme
- GEOMETRY
- CIRCLE - reprezentuje kruh, ten je definovaný ako jeden objekt typu VECPOSITION (= stred) a polomerom
- SPHERE
- LINE - predstavuje čiaru, pričom tá je definovaná predpisom ay + bx + c = 0, kde a, b, c sú základné konštanty. Ďalej obsahuje niekoľko prislúchajúcich metód na vytvorenie čiary z dvoch zadaných bodov, vypočítanie miesta preťatia, najbližší bod ku čiare, a iné.
- RECTANGLE - je tvorený dvoma bodmi (ľavý horný a pravý dolný), pričom každé body sú reprezentované objektmi VECPOSITION
- PLANE - reprezentuje plochu pomocou vektora a jedného ľubovoľného bodu

V tejto triede sú už podľa názvu združené základné geometrické tvary a funkcie, ako vytvorenie geometrických objektov, hľadanie bodov preťatia objektov, a podobne. V podstate je to analytická geometria.

Formations.cpp - Obsahuje 3 základné triedy

- PLAYERTYPEINFO - obsahuje údaje o hráčovi, t.j. jeho typ, obmedzenia (maximálna a minimálna x-ová súradnica, v ktorých by sa mal pohybovať, atď)
- FORMATIONTYPEINFO - obsahuje všetky informácie o jednej formácii, t.j. typ formácie, predvolené pozície všetkých hráčov a pod.
- FORMATIONS - táto trieda slúži ako kontajner na formácie, sem sa všetky ukladajú

Zo súboru `Formations` je zaujímavou akurát funkcia `getStrategicPosition`, táto funkcia vráti strategickú pozíciu hráča (SPH), kde by asi mal v danej formácii byť. Táto poloha sa vypočíta podľa troch údajov, a to aktuálnej pozície hráča, pozície lopty a tzv. `attraction` atribútu. `Attraction value` je hodnota, ktorú má každý typ hráča priradenú. Pomocou vzorca SPH = aktuálna pozícia + pozícia lopty * `attraction` sa teda vypočíta hodnota SPH, a porovná sa s maximálnou (minimálnou) x-ovou súradnicou hráča. (v rámci ktorých by sa mal pohybovať).

Ak je SPH väčšie ako maximálna povolená hranica pre daný typ hráča (alebo menšia ako minimálna) tak sa hráč "zaradí" na maximálnu (minimálnu) pozíciu.

Restore.cpp

`Restore.cpp` je len pomocná trieda na prácu so zálohou. Máme definovaný názov archívu (zálohy) a miesto (adresár) zálohovania. O ostatné sa už postará táto trieda.

ActHandler.cpp

Obsluhuje príkazy smerom ku serveru. Dokáže posielať príkazy jednotlivo, alebo si ich ukladať do frontu a poslať naraz.

Trieda WorldModel (WorldModel.h)

Táto trieda slúži hráčovi na vytvorenie modelu sveta. Model sveta obsahuje údaje o hráčoch, lopte, značkách a čiarach. Metódy triedy `WorldModel` vedia z týchto údajov získať ešte ďalšie užitočné informácie. Model sveta sa obnovuje podľa údajov posielaných v správach zo servera. Spracovanie správ v triede `WorldModel` má na starosti metóda

`bool updateFromServerMsg(char ** msg).`

Atribúty triedy sa rozdeľujú do týchto skupín:

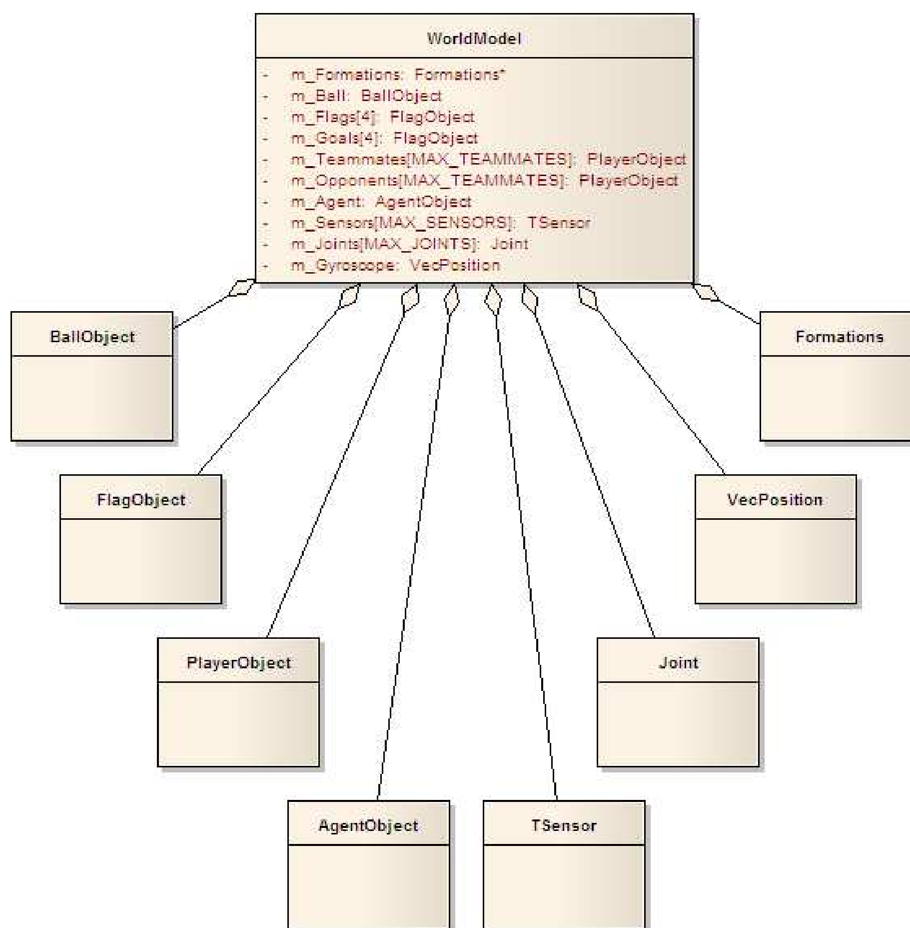
- Informácie o prostredí zo soccer servera
  - `double      m_SimulatorTime;        simulačný čas [sek]`
  - `double      m_GameTime;             čas hry v [sek]`
  - `double      m_SimulatorStep;        čas simulačného kroku [sek]`
  - `double      m_GoalWidth;            šírka brány`
  - `double      m_GoalHeight;           výška brány`
  - `double      m_FieldLength;          dĺžka ihriska`
  - `double      m_FieldWidth;           šírka ihriska`
  - `int         m_AgentNumber;          číslo hráča`
  - `unsigned    m_CurrentCycle;         číslo simulačného kroku`
  - `string      m_TeamName;             názov hráčovho tímu`
- Informácie o stave zápasu
  - `PlayModeT   m_PlayMode;             herný režim`
  - `SideT       m_Side;                 hráčova strana ihriska`
- Informácie o všetkých objektoch na ihrisku
  - `BallObject      m_Ball;                        lopta`
  - `FlagObject      m_Flags[4];                    rohy ihriska`
  - `FlagObject      m_Goals[4];                    žrde brán`
  - `PlayerObject    m_Teammates[MAX_TEAMMATES];    spoluhráči`
  - `PlayerObject    m_Opponents[MAX_TEAMMATES];    protihráči`
  - `AgentObject     m_Agent;                       hráč`
  - `Tsensor         m_Sensors[MAX_SENSORS];        senzory hráča`
  - `Joint           m_Joints[MAX_JOINTS];          kĺby hráča`
  - `VecPosition     m_Gyroscope;                   gyroskop hráča`
  - `Formations *    m_Formations;                  formácia hráča`
- Informácie o vykonaných akciách hráča

o nie je implementované

Metódy triedy sa tiež delia do niekoľkých skupín:

- Metódy na získavanie informácií o objektoch
- Obnovovacie na aktualizáciu modelu sveta podľa údajov zo senzorov
- Predikčné na predvídanie budúcich stavov podľa zozbieraných údajov
- Vysoko-úrovňové na vytváranie záverov z údajov v modeli sveta

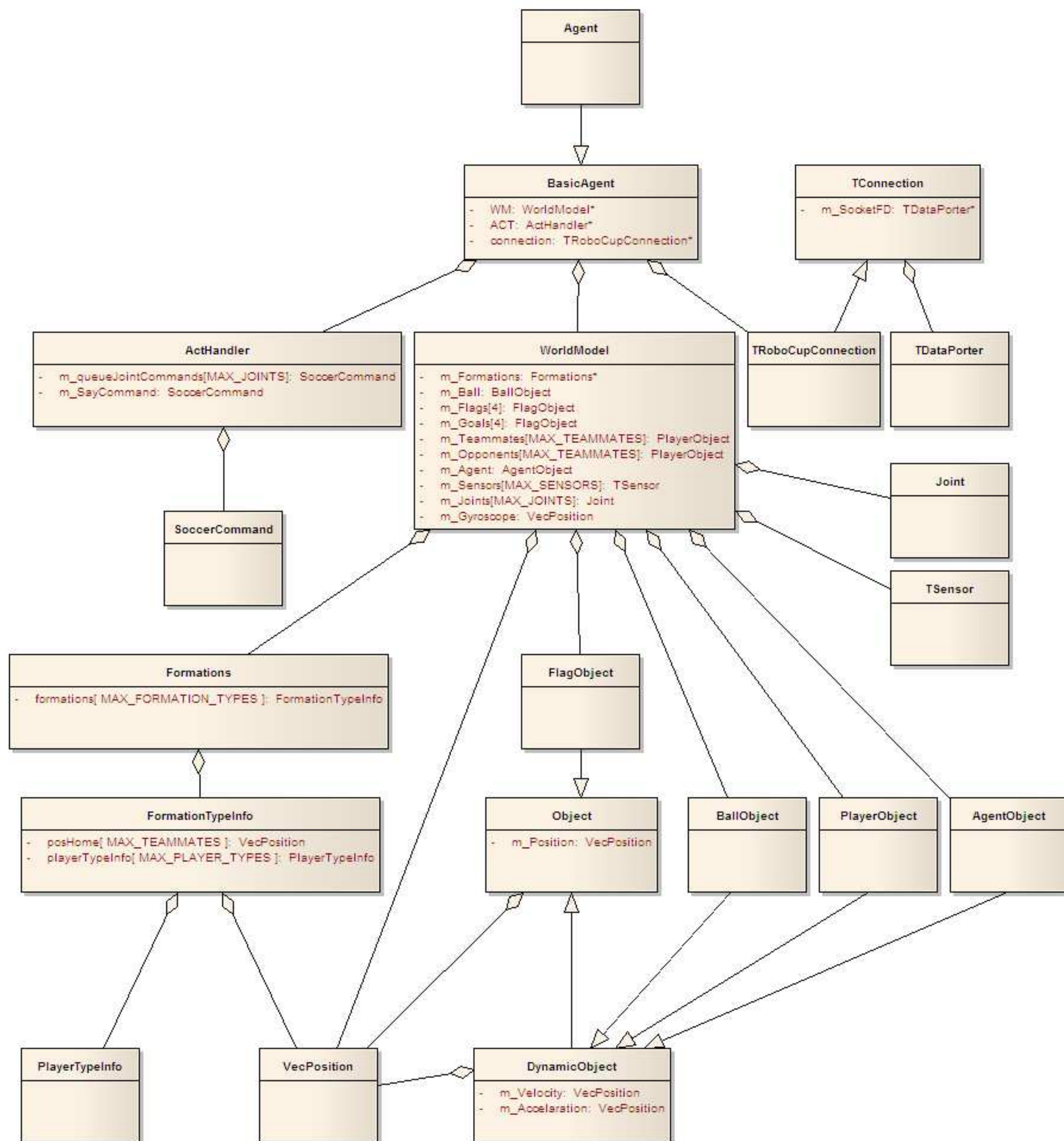
Metódy slúžiace na obnovu údajov v modeli sveta sú implementované v súbore `WorldModelUpdate.cpp`. Na Obr. 20 je znázornený diagram tried použitých v modeli sveta. Model sveta slúži hráčovi predovšetkým na získavanie aktuálnych informácií.



Obr. 20 Diagram tried

Architektúra hráča

Na Obr. 21 je znázornená architektúra hráča tímu Zigorat



Obr. 21 Architektúra hráča tímu Zigorat

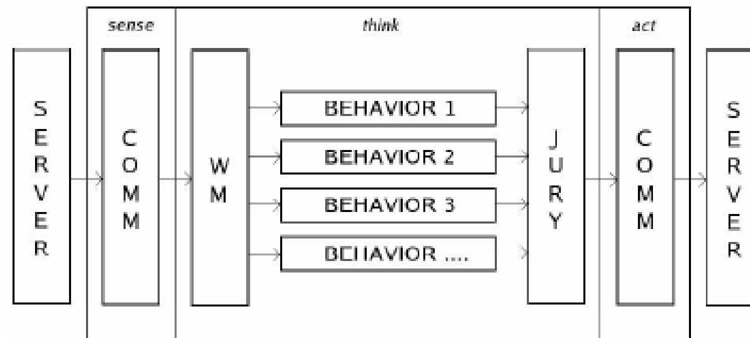
2.3.2.3 Tím Little Green Bats

Úvod

Tento holandský tím [17] 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 Brémach, 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.

Architektúra hráča

Tento hráč pozostáva z troch vrstiev a komunikácie so serverom. Architektúra tohto hráča je znázornená na Obr. 22.



Obr. 22 Architektúra hráča tímu Little Green Bats

Model sveta

Predstavuje vrstvu označenú ako *WM (World Model)*. Táto vrstva obsahuje všetky informácie, ktoré hráč dostáva zo servera. Teda nachádzajú sa tu pozície všetkých hráčov a lopty.

Správanie

Je to vrstva označená ako BEHAVIORS. Správanie predstavuje neusporiadanú skupinu objektov správania. Nezávislé správanie nemôže komunikovať priamo s iným nezávislým správaním. Môže komunikovať iba s modelom sveta, od ktorého dostáva informácie o hráčoch a o lopte, alebo môže komunikovať s ďalšou vrstvou, ktorej odovzdáva svoje dáta. Dáta, ktoré posiela nasledujúcej vrstve, pozostávajú z aktuálnej pozície, pozície do ktorej sa hráč chce dostať a hodnoty, ktorá vyjadruje vhodnosť dostania sa do tejto pozície. Hodnota vyjadrujúca vhodnosť dostania sa do danej pozície je určená na základe normálneho rozdelenia alebo rozdelenia Monte Carlo. Priradené hodnoty sú z intervalu <-1,1>. Hodnota nad 0 znamená, že do danej pozície je výhodné sa dostať, takže sa dané správanie má vykonať, 0 znamená, že nie je dobré sa dostať do tejto pozície, a teda sa dané správanie nemá vykonať a hodnota pod 0 znamená, že sa má vykonať presný opak tohto správania. Tu sa nevyužívajú vektory ale iba pozície, a preto má ďalšia vrstva viac práce.

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.

Typy správaní

Hráč je schopný používať niekoľko zadefinovaných správaní. Je tiež možné správania kombinovať a vykonať najlepšiu akciu podľa situácie. Hráč využíva nasledovné správania:

- *ballTowardsGoal* **–** snaha dostať hráča s loptou blízko súperovej brány a následne vystreliť na bránu.
- *stayInField* **–** sa stará 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

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 vykonať. Budú sa snažiť vylepšiť ešte aj komunikáciu so serverom.

2.4 Zhodnotenie a výber vhodného typu servera

V rámci analýzy sme sa venovali dvom typom serverov – s hráčmi typu „Sphere“ a s hráčmi typu „Humanoid“. Pre obidva typy hráčov sme analyzovali niekoľko tímov, ktoré sa venujú vývojom hráčov pre tieto typy serverov.

Výhodou serveru s hráčmi typu „Sphere“ je, že existuje pomerne veľa tímov, ktoré sa tomuto typu hráča venujú. Taktiež je k dispozícii väčšie množstvo informácií a dostupných zdrojových kódov. Na našej fakulte boli tri tímy, ktoré sa venovali tomuto typu hráčov a implementovali hráčom základné schopnosti. Zaujímavé by bolo pokračovať v práci vybraného tímu a implementovať niektoré vyššie schopnosti hráčov.

Server s hráčmi typu „Humanoid“ je pomerne „mladý“, neexistuje k nemu dokonca ani oficiálna dokumentácia. V porovnaní s predchádzajúcim typom serveru, sa tomuto novému typu zatiaľ venuje veľmi málo tímov. Vývoj tohto typu hráčov je takpovediac v začiatkoch a vyžaduje si začínať úplne od základov, od základných schopností hráča.

V našom projekte sme sa rozhodli venovať sa vývoju hráča pre server verzie 0.5.6 s hráčmi typu „Humanoid“, aj napriek tomu, že máme k dispozícii veľmi málo informácii či už o samotnom serveri alebo o postupe riešenia iných tímov. Ďalším dôvodom je, že po vydaní tohto nového typu servera sa už pravdepodobne nebudú vyvíjať nové verzie toho staršieho typu, takže ak by sme sa venovali hráčom typu „Sphere“, naša práca by mohla byť zbytočná pre použitie v budúcnosti.

© Hviezdna jedenástka, 2008

3. Špecifikácia požiadaviek

Výsledkom práce tohto tímového projektu by malo byť vytvorenie humanoidného hráča 3D RoboCup-u. Tento hráč bude komunikovať so serverom, a preto je potrebné vytvoriť ako základ hráča dobrú komunikáciu.

Táto komunikácia by mala byť v prvom rade stabilná, to znamená, že by sa spojenie medzi hráčom a serverom nemalo prerušovať, a taktiež by sa nemala strácať komunikácia po sieti. Taktiež by bolo dobré, aby komunikácia bola čo možno najrýchlejšia, aby hráč nemusel zbytočne dlho čakať na odpoveď od servera. Čo sa týka prijímaných dát, tieto dáta by mali byť vhodne uložené, aby bola s nimi jednoduchá manipulácia a aby sa dali aj jednoducho aktualizovať. Keďže server neposiela presné dáta, tak by bolo dobré, keby sa niečo staralo aj o odstraňovanie týchto nepresností.

Keďže hranie tohto robotického futbalu ovplyvňuje veľa faktorov a nie je čas vyskúšať všetky možné kombinácie, ktoré pripadajú do úvahy, hráč by mal mať implementovaný určitý model správania. Toto správanie by síce nemuselo byť ideálne t. j. hráč sa nezachová v danej situácii najlepšie ako sa dá, ale zachová sa dostatočne dobre vzhľadom na rýchlosť, ktorou sa rozhoduje urobiť určitú akciu. Pravdaže by bolo výhodné, aby sa hráč rozhodoval podľa svojich možností čo najlepšie a v najkratšom čase. Na základe rozhodnutia o vykonaní určitej akcie je ešte potrebné túto akciu vykonať. Preto by hráč mal mať implementované určité zručnosti, pomocou ktorých by tieto akcie vedel vykonávať.

Náš hráč by mal v prvom rade vedieť chodiť tak, že by pritom udržiaval aj rovnováhu a nepadal by, mal by sa vedieť otočiť na mieste alebo počas chôdze by mal vedieť ísť do oblúkov.

Poslednou vecou, ktorú by mal hráč zvládnuť, by bolo kopnutie do lopty bez straty stability a následného pádu na zem. Nad rámec požiadaviek by bola implementácia vstávania hráča. Keďže tento projekt bude ďalej rozširovaný, musí byť dostatočne prehľadne implementovaný, okomentovaný a zdokumentovaný.

3.1 Požiadavky na komunikáciu

V požiadavkách na komunikáciu odznelo, že by táto komunikácia mala byť stabilná, čo nepriamo hovorí o tom, že na komunikáciu by sa mal použiť TCP protokol. Tým sa odstráni aj problém so strácaním komunikácie po sieti. Nevýhodou tohto typu komunikácie je, že existujú aj rýchlejšie typy spojení. Príkladom rýchlejšieho spojenia je spojenie cez UDP protokol, avšak toto spojenie nie je natoľko spoľahlivé ako TCP. Otázne je, či je možné dovoliť si strácať dáta na úkor rýchlosti. Preto je potrebné zabezpečiť komunikáciu pomocou TCP protokolu. Je možné implementovať aj komunikáciu pomocou UDP protokolu.

3.2 Požiadavky na prácu s dátami

Dáta majú byť vhodne uložené, aby bola s nimi jednoduchá manipulácia. Taktiež by bolo dobré, keby sa odstraňovali nepresnosti v dátach. Na to by však bolo potrebné pamätať si aj predchádzajúce dáta, ktoré server posielal. Potom by sa dal vyrobiť filter, ktorý by dokázal tieto nepresnosti aspoň sčasti odstraňovať. Pamätať si predchádzajúce dáta by bolo vhodné aj na to, aby sa robila určitá predikcia dát pre rozhodovanie.

Vzhľadom na rozsiahlosť tejto práce bude postačovať, ak budú dáta len vhodne uložené. Ako doplnok sa môže vytvoriť história dát, na základe ktorej je potom možné dorobiť presnosť dát a tiež predikciu.

3.3 Požiadavky na správanie

Je potrebné, aby sa hráč vedel dobre rozhodovať v čo najkratšom čase. Keďže futbal je tímová hra, hráč by mal mať implementované určité tímové správanie. Mal by samozrejme aj svoje vlastné správanie, ale to by bolo podriadené tímovému, aby hráč, ktorý je súčasť tímu do tohto tímu zapadal. Vzhľadom na rozsiahlosť projektu nie je potrebné správanie implementovať, ale bude to potrebné pri ďalšom rozširovaní hráča.

