

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

Robocup 3D

The A Team

Dokumentácia k inžinierskemu dielu

Vedúci tímu:

Ing. Marián Lekavý, PhD.

Členovia tímu:

Bc. Anton Balucha

Bc. Peter Borga

Bc. Tomáš Florek

Bc. Adam Pagáč

Bc. Eduard Pribula

Bc. Maros Unčík

Bc. Miroslav Vasil'

Kontakt: fiit-tim-04@googlegroups.com

Obsah

1	Úvod.....	1
1.1	Účel dokumentu	1
1.2	Cieľ projektu	1
2	Šprinty.....	2
2.1	Šprint č. 01.....	2
2.1.1	Analýza	3
2.1.1.1	Nástroj pre podporu projektu	3
2.1.1.2	Analýza fakultných hráčov.....	3
2.1.1.3	Analýza zahraničných hráčov	8
2.1.1.4	Analýza fungovania systému a pravidiel Robocup	26
2.1.2	Návrh.....	32
2.1.2.1	Vytvorenie stránky tímu	32
2.1.2.2	Nástroj pre podporu projektu	32
2.1.2.3	Vytvorenie dokumentácie	32
2.1.2.4	Výber hráča k ďalšiemu vývoju.....	32
2.1.3	Implementácia	33
2.1.3.1	Vytvorenie stránky tímu	33
2.1.3.2	Nástroj pre podporu projektu	33
2.1.3.3	Vytvorenie dokumentácie	33
2.1.3.4	Inštalácia serveru, editora pohybov a hráčov	33
2.1.3.5	Vytvorenie a editácia pohybov hráča	34
2.1.4	Testovanie	35
2.1.4.1	Vytvorenie stránky tímu	35
2.1.4.2	Nástroj pre podporu projektu	35
2.1.4.3	Vytvorenie dokumentácie	35
2.2	Šprint č. 02.....	36
2.2.1	Analýza	37
2.2.1.1	Analýza prevzatých kódov hráča JIM.....	37
2.2.1.2	Analýza najdôležitejších tried	38
2.2.1.3	Analýza miest pre rozšírenie	40
2.2.1.4	Analýza testovania.....	40
2.2.1.5	Manažment verzií	42
2.2.2	Návrh.....	43
2.2.2.1	Návrh v analýze zdrojových kódov	43
2.2.2.2	Návrh testovania	43
2.2.2.3	Návrh manažmentu verzií.....	43
2.2.2.4	Vytvorenie dokumentácie k riadeniu	43
2.2.2.5	Aktualizácia webovej stránky	44

2.2.3	Implementácia	45
2.2.3.1	Implementácia úpravy zdrojových kódov	45
2.2.3.2	Implementácia testovania	46
2.2.3.3	Implementácia manažmentu verzií	47
2.2.3.4	Vytvorenie dokumentácie k riadeniu	48
2.2.3.5	Aktualizácia webovej stránky	48
2.2.4	Testovanie	49
2.2.4.1	Testovanie úpravy zdrojových kódov	49
2.2.4.2	Testovanie správnosti testov	49
2.2.4.3	Testovanie manažmentu verzií	49
2.2.4.4	Vytvorenie dokumentácie k riadeniu	49
2.2.4.5	Testovanie aktualizácie webovej stránky	49
3	Zdroje	50

1 Úvod

1.1 Účel dokumentu

Tento dokument je vytvorený pre účel projektovej dokumentácie projektu Robocup 3D v rámci predmetu Tímový projekt v akademickom roku 2010/2011. V dokumente sú zdokumentované priebehy jednotlivých šprintov metodiky SCRUM.

1.2 Cieľ projektu

RoboCup 3D je medzinárodná súťaž, ktorá vznikla za účelom rozvoja umelej inteligencie, zaujímavou formou – hraním robotického futbalu. Snahou je podporiť výskum v tejto oblasti, pričom súťaž nie je obmedzovaná na použité technológie. Súťaž RoboCup prebieha v niekoľkých kategóriách, od virtuálnych zápasov medzi robotmi na simulačnom serveri až po skutočných robotov, ktorý behajú po ihrisku a kopú do lopty.

Simulovaný robotický futbal má na našej fakulte dlhú tradíciu. Naši študenti sa mu venujú už desať rokov a za ten čas sa objavilo množstvo tímov študentov, ktorý sa snažili vytvárať a vylepšovať programy, ktoré simulujú správanie sa futbalového hráča. Aj v našom projekte sa zaoberáme práve simuláciou robotov, čo nám umožňuje lacno, efektívne a pružne vyvíjať nové postupy tak, aby sme priniesli pokrok v tejto oblasti vývoja.

Simulovaný robotický futbal nie je finančne náročný a to je aj jeden z hlavných dôvodov, prečo sa teší vysokej obľube. Simulačný server vytvára fyzikálne podmienky, ktoré sú zhodné s reálnym svetom a preto je veľkou výhodou, že odladený kód, ktorý sa vyvinie pre simulačné prostredie sa dá následne zobrať a implementovať v skutočnom robotovi. Ten potom koná tak ako by sa nachádzal v simulácii.

Hoci pôvodne existovala táto súťaž len v dvojrozmernom priestore, v súčasnosti sa presadzuje najmä tretí rozmer. Tretí rozmer však so sebou priniesol aj problémy, ktoré sa stali určitou výzvou pre súťažiacie tímy. Vznikla tak nová oblasť, v ktorej každý tím môže kreatívne zužitkovať svoje nápady.

Simulačná liga má svoje pravidlá, podobne ako skutočný futbal, ktoré treba rešpektovať. Je to aj z toho dôvodu, že hlavným cieľom RoboCup-u je do roku 2050 vyvinúť tím autonómnych robotov, ktorí budú schopní vyhrať ľudské majstrovstvá sveta vo futbale.

Náš tím sa v rámci tohto projektu bude snažiť priblížiť k naplneniu tohto sna. Hlavným cieľom projektu bude pokračovať v úspešných šľapajach predchádzajúcich tímov. Budeme tak pokračovať v jednom z vyvinutých hráčov z minulých rokov a doplníme jeho nedostatky. Budeme sa snažiť pretvoriť navrhnuté pohyby do skutočnej hry. Dôležitým faktorom bude využitie prístupov z umelej inteligencie a robotiky, pričom sa chceme inšpirovať aj zahraničnými tímami. Naším výstupom bude hráč, ktorý bude rozšíriteľný našimi nasledovníkmi v ďalších rokoch a to nielen na úrovni návrhu ale aj implementácie.

2 Šprinty

2.1 Šprint č. 01

Číslo šprintu	01
Počet príbehov	09
Začiatok šprintu	07.10.2010
Koniec šprintu	21.10.2010

Príbehy

Vytvorenie stránky tímu
Nástroj pre podporu projektu
Vytvorenie dokumentácie
Analýza fakultných hráčov
Analýza zahraničných hráčov
Analýza fungovania systému a pravidiel Robocup
Inštalácia serveru, editora pohybov a hráčov
Vytvorenie a editácia pohybov hráča
Výber hráča k ďalšiemu vývoju

2.1.1 Analýza

2.1.1.1 Nástroj pre podporu projektu

Jedno z dôležitých úloh v prvom šprinte je určenie a nainštalovanie nástroja, ktorý bude slúžiť na podporu procesov v našom tímovom projekte. Hlavnou úlohou takéhoto nástroja bude zadávanie úloh jednotlivým členom nášho tímu, sledovanie splnenie úloh v čase, prípadne zaznamenávanie stráveného času. Rovnako by bolo vhodné, aby takýto nástroj umožňoval sledovať problémy, ktoré vznikli v súvislosti s vývojom.

V súčasnosti existuje veľké množstvo voľne dostupných nástrojov, ktoré majú prepracované funkcie a je ťažšie si medzi nimi vyberať. Sú systémy, ktoré fungujú ako stand-alone aplikácie a viac menej nedovoľujú zdieľanie informácií medzi členmi tímu, rovnako aj aplikácie, ktoré sú prístupné pomocou webu. Takéto nástroje poskytujú zvyčajne úplne zdieľanie všetkých informácií medzi všetkými členmi tímu. Práve druhá skupina systémov bola pre nás zaujímavá a preto sme sa zamerali na systémy z tejto skupiny. Do úvahy sme zobrali systémy Redmine, Jira a Dot Project, ktoré poskytujú prístup prostredníctvom webu. Umožňujú široké možnosti zaznamenávania rôznych informácií spojených s manažmentom softvéru, ako napríklad plnenie úloh v čase, pridelovanie úloh, plánovanie úloh, určovanie odhadov a podobne. Systémy sú založené na rôznych technológiách (Ruby, Java a PHP), čo nás ale neobmedzuje pri ich nasadení.

Čo sa týka nástroja na komunikáciu v tíme, bolo treba vybrať nástroj, ktorý nám umožní efektívne komunikovať medzi všetkými členmi tímu súčasne. Na výber máme opäť veľké množstvo nástrojov, ako napríklad rôzne služby založené na instant messagingu, či mailové riešenia. Riešenia, ktoré sme identifikovali boli na jednej strane platené, ale taktiež sme našli aj voľne dostupné. V našom prípade pripadajú do úvahy iba voľne dostupné riešenia a preto sme sa ďalej zaoberali nástrojom GoogleGroups.

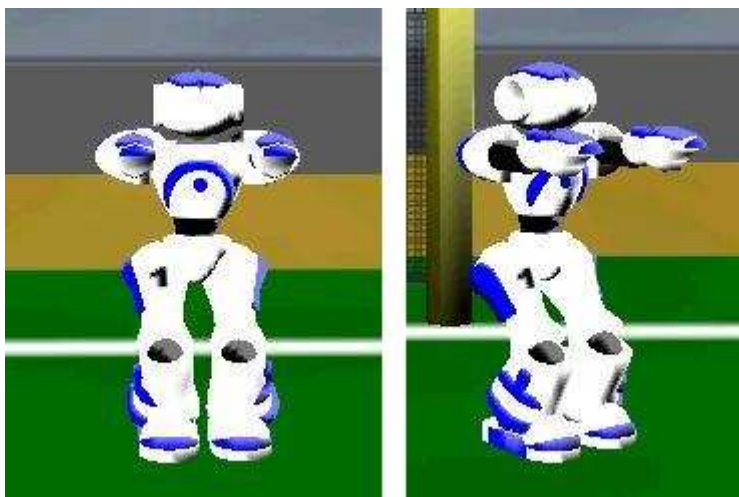
2.1.1.2 Analýza fakultných hráčov

Critical Error

Critical error je tím, ktorý bol vyvíjaný minulý rok na našej fakulte. Pri vytváraní sa jeho autori rozhodli editovať hráča tímu DreamTeam. DreamTeam využíval pri modelovaní pohybov XML súbor s presne definovaným formátom a cieľom bolo vytvoriť prepojenie medzi týmto súborom a editorom pohybov, ktorý vytvoril tím Agenty 007.

Pohyby hráča

Pohyby sú realizované v editore pohybov a následne exportované do XML. Všetky sú realizované zo základnej polohy hráča znázornenej na Obrázku 2.1 - 1.



Obrázok 2.1-1 – Základná poloha hráča

Základom tvorby pohybov hráča je udržanie stability, ktorá sa dosahuje presúvaním ťažiska na stranu. Samozrejme platí, že čím je ťažisko nižšie, tým je hráč v stabilnejšej polohe.

Chodenie do boku

Pri kráčaní doľava hráč preniesie svoje ťažisko na pravú nohu, zodvihne ľavú, nakloní sa doľava, preniesie ťažisko na svoju ľavú nohu a pritiahne pravú nohu. Tento cyklus sa opakuje, až kým sa hráč nedostane na želané miesto.

Otáčanie

Otočenie sa smerom doľava je realizované nasledovne. Hráč mierne od seba vytočí nohy a následne sa nakloní doľava, čím preniesie svoje ťažisko na ľavú nohu. Následne pokračuje zodvihnutím pravej nohy a otočením sa na ľavej nohe. Potom sa hráč vráti do základnej polohy a cyklus sa opakuje, až kým sa neotočí do želanej polohy. V jednom cykle sa hráč otočí približne o 30°.

Kopy hráča

Implementovaných bolo 5 kopov:

- Kopnutie do lopty bokom chodidla ľavou nohou v smere cca 45°
- Kopnutie do lopty bokom chodidla pravou nohou v smere cca 45°
- Kopnutie do lopty bokom chodidla ľavou nohou v smere 90°
- Kopnutie do lopty bokom chodidla ľavou nohou v smere 90°
- Kopnutie do lopty v smere dozadu ľavou nohou

Krížový kop doľava sa uskutočňuje prenesením ťažiska na ľavú nohu, zodvihnutím pravej nohy od tela do boku a následne sa už len natočí bedrový kĺb a noha v kolene. Obdobne je vykonávaný kop doprava.

Kop do boku do pravej strany je realizovaný v čase, keď sa hráč nachádza vedľa lopty. Pomocou členkov a bedrových kĺbov sa preniesie ťažisko na ľavú nohu a pravou nohou sa už len vykoná samotný kop do boku. Analogicky sa vykonáva kop do ľavej strany.

Kop dozadu je vytvorený len pre jednu nohu, a to pre ľavú. Lopta sa v tomto prípade nachádza za hráčom. Hráč prenáša ťažisko na pravú nohu, dvíha ľavú a následne ju ohýba v kolene. Nakoniec takto ohnutú nohu preniesie po osi bedrového kĺbu a loptu trafi spodkom chodidla.

Zoznam navrhnutých kopnutí

Critical Error sa snažil implementovať niekoľko ďalších kopov, v závislosti od postavenia hráča a lopty. Zoznam týchto kopov je uvedený v Tabuľke 2.1 – 1.

Id	Postavenie hráča	Pozícia lopty	Smer kopu	Realizácia
1	Vystretý	Pred hráčom	Dopredu (ďaleko)	Áno
2	Vystretý	Pred hráčom	Dopredu (do výšky)	Nie
3	Vystretý	Pred hráčom	Dopredu (presne)	Nie
4	Vystretý	Pred hráčom	K hráčovi	Nie
5	Skrčený	Pred hráčom	Dopredu	Áno
6	Skrčený	Pred hráčom	Do boku (popred hráča)	Áno
7	Skrčený	Pred hráčom	Šikmo vpred (popred hráča)	Áno
8	Skrčený	Vedľa hráča	Do boku	Áno
9	Skrčený	Vedľa hráča	Šikmo vpred	Nie
10	Skrčený	Vedľa hráča	Šikmo vzad	Nie
11	Skrčený	Za hráčom	Dozadu	Áno
12	Skrčený	Za hráčom	Do boku (poza hráča)	Nie
13	Skrčený	Za hráčom	Šikmo vzad (poza hráča)	Nie

Tabuľka 2.1-1 – Zoznam kopov

Chôdza

Chôdza bola prebratá od tímu SEU RedSun, ale kvôli šumu a nepresnostiam pri parsovaní hráč často padal na zem. Tento problém sa podaril odstrániť avšak na úkor značného úbytku rýchlosti. Tím videl využitie tejto stabilnej chôdze ako alternatívu, napríklad nastavenie hráča k lopte.

Finalizačné fázy

Finalizačné fázy slúžia na správne ukončenie vykonávaného pohybu hráča.

- Stabilization
- Rollback

Stabilization je fáza, ktorá sa uskutočňuje v prípade, že hráč už nepotrebuje vykonávať daný pohyb. Napríklad, hráč sa dostane na takú vzdialenosť k lopte, že už môže vykonávať kop, zmena polohy lopty alebo hráčov a podobne.

Rollback je fáza, ktorá sa vykonáva pri neočakávanej udalosti a hráč sa pri nej vráti do základnej polohy. Takouto udalosťou môže byť napríklad pád hráča.

Brankár

Brankár je rozdelený do troch aspektov správania sa:

- Chytanie – pohyb, postoj, zákroky
- Vybíhanie z bránky – zmenšenie streleckého uhla
- Rozohrávanie – snaha o čo najdlhší výkop

Pohyby brankára:

- Chôdza – dopredu, dozadu, doľava a doprava
- Pády – doľava, doprava
- Brankársky postoj
- Kopnutie – s čo najväčšou silou aj na úkor presnosti
- Brankárska roznožka
- Vstávanie

Hráč Agenty 007

Agent tímu Agenty 007 bol vyvinutý na predmete Tímový projekt v školskom roku 2008/2009. Tím sa inšpiroval niekoľkými ďalšími tímami, avšak najväčší vplyv na ich výsledok mal tím Hviezdna 11 z roku 2007. Na jeho vývoj bol použitý jazyk C++. Hlavným cieľom tímu, ktorý tohto hráča vyvíjal, bolo postaviť pevný a kvalitný základ pre ďalšie generácie riešiteľov tohto projektu. Do hráča bola preto vložená iba základná funkcionálna pre jeho bezchybné fungovanie. Sústredili sa na vytvorenie editora pohybov, udržiavanie rovnováhy hráča a naučili ho niekoľko základných pohybov ako vstať, zmeniť smer pohybu a kopnúť do lopty. Počas práce sa vyskytlo niekoľko problémov, ktoré neboli vyriešené a to:

- Dodržiavanie podmienok s počiatočným natočením kĺbov. Tento problém nemal v ich fáze vývoja dostatočnú prioritu.
- Výpočet zmeny natočenia kĺbu v aktuálnom cykle. Bol spravený iba výpočet na začiatku pohybu.
- Zaistenie, že sa pohyby jedného kĺbu vykonávajú sekvenčne, vykonajú korektne podľa požiadaviek. Problém je vo fáze riešenia nakoľko jeho riešenie je spojené s predchádzajúcim problémom.

Zdrojové súbory agenta boli dostatočne prehľadné a usporiadané v prehľadnej adresárovej štruktúry, avšak s minimálnym množstvom komentárov.

Tento hráč slúži ako dobrý základ, avšak je vo veľmi skorej fáze vývoja. Na tohto hráča však nadväzovali ďalšie tímové projekty a preto je pôvodný hráč tímu Agenty 007 pre náš projekt pomerne neperspektívny.