Veľmi dôležitou požiadavkou pri tomto je zabezpečiť, aby činnosť rozhodovania bola nezávislá od určitých základných zručností, napríklad chôdze. Zjednodušene povedané, výpočet súvisiaci s rozhodovaním by nemal hráčovi uberať veľa času na úkor výpočtov súvisiacich s chôdzou a udržiavaním rovnováhy, pretože je zrejmé, že udržanie rovnováhy má pre hráča väčšiu prioritu ako určité strategické správanie. To znamená, že treba zabezpečiť, aby tieto výpočty prebiehali v ideálnom prípade paralelne, napríklad implementáciou s využitím vlákien.

3.4 Požiadavky na zručnosti

Zručnosti hráča je možné rozdeliť do dvoch úrovní :

- Nižšie zručnosti
- Vyššie zručnosti

Samozrejme toto rozdelenie je len veľmi zjednodušené a môže existovať viac úrovní zručností. Pritom by však malo platiť, že zručnosti vyššej úrovne sú vždy „poskladané“ zo zručností vyššej úrovne.

V našom prípade medzi nižšie zručnosti môžme zaradiť jednotlivé časti chôdze. Na úplne najnižšej úrovni je ovládanie jednotlivých kĺbov, toto však zabezpečuje priamo server pomocou efektorov. Potom medzi nižšie zručnosti môžme zaradiť napríklad vykonanie jedného kroku, ktorý pozostáva vlastne z postupného ovládania jednotlivých kĺbov efektormi. Môžme rozoznávať minimálne tri druhy základných krokov:

- Krok pri rozbiehaní
- Krok udržiavací – klasická chôdza
- Krok pri zastavovaní

Hráč by mal dokázať chodiť tak, aby pritom udržiaval aj rovnováhu a napadal by, mal by sa vedieť otočiť namieste alebo počas chôdze by mal vedieť ísť do oblúkov, atď. Všetky tieto zručnosti však pozostávajú z elementárnejších zručností.

K elementárnejším zručnostiam patrí napr. krok. Preto je potrebné aby sa vyrobili takéto „nižšie“ zručnosti, pomocou ktorých sa poskladajú tie „vyššie“ zručnosti, aby bola zabezpečená funkcionalita hráča.

3.5 Požiadavky na architektúru

Celý návrh nadstavby hráča, požadovaných schopností ako aj implementácia by mali zohľadňovať jeho budúce rozširovanie o nové schopnosti nasledujúcimi tímami. Preto medzi požiadavky pri vytváraní hráča bude patriť aj vytvorenie takej architektúry hráča, ktorá umožní

jeho škálovateľnosť. Hráč by sa mal skladať z modulov, ktoré by sa mali dať ľahko rozširovať, prípadne vymieňať za iné a prepájať nové s existujúcimi.

Pri návrhu architektúry musíme brať do úvahy aj moduly, ktoré budú zabezpečovať vyššie schopnosti hráčov, napríklad určité správanie sa v tíme. Napriek tomu, že tieto schopnosti nebudeme implementovať, bolo by dobre navrhnúť architektúru tak, aby zahŕňala „predprípravu“ pre tieto vyššie zručnosti.

3.6 Požiadavky na dokumentáciu

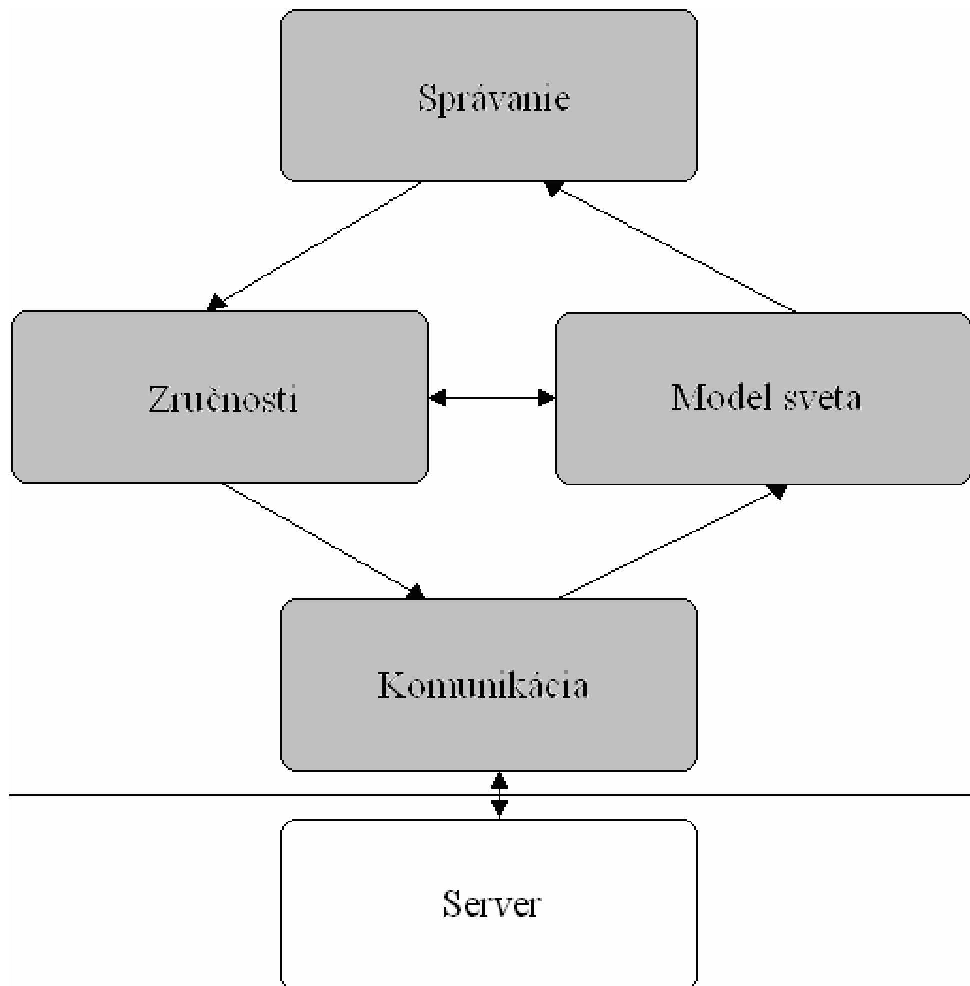
Požiadavky na kvalitu dokumentácie priamo nesúvisia s ostatnými požiadavkami na časti implementácie. Napriek tomu je vhodné ich spomenúť, pretože tento projekt má aj „výskumný“ charakter, nie je zameraný len na implementáciu určitej funkcionality, ale je zameraný aj na výber a zdokumentovanie vhodných algoritmov a prístupov. Z tohto dôvodu je vhodné mať dokumentáciu v prehľadnej forme, aby tímy, ktoré budú pokračovať v našej práci, mali pripravený dobrý základ pre ich prácu.

© Hviezdna jedenástka, 2008

4. Návrh

4.1 Hrubý návrh architektúry

Na základe špecifikácie požiadaviek sme vytvorili architektúru, ktorá pozostáva zo štyroch základných častí: Komunikácia, Modelu sveta, Správania a Zručností. Všetky časti spolu z väzbami sú znázornené na Obr. 23.



Obr. 23 Hrubý návrh architektúry hráča

Komunikácia je časť, ktorej hlavnou úlohou je zabezpečiť obojsmernú komunikáciu medzi hráčom a serverom. Prijaté a spracované údaje posiela modelu sveta. Táto časť komunikuje aj s modelom zručnosti, ktorý posiela údaje späť na server.

Model sveta predstavuje časť, v ktorej sa nachádzajú všetky dostupné informácie pre agenta o pozíciách hráčov a lopty, o ich rýchlostiach apod. Model sveta je prepojený s časťou komunikácia, od ktorej získava všetky potrebné dáta zo servera. Ďalej komunikuje s modulom správanie, nakoľko sa hráč musí rozhodnúť, akú akciu vykoná. Všetky potrebné údaje si preberie z modelu sveta. Ešte sa tu nachádza väzba so zručnosťami. Táto väzba je obojstranná. Väzba od modelu sveta k zručnostiam sa využíva v tom prípade, že na základe správania je už rozhodnuté,

aká akcia sa má vykonať. Potom dôjde k realizácii vybranej akcie, kedy sa tieto dáta využívajú. Väzba od zručností smerom k modelu sveta predstavuje skutočnosť, že potrebujeme pomocou predikcie určiť nejakú situáciu. Tak nebudeme čakať na server, kým nám pošle dáta, ktoré mu boli poslané zo zručností, ale použijeme dáta priamo zo zručností.

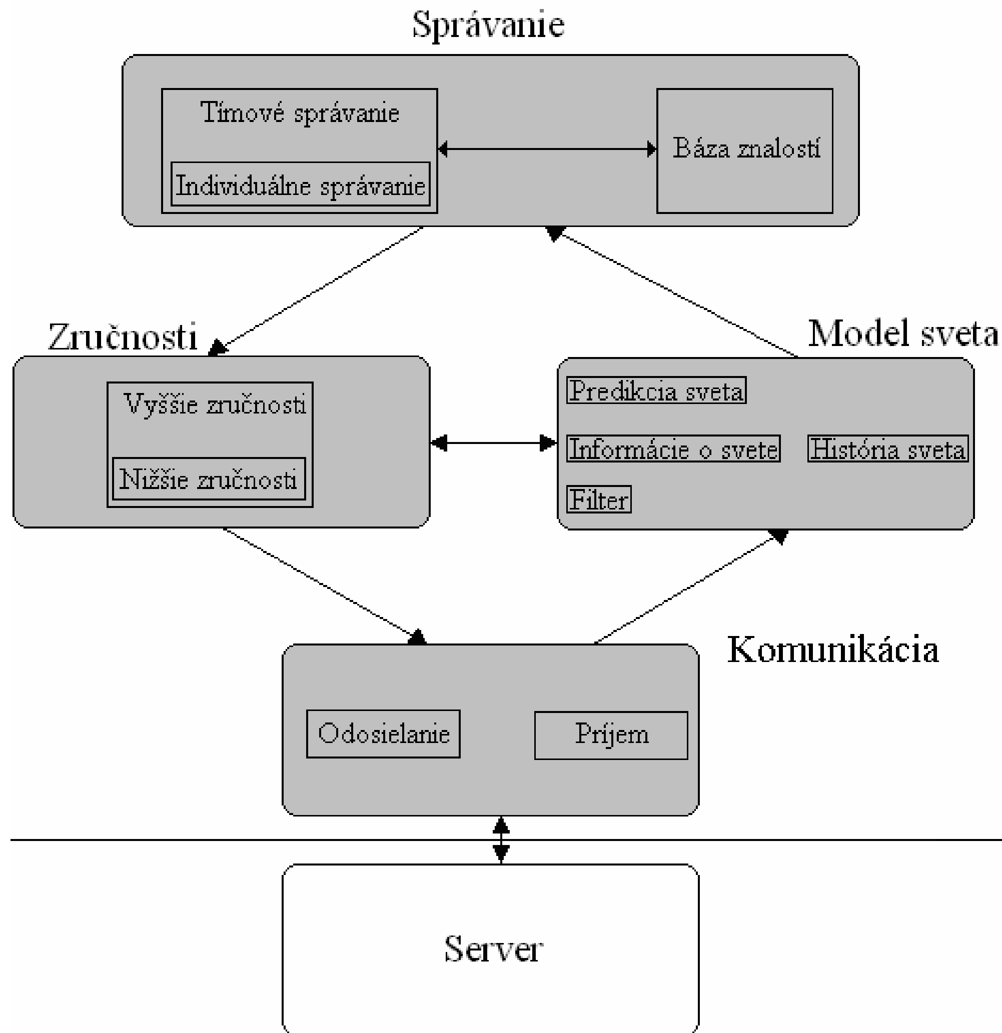
Správanie je časťou, ktorá sa stará o rozhodovanie hráča. Správanie komunikuje s modelom sveta, od ktorého berie informácie, na základe ktorých sa hráč rozhoduje o akciách, ktoré má vykonať. Správanie je ešte prepojené so zručnosťami, ktoré vybranú akciu realizujú.

Zručnosti reprezentujú časť, ktorá sa stará o realizáciu samotných akcií. Táto časť komunikuje so správaním, od ktorého dostáva akcie, ktoré má vykonať. Je prepojená aj s modelom sveta, od ktorého získava údaje, na základe ktorých jednotlivé akcie vykonáva. Ešte sa tu nachádza väzba s komunikáciou, ktorej po vykonaní určitého kroku odovzdá svoje údaje, aby boli odoslané serveru.

Toto bol hrubý návrh architektúry hráča. Avšak špecifikácia umožňuje aj podrobnejší návrh, ktorý bude v tejto časti rozoberaný. Nebol rozoberaný hneď, aby ho čitateľ lepšie pochopil. Na Obr. 24 je znázornená podrobnejšia architektúra hráča.

4.2 Spresnenie hrubého návrhu architektúry hráča

Na Obr. 24 je znázornený návrh architektúry hráča, v ktorom sme spresnili význam jednotlivých modulov.



Obr. 24 Spresnenie hrubého návrhu architektúra hráča.

Časť Komunikácia sa delí na dva moduly. Modul *príjem* zabezpečuje spracovanie údajov zo servera, ktoré sú preposielané do časti Model sveta. Druhým modulom je *Odosielanie*, ktorý plní opačnú funkciu ako modul Príjem. Spracováva príkazy z časti Zručnosti a odosiela ich na server.

Model sveta obsahuje štyri vnorené moduly. Nakoľko server posiela nepresné informácie o dianí na hracej ploche (aplikuje na posielané údaje šum), je potrebné tento šum čiastočne odstrániť, o ktorý sa bude starať modul *Filter*. V minulosti sa osvedčil Kalmanov filter, ktorý vo väčšej miere spresnil údaje. Modul *Informácie o svete* obsahuje aktuálny stav modelu sveta po príchode nových a odfiltrovaných informácii zo strany servera. Predchádzajúci stav modelu sveta sa uloží do modulu *História sveta*. Na základe predchádzajúceho a aktuálneho stavu modelu sveta môžeme predvídať správanie sveta v nasledujúcom takte. Na jeho určenie slúži modul *Predikcia sveta*. Potreba spomínaných modulov v časti Modelu sveta sú potrebné pre rýchlu komunikáciu

© Hviezdna jedenástka, 2008

medzi časťami Správanie a Zručnosti. Dôvodom je predpovedanie nových údajov zo servera (raz za 20ms), na základe ktorých sa môže resp. bude hráč rozhodovať.

Po vytvorení a aktualizovaní údajov v modeli sveta sa hráč rozhodne, aká bude jeho ďalšia aktivita. Jeho rozhodnutie zabezpečuje časť Správanie. Základným modulom je *Báza znalostí*, ktorá obsahuje napr. rozdelenie hráčov vo formáciách. Priamo komunikuje s modulom *Tímové správanie*, ktorý obsahuje modul *Individuálne správanie*. Tímové správanie rozhodne, ktorú akciu vyberie z bázy znalostí a použije pre účely vhodné pre tím. Individuálne správanie vyberie bližšie špecifikovanú činnosť (napr. pokrytie protihráča, presunúť sa na vhodné miesto za účelom zachytenie lopty alebo postavenie sa pred súperovu bránku), ktorú zrealizuje časť Zručnosť.

Vybraná akcia postupuje do časti Zručnosť, ktorá sa stará o realizáciu samotných akcií. Skladá sa z modulu *Nižšie zručnosti*, ktorý je vnoreným modulom *Vyšších Zručnosti*. Do modulu nižšie zručnosti zaraďujeme akcie pohyb jednotlivých kĺbov a premiestni sa na určené miesto (napr. pri rozohrávaní). Druhý modul sa skladá z akcií nižších zručností. Patria sem akcie otáčanie, krok, beh, kopni do lopty, udržiavanie stability a vstávanie zo zeme. Po vykonaní sa odovzdajú informácie do modulu Odosielanie v časti Komunikácia, ktorá pošle údaje serveru. Medzi časom, ktorý je potrebný na ďalší takt servera (20ms) sa využije na komunikáciu medzi časťami Zručnosti a Modelom sveta (s využitím modulu predikcie sveta), čím si hráč pripraví nasledujúce akcie.

© Hviezdna jedenástka, 2008

5. Prototyp

V tejto kapitole sa budeme venovať prototypu vytvorenému v zimnom semestri. Uvedieme, aké sú hlavné ciele prototypovania, opíšeme časti implementované v rámci prototypu a spôsob akým sme testovali ich funkčnosť. Na záver zhrnieme dosiahnuté výsledky

5.1 Ciele prototypovania

Hlavným cieľom prototypu je overenie informácií získaných pri analyzovaní servera a vytvorenie základnej štruktúry hráča. Pri analýze sme vychádzali predovšetkým zo zdrojových kódov servera a iných tímov a nemali sme k dispozícii oficiálnu dokumentáciu k serveru. Preto bolo nutné si tieto predpoklady overiť vytvorením prototypu.

Medzi najdôležitejšie informácie, ktoré sú potrebné pre korektné fungovanie agenta, patrí spôsob, akým sa ovládajú kĺby. V rámci prototypovania sme sa venovali podrobne zisteniu týchto informácií. Vychádzali sme pritom z implementácie agenta tímu Zigorat. Tento tím mal kompletne spravenú komunikáciu a základné funkcie pre ovládanie kĺbov. Použili sme tieto funkcie a pomocou nich sme implementovali vstávanie agenta zo zeme po páde.

V priebehu analýzy riešení iných tímov sme plánovali pri implementácii nášho hráča pokračovať v riešení vytvorenom tímom Zigorat (kap. 2.3.2.2). Na základe podrobnejšej analýzy ich zdrojových kódov sme sa rozhodli, že nepoužijeme toto riešenie, ale začneme implementovať hráča od základov. Dôvodom pre toto rozhodnutie bolo zistenie, že v ich implementácii boli jednotlivé funkčné časti veľmi úzko zviazané a myslíme si, že to znižuje kvalitu vytvoreného riešenia, keďže je potom zdrojový kód neprehľadný a ťažšie rozširovateľný. Ďalším problémom bolo, že zdrojový kód pre hráča Zigorat bol skompilovateľný iba pod Linuxom, čo nám nevyhovuje. Z týchto dôvodov sme v rámci prototypu vytvorili základnú funkcionalitu nášho hráča. Je tu zahrnutá komunikácia so serverom, parser pre prichádzajúce správy a implementácia vlákien.

5.2 Implementácia prototypu

V tejto časti uvedieme, aké vývojové prostredia používame pri implementácii a opíšeme jednotlivé časti prototypu.

5.2.1 Výber vývojového prostredia a štruktúra projektu

Hráč bude implementovaný v jazyku C++. Pre tento jazyk sme sa rozhodli z toho dôvodu, že je v ňom napísaný kód simulačného servera a taktiež všetky analyzované tímy.

Pri výbere vývojového prostredia sme vychádzali z definovaných požiadaviek na vytváraného hráča. Jedna z hlavných požiadaviek bola prenositeľnosť systému. V našom tíme sú členovia, ktorí preferujú prácu pod operačným systém Windows, iní pod Linuxom, dokonca máme člena v tíme, ktorý pracuje pod Mac OS. Samozrejme k efektivite práce prispieva, ak každý pracuje pod operačným systémom a vývojovým prostredím, na ktoré je zvyknutý.

Na základe týchto požiadaviek sme sa rozhodli dodržiavať nasledovné body:

- Oddeliť zdrojové kódy od súborov súvisiacich s projektom v konkrétnom vývojovom prostredí. Máme vytvorený adresár, v ktorom sú všetky zdrojové kódy a zvlášť adresáre s projektmi pre jednotlivé vývojové prostredia.

- Udržiavať projekty vo všetkých vývojových prostrediach – každý člen tímu bude zodpovedný za udržiavanie projektu v danom vývojovom prostredí, v ktorom pracuje. Jedná sa predovšetkým o pridávanie nových zdrojových súborov a knižníc do projektu
- Písať prenositeľný kód – vytváraný kód by mal používať iba štandardné knižnice a treba zaistiť, aby bol skompilovateľný vo všetkých uvedených operačných systémoch.

Použité vývojové prostredia sú:

- Visual Studio 6.0 (Windows)
- Visual Studio 2005 (Windows)
- Visual Studio 2008 (Windows)
- Eclipse (Linux)
- Xcode (Mac OS)

5.2.2 Architektúra hráča

Na (Obr. 25) je znázornený logický model architektúry hráča. Nasleduje stručný opis jednotlivých tried.

WorldModel

Táto trieda uchováva informácie o modely sveta. Model sveta je pasívna trieda, v ktorej sa uchovávajú informácie o polohe objektov vo svete (v našom prípade na ihrisku). Model sveta obsahuje informácie o hracej lopte, tímových hráčoch, oponentoch, o pozícií vlajok a bránok na ihrisku. V triede sa nachádzajú informácie o agentovi. Medzi tieto informácie patria nastavenia kĺbov, vektor gyroskopu z trupu agenta a sily a vektory tlakových perceptorov na chodidlách.

Communication

Táto trieda sa stará o komunikáciu so serverom. Trieda zahrňuje funkcie na prijímanie a odosielanie správ na server. Odosielanie a prijímanie správ prebieha pomocou soketov pričom sokety sa vytvárajú pomocou tried TCPSocket a UDPSocket podľa toho či chceme vytvoriť TCP alebo UDP spojenie so serverom. V triede Communication sa nachádzajú odkazy na triedy Parser a Wrapper, ktoré slúžia na rozparsovanie prijatej správy respektíve zabalenie odchádzajúcej správy. Ďalej trieda obsahuje odkaz na triedu Logger, ktorá slúži na logovanie správ do súboru pre neskoršiu analýzu.