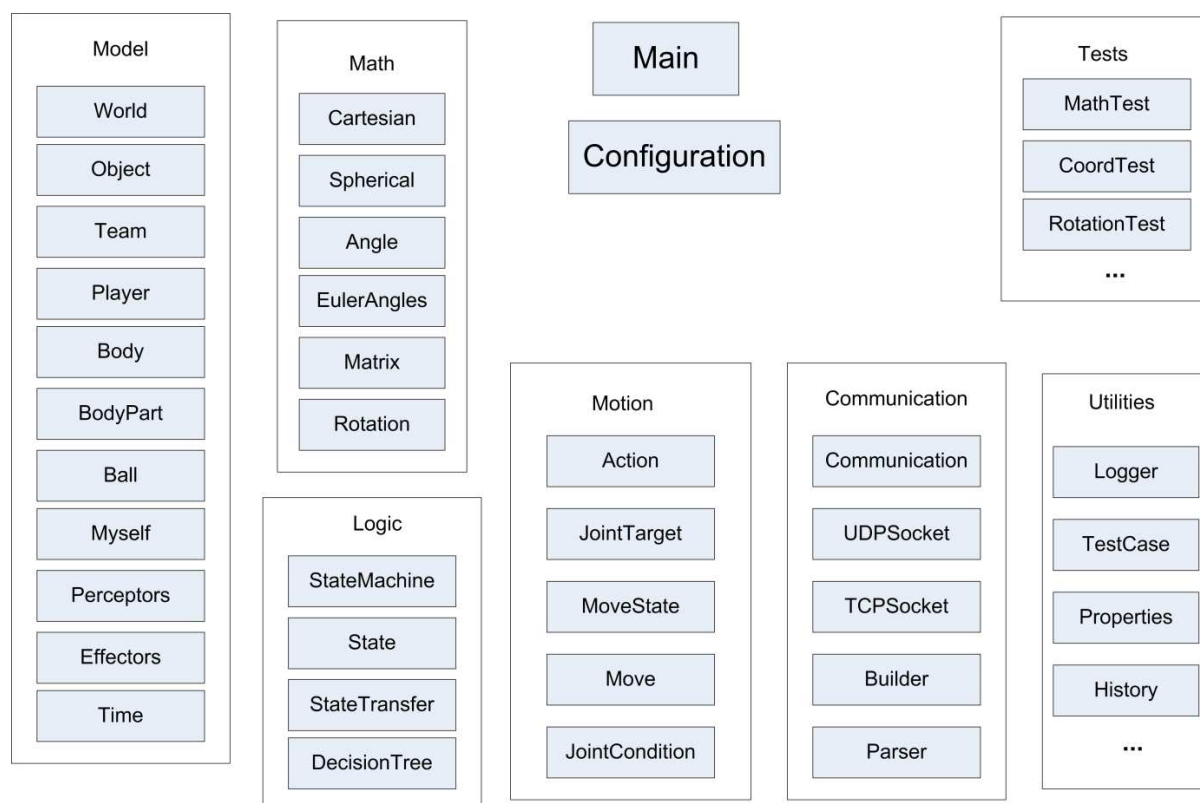
Hráč RoboKopy

Agent tímu Robokopy bol vyvíjaný týmto tímom v školskom roku 2009/2010 na predmete Tímový projekt. Jeho základ vychádza z ďalšieho fakultného hráča Agenty 007 z predchádzajúceho roku. V hráčovi bolo opravených niekoľko chýb, taktiež boli refaktorované zdrojové kódy. Spolu s agentom bol použitý a ďalej zdokonalený aj editor pohybov. Hráč bol implementovaný v jazyku C++.

Hráč sa skladá z množstva malých špecializovaných tried. Zdrojové kódy sú prehľadné aj keď pomerne slabo okomentované. Obrázok 2.1 - 2 ukazuje jednotlivé moduly projektu.

Úlohy, ktoré zabezpečujú jednotlivé moduly sú nasledovné:

- Model - uchováva informácie o stave sveta a objektov v ňom,
- Math - pomocné matematické operácie a štruktúry,
- Logic - stavový automat a prechodové funkcie,
- Motion - zabezpečenie pohybu,
- Communication – komunikácia so serverom,
- Utilities – pomocné triedy,
- Test – obsahuje TestCase niektorých tried a algoritmov.



Obrázok 2.1-2 – Moduly projektu tímu Robokopy

súbory projektu sa síce nachádzali v adresárovej štruktúre podľa zaradenia do modulu, ale po otvorení v programe Visual Studio 2008 boli všetky zliate v jednom adresári. Tento

stav mohol byť spôsobený tým, že neboli poskytnuté všetky súbory patriace k projektu tohto programu.

Oproti hráčovi Agenty 007 boli zlepšené niektoré základné pohyby:

- Chôdza – sústredili sa na stabilitu čím utrpela rýchlosť. Testy z dokumentácie však ukázali, že aj stabilita je relatívna a pri menej výkonnom PC nebola dostatočná,
- Otáčanie – maximálny uvádzaný uhol je 90°, ale kvôli strednej logike boli spravené ďalšie otočenia, pravdepodobne odstupňovanie od 0° do 90°,
- Vstávanie – vstávanie z brucha bolo len mierne upravené, avšak stále nie je stopercentne stabilné. Neskôr bolo implementované aj vstávanie z chrbta,
- Kop do lopty – boli vytvorené dve verzie – jedna stabilná a druhá nestabilná, tá druhá však neposkytovala žiadne výhody.

Okrem týchto pohybov boli implementované ešte exhibičné pohyby, z ktorých by mohol byť zaujímavý krok v bok.

V rámci ďalšej práce bol na hráčovi vylepšený matematický model a história kinetických údajov pre každý dynamický objekt. Implementované boli tiež funkcie na určenie a predikciu orientácie kamery, pozície a rýchlosti objektov. Taktiež bola implementovaná stredná logika, pričom tím umožnil jej vizualizáciu a jej tvorbu cez editor pohybov. Kvôli strednej logike boli tiež navrhnuté funkcie, určujúce v akom stave sa nachádza hráč. Taktiež bol vytvorený systém spájania pohybov.

Hráč vyzerá byť vcelku na dobrej úrovni spracovania. Je pripravený hlavne na ďalší rozvoj strednej, prípadne vyššej logiky.

2.1.1.3 Analýza zahraničných hráčov

Väčšina zahraničných tímov si svoje detailné znalosti pri vývoji Robocupu chráni. Dôvodom je, aby sa nedostali ku konkurenčným tímom, ktorým by umožnili ľahšie poraziť ostatné tímy. Preto väčšina zahraničných tímov neposkytuje detailne informácie. Na ich stránkach sa väčšinou nachádzajú len logy z predchádzajúcich zápasov a stručné predstavenie tímov. Niektoré tímy však poskytujú aj informácie o architektúre hráčov, základných princípoch nižších a vyšších znalosti hráčov, podporných nástrojoch pri vývoji hráčov, alebo aj samotných hráčov. Analýzu zahraničných tímov sme sústredili na najúspešnejšie tímy minuloročného turnaja Robocup 3D 2010 v Singapore, ale aj ďalšie zaujímavé tímy so zaujímavými technikami.

Apollo 3D

Tento tím sa stal víťazom turnaja Robocup 3D 2010 v Singapore, keď vo finále zvíťazil nad tímom NAO-Team Humboldt v predĺžení 1:0. Tím Apollo 3D [3] bol založený študentmi na Univerzite pošt a telekomunikácií Nanjing v Číne. Avšak veľa konkrétnych informácií tento tím neposkytuje. Stránka tímu je vo vývoji a poskytuje len logy z predchádzajúcich zápasov. Keďže konkrétne informácie nie sú poskytnuté, analýza tohto tímu prebiehala z veľkej časti sledovaním odohratých zápasov.

Hráč je založený na princípoch inverznej kinetiky v kombinácii s analytickými a numerickými metódami. Pri chôdzi je využitá metóda plánovania trajektórií, ktorá umožňuje dosiahnuť oveľa lepšie výsledky chôdze v 3D prostredí. Základné postavenie tímu pri zápase šiestich hráčov je 4 + 1 + 1. Chôdza hráča je relatívne rýchla a stabilná, ale hráč v konfliktných situáciách s ostatnými hráčmi padá. Hráč má definované veľmi rýchle pohyby vstavenia, ktoré je možné vidieť na Obrázkoch 2.2 - 3., 2.2 - 4. Kopy hráča sú plynulé a hráč plynule prechádza z pohybu chôdze do pohybu kopu.

Správanie hráča z taktického hľadiska je také, že sa snaží riešiť útočné akcie najčastejšie individuálne, hoci v niektorých prípadoch možno spozorovať nahrávky. V útočných akciách sa snaží obchádzať ostatných hráčov za účelom eliminácie konfliktných situácií. V rámci ušetrenia času, v niektorých situáciách nedochádza otočeniu hráča a následnom pohybe chôdze za loptou, ale okamžitým pohybom chôdze vzad.



Obrázok 2.1-3– Pohyb vstávania z brucha



Obrázok 2.1-4 – Pohyb vstávania z chrbta

NAO-Team Humboldt

NAO-Team Humboldt [4] je tím, ktorý bol založený v roku 2007 na Humboldtskej univerzite v Berlíne. Členmi tímu sú študenti a vedeckí pracovníci, ale nie len z Nemecka, ale aj Ruska, Egypta, Číny, Iraku a Iránu. Posledným úspechom tohto tímu je druhé miesto v turnaji Robocup 2010, ktoré sa konalo v Singapore. Výskum tímu je orientovaný hlavne na techniku hráča a strojové učenie s aplikáciou v kognitívnej robotike.

Hráč tímu NAO-Team Humboldt je implementovaný v C++. Architektúra hráča je rozdelená na platformovo nezávislú časť, ktorá tvorí jadro a platformovo závislú časť, ktorá tvorí logiku hráča. Architektúra hráča taktiež umožňuje ladenie kódu hráča za behu. Získané výsledky umožňuje monitorovať a vizualizovať pomocou RobotControl.