TPCSocket

Trieda slúži na komunikáciu pomocou TCP spojenia. Trieda sa stará o vytvorenie a inicializáciu spojenia, o samotnú komunikáciu a o korektné ukončenie spojenia.

UDPSocket

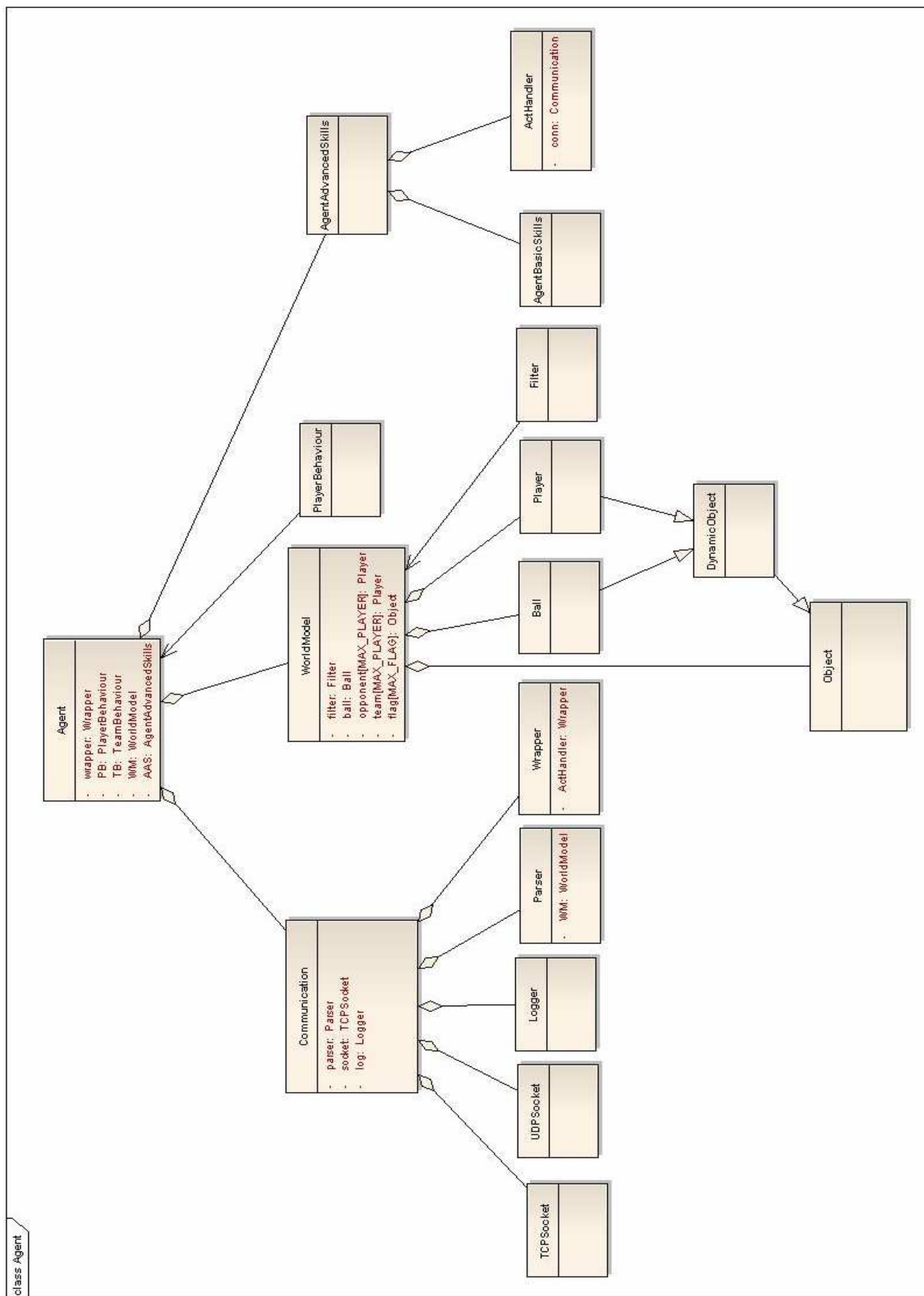
Trieda je analogická k triede TCPSocket len s tým rozdielom, že namiesto TCP spojenia vytvára UDP spojenie.

Parser

Trieda parser slúži na rozloženie prijatej správy na jednotlivé zložky a následne uloženie do štruktúry, s ktorou pracuje hráč.

Wrapper

Táto trieda slúži na skonvertovanie správy, ktorú chceme poslať serveru. Trieda dostane na vstupe správu vo forme s akou pracuje náš agent a na výstupe vznikne správa vo formáte, s ktorým pracuje server.



Obr. 25 Architektúra hráča.

Logger

Táto trieda slúži na vytváranie záznamov pre neskoršiu analýzu. Trieda bude podľa nastavenia zaznamenávať niektoré alebo všetky prichádzajúce a tiež odchádzajúce správy. Tieto záznamy sa budú ukladať do súboru.

Filter

Táto trieda slúži na filtrovanie prijatých správ. Filtrovanie správ je potrebné na odstránenie nechceného šumu z komunikácie.

Agent

Trieda Agent je základná trieda (mozog agenta). Obsahuje hlavný cyklus, v ktorom sa čítajú správy zo servera, aktualizuje model sveta, prebieha rozhodovanie a správy sa posielajú späť na server. Úlohou agenta je rozhodovanie o jeho správaní. Agent obsahuje odkazy na triedu PlayerBehaviour s rôznymi správaniami (útočné, obranné, brankár, ...). Správanie sa realizuje pomocou zručnosti (skills), ktoré sú uložené v triede AgentAdvancedSkills.

PlayerBehaviour

Účelom triedy PlayerBehaviour je realizácia správania sa hráča. Agent bude obsahovať odkazy na viacero správaní a z nich si podľa aktuálnej situácie vyberie jedno správanie.

ActHandler

Trieda ActHandler slúži ako zásobník na správy. Do tejto triedy sa ukladajú správy, ktoré chceme odoslať serveru. Tieto správy sa neodosielajú automaticky ale odošlú sa nakoniec všetky naraz.

AgentBasicSkills

Trieda AgentBasicSkills reprezentuje základné (nižšie) zručnosti hráča. Do tejto kategórie patria zručnosti ako pohyb kĺbmi, či premiestňovanie sa na hracej ploche (beam).

AgentAdvancedSkills

Trieda AgentAdvancedSkills obsahuje vyššie zručnosti hráča. Vyššie zručnosti sa skladajú z nižších, teda trieda AgentAdvancedSkills využíva triedu AgentBasicSkills. Medzi vyššie zručnosti patrí chôdza, otáčanie sa či vstávanie zo zeme.

Object

Od tejto triedy sú odvodené všetky objekty nachádzajúce sa v modely sveta. Trieda Objekt obsahuje pozíciu objektu a čas kedy bola pozícia naposledy aktualizovaná.

DynamicObject

Trieda DynamicObject slúži na uchovávanie informácií o pohybujúcich sa objektov. Táto trieda je odvodená od triedy Object. Triedu Object rozširuje o ďalšie vlastnosti, ktoré sú potrebné pre sledovanie dynamiky pohybu. Trieda obsahuje okrem aktuálnej pozície objektu aj jeho rýchlosť a aktuálne zrýchlenie. V triede sa taktiež nachádza informácia o hmotnosti objektu. Trieda pomocou týchto parametrov dokáže odhadnúť budúcu pozíciu objektu.

Ball

Trieda Ball predstavuje hraciu loptu takže je odvodená od triedy DynamicObject. Triedu DynamicObject rozširuje o parameter radius, ktorý udáva polomer lopty.

© Hviezdna jedenástka, 2008

Player

Trieda Player slúži na uchovávanie informácií o hráčoch na hracej ploche. Trieda je odvodená od triedy DynamicObject keďže každý hráč je dynamický sa pohybujúci objekt a je potrebné sledovať jeho rýchlosť a smer pohybu. Každý hráč má svoje unikátne číslo, obsahuje informácie o aktuálnom stave jeho batérie, o jeho teplote. Každý hráč obsahuje tiež uhol, ktorým smerom je natočený.

Vec3f

Trieda Vec3f je trieda, ktorá predstavuje vektor v 3-rozmernom priestore. Obsahuje 3 čísla s pohyblivou desatinnou čiarkou (x, y, z). Spolu tieto tri čísla tvoria vektor do 3-rozmenrného priestoru. Táto trieda sa využíva ako informácia o pozícií v priestore, ale tiež ako vektor rýchlosti či zrýchlenia.

Vec3fTime

Táto trieda je odvodená od triedy Vec3f a rozširuje ju o ďalšiu položku time, ktorá slúži ako časová pečiatka. Túto triedu používame všade tam kde okrem pozície či rýchlosti objektu potrebujeme vedieť aj čas kedy bola táto pozícia zaznamenaná.

5.2.3 Implementácia vlákien

Pri realizácii projektov v jazyku C pre rôzne platformy vzniká problém s portovaním kódu. Je niekoľko možností ako portovanie vyriešiť. Buď sa kód aplikácie kompletne prepíše pre danú platformu alebo sa použijú špeciálne knižnice podporované medzi migrovanými platformami. Ďalšou možnosťou je spustenie aplikácie v subsystéme (napr. Cygwin).

Rozhodli sme sa pre použitie štandardizovaných knižníc, ktorými budeme riešiť vlákna v programe. Týmto spôsobom bude umožnené ľahšie udržiavanie kódu a odpadne aj nutnosť inštalácie ďalšieho potrebného softvéru. Navyše bude kód hráča, bez akýchkoľvek úprav, skompilovateľný na rôznych platformách. Konkrétne budeme vyvíjať kód pre platformy Linux (distribúcia Ubuntu), Windows (XP, Vista) a Mac OS X (Tiger).

Pre vlákna existuje štandard IEEE POSIX 1003.1c (1995). Implementácia v jazyku C, ktorá dodržiava tento štandard, sa označuje ako POSIX threads alebo Pthreads. Pomocou tejto knižnice je možné implementovať vlákna spomínaným štandardom na rôznych operačných systémoch, ktoré tento štandard oficiálne podporujú:

- A/UX
- AIX
- BSD/OS
- HP-UX
- INTEGRITY
- IRIX
- LynxOS
- Mac OS X
- MINIX
- OpenVMS (nutné zvoliť pri inštalácii POSIX balíček)
- QNX
- RTEMS (POSIX 1003.1-2003 Profile 52)
- Solaris
- OpenSolaris
- UnixWare

- velOSity
- Windows NT (okrem niektorých POSIX vlastností)
- Windows verzie obsahujúce Subsystem for UNIX-based Applications:
  - Windows Server 2003 R2
  - Windows Vista Enterprise a Ultimatellet 1

Systémy, ktoré neoficiálne podporujú POSIX štandard:

- Nucleus RTOS
- FreeBSD
- Linux (väčšina distribúcií)
- NetBSD
- BeOS
- OpenBSD
- SkyOS
- Syllable
- VSTa

Systémy, ktoré neoficiálne podporujú POSIX štandard pod podmienkou inštalovania podpory:

- eCos
- Plan 9 od Bell Labs APE - ANSI/POSIX Environment
- Symbian OS s PIPS
- Windows NT kernel s nainštalovaným balíkom Microsoft SFU 3.5
- Windows 2000 Server alebo Professional so Service Pack 3 a neskorším. Pre podporu POSIX treba aktivovať niektoré voliteľné funkcie Windows NT a Windows 2000 Server.
- Windows XP Professional so Service Pack 1 a neskorším
- Windows Server 2003
- Windows Vista

Okrem toho, že kód s POSIX vláknami je skompilovateľný a spustiteľný na podporovaných operačných systémoch, prináša aj zvýšenie výkonu v porovnaní s procesovým riešením, pri ktorom manažovanie zaberie viac systémových prostriedkov. V Tab. 4 je porovnanie, koľko sekúnd trvá vytvorenie 50 000 procesov resp. vlákien.

Platforma	fork()			pthread_create()		
	real	user	sys	real	user	Sys
AMD 2.4 GHz Opteron (8cpus/node)	41.07	60.08	9.01	0.66	0.19	0.43
IBM 1.9 GHz POWER5 p5-575 (8cpus/node)	64.24	30.78	27.68	1.75	0.69	1.10
IBM 1.5 GHz POWER4 (8cpus/node)	104.05	48.64	47.21	2.01	1.00	1.52
INTEL 2.4 GHz Xeon (2 cpus/node)	54.95	1.54	20.78	1.64	0.67	0.90
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.54	1.07	22.22	2.03	1.26	0.67

Tab. 4: Porovnanie fork() a pthread_create()

Pre systémy Windows existuje POSIX knižnica PthreadsWin32, ktorá implementuje väčšinu POSIX štandardu. V súčasnosti existuje Pthreads-w32 2.8.0 verzia a dá sa stiahnuť zo stránky http://sourceware.org/pthreads-win32/. V programe sa môže použiť dynamická alebo statická knižnica. Knižnica sa použila vo viacerých projektoch vyvíjaných na rôznych platformách.

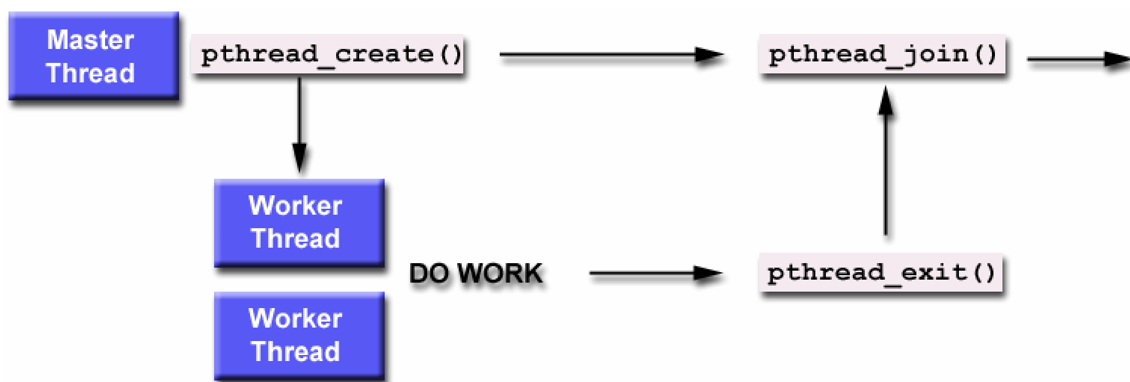
© Hviezdna jedenástka, 2008

Návrh triedy implementujúcej POSIX vlákna

Trieda `Thread` bude mať za úlohu vytvorenie, spustenie a ukončenie POSIX vlákna. Vlákna budú typu Joinable a teda hlavné vlákno bude môcť získať výstup svojich podvlákien (Obr. 26). Hlavné vlákno sa synchronizuje so svojimi podvláknami funkciou `pthread_join()`, ktorej sa ako parameter zadá deskriptor podvlákna a smerník na premennú návratového kódu. Hlavné vlákno potom čaká na ukončenie zadaného podvlákna. Vytvorenie a spustenie vlákna má na starosti metóda `start()`. Metóda stop vlákno zastaví. Trieda `Thread` obsahuje virtuálnu metódu `run()`, ktorej implementácia v zdedenej triede vytvorí kód spustený vo vlákne. Metóda `run()` sa volá zo spoločnej funkcie pre vlákna, `threadMain(void * _params)`.

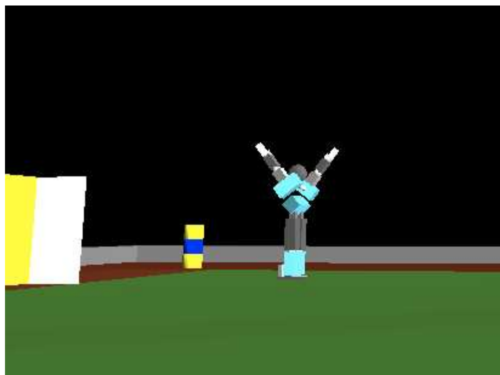
```
void * threadMain(void * _params) {

    ThreadParams * params = (ThreadParams *) _params;

      int rc = params->thread->run();

    pthread_exit((void *) rc);
    return (void *) rc;
}
```

Funkcii `threadMain` sa predáva smerník na štruktúru `ThreadParams`.



Obr. 26: Synchronizácia vlákien typu Joinable

Funkčnosť implementácie POSIX vlákien triedou Thread sme overili na systémoch Linux (distribúcia Ubuntu), Windows (Vista a XP) a Mac OS X (Tiger). Takto sme otestovali aj modul komunikácie hráča so serverom. Serveru sme posielali správy pre kĺby rúk robota a na monitore sme videli, ako nimi hráč hýbe (Obr. 27).



Obr. 27: Pohyb hráča rukami

© Hviezdna jedenástka, 2008

Trieda modulu komunikácie je odvodená od triedy `Thread` a v metóde `run()` je slučka, ktorá počúva a odosiela správy. Slučka skončí ak vlákno dostane signál na ukončenie (THREADSIG_STOP). Signály sa vláknu posielajú prostredníctvom premennej `signal`. Na (Obr. 28) vidno vzťahy medzi triedami.

Synchronizácia vlákien

Synchronizácia vlákien v programe bude prebiehať pomocou zámkov (mutexes). Je dôležité, aby vlákno po uzamknutí zámku, zámok aj odomklo. Ďalej treba zámok použiť všade tam, kde sa pristupuje a narába so spoločnými prostriedkami vlákien.

Použitie zámkov

1. Vytvorenie zámku mutexsum:

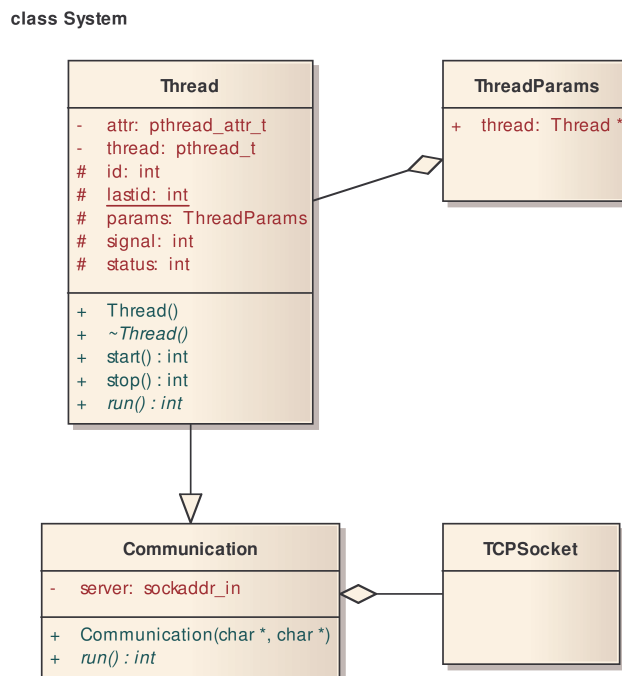
   ```
   pthread_mutex_t mutexsum;
   pthread_mutex_init(&mutexsum, NULL);
   ```

2. Uzamknutie a odomknutie zámku:

   ```
   pthread_mutex_lock (&mutexsum);
   sum += mysum;
   pthread_mutex_unlock (&mutexsum);
   ```

3. Zrušenie zámku:

   ```
   pthread_mutex_destroy(&mutexsum);
   ```



Obr. 28: Diagram tried prototypu na testovanie komunikácie vo vlákne

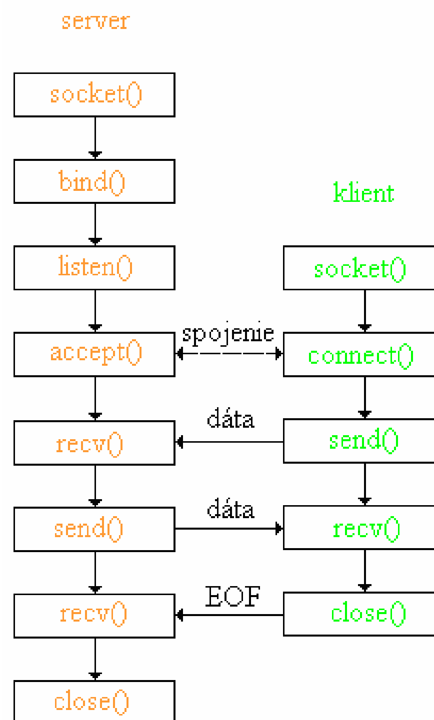
5.2.4 Komunikácia so serverom

Sokety

Pre sokety existuje v jazyku C implementácia štandardu IEEE POSIX (Portable Operating System Interface) 1003.1g (2001). Táto implementácia sa označuje pojmom POSIX sockets alebo BSD (Berkeley Software Distribution) sockets. Tým, že simulačný futbal je schopný komunikovať prostredníctvom TCP a UDP protokolov, boli pre model hráča implementované TCP a UDP sokety.

TCP soket

TCP protokol je protokol so spojením, ktorý zabezpečuje komunikáciu medzi klientom a serverom bez straty paketov v sieti. Jednoduchá schéma komunikácie klienta so serverom pomocou tohto protokolu je znázornená na (Obr. 29).



Obr. 29: Schéma TCP komunikácie medzi klientom a serverom

Pre tento typ komunikácie bola pre hráča vytvorená trieda TCPSocket, ktorá pomocou vyššie spomínaných metód (socket(), connect(), ...) zabezpečuje pohodlnú komunikáciu so serverom.

Trieda TCPSocket obsahuje:

Konštruktor implementujúci metódu socket() z POSIX štandardu, ktorého úlohou je vytvoriť soket.

Metódu *connect* využívajúcu metódu connect(). Jej úlohou je združiť server so soketom a pripojiť sa naň.

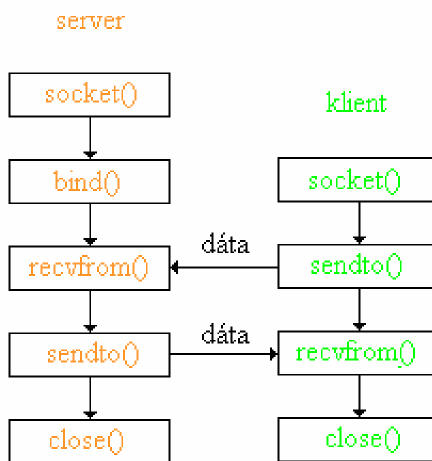
Metódu *send* implementujúcu metódu send() z daného štandardu, ktorej úlohou je posielať dáta serveru.

Metódu *receive*, ktorá využíva metódu recv(). Jej úlohou je prijímať prichádzajúce dáta zo servera.

A nakoniec metódu *close*. Táto metóda implementuje POSIX metódu s rovnomenným názvom. Jej úlohou je zavrieť soket.

UDP soket

UDP protokol je protokol bez spojenia, ktorý zabezpečuje rýchlu komunikáciu medzi klientom a serverom s možnou stratou paketov v sieti. Jednoduchá schéma komunikácie klienta so serverom pomocou tohto protokolu je znázornená na (Obr. 30) resp. (Obr. 31).



Obr. 30: Schéma UDP komunikácie medzi klientom a serverom (verzia 1).

Pre tento typ komunikácie bola pre hráča vytvorená trieda UDPSocket.

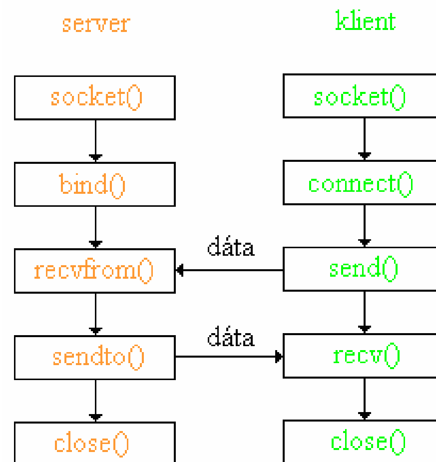
Trieda UDPSocket obsahuje:

Konštruktor implementujúci metódu socket() z POSIX štandardu, ktorého úlohou je vytvoriť soket.

Metódu *sendto* implementujúcu metódu sendto() z daného štandardu, ktorej úlohou je posielať dáta serveru.

Metódu *receivefrom*, ktorá využíva metódu recvfrom(). Jej úlohou je prijímať prichádzajúce dáta zo servera.

A nakoniec metódu *close*. Táto metóda implementuje POSIX metódu close(). Jej úlohou je zavrieť soket.



Obr. 31: Schéma UDP komunikácie medzi klientom a serverom (verzia 2).

Pre tento typ komunikácie bola pre hráča vytvorená trieda UDPSocket2.

Trieda UDPSocket2 obsahuje:

Konštruktor implementujúci metódu socket() z POSIX štandardu, ktorého úlohou je vytvoriť soket.

Metódu *connect* využívajúcu metódu connect(). Jej úlohou je združiť server so soketom.

Metódu *send* implementujúcu metódu send() z daného štandardu, ktorej úlohou je posielať dáta serveru.

Metódu *receive*, ktorá využíva metódu recv(). Jej úlohou je prijímať prichádzajúce dáta zo servera.

A nakoniec metódu *close*. Táto metóda implementuje POSIX metódu s rovnomenným názvom. Jej úlohou je zavrieť soket.

5.2.5 Parser

Pri vytváraní parsera bol použitý open-source program s názvom APG - ABNF Parser Generator [18]. Tento program vygeneruje parser na základe vstupného súboru, v ktorom je gramatika vo forme ABNF (Augmented Backus-Naur Form) zápisu. ABNF je formalizmus pre zápis gramatík vo forme terminálnych a neterminálnych symbolov.

APG na základe gramatiky zapísanej v súbore s príponou *.bnf vygeneruje parser pre túto gramatiku. Parsovanie funguje tak, že na základe danej gramatiky sa vytvára abstraktný syntaktický strom, ktorý sa postupne prechádza. Pre jednotlivé neterminálne symboly sa pri prechode týmto stromom volajú callback funkcie, do ktorých napíšeme, čo sa má s vyparsovanými dátami spraviť.

Vyparsované údaje sa budú ukladať do štruktúry, ktorá je definovaná ako trieda `ServerMsgParserData`.

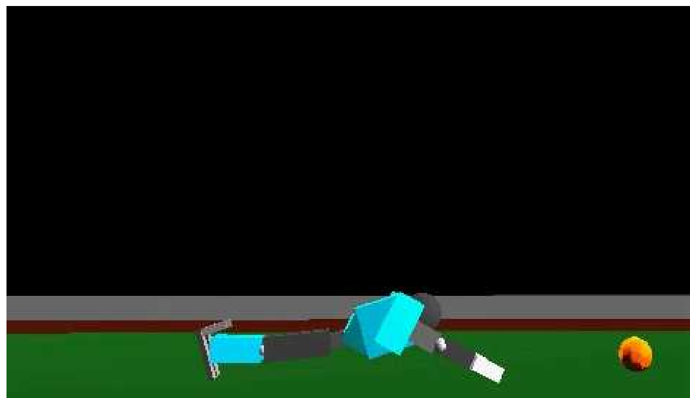
5.2.6 Ovládanie kĺbov, implementácia vstávania

Vstávanie je dôležitou súčasťou hry v robotickom futbale. Hráči zatiaľ nie sú ani zďaleka tak dokonalí, aby dokázali odohrať zápas bez pádu, či len s výnimočným pádom. Pády sú

momentálne veľmi častou súčasťou hry, je preto potrebné sa s nimi určitým spôsobom vysporiadať. Pri vstávaní sme sa inšpirovali postupným typom vstávania jedného z tímov zo záznamu finále Robocupu 3D z roku 2007.

Vstávanie zabezpečuje samostatná funkcia `fall`. Samotný pohyb vstávania je rozdelený na viacero fáz, konkrétne na osem, ktoré teraz bližšie popíšeme. Dôležitou informáciou je, že rozoznávanie fáz je založené na aktuálnom čase. Na začiatku funkcie vstávania sa zistí aktuálny čas, a potom sa postupne prechádza v závislosti na čase jednotlivými fázami.

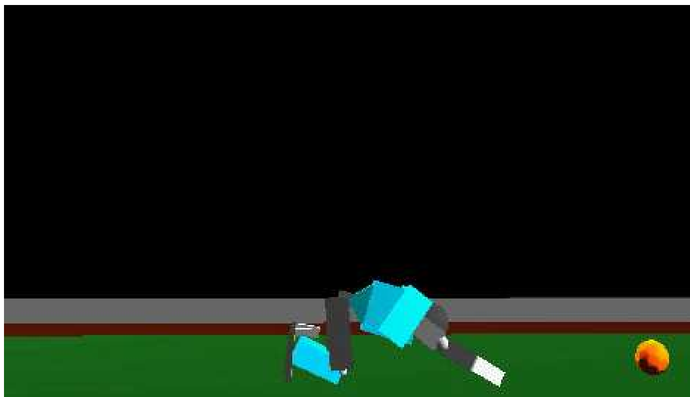
Táto fáza je prvá, jej úlohou je posunúť ruky popod telo robota tak, aby sa nimi zospodu podopieral, viď (Obr. 32).



Obr. 32: Prvá fáza vstávania hráča z ľahu na bruchu

Z ľahu na bruchu sa postupne posúvajú obe ruky popod telo, až do konečného stavu na (Obr. 32). Tento pohyb je docielený iba ovládaním plecného kĺbu. Táto fáza trvá asi 2 sekundy.

Ďalšia fáza má za účel ohnutie kolien a bokov tak, aby bola výsledná poloha kľačanie štvornožky. Táto fáza trvá asi 3 sekundy. Cieľom je poloha na (Obr. 33):

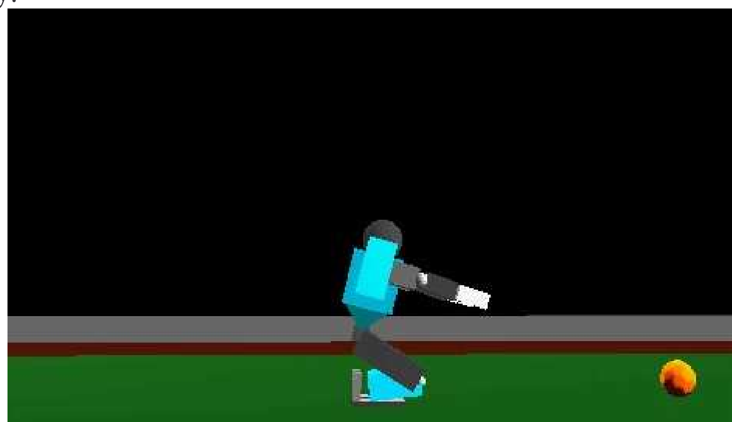


Obr. 33: Druhá fáza vstávania hráča z ľahu na bruchu

Tretia fáza má za úlohu prisunutie rúk bližšie popod telo, z čoho následne ťaží štvrtá fáza, ktorá posúva nohy popod telo. Výsledkom je poloha, keď hráč kvočí, a podopiera sa rukami. Piata fáza už len vyrovná lakeť, posunie ruky ešte bližšie pri telo, a pripraví tak hráča na finálnu fázu samotného vstávania. Z tejto polohy je teda už možné vstať. Cieľová poloha fázy 5 je znázornená na (Obr. 34):

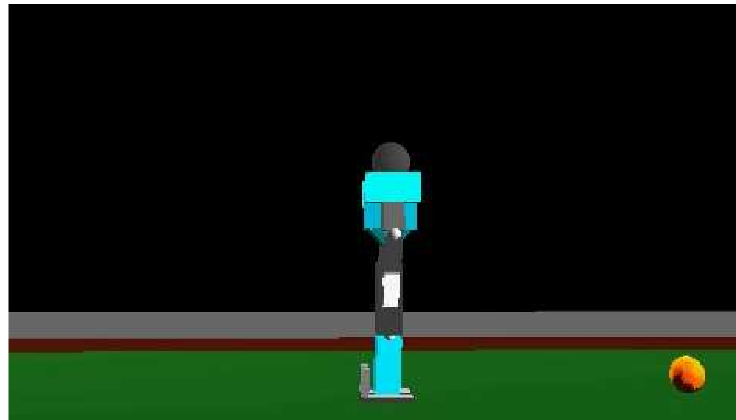
Obr. 34: Piata fáza vstávania hráča z ľahu na bruchu

Šiesta fáza je už len stabilizačná. Keďže ťažisko hráča je pomerne vysoko, popod hlavou, je nutné polohu hráča pred vstaním stabilizovať. To je dosiahnuté vyrovnávaním lakťa a vystretím rúk pred telo ohnutím pliec do pravého uhla. Je to posledný stupeň pred vzpriamením sa hráča do stojacej polohy.



Obr. 35: Siedma fáza vstávania hráča z ľahu na bruchu

Posledná fáza je najrýchlejšia – trvá asi len 2 sekundy. Poloha tela hráča je stabilizovaná, viď (Obr. 35), je teda možné vstať. Vstanie je dosiahnuté vyrovnaním kolenných kĺbov a kĺbov bokov. Následne sa pripažia ruky, a hráč je konečne vo vzpriamenej polohe. Výsledok vstávania je znázornený (Obr. 36):



Obr. 36: Posledná – ôsma fáza vstávania hráča z ľahu na bruchu

Cele vstávanie trvá približne 20 sekúnd. Na záver pripájam ukážku zdrojového kódu poslednej fázy vstávania, kvôli utvoreniu si lepšej predstavy ako vlastne vstávanie funguje:

```
// zaciatok vstavania, urcime zaciatocny cas
Time time  = WM->getSimulatorTime();
...
...
fazy vstavania
...
...
// posledna osma faza
if( WM->getSimulatorTime() - time < 21)
        {
        printf ("\nosma faza");
        ankle   ( SIDE_LEFT,     0,   0, 1.30 * gain );
        ankle   ( SIDE_RIGHT,    0,   0, 1.30 * gain );
        knee    ( SIDE_LEFT,     0,       1.30 * gain );
        knee    ( SIDE_RIGHT,    0,       1.30 * gain );
        hip     ( SIDE_LEFT,     0,  0,   1.30 * gain );
        hip     ( SIDE_RIGHT,    0,  0,   1.30 * gain );
        shoulder( SIDE_LEFT,     0,    0, 1.30 * gain );
        shoulder( SIDE_RIGHT,    0,    0, 1.30 * gain );
        }
```

Táto časť kódu spraví nasledovné:

1. ako už bolo spomenuté, ešte na začiatku funkcie vstávania, t.j. ešte pred prvou fázou, sa zistí aktuálny čas simulátora – a ten sa uloží do premennej `time`. To sa vykoná v riadku `Time time  = WM->getSimulatorTime();` Všetko ostatné sa riadi podľa tejto premennej.
2. ak je rozdiel medzi aktuálnym časom, a časom začiatku vstávania menší ako x sekúnd, vykonáva sa príslušná fáza. Napríklad aj je rozdiel menší ako 3, vykonáva sa prvá fáza, ak je väčší ako 3 a menší ako 5, vykonáva sa druhá fáza, atď. Na ukážke zdrojového kódu je napríklad vidieť, že ak je rozdiel väčší ako 19 sekúnd a menší ako 21, vykonáva sa fáza osem – fáza vstávania.
3. Samotný kód fázy 8 predstavuje vzpriamenie. To je zložené zo 4 elementárnych pohybov a to vyrovnávanie členkov, vyrovnávanie kolien, vyrovnávanie bokov a pohyb pliec, ktorý zabezpečí, že sa vystreté ruky vrátia do polohy pri tele.

Všetky tieto pohyby sú dosiahnuté prostredníctvom ovládacích kĺbových procedúr – každý kĺb má vlastnú, avšak ich volanie je veľmi podobné. Popíšeme to na ukážke ovládania kolenného kĺbu:

```
knee    ( SIDE_LEFT,     0,       1.30 * gain );
knee    ( SIDE_RIGHT,    0,       1.30 * gain );
```

`Knee` je názov funkcie, tá má 3 parametre. Prvým je ktorý kĺb chceme ovládať, ľavý, alebo pravý (`SIDE_LEFT, SIDE_RIGHT`). Druhý parameter je koncový uhol, do ktorého sa má kĺb dostať (tu napríklad chceme, aby sa kĺb vrátil do pôvodnej polohy, tá je určená uhlom 0). Pohyb vpravo, či vľavo je potom definovaný posunom do kladného, či záporného uhla. Posledným parametrom je rýchlosť pohybu (`1.30 * gain`). Na ukážke vyššie vidieť, že v tomto prípade sa kolenný kĺb hýbe zrýchlene – 1.3 násobkom defaultnej hodnoty.

Každá fáza vstávania teda pozostáva zo série elementárnych pohybov jednotlivých kĺbov, podobných vyššie popísanému. Potrebné množstvo kĺbov v rovnakom čase sa zapojí, vykoná pohyb do požadovanej polohy, a výsledkom tejto spolupráce je určitá fáza pohybu. Každá fáza sa vykonáva iba určitú dobu. Celé vstávanie potom tvorí viacero fáz zoradených za sebou.

Vstávanie, ktoré sme implementovali, trvá spolu približne 20 sekúnd a je typom postupného vstávania. „Ninja" vstávanie, ktoré bolo vidieť vo videoukážke z finále 3D Robocupu sa nám nepodarilo napodobniť, hlavne kvôli veľmi vysoko položenému ťažisku hráča. Implementované vstávanie je statické, t.j. pozostáva z vopred určenej postupnosti pohybov.

Videoukážka vstávania hráča z ľahu na bruchu je umiestnená aj na našej tímovej stránke v sekcii odkazy, konkrétny link je http://www2.dcs.elf.stuba.sk/TeamProject/2007/team11is-si/download/vstavanie01.avi .

5.3 Zhodnotenie prototypu

Výsledkom prototypu bolo overenie informácií získaných pri analyzovaní servera, vytvorenie základnej štruktúry hráča a implementácie modulu komunikácie tohto hráča. Informácie sme overili na zdrojových kódoch tímu Zigorat. Staticky sme vytvorili sekvenciu pohybu kĺbov a demonštrovali ju na príklade vstávania agenta zo zeme z polohy na bruchu. Agent sa úplne postavil za 20 sekúnd, čo považujeme za dostatočne krátky čas.

Hlavnou motiváciou tímu bolo vytvorenie nového agenta, skompilovateľného pod vyššie popísanými operačnými systémami v rôznych vývojových prostrediach. Tento cieľ sa podarilo uskutočniť pod operačnými systémami Windows (XP), Linux (distribúcia Ubuntu) a MacOS (Tiger) vo vývojových prostrediach Visual Studio (6.0, 2005, 2008), Eclipse a Xcode.

V prototype hráča sme oddelili modul komunikácie od ostatných modulov uložením do samostatného vlákna. Je vytvárané podľa štandardu pre vlákna IEEE POSIX 1003.1c (1995), čím sme zabezpečili nezávislosť na operačnom systéme. Overenie funkčnosti modulu komunikácie sme otestovali v posielaní správ na server pre kĺby rúk robota. Výsledkom bolo rotovanie rúk hráča. Nakoniec sme vytvorili modul parser, ktorý je nezávislý na architektúre agenta.

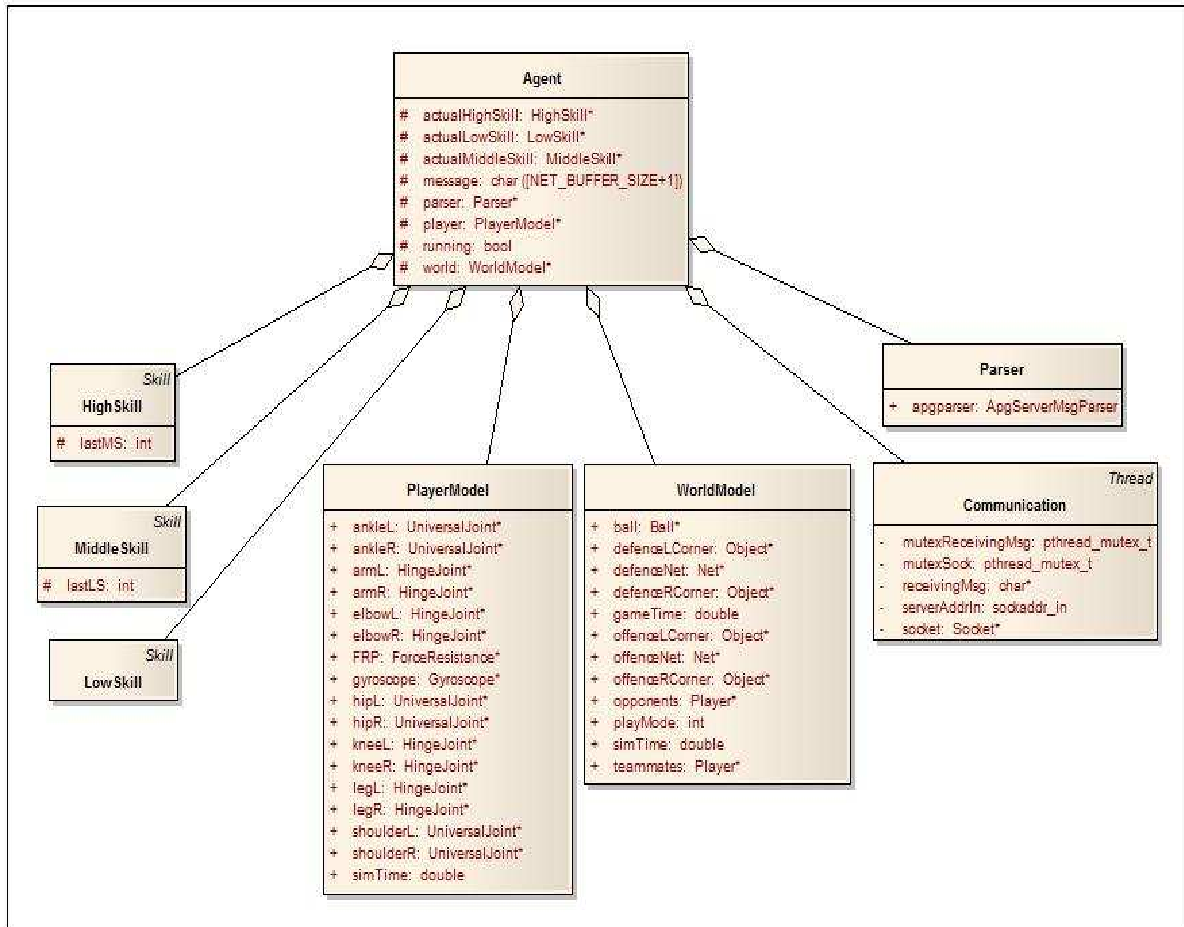
© Hviezdna jedenástka, 2008

6. Implementácia

V tejto kapitole budú opísané všetky implementované časti hráča. Taktiež opíšeme konečnú architektúru hráča, nakoľko sú v nej zmeny oproti návrhu.

6.1 Architektúra hráča

Pri implementácii sme boli nútení doplniť a čiastočne zmeniť architektúru hráča oproti architektúre uvedenej v prototype. Na Obr. 37 je uvedená výsledná architektúra hráča



Obr. 37: Výsledná architektúra hráča

Modul komunikácie a parser sa stará o posielanie a získavanie informácií zo správ posielaných serverom. Model sveta – World model umožňuje hráčovi ukladať si informácie o okolitom svete a objektoch v ňom. Model hráča – Player model zase obsahuje informácie o hráčovi samotnom, napríklad informácie o polohách jeho jednotlivých kĺbov. Triedy z množiny Skill reprezentujú správanie, schopnosti agenta.