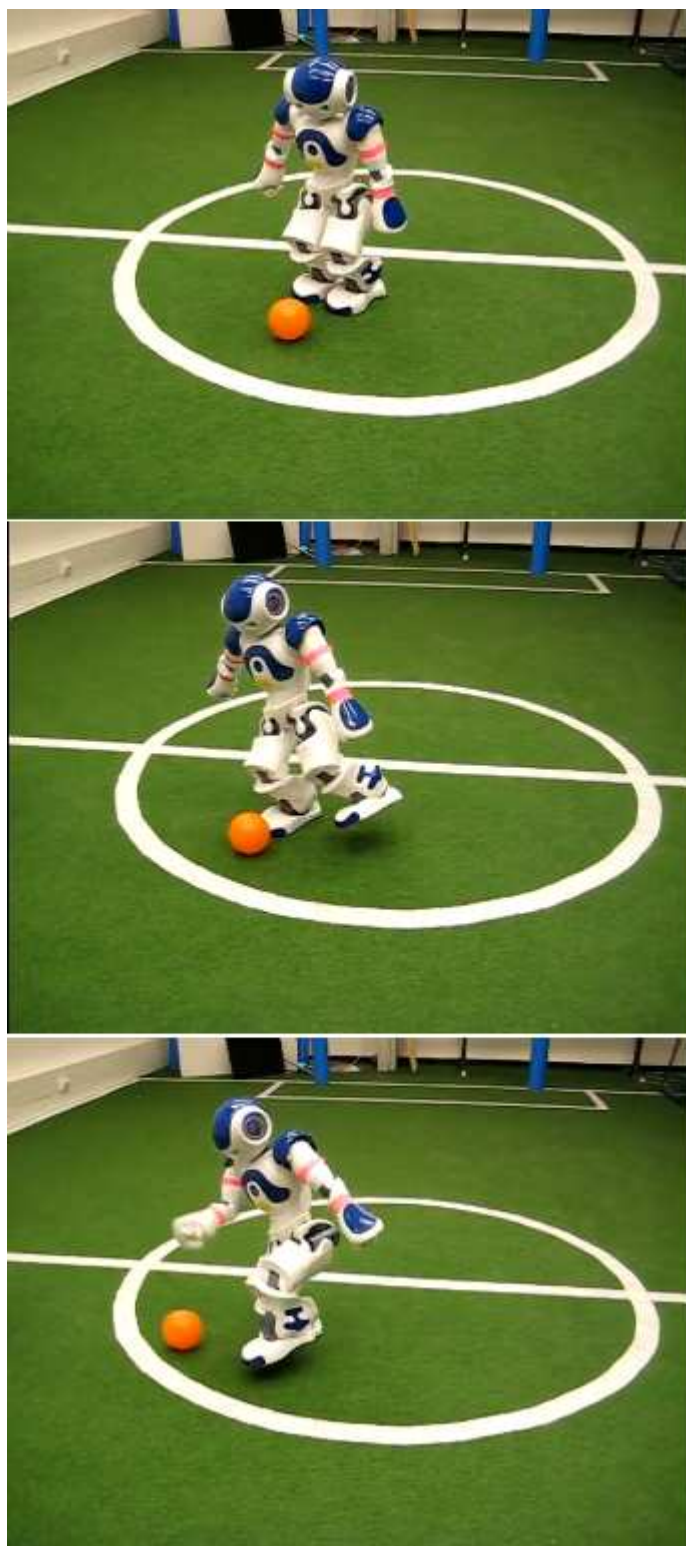
RobotControl je nástroj práve tohto tímu, ktorý bol vytvorený za účelom lepšieho testovania a analyzovania hráča. Implementovaný je v jazyku Java z dôvodu viacplatformovej podpory.

Hráč tohto tímu využíva na spracovanie obrazu rozpoznávanie farieb na mriežke o veľkosti osemdesiat na šesťdesiat bodov. Objektmi spracovania sú hlavne lopta, čiary a ostatní hráči. Pomocou rôznych filtrov získava informácie o relatívnych a absolútnych pozíciách. Napríklad informácia o globálnej pozícii lopty sa využíva na koordináciu všetkých hráčov tímu.

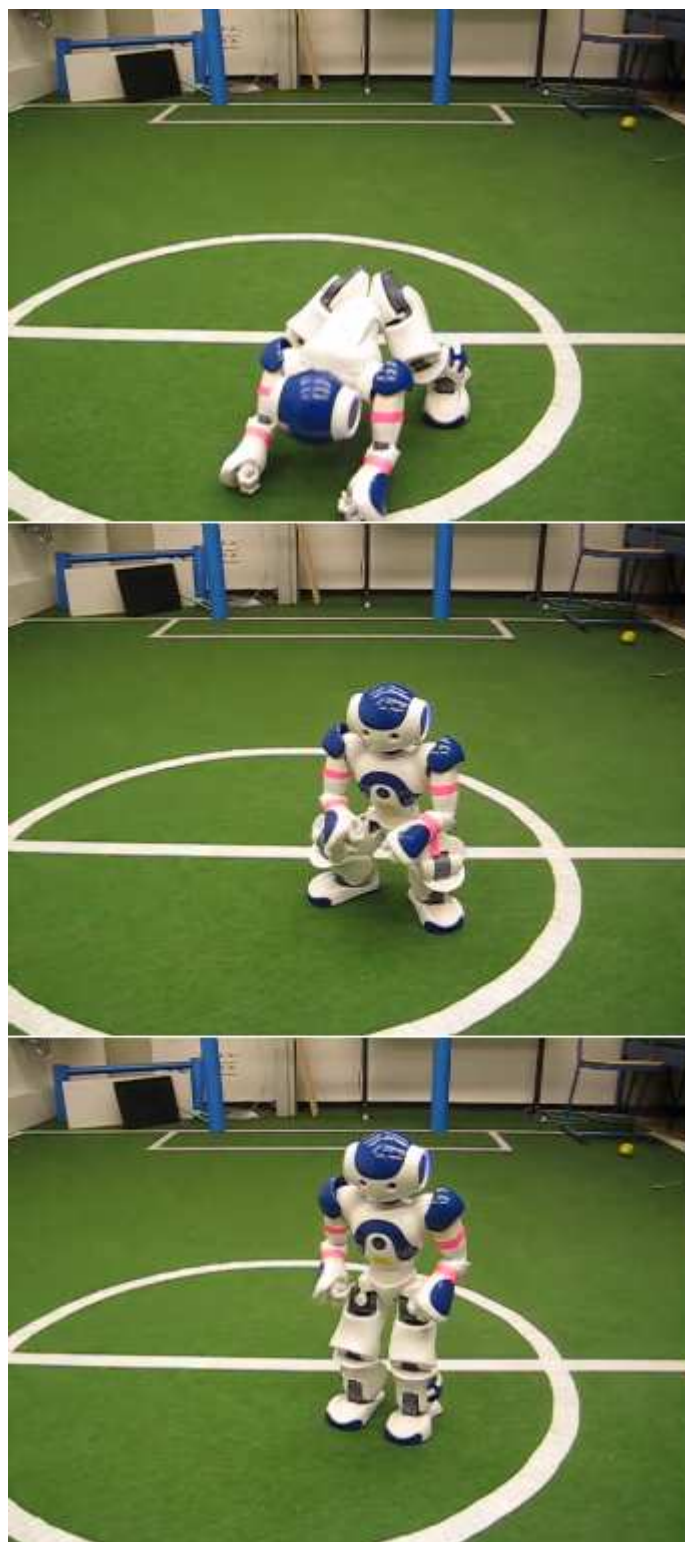
Pohyby hráča sú vytvárané pomocou nástroja vytvoreného týmto tímom. Dynamické pohyby sú vytvárané z predefinovaných kopov a pohybov do rôznych strán s využitím inverznej kinetiky a výpočtov na základe údajov zo snímačov. Správanie hráča je vykonávané pomocou stavového stroja, ktorý je rozdelený na vrstvy. Pri určení správania sa využívajú najnižšie vrstvy, ale aj najvyššie vrstvy, ktoré už zabezpečujú spoluprácu medzi hráčmi tímu. Tím sa sústreďuje na vytváranie dynamických pohybov vzhľadom na to, že presne predefinované pohyby sú využiteľné len v špecifických situáciách a špecifických podmienkach. Zameriava sa napríklad na kontrolu stability hráča využitím rôznych rýchlostí vykonania pohybov a kontrolou podkladu hráča alebo na dynamické kopy s prihliadaním na stav lopty a plynulý prechod od chôdze ku kopu.

Na stránke tímu NAO-Team Humboldt sú prístupné aj videá zobrazujúce najrozvinutejšie pohyby hráča ako je kop, bočný kop, hľadanie lopty, nasledovanie lopty, vstávanie z brucha, vstávanie z chrbta a iné. Obrázky 2.2 - 5, 2.2 - 6, 2.2 - 7 zobrazujú niektoré z pohybov hráča. Tento tím dokonca poskytol aj samotného hráča spolu s niektorými najrozvinutejšími pohybmi. Po spustení len samotnej chôdze je viditeľné, že sa jedná o veľmi presné a stabilné pohyby. Chôdza je relatívne rýchla a plynulá.

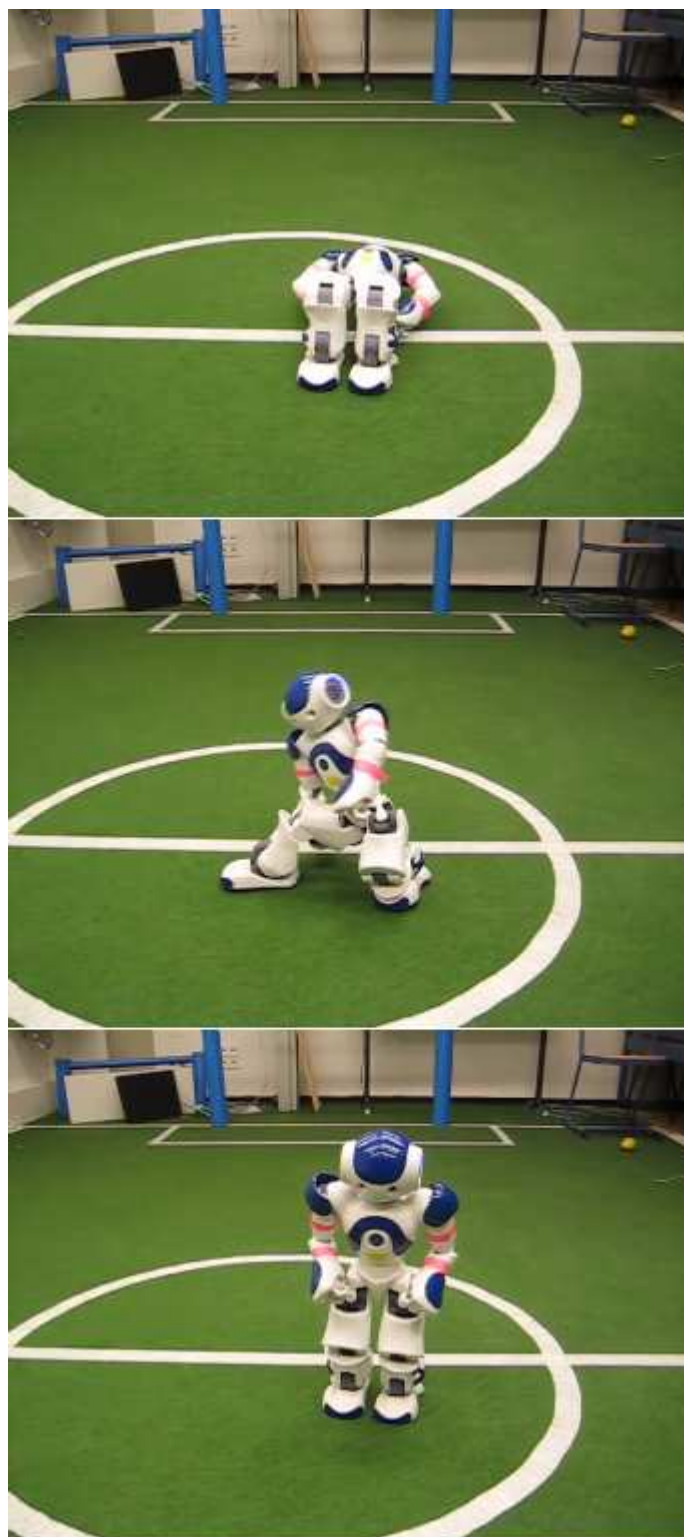
Zaujímavosťou, ktorou sa začal zaoberať tento tím je rozoznávanie gest hráčov. Táto metóda nie zatiaľ využiteľná pri súčasných robotoch, ale v budúcnosti môže pomôcť pri vzájomnej komunikácii hráčov.