V nasledujúcich kapitolách opíšme podrobnejšie implementáciu jednotlivých modulov v architektúre.

6.2 Implementácia modulu komunikácie

Komunikácia so serverom

Hráč prijíma a posiela správy vo formáte:

- Prvé štyri byty predstavujú dĺžku správy v network order (treba prekonvertovať)
- Ostatné byty predstavujú už obsah správy, čiže s-výrazy.

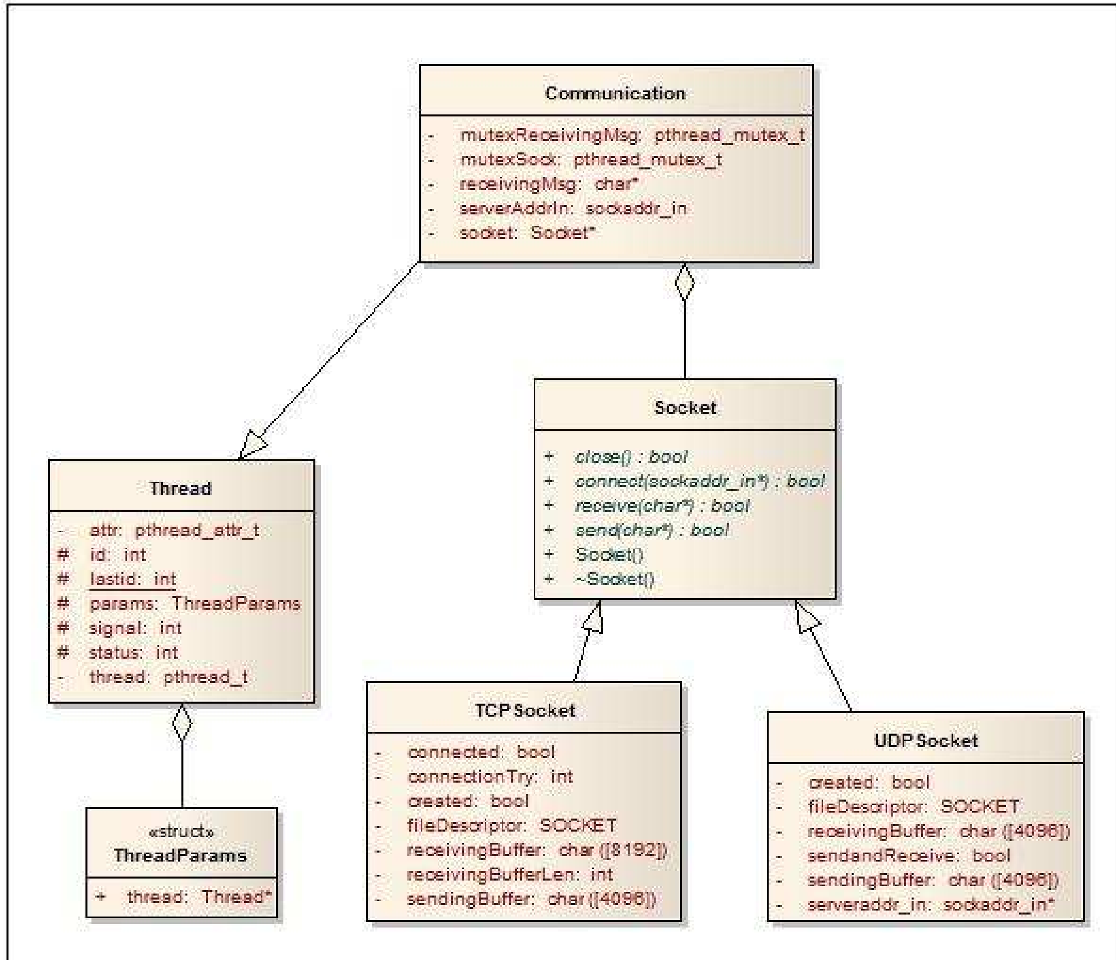
Obsah prijatej správy môže vyzerať nasledovne:

```
(time (now 1.68))(GS (t 0.00) (pm BeforeKickOff))(GYR (n torso) (rt -0.41 -
0.68 -0.08))(See (G1L (pol 0.62 -172.55 4.76)) (G2L (pol 1.62 -114.06 1.83))
(G2R (pol 5.59 -17.36 0.53)) (G1R (pol 5.39 -2.87 0.56)) (F1L (pol 1.39
115.29 -14.48)) (F2L (pol 2.89 -104.17 -6.93)) (F1R (pol 5.53 10.76 -3.60))
(F2R (pol 6.08 -29.27 -3.28)) (B (pol 2.54 -20.34 -6.93)))(UJ (n laj1_2) (ax1
75.60) (ax2 -0.05))(UJ (n raj1_2) (ax1 -75.60) (ax2 0.05))(HJ (n laj3) (ax -
0.00))(HJ (n raj3) (ax -0.00))(HJ (n laj4) (ax -0.08))(HJ (n raj4) (ax
0.04))(HJ (n llj1) (ax -0.00))(HJ (n rlj1) (ax 0.00))(UJ (n llj2_3) (ax1 -
0.00) (ax2 0.00))(UJ (n rlj2_3) (ax1 -0.00) (ax2 0.00))(HJ (n llj4) (ax -
0.00))(HJ (n rlj4) (ax -0.00))(UJ (n llj5_6) (ax1 0.00) (ax2 -0.00))(UJ (n
rlj5_6) (ax1 0.00) (ax2 -0.00))
```

Server posiela správy každých 20 ms (prednastavené). Počas tohto intervalu je schopný prijať a spracovať len jednu správu, ktorú mu hráč pošle. Správa je zložená z s-výrazov, pre každý efektor samostatne.

Server komunikuje pomocou TCP a UDP protokolov, preto boli vytvorené triedy *TCPSocket* a *UDPSocket*, aby hráč mohol pomocou daných protokolov so serverom komunikovať. Bola vytvorená aj spoločná trieda *Socket*, od ktorej obe spomínané triedy dedia.

Diagram tried pre modul komunikácie je znázornený na Obr. 38



Obr. 38: Diagram tried pre modul komunikácie

Trieda Socket

Trieda *Socket* je abstraktná trieda, v ktorej sú definované spoločné metódy pre oba typy socketov. Táto trieda má štyri virtuálne metódy a tie musia byť preto implementované aj v dedičných triedach. Ide o metódy *connect, send, receive* a *close*. Všetky metódy sú popísané v Tab. 5.

Názov	Návratová hodnota	Popis
Socket		Konštruktor.
~Socket		Deštruktor.
connect	Bool	Metóda pripojenia sa na server resp. metóda určenia na aký server sa budú posielať správy a z akého servera sa budú prijímať správy.
send	Bool	Metóda posielania správ serveru.
receive	Bool	Metóda prijímania správ.
close	Bool	Metóda uzatvorenie socketu.

Tab. 5: Metódy triedy Socke

Trieda TCPSocket

Ako už bolo spomenuté trieda *TCPSocket* dedí od abstraktnej triedy *Socket*. Táto trieda implementuje metódy na komunikáciu so serverom. Implementuje všetky štyri virtuálne metódy: *connect, send, receive* a *close*.

V Tab. 6 sú popísané všetky atribúty tejto triedy.

Názov	Typ	Popis
fileDescriptor	SOCKET (Win) / int (inak)	Vyjadruje file descriptor socketu.
created	bool	Vyjadruje, či bol socket vytvorený.
connected	bool	Vyjadruje, či sa socket pripojil na server.
connectrionTry	Int	Určuje počet pokusov o pripojenie.
sendingBuffer	char [4096]	Zásobník pre odosielané dáta. Obsahuje dĺžku správy v network order a samotnú správu.
receivingBuffer	char [8192]	Zásobník pre prijaté dáta.
receivingBufferLen	Int	Počet bytov v zásobníku pre prijímanie dát.

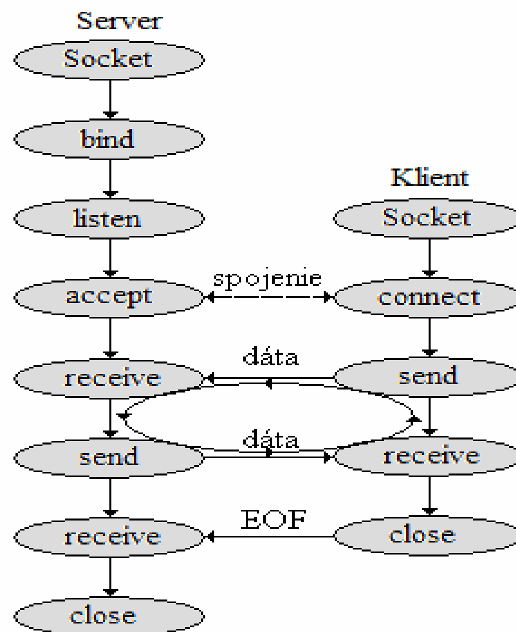
Tab. 6: – Atribúty triedy TCPSocket

V Tab. 7 sú popísané všetky metódy tejto triedy.

Názov	Návratova hodnota	Popis
TCPSocket		Konštruktor, v ktorom sa vytvára socket, atribút created sa nastavuje podľa výsledku vytvorenie socketu, atribút connected sa nastavuje na false, atribút connectionTry sa nastavuje na 0 a atribút receivingBufferLen sa tiež nastavuje na 0.
connect	Bool	Metóda, ktorá ak je socket vytvorený a počet pokusov o pripojenie je rovný nule sa pokúsi pripojiť k serveru. Počet pokusov o pripojenie sa zvyšuje a vracia sa výsledok pokusu pripojenia. Táto metóda zabezpečuje, že hráč sa nemôže pripojiť druhý-krát na nejaký server, lebo by to aj tak nevyšlo. Čiže ak sa pripojí na prvý-krát, ostane pripojený, ak sa nepripojí ostane nepripojený.
send	Bool	Metóda, ktorá ak je socket pripojený, sa pokúsi odoslať správu a vráti výsledok odoslania správy.
receive	Bool	Metóda, ktorá ak je socket pripojený, sa pokúsi prijať správu. Ak na sieťovej karte nie sú žiadne dáta pre tento socket od daného servera, tak sa čaká kým dáta neprídu a potom sa z nich vygeneruje správa. Ak tam sú dáta, tak sa spracujú do správ, ostane len posledná správa, ostané sa zahodia. Nakoniec metóda vráti výsledok o prijatí správy.
close	Bool	Metóda sa pokúsi odpojiť socket od servera a zavrieť socket. Vráti výsledok tejto operácie.

Tab. 7: – Metódy triedy TCPSocket

Na obrázku nižšie Obr. 39je znázornený princíp TCP komunikácie.



Obr. 39: Schéma TCP komunikácie

Trieda UDPSocket

Trieda *UDPSocket* dedí od abstraktnej triedy *Socket*. Táto trieda implementuje metódy na komunikáciu so serverom. Implementuje všetky štyri virtuálne metódy: *connect, send, receive* a *close*.

V Tab. 8 sú popísané všetky atribúty tejto triedy.

Názov	**Typ**	**Popis**
fileDescriptor	SOCKET (Win) / int (inak)	Vyjadruje file descriptor socketu.
created	bool	Vyjadruje, či bol socket vytvorený.
sendandReceive	bool	Vyjadruje, či sa možu posielať a prijímať dáta zo servera.
sendingBuffer	char [4096]	Zásobník pre odosielané dáta. Obsahuje dĺžku správy v network order a samotnú správu.
receivingBuffer	char [4096]	Zásobník pre prijaté dáta. Obsahuje dĺžku správy v network order a samotnú správu.
serveraddr_in	sockaddr_in *	Server, ktorému sa posielajú dáta a od ktorého sa prijímajú dáta.

Tab. 8: – Atribúty triedy UDPSocket

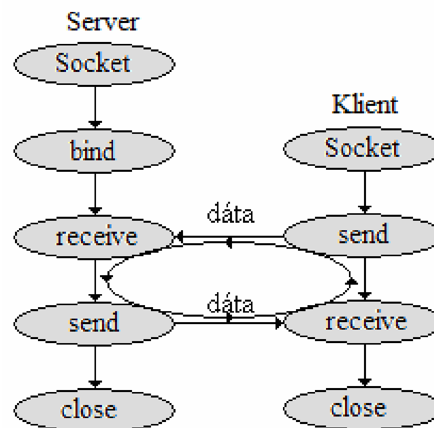
V Tab. 9sú popísané všetky metódy tejto triedy.

Názov	**Návratova hodnota**	**Popis**
UDPSocket		Konštruktor, v ktorom sa vytvára socket, atribút created sa nastavuje podľa výsledku vytvorenie socketu, atribút sendandReceive sa nastaví na hodnotu atribútu created.
connect	bool	Metóda, ktorá nastavuje atribút serveraddr_in na server, s ktorým sa bude komunikovať.
send	bool	Metóda, ktorá ak je socket schopný posielať a prijímať dáta, sa pokúsi odoslať správu a vráti výsledok odoslania

		správy.
receive	bool	Metóda, ktorá ak je socket schopný posielať a prijímať dáta, sa pokúsi prijať správu. Ak na sieťovej karte nie sú žiadne dáta pre tento socket od daného servera, tak sa čaká kým dáta neprídu a potom sa z nich vygeneruje správa. Čaká sa maximálne jednu sekundu. Ak do tohto časového limitu neprídu dáta, predpokladá sa, že server už nie je schopný posielať dáta (napr.: padol) a vráti sa návratová hodnota false. Ak tam sú dáta, tak sa spracujú do správ, ostane len posledná správa, ostané sa zahodia. Nakoniec metóda vráti výsledok o prijatí správy.
close	bool	Metóda sa pokúsi zavrieť socket. Vráti výsledok tejto operácie.

Tab. 9: – Metódy triedy UDPSocket

Na obrázku Obr. 40 je znázornený princíp UDP komunikácie.



Obr. 40: Schéma UDP komunikácie

6.3 Parser správ prichádzajúcich zo servera

Pri implementácii parsera správ bol použitý nástroj APG. Implementované boli všetky typy perceptorov, ktoré sme potrebovali pri vývoji hráča. Nie je implementované parsovanie správ o polohe iných hráčov na ihrisku (časť percerptoru See), pretože sme neimplementovali žiadne schopnosti hráča, ktoré by vyžadovali interakciu s inými hráčmi a ani pri spúšťaní simulátora sme nemali viacerých hráčov na ihrisku.

Pomocou APG nástroja sa zo vstupnej gramatiky vygenerujú dva zdrojové súbory ServerMsgParser.cpp ServerMsgParser.h. Tieto súbory vlastne definujú lexikálny a syntaktický analyzátor správ zo servera.

Aby sme mohli vyparsované údaje nejakým spôsobom uložiť, sú v týchto súboroch vygenerované callback funkcie pre každý neterminálny symbol zadaný v gramatike. Do danej callback funkcie dopíšeme vlastný kód, ktorý bude definovať to, čo sa má s vyparsovaným údajom spraviť. Pri prechádzaní správou je callback funkcia pre každý neterminálny symbol volaná vo viacerých stavoch, ktoré sú označené nasledovne:

- SYN_MATCH – časť reťazca sa zhoduje s pravidlom v gramatike, tento stav je najdôležitejší, pretože nám hovorí o tom, že príslušný údaj bol správne vyparsovaný. Pri

tomto stave sa callback funkcia volá s parametrami – dĺžka vyparsovaného podreťazca (ulLen) a pozícia začiatku podreťazca od začiatku celej správy

- SYN_NOMATCH – keď sa nenájde zhoda medzi pravidlom a príslušným reťazcom
- SYN_EMPTY – nájdenie prázdneho reťazca
- SYN_PRE – stav pred začiatkom parsovania príslušného neterminálneho symbolu

Vo vygenerovanom kóde je potrebné vlastný dopísaný kód písať medzi vygenerované riadky, ktoré začínajú znakmi „//{{„ a končia znakmi „//}}“. Je to kvôli tomu, že APG dokáže nanovo vygenerovať tieto súbory pre gramatiku tak, aby používateľom dopísaný kód ostal zachovaný. Napríklad ak chceme pridať jedno pravidlo do gramatiky, bude nutné vyššie spomenuté dva súbory vygenerovať znovu, no časti kódu, ktoré boli dopísané sa takto „nestratia“. Spomenutá štruktúra sa vygeneruje na viacerých vhodných miestach, nie len pre callback funkcie, napríklad na začiatku súboru, v konštruktore hlavnej triedy a pod. Týmto je možné si vygenerovaný parser plne prispôsobiť pre naše potreby.

Príklad použitia callback funkcie

Teraz opíšme, akým spôsobom definujeme gramatiku a ako je možné používať callback funkcie a získať pomocou nich vyparsované dáta.

Použijeme perceptor time. Príklad poslanej správy tohto perceptora je:

```
(time (now 219.93))
```

ABNF zápis gramatiky pre tento perceptor je :

```
MsgTime = "(" timeStr *SP "(" "now" *SP Time "))"
timeStr = "time"
Time    = realNr
SP      =  %x20
```

Všetky symboly, ktoré nie sú uzatvorené v úvodzovkách predstavujú neterminálne symboly. Zápis je vo forme pravidla. Terminálne symboly sa okrem reťazca v úvodzovkách môžu zapisovať aj pomocou ASCII hodnoty znaku tak, že sa použije symbol „%“. Napríklad %x20 pre ASCII znak „Space“.

Vidíme, že v pravidle MsgTime definujeme, aké znaky alebo podreťazce majú prísť za sebou. Nás bude zaujímať neterminálny symbol Time, pretože ten predstavuje tú konkrétnu hodnotu, ktorú si chceme uložiť. Time je reálne číslo.

Pre neterminálny symbol Time máme takúto callback funkciu:

```
// call back function for rule "Time"
ulong ApgServerMsgParser::pfn_Time(void* vpData, ulong ulState, ulong
ulOffset, ulong ulLen)
{
  ulong ulReturn = PFN_OK;
//{{CallBack.pfn_Time
  if(ulState == SYN_MATCH){
        strncpy(parse_buffer, &psr_msg_string[ulOffset], ulLen);
        parse_buffer[ulLen] = '\0';
        pStoredData->SetSimTime(atof((char*)parse_buffer));
  }
//}}CallBack.pfn_Time
  return ulReturn;
}
```

Časti kódu uzatvorené v „//{{„ a „//}}“ boli ručne písané, ostatné je vygenerované. V kóde je v podstate zapísané to, že ak nám parser pre daný neterminálny symbol Time vráti SYN_MATCH, t.j. v správe sa naozaj nachádza na príslušnom mieste reálne číslo, tak definujeme, čo sa s týmto reťazcom má spraviť. V tomto prípade reťazec prekonvertujeme na typ float a uložíme do nami definovanej štruktúry.

6.4 Logovanie do súboru

Keďže implementovaný hráč predstavuje určitú „realtime" aplikáciu, nie je možné používať pri ladení chýb debugger. Kvôli tomu sme vytvorili logovanie do výstupného súboru. Logovanie bolo implementované tak, že sme pre časti kódu týkajúce sa logovania, použili podmienenú kompiláciu. Je to kvôli tomu, aby bolo možné hráča skompilovať bez častí kódu loggera, pretože to môže spomaľovať beh aplikácie.

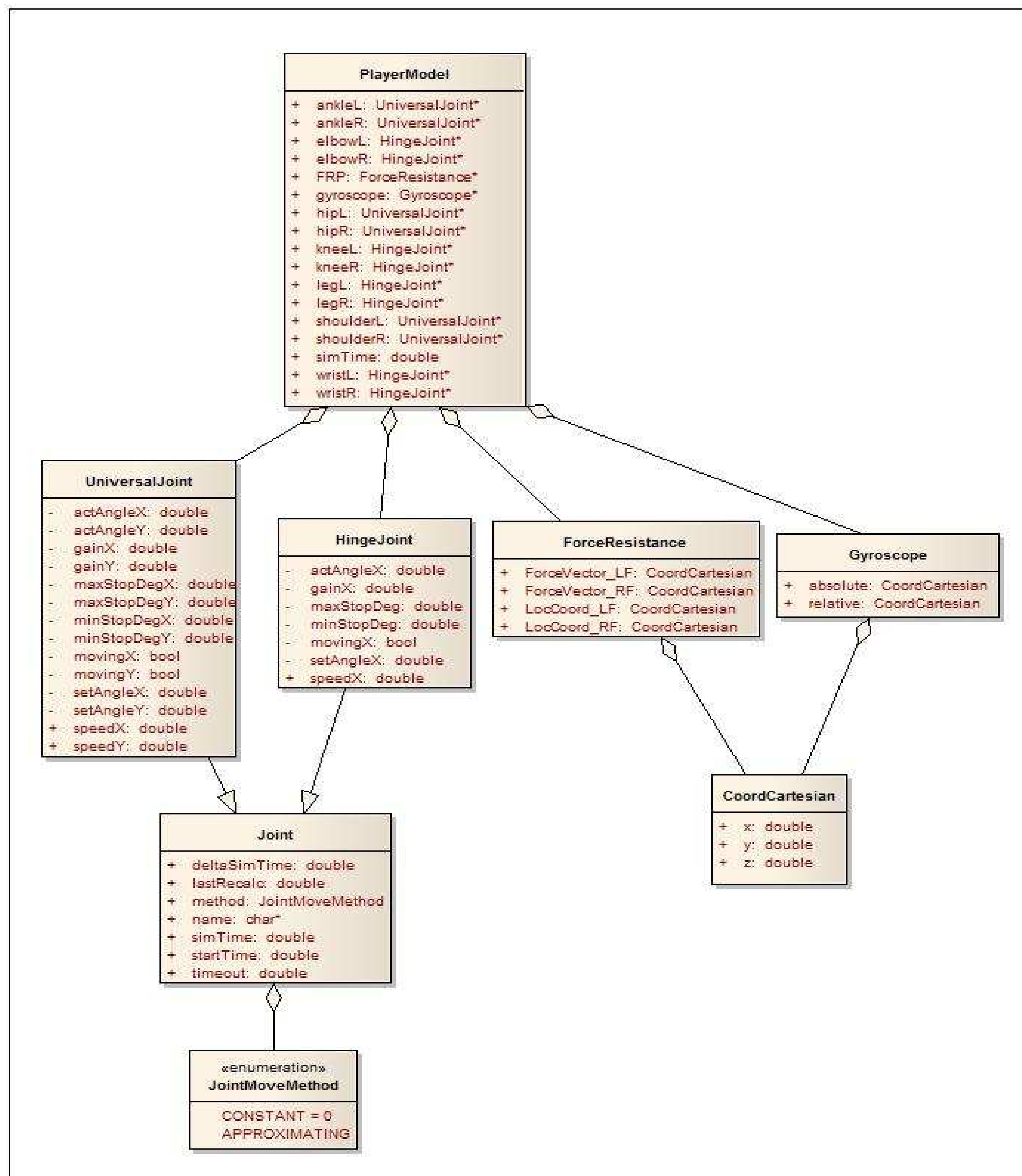
Pri implementácii triedy pre logger sme použili návrhový typ singleton. To znamená, že je možné vytvoriť iba jednu inštanciu tejto triedy, ktorá sa bude používať.