Obrázok 2.1-5 – Pohyb bočného kopu [4]



Obrázok 2.1-6 – Pohyb vstávania z brucha [4]



Obrázok 2.1-7 – Pohyb vstávania z chrbta [4]

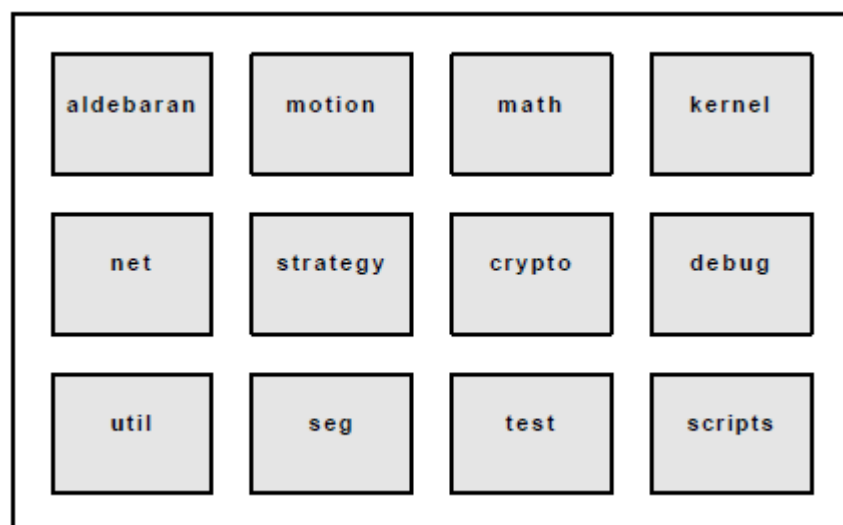
Nao-Team HTWK

Tím Nao-Team HTWK [5] je tím tvorený študentmi a absolventmi Univerzity aplikovaných vied v Leipzigu. Tím bol založený začiatkom roka 2009 a odvtedy dokázal dosiahnuť druhé miesto v RoboCup German Open 2009 a prvé miesto v Technical Challenge of RoboCup 2009.

Architektúra

Hráč je implementovaný vo viacerých jazykoch a to v C, Java a Perl. V Perl sú implementované len skripty na kompilovanie a zabezpečenie závislosti pri medzi platformovom kompilovaní. Jadro je implementované v C z dôvodu zabezpečenia vysokej rýchlosti a samotná logika hráča je implementovaná v jazyku Java. Celý hráč je implementovaný ako framework, ktorý je úplne nezávislý od framework, ktorý je v samotných Nao robotoch. Rozširuje však NaoQL framework, ktorý neumožňuje implementáciu v C, má nedostačujúce možnosti ladenia a nutnosť reštartu, len pri akejkoľvek úprave pohybov alebo stratégie.

Základom frameworku je komunikácia pomocou Unix Domain Socket Client Server, ktorý slúži na odosielanie dotazov nie len z pôvodných časti frameworku, ale aj z rozšírenej časti. Odosielané dotazy sú jednoduché a dosahujú vysokých výkonov. Obrázok 2.2 - 8 zobrazuje architektúra frameworku.



Obrázok 2.1-8 – Architektúra frameworku [5]

Aldebaran – časť systému, ktorá obsahuje Unix Domain Socket Interface s implementáciou klientskej a serverovej komunikácie.

Motion – časť systému, ktorá sa zabezpečuje načítavanie pohybov, nastavovanie jednotlivých kľbov a kontrolu stability.

Math – časť systému s vysokým výkonom, ktorá sa stará o matematické operácie.

Kernel – časť systému, ktorá sa stará o vzájomnú komunikáciu medzi časťami systému.

Net – časť systému zabezpečujúca šifrovanú komunikáciu medzi hráčmi.

Strategy – časť systému, ktorá zabezpečuje stratégiu hráča. Stratégie je možné meniť aj počas turnajov.

Crypto – časť systému, ktorá zabezpečuje samotné šifrovacie funkcie pre komunikáciu medzi hráčmi

Debug – časť systému, ktorá je slúži na ladenie systému a hráča.

Util – časť systému, ktorá uchováva špeciálne dátové štruktúry v podobe zoznamov, previazaných zoznamov, hešovacích tabuliek, stromov atď.

Seg – časť systému, ktorá zabezpečuje vysokovýkonné rozoznávanie obrazu.

Test – časť systému v podobe Unit testov, ktorá zabezpečuje test frameworku.

Scripts – časť systému v podobe skriptov v Perl, ktorá zabezpečuje kompilovanie.

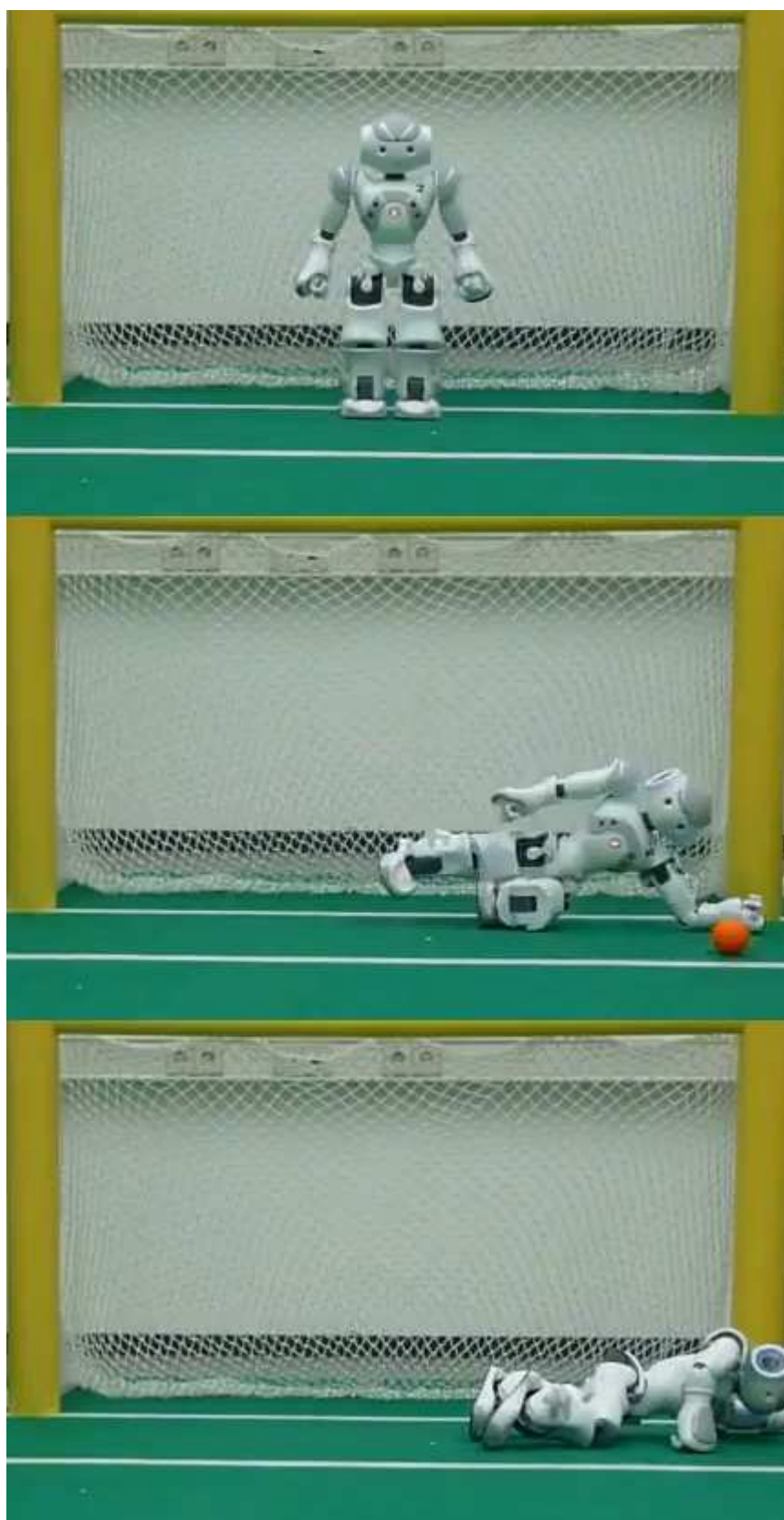
Pohyby

Väčšina pohybov, ktoré robot používa sú uzavreté cykly pohybov, ktoré sú navrhnuté pomocou špeciálnych evolučných algoritmov. Pre dosiahnutie čo najvyššej rýchlosti boli evolučné algoritmy optimalizované v simulačnom prostredí Webots. Optimalizácie prebiehali tiež pomocou symetrie a mutačného jadra založeného na B-Splines. Použité evolučné algoritmy využívajú fitness funkciu závislú od stability a taktiež od rýchlosti pohybov. Reálne roboty dosahujú pri týchto pohyboch rýchlosť až 32 centimetrov za sekundu.

K dosiahnutiu plynulej chôdze do strany sú využité optimalizované sínusové funkcie, ktoré upravujú pohyb bedrových kĺbov. Stabilita pri chôdzi je zabezpečená aj upravovaním sklonu ramien pomocou PD regulátora. Rozpoznávanie pádu je rozpoznávané sklonom trupu a znížením tuhosti kĺbov.

Pohyby vstávania z chrbta a z brucha nie sú riešené pomocou evolučných algoritmov, ale sú navrhnuté napevno. Časy týchto pohybov sú 5,2 sekundy z chrbta a 4,5 sekundy z brucha.

Brankár tohto tímu má definovaný efektívny pohyb chytania lopty. Ukážka tohto pohyby je zobrazená na Obrázku 2.2 - 9.



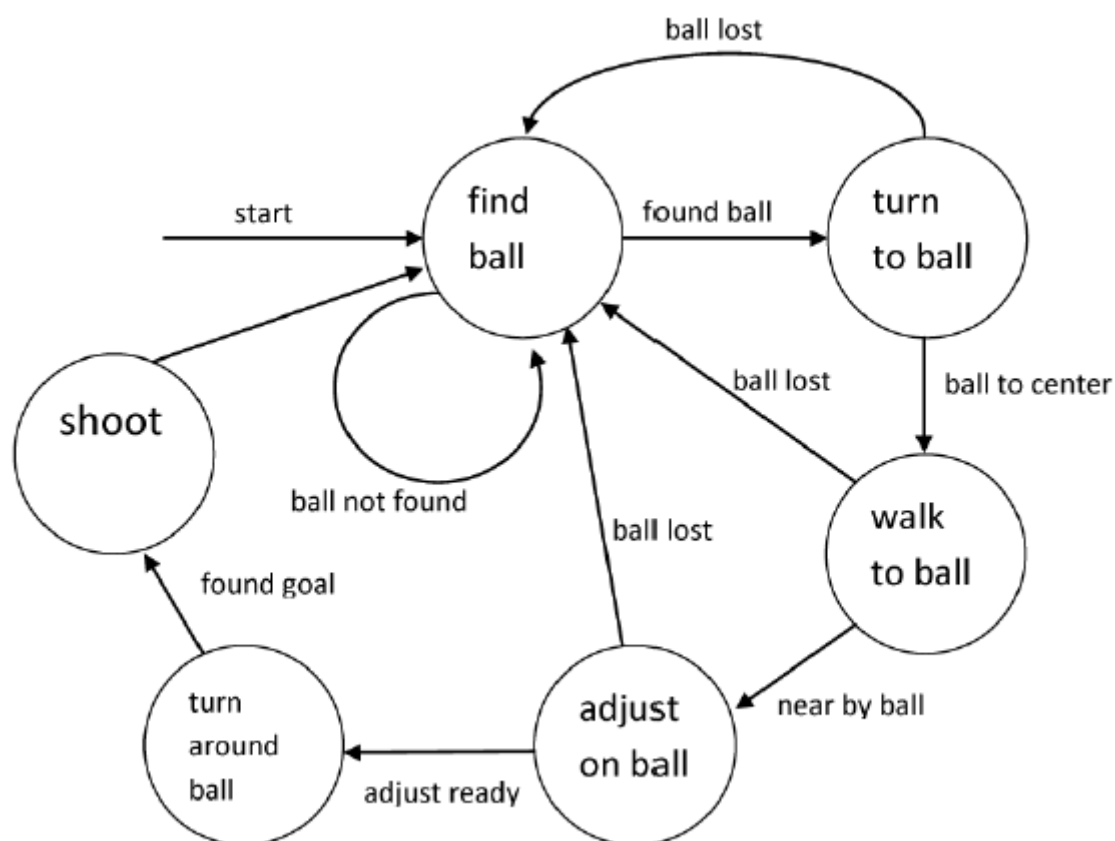
Obrázok 2.1-9 – Pohyb brankára [5]

Rozpoznávanie

Reálne roboty sú vybavené dvoma kamerami, ale pre dlhé časy medzi prepínaním jednotlivých kamier využíva tento tím len jednu. Na rozoznávanie objektov ako sú hráči, čiary, lopta sa využíva segmentácia obrazu a rozoznávanie farieb. Každý objekt má definovanú farbu a na základe nej je možné rozoznať, že sa pravdepodobne jedná o daný typ objektu. Následne sú využívané dva rôzne filtre na overenie, či sa jedná o objekty daného typu.

Stratégia

Výber stratégie tímu je založený na konečnom stavovom diagrame, ktorý zobrazuje Obrázok X. Je zložený zo šiestich stavov. K rýchlejšiemu vyhodnocovaniu stratégie sa využíva metóda, pri ktorej sa v jednotlivých stavoch zanedbávajú určité faktory. To znamená, že v niektorých stavoch sa vôbec neprihliada na ostatných hráčov alebo na bránu a iné. Táto metóda umožňuje značné zníženie náročnosti výpočtu, čo má za následok rýchlejšie reakčné doby a menej pohybov hráča.



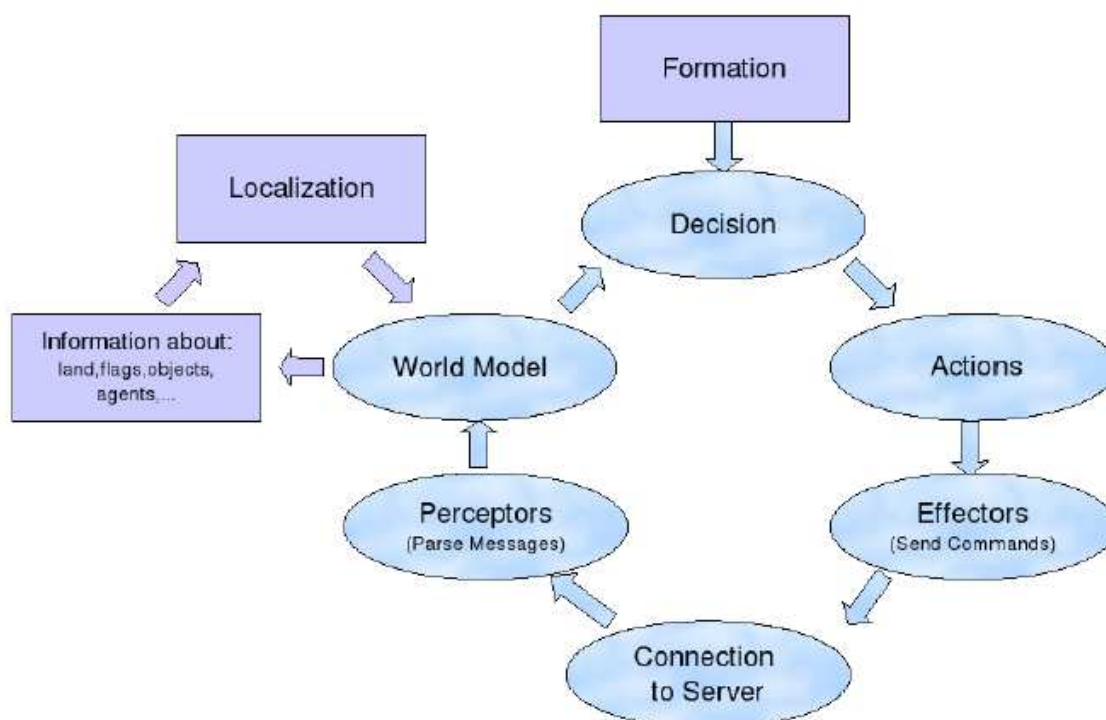
Obrázok 2.1-10 – Stavový diagram stratégie [5]

Alzahra

Tím univerzity Alzahra [6] bol vytvorený v lete 2009 v spolupráci so študentmi matematických vied a výpočtovej techniky na technickej vysokej škole Alzahra. V krátkej dobe tento tím vytvoril vlastných hráčov založených na kóde tímu Zigorat (Inter-University AI), vytvoril vlastné pohyby a naprogramoval nové zručnosti pre robota. Tím Alzahra sa zúčastnil súťaže RoboCup 2010 v simulačnej lige a hoci nepatril k úspešným tímom (nepostúpil do druhého kola), myslíme si, že je potrebné opísať aj takýto tím, aby sme mohli porovnať prípadne nedostatky s inými tímami.

Architektúra

V počiatočných dňoch sa tím Alzahra zamerával najmä na architektúru agenta, ktorú pokladajú jeho autori za kľúčovú. Vychádzajú z predpokladu, že ak bude táto architektúra výborne navrhnutá, potom aj správanie robota bude viac podobné správaniu človeka. Jednoduchá architektúra agenta je zobrazená na Obrázku 2.2 - 11.



Obrázok 2.1-11 – Architektúra hráča tímu Alzahra [6]

Najdôležitejšie časti tejto architektúry sú nasledovné:

Formation – táto časť zodpovedá za pozíciu hráča na ihrisku a rovnako aj sprostredkúva informáciu o type hráča (brankár, útočník), pričom sa berie do úvahy aj typ hry (obrana, alebo útok).

World Model – obsahuje všetky potrebné informácie pre agenta. Sú to informácie o ihrisku, polohe iných hráčov a lopty. Pomocou týchto informácií tak agent môže zistiť svoju polohu na ihrisku a stav ostatných objektov.

World Model – obnovuje informácie na základe správ, ktoré získa zo simulačného serveru. Hráč má k tomuto modelu prístup a na základe neho robí rozhodnutia.

Decision – táto časť definuje aktuálnu rolu, ktorú hráč bude vykonávať použitím informácií z častí *Formation* a *World Model*.

Action – potom, ako sa určí rozhodnutie v časti *Decision*, akcie, ktoré robot vykoná sú poslané na simulačný server.

Send commands – vytvára spojenie hráča so serverom, a zasiela serveru informácie.

Server communication – táto časť samotná slúži na komunikáciu so serverom, implementuje sieťový protokol a analyzuje správy.