Logovanie je spravené pomocou prepínačov, ktoré sa nastavia na hodnotu true alebo false podľa toho, či chceme príslušné informácie logovať alebo nie. Je to robené kvôli tomu, aby sme nemuseli logovať vždy všetky informácie ale len tie, ktoré v danej chvíli potrebujeme.

6.5 Model hráča

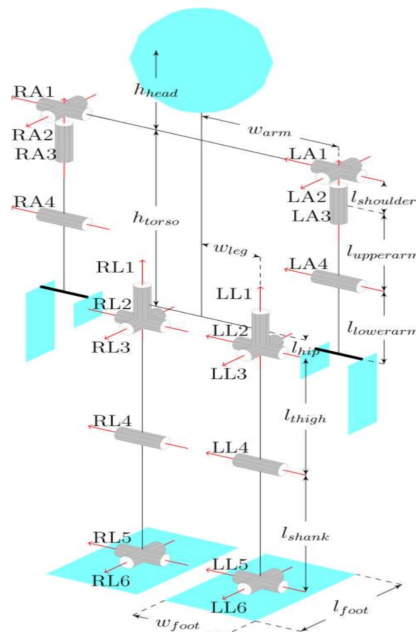
Model hráča pozostáva z kĺbov, gyroskopu umiestneného v trupe a silového perceptora umiestneného v chodidlách (Obr. 41).

© Hviezdna jedenástka, 2008



Obr. 41: Diagram tried modelu hráča

Pri implementácii modelu agenta sme vychádzali zo štandardného modelu používaného pre danú verziu servera. Tento model má názov soccerbot056 (Obr. 42) a jeho parametre sú definované v súbore soccerbot056.rsg.



Obr. 42: Model hráča soccerbot056

Trieda PlayerModel

Trieda PlayerModel obsahuje všetky informácie o agentovi samotnom. Tieto informácie zahŕňajú:

- Informácie o kĺboch agenta
- Gyroskop – informácie o zmenách polohy torza agenta
- ForceResistance perceptor – informácie o sile pôsobiacej na chodidlá agenta

Model soccerbot056 má celkovo 14 kĺbov. Ako vidieť v Tab. 1, pre každý kĺb je v PlayerModel vytvorená inštancia s príslušným názvom. Presnejšie povedané, ide o smerníky na inštancie jednotlivých kĺbov. Každý kĺb má definované obmedzenia, do akého minimálneho a maximálneho uhla sa môže natočiť. Horné a dolné obmedzenia sú uvedené v tabuľke. Pri kĺboch s dvoma osami sú uvedené obmedzenia pre každú os.

Názov kĺbu	Typ	Inštancia v triede PlayerModel	Osi	Min	Max
lle5_6	Universal	ankleL	LL5, LL6	-90, -45	90, 45
rle5_6	Universal	ankleR	RL5, RL6	-90, -45	90, 45
lle4	Hinge	kneeL	LL4	-160	10
rle4	Hinge	kneeR	RL4	-160	10
lle1	Hinge	legL	LL1	-90	60
rle1	Hinge	legR	RL1	-90	60
lle2_3	Universal	hipL	LL2, LL3	-45, -45	120, 75
rle2_3	Universal	hipR	RL2, RL3	-45, –75	120, 45
lae1_2	Universal	shoulderL	LA1, LA2	-90, -10	180, 180
rae1_2	Universal	shoulderR	RA1, RA2	-90, -180	180, 10
lae4	Hinge	elbowL	LA4	-10	130
rae4	Hinge	elbowR	RA4	-10	130
lae3	Hinge	armL	LA3	-135	135
rae3	Hinge	armR	RA3	-135	135

© Hviezdna jedenástka, 2008

Tab. 10: Kĺby v modeli soccerbot056

Okrem informáciách o kĺboch model hráča obsahuje aj atribút simTime udávajúci aktuálny čas simulácie v sekundách a inštancie tried Gyroscope a ForceResistance.

Všetky metódy triedy PlayerModel sú uvedené a popísané v Tab.2.

Názov	Návratová hodnota	Popis
PlayerModel		Konštruktor. Inicializuje všetky kĺby (názov kĺbu a jeho obmedzenia), gyroskop a forceResistance perceptor
~PlayerModel		Deštruktor
update	void	Aktualizácia kĺbu. Aktualizuje sa simulačný čas podľa parametra simTime
buildMessage	void	Metóda vytvorí pre server správu o kĺbe a zapíše ju ako reťazec na určené miesto podľa parametra message.
logPlayerModel	void	Ukladá do logu všetky údaje týkajúce sa modelu hráča (simulačný čas, absolútne hodnoty súradníc gyroskopu, súradnice ForceResistance perceptora pre ľavé a pravé chodidlo, všetky uhly kĺbov)
jointsMoving	bool	Vracia true, ak je akýkoľvek kĺb v pohybe

Tab. 11: Trieda PlayerModel

Trieda ForceResistance

Trieda ForceResistance slúži na uchovávanie hodnôt ForceResistance perceptora. Atribúty s popisom sú uvedené v Tab. 12.

Názov	Typ	Popis
LocCoord_LF	Vector3D	Lokálne koordináty koncentračného bodu sily pre ľavé chodidlo
LocCoord_RF	Vector3D	Lokálne koordináty koncentračného bodu sily pre pravé chodidlo
ForceVector_LF	Vector3D	Vektor sily pre ľavé chodidlo
ForceVector_RF	Vector3D	Vektor sily pre pravé chodidlo

Tab. 12: Trieda ForceResistance

Obsahuje metódu update, ktorá aktualizuje atribúty po každej prijatej správe od servera.

Trieda Gyroscope

Trieda Gyroscope obsahuje dva atribúty gyroskopu:

- absolútne (atribút absolute) koordináty gyroskopu, uchováva aktuálne uhlové zrýchlenie gyroskopu
- relatívne (atribút relative) koordináty gyroskopu sa vypočítajú ako rozdiel predchádzajúcej a aktuálnej absolútnej hodnoty gyroskopu.

6.6 Model sveta hráča

Trieda WorldModel

Trieda WorldModel predstavuje model sveta, t.j. všetky objekty, ktoré je hráč schopný vidieť, a všetky potrebné informácie pre to, aby sa mohol a vedel rozhodovať podľa aktuálnej situácie. Trieda obsahuje tieto objekty:

- `Ball * ball;` - lopta
- `Net * defenceNet;` - vlastná brána
- `Net * offenceNet;` - súperova brána
- `Object * defenceLCorner;` - ľavý roh na vlastnej polovici (v obrannej zóne)
- `Object * defenceRCorner;` - pravý roh na vlastnej plovici (v obrannej zóne)
- `Object * offenceLCorner;` - ľavý roh na súperovej plovici (v útočnej zóne)
- `Object * offenceRCorner;` - pravý roh na súperovej olovici (v útočnej zóne)
- `Player * opponents;` - protihráči
- Player * teammates; - spoluhráči

Ako vidieť, hráč model sveta obsahuje informácie o lopte, vlastnej a súperovej bránke, rohových zastávkach, spoluhráčoch a protihráčoch. Všetky tieto objekty budú podrobne popísané nižšie. Okrem toho obsahuje trieda aj tieto atribúty:

```
double simTime;
double gameTime;
int playMode;
```

Vyššie uvedené atribúty predstavujú simulačný čas, čas hry a mód hry. Simulačný čas je čas v milisekundách odkedy sa pustil simulačný server, t.j. simspark. Čas hry je čas, odkedy sa spustila hra, t.j. doba od prvého výkopu. Mód hry určuje aktuálnu hernú situáciu – napr. play on, kick left, free kick, atď. Podľa pravidiel, a nastavuje ho simulačný server. Podrobnejšie je popísaný v dokumentácii ku serveru. Trieda ďalej obsahuje okrem konštruktora a deštruktora jedinú ale o to dôležitejšiu metódu:

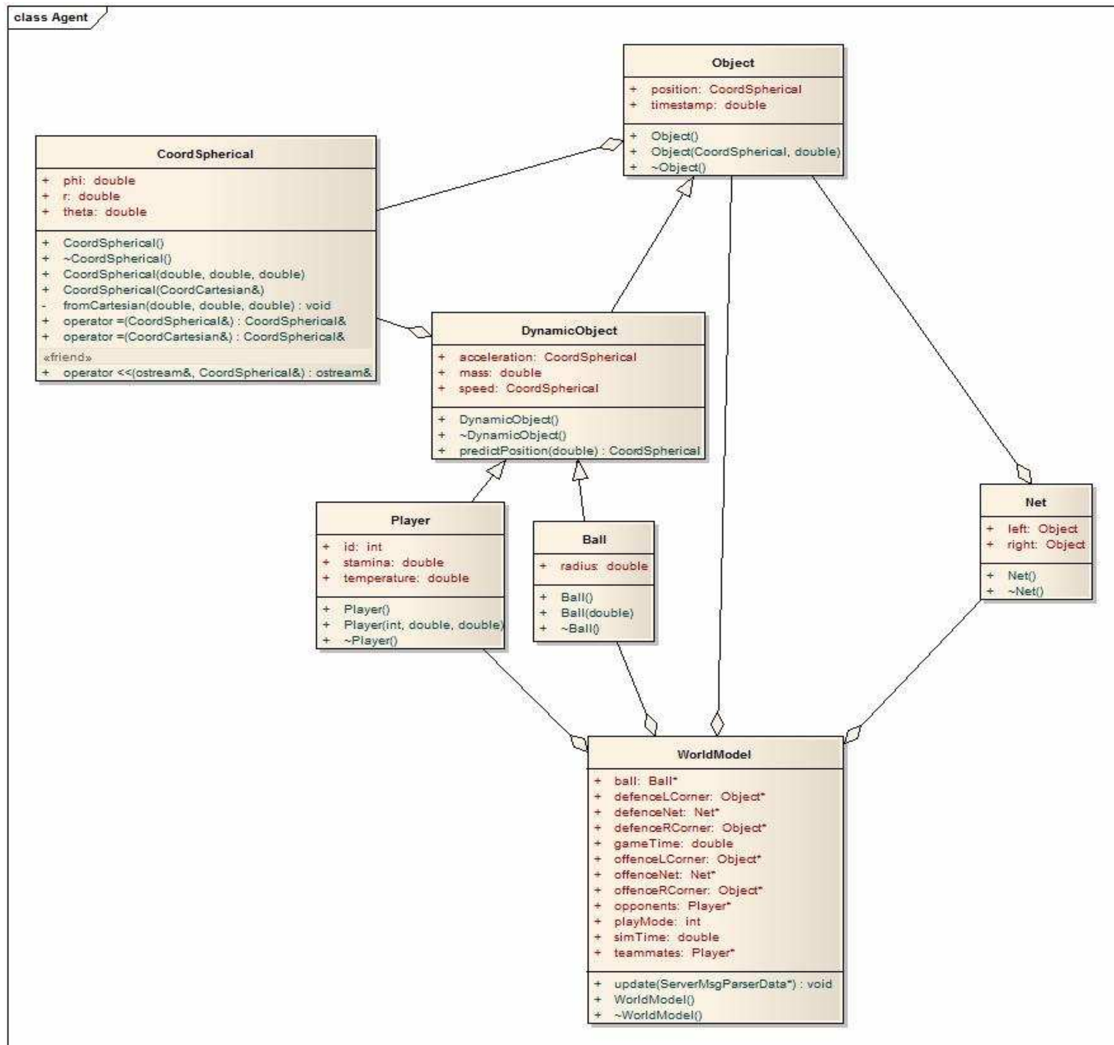
```
void WorldModel::update(ServerMsgParserData * parsedData)
```

Táto metóda umožňuje hráčovi "vidieť" – updatuje všetky informácie o objektoch v modeli sveta. Ako parameter dostane vyparsované dáta z parsera, a následne aktualizuje simulačný a herný čas, herný mód, pozíciu lopty, a pod. Volá sa v každej simulačnej slučke, takže informácie sú vždy aktuálne.

Väčšina objektov v modele sveta je zložených z niekoľkých menších objektov, prípadne dedí od iných. Všetky typy objektov, ktoré sme vytvorili, sme umiestnili do balíka Objects.

Objekty v modeli sveta

Tento balík obsahuje všetky nami používané objekty. Koreňom hierarchie je trieda Object – od neho je potom odvodený DynamicObject – dynamický objekt, ktorého konkrétne inštancie sú lopta a hráča. Zvyšné objekty ako rohové zastávky, bránky a pod. nie sú pohyblivé, a teda sú odvodené od typu Object. Hierarchia objektov je prehľadne znázornená na obrázku Obr. 43:



Obr. 43: Hierarchia základných typov objektov v modeli sveta

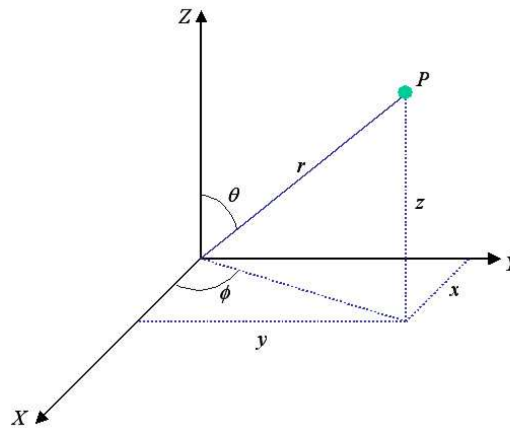
Trieda Object reprezentuje všeobecný typ objektu. Najdôležitejšie atribúty sú pozícia, a timestamp, čo je vlastne čas, kedy bol daný objekt videný na danej pozícii naposledy. V praxi to znamená, kedy bola informácia o objekte naposledy aktualizovaná. Aktualizácia sa deje vždy, keď príde správa zo servera, t.j. defaultne v časových intervaloch 20 ms. Keďže sa implicitne predpokladá, že Object reprezentuje statický, polohu nemeniaci objekt, tak inštancie tohto objektu sa využívajú iba v triede Net, ktorá predstavuje bránu. Keďže najdôležitešie informácie o bráne sú jej rozmery, rozhodli sme sa bránu reprezentovať dvoma objektami Object – každý z nich predstavuje 1 tyčku. Je tak jednoznačne definovaný priestor, kam môže hráč umiestniť loptu. Hornú žrď sme sa rozhodli nebrať do úvahy, keďže hráč momentálne nevie kopať loptu inak ako po zemi. Samozrejme, je možné ju kedykoľvek pridať.

Od objektu Object je ďalej odvodený špeciálny typ objektu – dynamický objekt reprezentovaný triedou DynamicObject. Pridáva atribúty špecifické pre hýbajúce sa objekty, ako je napríklad lopta či hráč, t.j. zrýchlenie, hmotnosť, a rýchlosť. Obsahuje aj metódu predictPosition, ktorá má v budúcnostri slúžiť na to, aby na základe predchádzajúcecj rýchlosti a smeru dokázala predpovedať predpokladanú pozíciu hráča. Keďže úlohou nášho tímu bolo implementovať základné zručnosti hráča ako je ovládanie kĺbov, a následne nízkoúrovňové pohyby ako chôdza

či vstávanie, a nezaoberali sme sa tímovou hrou, implementovanie tejto metódy nemalo pre nás nijaký význam a prenechali sme ho na budúcoročný tím.

DynamicObject má 2 inštancie – hráča (Player) a loptu (Ball). Hráča obsahuje základné atribúty ako je jeho id (tím by mal mať 11 jednoznačne identifikovateľných hráčov), stamina (energia hráča, ktorá sa znižuje priamo úmerne s jeho aktivitou) a temperature (teplota hráča). Trieda Ball popisujúca loptu pridáva prakticky iba jeden atribút – a to polomer lopty.

Poloha všetkých objektov môže byť reprezentovaná dvoma triedami CoordSpherical a CoordCartesian. Trieda CoordSpherical reprezentuje každú polohu prostredníctvom sférických súradníc. Jej základnými atribútmi sú teda radius a 2 uhly – theta a phi Obr. 44



Obr. 44: Sférický súradnicový systém

Ak máme teda 2 body, bod S a bod P, tak potom vo sférickom súradnicovom systéme je ich vzájomná poloha reprezentovaná usporiadanou trojicou (r, φ , θ), pričom:

- r predstavuje vzdialenosť oboch bodov
- uhol φ predstavuje orientovaný uhol určený vrcholom S, začiatočným ramenom v priemete priamky r do roviny xy a koncovým ramenom v smere osi x
- uhol θ predstavuje orientovaný uhol určený vrcholom S, začiatočným ramenom r, a koncovým ramenom v smere osi z

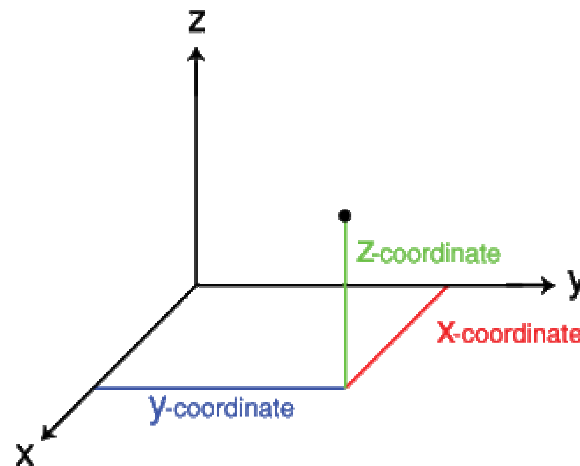
Trieda CoordSpherical poskytuje aj konverznú metódu na vytvorenie sférických súradníc z karteziánskych. Konverzný vzorec je nasledovný:

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\phi = \tan^{-1}\left(\frac{y}{x}\right)$$

$$\theta = \cos^{-1}\left(\frac{z}{\sqrt{x^2 + y^2 + z^2}}\right)$$

Trieda CoordCartesian zase reprezentuje každú polohu prostredníctvom známeho súradnicového systému troch osí. Každý bod v karteziánskom súradnicovom systéme je reprezentovaný usporiadanou trojicou (x,y z) - Obr. 45.



Obr. 45: Karteziánsky súradnicový systém

Vzdialenosť dvoch bodov sa vykonáva vzájomným odčítaním ich x-ových, y-ových a z-ových súradníc. Trieda CoordCartesian poskytuje aj konverznú metódu na vytvorenie karteziánskych súradníc zo sférických. Konverzný vzorec je nasledovný:

$$x = r\sin(\theta)\cos(\phi)$$

$$y = r\sin(\theta)\sin(\phi)$$

$$z = r\cos(\theta)$$

6.7 Ovládanie kĺbov hráča

Model hráča soccerbot056 (Obr. 1), ktorý používame, má dva typy kĺbov, Hinge (Obr. 46) a Universal (Obr. 47). Kĺb typu Hinge má jednu os otáčania a kĺb typu Universal ich má dve. Tieto dva typy kĺbov implementujú triedy HingeJoint a UniversalJoint, ktorých spoločným predkom je trieda Joint.

Trieda Joint

Trieda Joint je abstraktná trieda, v ktorej sú definované spoločné atribúty a metódy oboch typov kĺbov. Spoločné atribúty s popisom sú uvedené v Tab. 13.

Názov	Typ	Popis
name	char *	Názov kĺbu.
simTime	double	Aktuálny čas simulácie v sekundách.
deltaSimTime	double	Zmena simulačného času od poslednej aktualizácie kĺbu v sekundách.
method	enum JointMoveMethod	Spôsob pohybu kĺbu. Ten može byť: CONSTANT alebo APPROXIMATING
startTime	double	Začiatok vykonávania pohybu. Nastaví sa simulačný čas.
timeout	double	Maximálny čas pohybu v sekundách.
lastRecalc	double	Čas posledného prerátania pohybu.