Perceptors – táto časť prijíma informácie od servera – sú to informácie o hre, hráčoch a iných objektoch, ktoré sú vidieť na ihrisku.

Určovanie pozície

Za určovanie pozície je zodpovedná časť **Localization**, ktorá je jedna z najdôležitejších častí robota. Keďže v novšej verzii servera mal robot 120° zorné pole v každej dimenzii a zároveň boli zavedené určité obmedzenia k určovaniu polohy (robot nie vždy vidí tri body, podľa ktorých môže zistiť svoju polohu) navrhol tím Alzahra spôsob, ako dokáže robot určiť svoju polohu pomocou 2 bodov. Určenie polohy prebieha tak, že ju odhadujú z výšky robota a bodov, ktoré sa nachádzajú na ihrisku.

Nájdenie cesty

Ďalší problém, ktorý tím určitým spôsobom vyriešil je hľadanie cesty, tak aby nenarazil do iného objektu (napr. hráčov). Tento problém sa snažia riešiť na základe metód výpočtovej geometrie (computational geometry).

V tejto metóde pozorujú ruky a nohy stojaceho robota a ruky, nohy a hlavu ležiacich robotov. Na základe toho je určená ich poloha ako vrcholy konvexného polygónu. Podobne sú vypočítané aj polygóny lopty a ostatných prekážok. Na Obrázku 2.2 - 12 predstavuje A agenta, ktorý hľadá cestu. Polygón O_1 je ležiaci robot a O_2 a O_3 sú stojace roboty. Keďže robot nemôže vnímať všetky body ostatných hráčov súčasne, je takýto spôsob vykreslenia si najväčších polygónov oveľa výhodnejší. Hoci sa týmto spôsobom úloha zjednodušila, je stále pomerne ťažké formulovať miesta, ktoré obsahujú prekážky, vzhľadom na meniace sa podmienky v hre, ako aj to, že robot musí neustále sledovať svoje hrany, tak aby nenarazil do týchto prekážok.

K tomu, aby sa zabránilo robotovi naraziť, využíva tento tím, ako bolo spomenuté, metódy výpočtovej geometrie. Podľa nej je potrebné najprv určiť priestor prekážok (configuration-space obstacle). To urobia v niekoľkých krokoch:

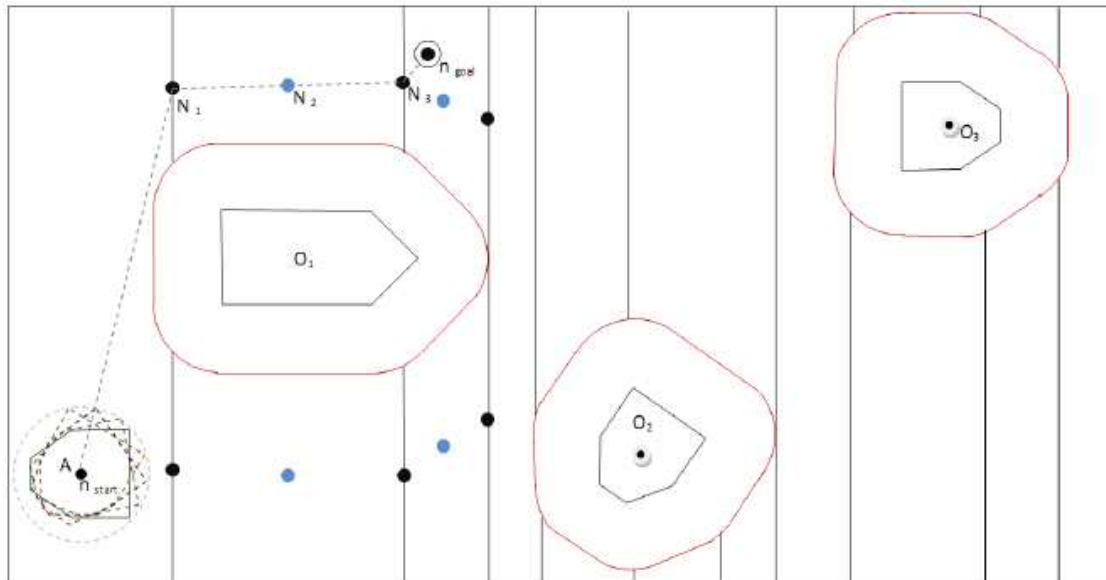
1. Najprv si vypočítajú tzv. Minkowského sumu pre agenta A a každý objekt O_i

$$P_i = A \oplus O_i$$

2. Následne odčítajú priestor prekážok od celkového priestoru M

$$C_{free} = M \setminus \bigcup_{i=1}^m P_i$$

Kde m predstavuje počet všetkých prekážok na ihrisku. Následne sme dostali tzv. voľnú konfiguráciu, ktorú rozdelia pomocou tradičnej triangulačnej metódu na tzv. Trapezoid Map. V tejto mape na základe algoritmu pre hľadanie najkratšej cesty v grafe určujú dráhu agenta. Na obrázku Obrázku 2.2 - 12 sú jednotlivé body pohybu znázornené vrcholmi N_i .



Obrázok 2.1-12 – Ukážka nájdenia cesty [6]

Riadenie pohybu

Tím Alzakra zvažovali na výber dva typy pohybu pre svojho agenta – statický a dynamický. Pri statickom type pohybu robot vykonáva len stabilné pohyby, kde centrum váhy je v stabilnej oblasti. To znamená aj pomalšie pohyby. Naproti tomu dynamický typ pohybu nie je limitovaný týmto obmedzením. Agentova rovnováha závisí od jeho rýchlosti a zrýchlenia, čo ale umožňuje agentovi pohybovať sa rýchlejšie. Tento tím si vybral dynamický typ pohybu a riadenie pohybu tak rozdelili na tri časti:

1. Určenie stratégie pohybu (či agent kráča, uteká, kope do lopty)
2. Plánovanie (kedy sa čo vykoná)
3. Sledovanie a stabilizácia pohybu (v tejto časti je aj riadenie pohybu rúk)

Pri návrhu pohybov tento tím využíva tri typy metód:

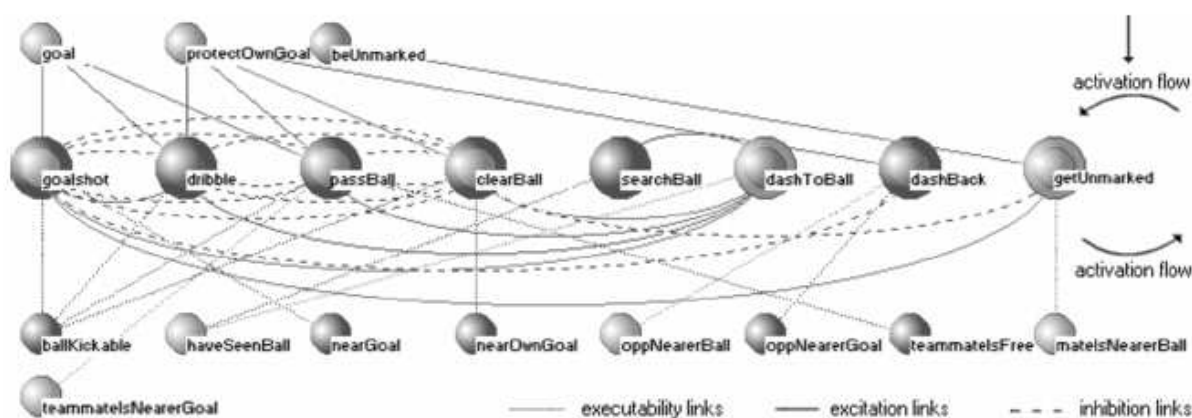
1. Metódy založené na trajektórií pohybu
2. Mentálne metódy (bližšie ich nešpecifikujú)
3. Metódy založené na CPG (založené na neurónových sieťach)

MagmaOffenburg

Tím MagmaOffenburg [8, 9] je nemecký tím, ktorý bol v mnohých národných súťažiach úspešný a kvalifikoval sa dokonca na súťaž RoboCup 2010. Okrem vylepšovania svojho hráča idú aj cestou tvorby nástrojov pre podporu vývoja, čo im do značnej miery automatizuje niektoré činnosti spojené s vývojom v tejto oblasti. Hlavnými časťami, na ktoré sa tím zameriava v rámci vývoja hráča sú rozhodovanie sa pomocou rozšírených sietí správania sa (extended behavior networks) a zároveň implementáciu v Jave, aby umožnili aj Java komunite podieľať sa na vývoji.

Rozhodovanie sa hráča

Rozhodovanie hráča pri určení, t.j. ktorú akciu vykoná, je založená na tradíciách tohto tímu už z RoboCupu 2D. Využívajú na to rozšírené siete správania, ktoré umožňujú definovať široké vzory správania. Ich princíp spočíva v explicitnej reprezentácii cieľov s dynamickými funkciami (teda takými, ktoré závisia od situácie). Ciele, ktoré sa agenti snažia splniť, môžu byť zoradené podľa priority na základe statických informácií, ale rovnako aj na základe meniacich sa podmienok v hre. To umožňuje zamerať sa agentovi na ciele, ktoré sú relevantné v danej situácii (Obrázok 2.2 - 13).