Tab. 13: Atribúty triedy Joint

Trieda Joint má štyri virtuálne metódy a tie musia byť preto implementované aj v dedičných triedach. Ide o dve metódy prepočítavania pohybu kĺbu recalculateConstant, recalculateApproximating a dve metódy buildMessage a isMoving. Všetky metódy triedy Joint sú uvedené a popísané v Tab. 14

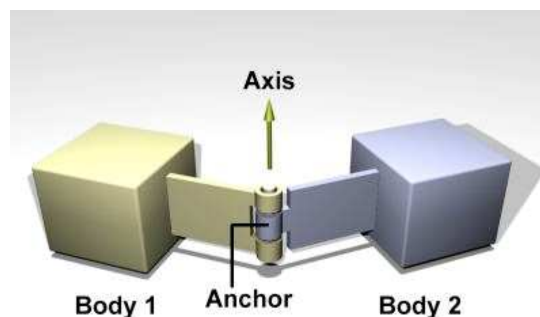
Názov	Návratová hodnota	Popis
Joint		Konštruktor. Inicializujú sa tu všetky atribúty triedy.
~Joint		Deštruktor.
update	void	Aktualizácia kĺbu. Aktualizuje sa simulačný čas podľa parametra simTime.
buildMessage	void	Metóda vytvorí pre server správu o kĺbe a zapíše ju ako reťazec na určené miesto podľa parametra message.
isMoving	bool	Vracia sa true, ak je kĺb v pohybe, inak false.
recalculate	void	Prepočítanie pohybu.
recalculated	bool	Vracia true, ak bol kĺb v aktuálnom simulačnom čase (kroku) prepočítaný, inak false.
recalculateConstant	void	Výpočet pohybu kĺbu spôsobom CONSTANT.
recalculateApproximating	void	Výpočet pohybu kĺbu spôsobom APPROXIMATING.

Tab. 14: Metódy triedy Joint

Metóda recalculate má na starosti výber spôsobu prepočítavania pohybu podľa atribútu method. Implementované sú dva spôsoby pohybu, CONSTANT a APPROXIMATING. Tie musia byť pre každý typ kĺbu naprogramované samostatne v metódach recalculateConstant a recalculateApproximating (v príslušných dedičných triedach). Oba spôsoby pohybu sú vysvetlené nižšie.

Trieda HingeJoint

Ako už bolo spomenuté, trieda HingeJoint dedí od abstraktnej triedy Joint. Táto trieda implementuje ovládanie kĺbu typu Hinge, teda kĺb s jednou osou otáčania - Obr. 46. Na ovládanie kĺbov tohto typu sú implementované dve metódy moveTo a moveBy. Metóda moveTo slúži na nastavenie kĺbu na zadaný uhol a metóda moveBy relatívne posunie kĺb o nastavený uhol.



Obr. 46: Kĺb typu Hinge

V Tab. 15 je opísaný význam jednotlivých parametrov metódy moveTo. Oproti metóde moveTo je rozdiel metódy moveBy v prvom parametri. V metóde moveBy je prvým parametrom relatívny uhol. Ten sa použije na výpočet konečného uhla, ktorý sa predá metóde moveTo. Teda metóda moveBy je implementovaná pomocou metódy moveTo.

Parameter	Typ	Popis
_setAngleX	double	Uhol, na ktorý sa má kĺb nastaviť.
_gainX	double	Počiatočná rýchlosť kĺbu udaná v percentách. 100% znamená otočenie na požadovaný uhol za jeden simulačný krok. Teda percento rýchlosti súvisí s konštantou SIMULATION_STEP definovanou v súbore Joint.h
interceptMove	bool	Nastaví sa na true, ak chceme prerušiť práve vykonávaný pohyb. Ak je tento parameter nastavený na hodnotu false, tak sa dokončí už začatý pohyb.
_method	enum JointMoveMethod	Spôsob pohybu kĺbu. Ten môže byť: CONSTANT APPROXIMATING
_timeout	double	Maximálny čas pohybu v sekundách. Po prekročení tohoto času vracia metóda moveTo hodnotu true tak, ako keby bol pohyb dokončený. Kĺb sa však nezastaví, ale pokračuje v pohybe.

Tab. 15: Parametre metódy moveTo pre kĺb typu Hinge

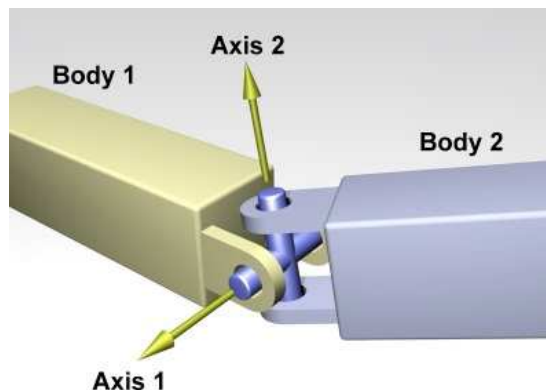
Jednotlivé atribúty triedy HingeJoint, tak ako aj ich význam, sú popísane v Tab. 16.

Názov	Typ	Popis
actAngleX	double	Aktuálny uhol v osi kĺbu v radiánoch.
setAngleX	double	Nastavený uhol v osi kĺbu
gainX	double	Premenná na výpočet rýchlosti pohybu.
movingX	bool	True ak sa kĺb hýbe, inak false.
timeoutReseted	bool	True ak bol vynulovaný timeout, inak false.
minStopDeg	double	Ohraničenie pohybu kĺbu – minimálny uhol v radiánoch.
maxStopDeg	double	Ohraničenie pohybu kĺbu – maximálny uhol v radiánoch.
speedX	double	Aktuálna rýchlosť otáčania kĺbu v radiánoch za sekundu.

Tab. 16: Atribúty triedy HingeJoint

Trieda UniversalJoint

Podobne ako trieda HingeJoint, aj táto trieda vychádza z rodičovskej triedy Joint. Implementuje ovládanie kĺbu typu Universal, teda kĺbu s dvomi osami otáčania (Obr. 47). Princíp ovládania kĺbu je tiež rovnaký ako pri triede HingeJoint s tým rozdielom, že trieda UniversalJoint manažuje o jednu os kĺbu navyše. Parametre a atribúty tejto triedy súvisiace s osami x, y sú rozlíšené sufixom x resp. y.



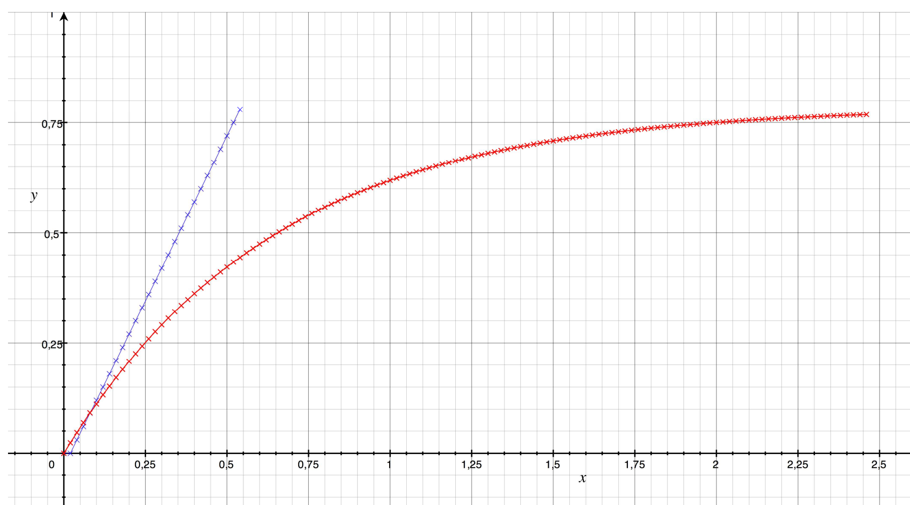
Obr. 47: Kĺb typu Universal

Pohyb typu CONSTANT

Spôsob CONSTANT funguje tak, že sa kĺb počas celého pohybu otáča konštantnou rýchlosťou. Pohyb sa zastaví, keď sa uhol osi kĺbu priblíži na požadovaný uhol s presnosťou určenou konštantou JOINT_ANGLE_ERROR_CONSTANT. Táto konštanta je definovaná v súbore Joint.h. Na Obr. 48 vidíme modrou farbou znázornený graf pre spôsob CONSTANT. Na osi x je čas v sekundách a na osi y uhol osi otáčania kĺbu v radiánoch. Obrázok bol vytvorený zo zaznamenaných hodnôt pri otáčaní kĺbu ľavého lakťa hráča. Pohyb bol iniciovaný kódom v metóde calculate triedy odvodenej od triedy LowSkill:

```
double timeout = 20.0;
SKILL_BEGIN_PHASE(0)
      SKILL_TIMEOUT_RESET
SKILL_END_PHASE_CONTINUE
SKILL_BEGIN_PHASE(1)
      SKILL_CALCULATE(elbowL->moveTo(45.0, 3.0, false, CONSTANT,
timeout))
SKILL_END_PHASE
SKILL_END(2)
```

Význam a použitie jednotlivých makier je vysvetlený ďalej.



Obr. 48: Dva typy pohybu kĺbov

Pohyb typu APPROXIMATING

Princíp tohoo spôsobu pohybu kĺbu spočíva v spomaľovaní rýchlosti otáčania s približovaním sa ku koncovému uhlu. Rýchlosť tak nie je závislá od času, ale od rozdielu uhlov. Aktuálna rýchlosť pohybu sa vypočíta vynásobením zosilnenia s rozdielom medzi aktuálnym uhlom a nastaveným uhlom kĺbu:

```
speedX = gainX * (setAngleX - actAngleX) / (SIMULATION_STEP * 100.0);
```

Následne sa vypočítaná rýchlosť percentuálne znormalizuje podľa veľkosti simulačného kroku. Na Obr. 48 vidíme červenou farbou znázornený pohyb kĺbu ľavého lakťa hráča práve spôsobom APPROXIMATING. Vidíme, že sa aktuálny uhol kĺbu každým krokom pomalšie a pomalšie približuje k nastavenému uhlu, teda konverguje k cieľu.

Zložený pohyb

K vykonaniu zloženého pohybu hráča sme pristupovali dvoma spôsobmi. Prvý bol taký, že sme si pre zvolené časové intervaly naprogramovali pohyby jednotlivými kĺbmi:

```
  // padnutie - 4 sekundy
        if (time<4)
                {
                player->ankleL->moveTo(90,100,0,0,false, APPROXIMATING);
                player->ankleR->moveTo(90,100,0,0,false, APPROXIMATING);
                }
   // vstavanie - 10 sekund
        if (time>4 && time<6)
                {
                player->shoulderL->moveTo(-230,170,0,0,false,APPROXIMATING);
                player->shoulderR->moveTo(-230,170,0,0,false,APPROXIMATING);
                }
        if (time>6 && time<9)
                {
  ...
```

Každý časový interval predstavoval fázu zloženého pohybu. V jednej fáze sa mohlo pohnúť viacerými kĺbmi naraz.

Druhý spôsob, ktorý sme použili, už nebol závislý od času, ale od ukončenia pohybu všetkých kĺbov v danej fáze. Kĺb, pokiaľ sa hýbe, vracia hodnotu false (metóda moveTo alebo moveBy). Až keď sa dostane do konečného uhla, teda sa prestane hýbať, vracia hodnotu true. Výnimku tvorí prípad, kedy vyprší časový limit timeout. Vtedy vracia kĺb takisto hodnotu true a to kvôli tomu, aby bolo možné nejakým spôsobom fázu ukončiť (príkazy vo fáze sa volajú každý simulačný krok). Mohol by totiž nastať prípad, kedy by sa kĺb do požadovaného stavu nepodarilo dostať v rozumnom čase. Ak pre kĺb vypršal časový limit a chceme s ním v ďalšej fáze pohnúť s inými parametrami, musíme časový limit vynulovať (metóda resetTimeout) alebo prerušiť jeho aktuálny pohyb (interceptMove = true v metóde moveTo alebo moveBy). Na zjednodušenie programovania fáz sú vytvorené makrá (Tab. 17).

Názov	Popis
SKILL_BEGIN(X)	Zahájenie pohybu zloženého z fáz. Parameter X je číslo fázy. Čísluje sa od nuly.
SKILL_BEGIN_PHASE(X)	Začiatok fázy X.
SKILL_CALCULATE(...)	Pohyb kĺbom. Ako parameter sa uvedie metóda na pohyb doného kĺbu.
SKILL_WAIT(X)	Čakanie X sekúnd.
SKILL_END_PHASE	Ukončenie fázy a návrat.
SKILL_END_PHASE_CONTINUE	Ukončenie fázy a pokračovanie na ďalšiu fázu v tom istom simulačnom kroku.

SKILL_END(X)	Ukončenie pohybu zloženého z fáz.
SKILL_TIMEOUT_RESET	Vynulovanie timeout-u pre všetky kĺby.

Tab. 17: Makrá na programovanie fázového pohybu

V nasledujúcom príklade si ukážeme zložený pohyb cez fázy. Najprv sa inicializuje pohyb:

```
SKILL_BEGIN(0)
```

Tu sa testuje, či sme vo fáze 0 a vynulujú sa všetky timeout-y kĺbov, aby bolo možné v nasledujúcej fáze uskutočniť pohyb. Potom nasledujú jednotlivé fázy zloženého pohybu:

```
//starting position
SKILL_BEGIN_PHASE(1)
        SKILL_CALCULATE(ankleL->moveTo(90.0, 5.0, 0.0, 5.0, false, APPROXIMATING, timeout))
        SKILL_CALCULATE(ankleR->moveTo(90.0, 5.0, 0.0, 5.0, false, APPROXIMATING, timeout))
        SKILL_CALCULATE(kneeL->moveTo(0.0, 5.0, false, APPROXIMATING, timeout))
        SKILL_CALCULATE(kneeR->moveTo(0.0, 5.0, false, APPROXIMATING, timeout))
        SKILL_CALCULATE(legL->moveTo(0.0, 5.0, false, APPROXIMATING, timeout))
        SKILL_CALCULATE(legR->moveTo(0.0, 5.0, false, APPROXIMATING, timeout))
        SKILL_CALCULATE(hipL->moveTo(0.0, 5.0, 0.0, 5.0, false, APPROXIMATING, timeout))
        SKILL_CALCULATE(hipR->moveTo(0.0, 5.0, 0.0, 5.0, false, APPROXIMATING, timeout))
        SKILL_CALCULATE(armL->moveTo(0.0, 5.0, false, APPROXIMATING, timeout))
        SKILL_CALCULATE(armR->moveTo(0.0, 5.0, false, APPROXIMATING, timeout))
        SKILL_CALCULATE(elbowL->moveTo(0.0, 5.0, false, APPROXIMATING, timeout))
        SKILL_CALCULATE(elbowR->moveTo(0.0, 5.0, false, APPROXIMATING, timeout))
        SKILL_CALCULATE(shoulderL->moveTo(0.0, 5.0, 0.0, 5.0, false, APPROXIMATING, timeout))
        SKILL_CALCULATE(shoulderR->moveTo(0.0, 5.0, 0.0, 5.0, false, APPROXIMATING, timeout))
SKILL_END_PHASE
…
```

Zložený pohyb sa ukončí makrom:

```
SKILL_END(2),
```

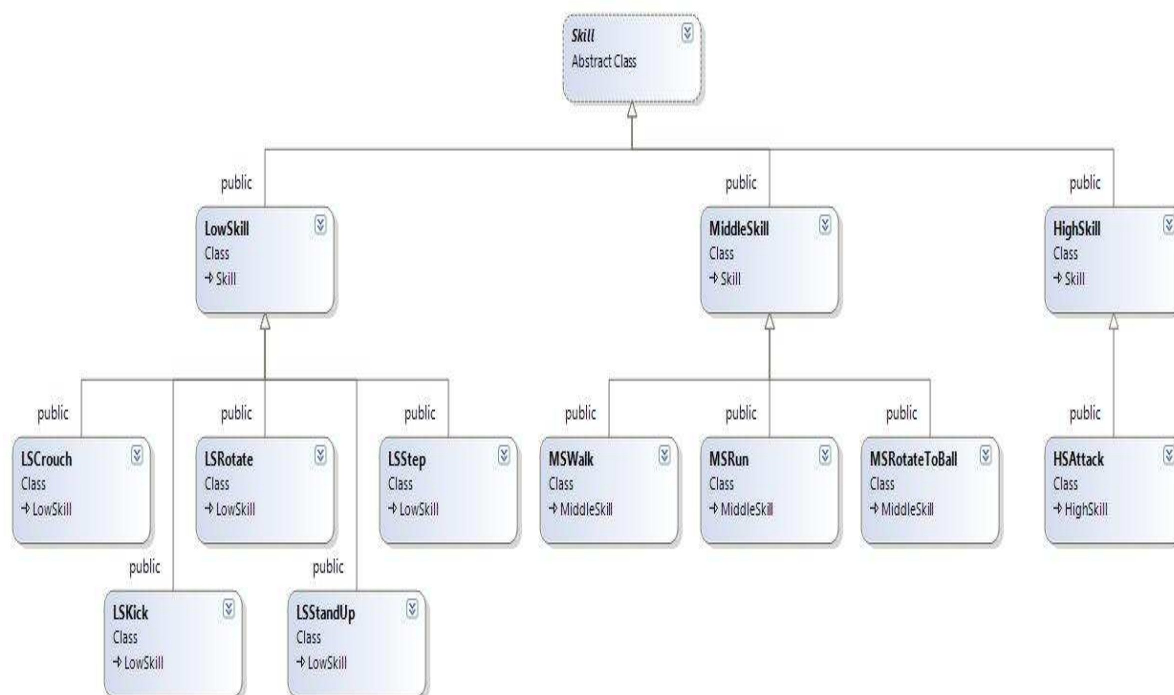
kde 2 je číslo fázy po poradí a ukončenie pohybu sa berie ako samostatná fáza.

6.8 Zručnosti agenta

Správanie hráča je jeho schopnosť vykonávať určité zručnosti. Na zručnosti využíva agent triedu Skills. Samotná trieda Skills nemá sama o sebe žiadnu funkcionalitu, táto trieda je abstraktná. Všetky zručnosti agenta sú odvodené od triedy Skills. Trieda Skills má abstraktnú funkciu *virtual int calculate (Skill ** lowerSkill)* , ktorú musia implementovať všetky zdedené zručnosti. Výnimkou sú zručnosti LowSkill, MiddleSkill a HighSkill, ktoré slúžia na rozdelenie zručnosti na nízke, stredné a vyššie. Funkcia calculate vrácia hodnotu 0 ak sa daná zručnosť ukončila. Trieda Skills má tiež virtuálnu funkciu *virtual int skillType ()*, ktorá vracia typ aktuálnej zručnosti.

Nižšie zručnosti slúžia na základnú funkcionalitu agenta. Sem patria zručnosti LSStand, LSCrouch, LSFallforward, LSStandup, LSKick, LSRotate, LSStep, LSMoveup, LSMovedown. Medzi stredné zručnosti patria MSKneebending, MSWalk, MSStandup, MSRun a medzi vyššie zručnosti patria HSAttack, HSTest. Zručnosti sú teda hierarchizované a platí, že zručnosti nižšieho stupňa sa môžu volať len zo zručností vyššieho stupňa. Napríklad zručnosť HSAttack slúžiaca na útočenie na súpera zistí, že agent je ďaleko od lopty tak zavolá zručnosť MSRun na miesto k lopte. Zručnosť MSRun potom zistí, či je agent na zemi – vtedy zavolá zručnosť LSStandup, alebo ak je v stabilnom stoji bude volať zručnosť LSStep až kým sa nepriblíži k lopte.

Príklad hierarchie zručností je na Obr. 49. Tento diagram neobsahuje úplne všetky vytvorené triedy pre zručnosti, ale len vybrané z nich.

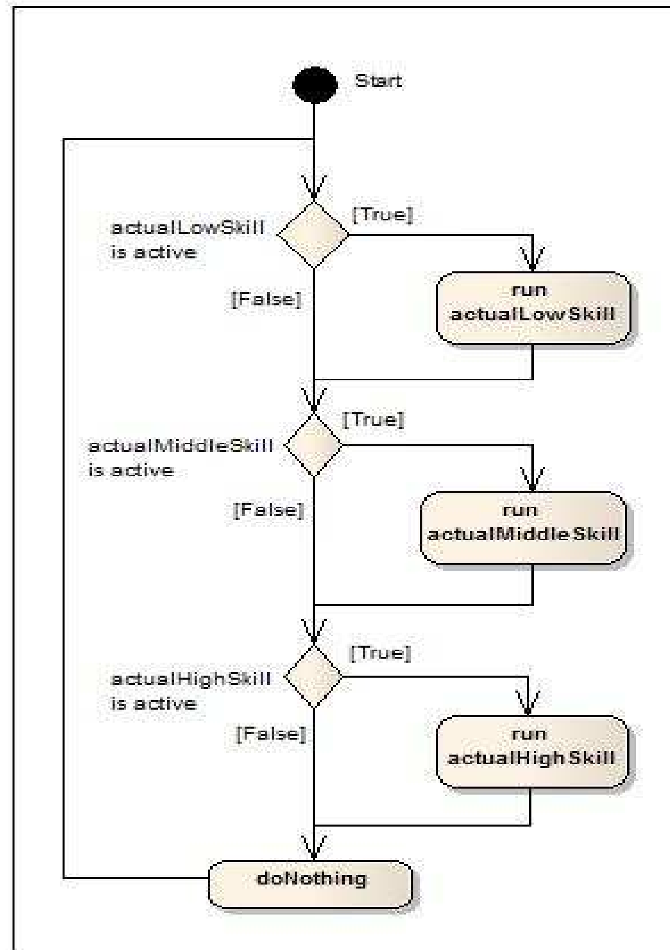


Obr. 49: Hierarchia zručností agenta

V rámci implementácie sme implementovali triedy pre zručnosti uvedené na Obr. 49. No z časových dôvodov nebola implementovaná ich celá funkcionalita. Niektoré zručnosti boli implementované ako testovacie, napríklad jednoduchý typ chôdze alebo vstávania.

Správanie agenta

Volanie zručnosti sa vykonáva v metóde behave() triedy Agent, ktorá zabezpečuje vykonávanie jednotlivých zručností a teda správanie sa agenta. Zručnosti sa volajú od najnižších, ktoré sú prvoradé, až po najvyššie. Diagram činností metódy behave() je na Obr. 50



Obr. 50: Správanie sa agenta

Z diagramu vidíme postupnosť vykonávania jednotlivých zručností, pričom sa postupuje vždy od najnižších po vyššie. Aj máme nejakú aktívnu nižšiu zručnosť, tak ju vykonávame. Až keď sa skončí jej vykonávanie, vtedy sa spustí nejaká stredná zručnosť hráča, ktorá väčšinou predstavuje len výber a nastavenie nejakej najnižšej zručnosti.

Môžeme si to predstaviť na prípade chôdze. Chôdza je sekvencia krokov a jeden krok nech predstavuje najnižšiu zručnosť. Chôdza predstavuje strednú zručnosť. Keď sa nejaký krok práve vykonáva, vtedy je síce aktívna aj stredná zručnosť chôdze, ale je v nejakom stave „čakania“, kým sa neskončí vykonávanie toho konkrétneho kroku. Keď je krok ukončený, začne sa vykonávať stredná zručnosť chôdze a to takým spôsobom, že znovu spustí najnižšiu zručnosť kroku s nejakými vhodnými parametrami.

Samozrejme z principiálneho hľadiska je možné, aby stredná zručnosť mala možnosť „zasiahnuť“ do vykonávania najnižšej zručnosti, no my sme sa sústredili len na ten prípad keď sa vždy čaká na ukončenie konkrétnej zručnosti.

Implementované zručnosti

V tejto časti uvedieme tie zručnosti, ktoré sme implementovali. Väčšina zručností predstavujú len určitý prototyp a implementovali sme ich kvôli otestovaniu implementácie hýbania kĺbmi. Implementované zručnosti sú nasledovné:

- Vstávanie zo zeme – dva typy vstávania

- Chôdza – dva typy chôdze
- Otáčanie sa na mieste

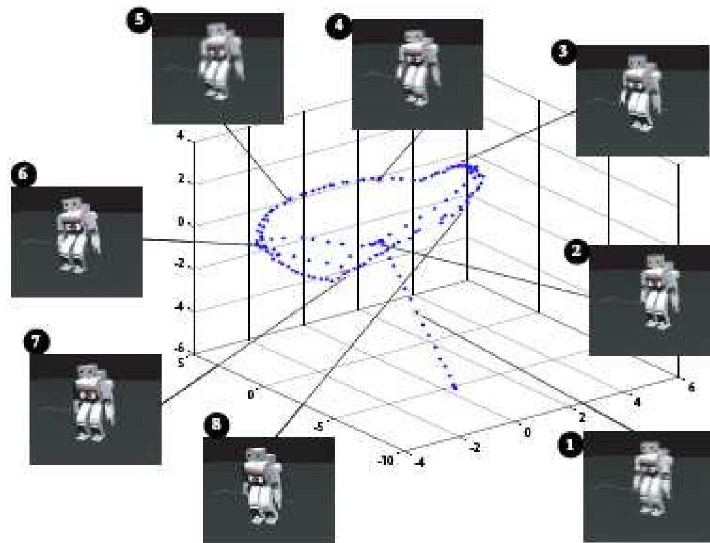
Všetky pohyby boli implementované staticky, to znamená nejakou dopredu danou postupnosťou pohybov kĺbov.

6.9 Učenie sa pohybov pomocou prediktívneho riadenia

Učenie sa humanoidných pohybov pomocou prediktívneho riadenia humanoidným robotom znamená, že sa odoberú dáta z humanoida (najčastejšie človeka), ktorý vykonáva pohyb, ktorý chceme nášho robota naučiť [22]. Tieto dáta sa odoberajú klasicky tak, že sa na telo (napr. na kĺby) pripevnia nejaké značky, ktoré sa zosnímajú. Takto zosnímané značky sa spracujú a vyhodnotí sa, kde sa jednotlivé časti humanoida pri pohybe nachádzali. Na Obr. 51 je znázornený zosnímaný pohyb robota.

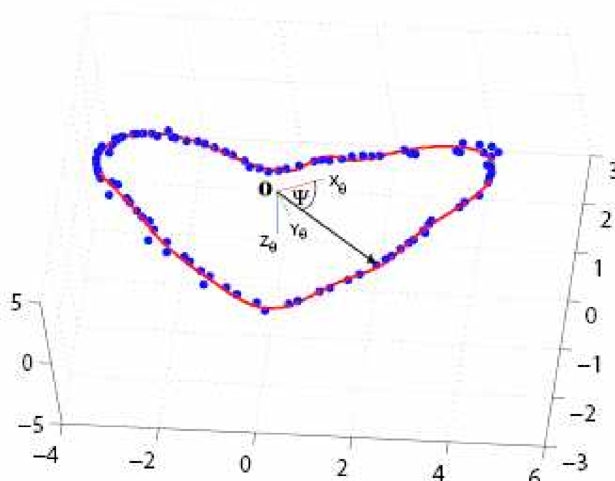
Obr. 51: Zosnímaný pohyb robota

Keď už máme požadované dáta a vieme, kde sa jednotlivé časti humanoida v danom čase nachádzali, pristúpime k dimenzionálnej redukcii, aby sme s dátami pracovali ľahšie. To znamená, že zosnímané dáta transformujeme na výchylky oproti nejakej východzej pozící (Obr. 52) (Dáta z gyroskopu).



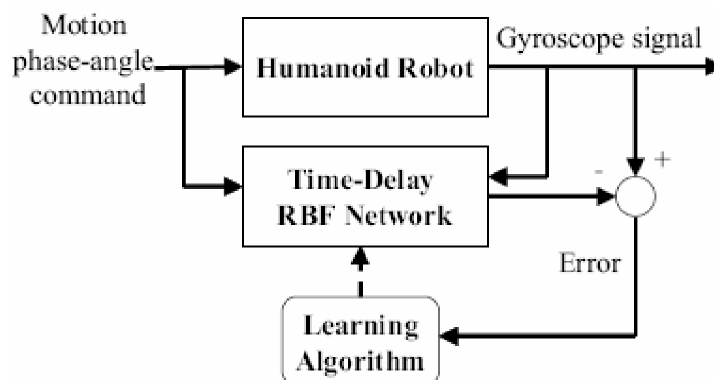
Obr. 52 Zosnímaný pohyb transformovaný na výchylky

Ak zosnímaný pohyb bol periodický, tak výchylky resp. transformované dáta zo zosnímaného pohybu vytvoria uzatvorenú krivku. Potom sa tieto výchylky aproximujú tak, aby sme z nich boli schopní vygenerovať vzor pre náš pohyb (Obr. 53).



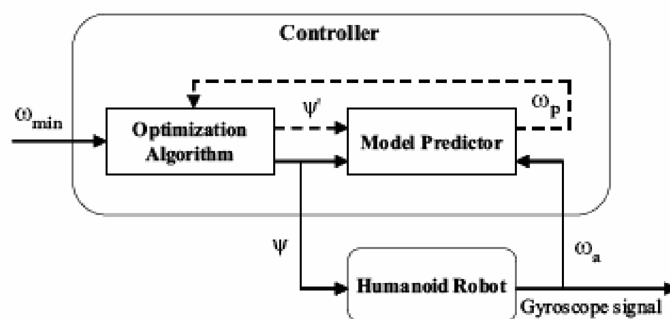
Obr. 53 Vygenerovaný vzor z výchyliek

Keď už máme vygenerovaný vzor, je možné ho spätne namapovať z redukovaného priestoru do priestoru, v ktorom sme zosnímali dáta. Teraz sa použije prediktor. Úlohou prediktora je z možných akcií vybrať takú, ktorá umožní robotovi dostať sa do ďalšej požadovanej pozície. Takže prediktor vyberie akciu na vykonanie. Prediktor je znázornený na nasledujúcom obrázku (Obr. 54).



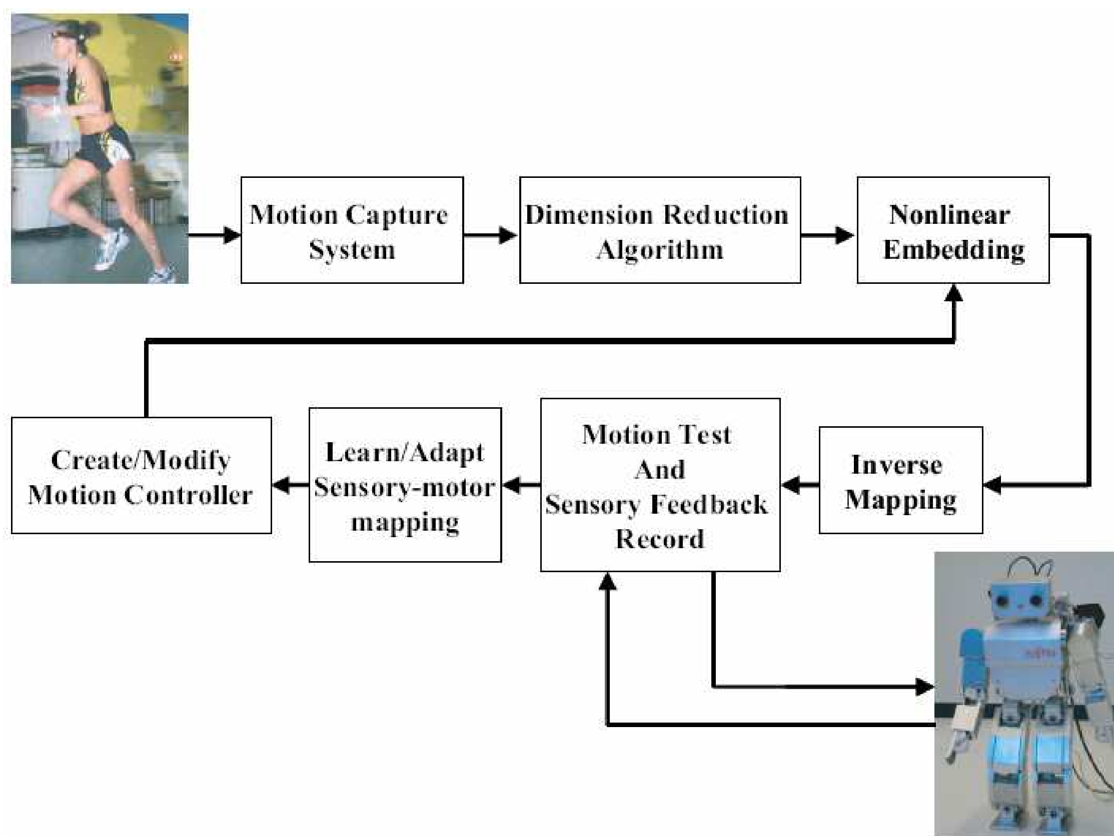
Obr. 54 Prediktor na vybratie najvhodnejšej akcie

Daná akcia sa vykoná a výsledok akcie sa zaznamená. Na základe záznamu sa prediktor učí, aká akcia je na vykonanie najvhodnejšia. Tieto kroky sa opakujú dovtedy, pokiaľ robot neoptimalizuje svoj pohyb. Zjednodušený model prediktívneho riadenia je na nasledujúcom obrázku (obr. 5).



Obr. 55 Model prediktívneho riadenia

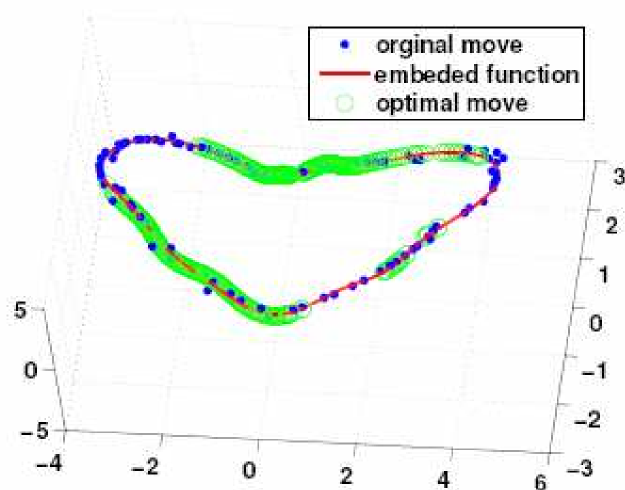
Proces algoritmu ilustruje obrázok nižšie (Obr. 56).



Obr. 56 Postup pri učení pomocou prediktívneho riadenia

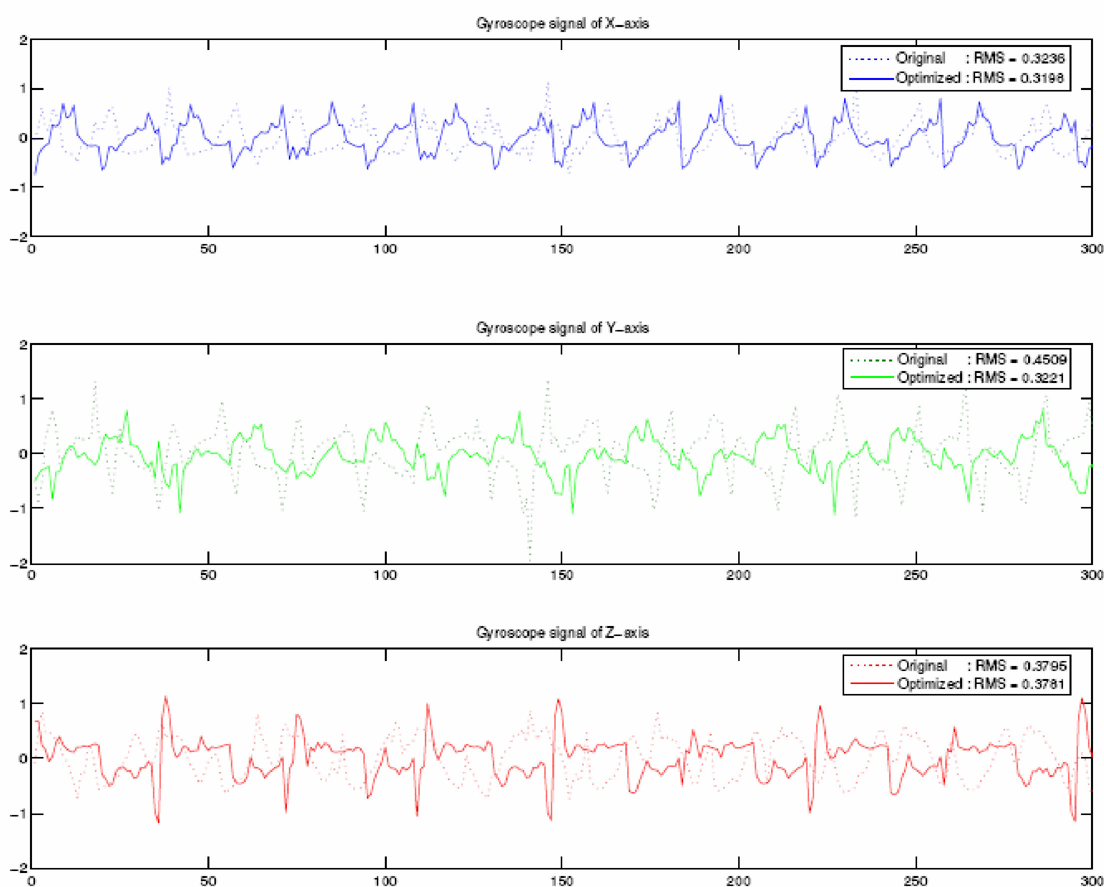
Výsledky

Pri učení sa prediktora, prediktor sa správne naučil vyberať vhodné akcie a optimalizoval svoj pohyb (Obr. 57).



Obr. 57 Optimalizovaný pohyb robota

Oproti pôvodnému pohybu, ktorý mal robot vykonávať zo zosnímaných dát sa zmenšili výchylky (vychýlenia gyroskopu), čím sa uvažovaný pohyb stal stabilnejším (Obr. 58).



Obr. 58 Stabilizovanie gyroskopu po optimalizácii pohybu

© Hviezdna jedenástka, 2008

Obr. 8 – Stabilizovanie gyroskopu po optimalizácii pohybu.

© Hviezdna jedenástka, 2008

7. Testovanie

Testovanie prebiehalo počas celého trvania implementácie. Pri testovaní a overení správnosti sme používali hlavne logovacie súbory, ktoré sa vytvárali pri behu programu. Pri testovaní implementácie zručností agenta sme využívali vizualizáciu simulácie v monitore.

Testovanie komunikácie a parsera

Pri testovaní komunikácie smerom „k nám“, t.j. či boli korektne prijaté správy, a taktiež pri testovaní parsera sme používali logovací súbor, kde sme si dali vypísať prijaté správy a taktiež vyparsované údaje. Pri tomto testovaní sme narazili na problém, že server niekedy nestíhal posielať všetky správy v prednastavenom intervale 20ms. Tento problém mali tí členovia tímu, ktorí používali virtual machine pod windowsom a tam spúšťali server. Keď bol server spustení pod OS Linux alebo Mac OS, taký problém sa nevyskytol

Testovanie funkcií pre hýbanie kĺbmi

Funkcie pre hýbanie kĺbmi predstavujú veľmi podstatnú časť implementácie, preto sme venovali veľkú pozornosť ich testovaniu. V prvom rade sme testovali, či sa dobre prerátavajú hodnoty rýchlostí pre pohyb kĺbov. Na toto sme použili výpisy z logovacieho súboru.

Ďalej sme testovali samotnú funkčnosť týchto funkcií a to tak, že sme implementovali jednoduché pohyby hráča. Napríklad sme implementovali otáčanie rukou a skúšali sme rôzne počiatočné rýchlosti pohybu kĺbu. Testovanie sme robili zvlášť pre lineárny a zvlášť pre aproximatívny typ pohybu kĺbmi. Výsledky z tohto testovania boli vizuálne, to znamená , že sme sledovali na monitore servera, aké skutočné pohyby vykonáva agent a či to súhlasí s tým, čo sme mu zadali, aby mal vykonávať

Testovanie pohybov

Implementované pohyby agenta v podstate tiež slúžili pre otestovanie funkcií pre hýbanie kĺbmi. Je to však testovanie z globálneho hľadiska, pretože tieto pohyby sú zložené z pohybov viacerých kĺbov naraz.

Takto sme empiricky zisťovali, akým spôsobom sa pohyby kĺbov navzájom ovplyvňujú a takisto, akým spôsobom vplývajú na ťažisko a stabilitu hráča. Výsledky testovania znovu boli iba vizuálne, na monitore sme pozorovali skutočne vykonávané pohyby agenta.

© Hviezdna jedenástka, 2008

8. Zhodnotenie

Počas dvoch semestrov trvania tímového projektu sme sa snažili vytvoriť architektúru hráča, ktorého sme nazvali Sirius, pre nový simulačný server, ktorého posledná verzia bola vydaná v júni predchádzajúceho roku. Tento náš vytvorený agent má predstavovať akýsi základ pre implementáciu vyšších schopností agenta.

Ako jednu z hlavných priorít sme si určili vytvorenie hráča tak, aby jeho zdrojové kódy boli skompilovateľné pod rôznymi operačnými systémami. Je to dobré kvôli tomu, aby náš zdrojový kód mohol použiť ktokoľvek bez ohľadu na to, aký operačný systém preferuje.

Ďalším cieľom bolo navrhnúť a vytvoriť architektúru hráča tak, aby obsahovala všetky dôležité časti (moduly), ktoré hráč potrebuje a aby bolo jednoduché pridávať nové súčasti.

Implementovali sme všetky základné moduly, ktoré hráč potrebuje. V prvom rade to je komunikácia hráča so serverom, to znamená prijímanie a posielanie správ vo vhodnom formáte. Ďalším modulom je parser pre správy prichádzajúce zo servera, ktorý slúži na získavanie údajov z prijatého reťazca. Ďalšími dôležitými modulmi sú model sveta a model hráča.

Ďalej sme implementovali funkcie pre ovládanie pohybu kĺbov a tieto funkcie sme následne použili pre implementáciu niektorých zručností agenta.

Počas semestra sa nám podarilo implementovať všetky časti, ktoré sme plánovali. Pri implementácii sme často museli používať a hľadať informácie v logovacích súborov. Keďže hráč je „realtime“ aplikácia, nie je možné použiť klasické debugovanie. Hľadanie chyby v tomto prípade je nie vždy jednoduchá záležitosť a museli sme riešiť viacero problémov, ktoré sa nám vyskytli počas implementácie.

Keď sme mali implementované všetky časti architektúry hráča, pokúsili sme sa vytvoriť jednoduché typy zručností, ako napríklad chôdza a vstávanie agenta. Tieto nami implementované pohyby považujeme len za určitý prototyp, ktorým sme chceli demonštrovať a overiť funkčnosť ovládania kĺbov a celej architektúry ako celku. Tieto pohyby sú implementované iba ako statické pohyby, to znamená vopred daná postupnosť elementárnych pohybov a nevyužívame pri nich žiadne informácie o ťažisku agenta. To znamená, že tieto pohyby sú veľmi citlivé na správnosť poslaných údajov zo servera. Keďže vieme, že server vnáša do údajov určitý šum, toto robí tieto pohyby dosť nestabilnými.

Možné vylepšenia

Ako bolo spomenuté vyššie, náš hráč tvorí základ pre implementáciu zručností a správania hráča. Preto vylepšenia vlastne predstavujú implementovanie týchto zručností.

Keďže približne týždeň pred odovzdaním celého projektu, vyšla nová verzia serveru, možné vylepšenie by sa mohli týkať prispôsobenia zdrojových kódov pre túto verziu servera.