Obrázok 2.1-13 – Rozšírená sieť správania [8]

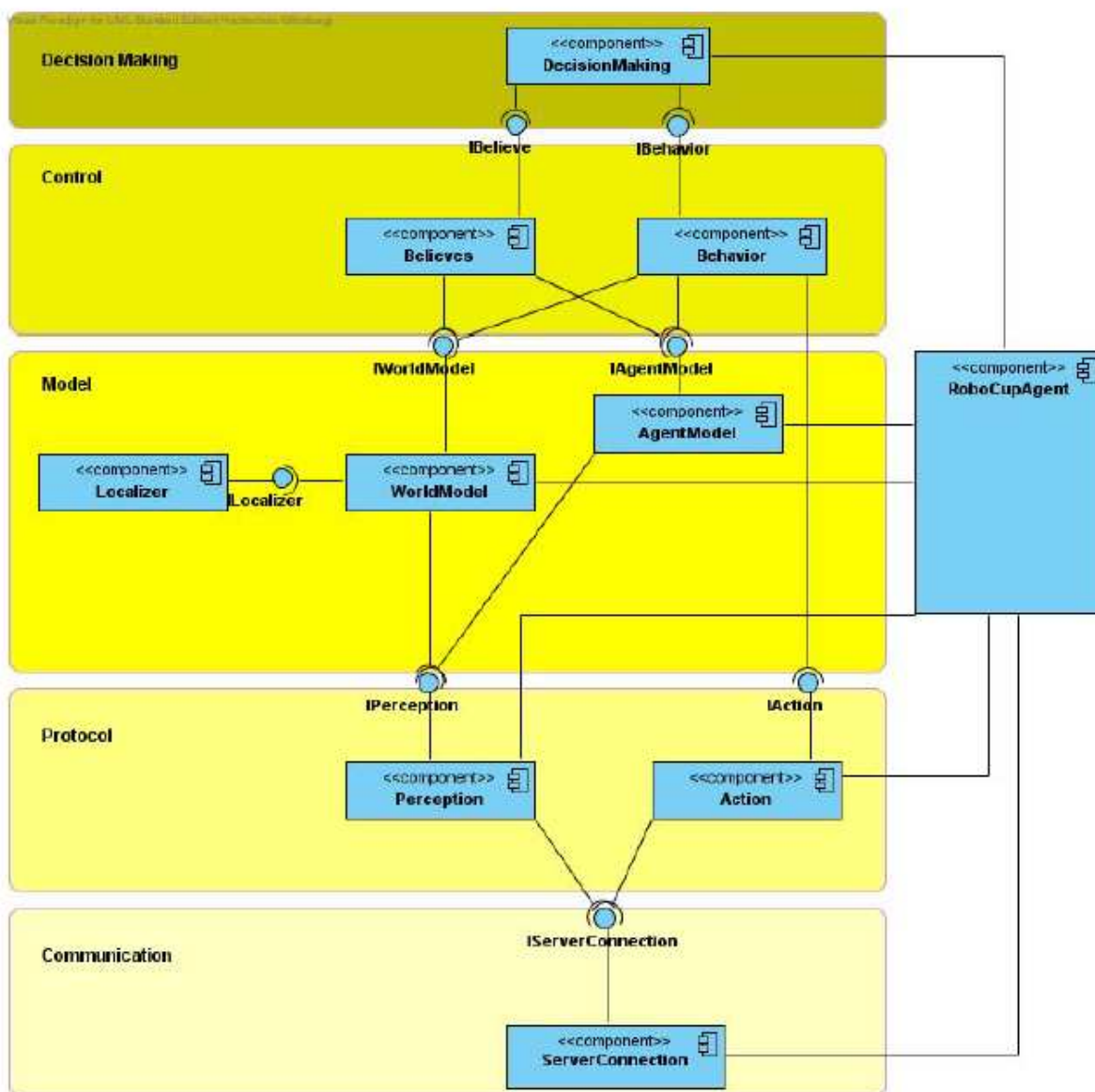
Architektúra

Z pohľadu architektúry robí tento tím priekopnícku prácu, keďže sa zameriaval na programovací jazyk Java, pretože väčšina iných hráčov je implementovaná v jazuku C++. Tento tím zároveň uverejnil svoj kód, ktorý bol použitý v súťaži RoboCup 2010. Hlavné body ich architektúry sú:

- Vrstvová architektúra
- Java ako implementačný jazyk
- JUnit testy pre zabezpečenie kvality dynamickým testovaním

Vrstevná architektúra je znázornená na Obrázku 2.2 - 14. Momentálne sa ich agent skladá z 5 vrstiev:

- Komunikačná vrstva
- Sieťový protokol
- Model
- Riadenie agenta
- Vykonanie rozhodnutí



Obrázok 2.1-14 – Architektúra hráča [8]

Tieto vrstvy sú navrhnuté a skonštruované tak, aby sa zabránilo závislosti z nižších vrstiev do vyšších vrstiev. To je výhoda, ktorá umožňuje aj iným tímom použiť kód tohto tímu a začať stavať na niektorej z vrstiev.

Tok informácií z jednej vrstvy do druhej je realizovaný pomocou návrhového vzoru observer, tak aby boli nižšie vrstvy nezávislé od vrstiev vyšších. To znamená, že napríklad GUI, ktoré vytvorili, zobrazuje stav sveta v stave v akom sa nachádza a ihneď reaguje

na zmeny, ktoré nastanú. Zároveň však sú model sveta a GUI nezávislé, resp. GUI je úzko zviazané s modelom (loosely coupled).

Architektúra tohto tímu je zároveň založená na komponentoch (Obrázok 2.2 - 4), to znamená, že ľubovoľný komponent, môže byť nahradená vlastnou implementáciou. K tomu využívajú rozhrania, na vytvorenie voľného prepojenie jednotlivých komponentov. Väčšina komponentov poskytuje úzku viazanosť s ostatnými komponentmi, ako napríklad vrstva pre vykonávanie rozhodnutí, model sveta, či komunikačná vrstva.

Vyvinuté nástroje

Tím vyvinul niekoľko nástrojov, ktoré umožňujú stiahnuť zo svojej stránky.

Agent GUI – je to nástroj na vizualizáciu, ktorú umožňuje zobrazovať správanie agentov v 2D a rovnako aj 3D. Slúži najmä na analýzu a ladenie správania hráča. Obsahuje aj funkcie na prehrávanie zaznamenaného správania (play, pause, forward), ktoré sa môžu hodiť pre analýzu.

Benchmark team – slúži na vyhodnocovanie a hodnotenie metód, ktoré slúžia na určenie polohy.

EBN Development Kit – nástroj na vizualizáciu, vytváranie a ladenie rozšírených sietí správania sa.

Live-Movement – slúži na interpretovanie rýchlych pohybov hráča, ktoré sa inak nedali interpretovať dostatočne rýchlo, umožňuje prerušiť správanie hráča, sledovať jeho momentálny stav a znovu pokračovať v pohybe.

Motorfile editor – slúži na vytvorenie správania robotov, z existujúcich vzorov správania sa dá poskladať v tomto editore nové správanie, a toto správanie následne exportovať do súboru, ktorý použije agent.

Polytechnic-Parsian

Iránsky tím Polytechnic-Parsian sa zúčastňuje na súťažiach v simulovanom futbale po celom svete a s výbornými výsledkami. Je to iránsky tím, ktorý sa zaoberá najmä správnym navrhnutím pohybu pre robota. K tomu využívajú matematické modely založené na Fourierových radoch. V súčasnosti majú zvládnutú najmä chôdzu robota a pracujú na ďalších pohyboch, ako sú kopy do lopty.

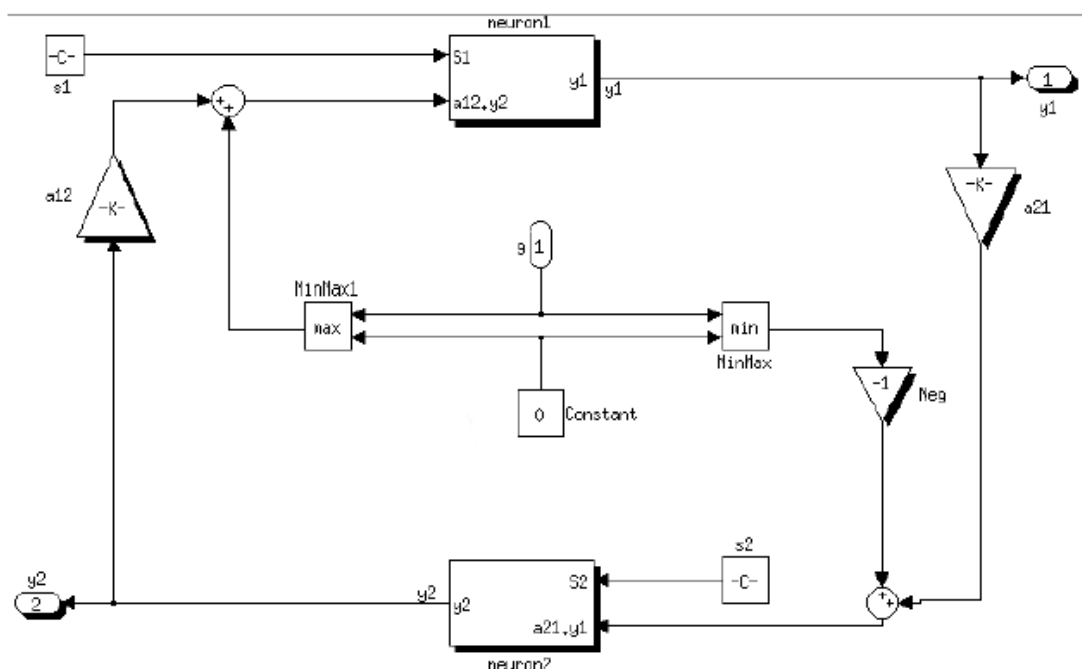
Riešenie zložitých problémov

Keďže pohyb robota je náročná vec, vo svojom riešení tento tím vyberá množstvo metód, ktoré sa snažia prispôbiť a implementovať v ich hráčovi. Týmito metódami, ktoré sa používajú na riešenie zložitých problémov, sú najmä:

- Central Pattern Generator – pre generovanie akcií, ktoré sú stabilné aj v prostredí, kde sa neustále menia podmienky
- Comprehensive Learning Particle Swarm Optimization (CLPSO) – nastavovanie parametrov systému, ktoré by sa inak muselo vykonávať experimentálne
- Reinforcement Learning – pre akcie v súvislosti s kopmi, ktoré musia rátať so zmenou v prostredí

Generovanie akcií

Tím pristúpil k riešeniu návrhu hráča podobne, ako ku simulácii správania sa živých organizmov. Výskum naznačuje, že pohyb všetkých stavovcov, vrátane človeka, je vo všeobecnosti determinovaný tzv. centrálnym generátorom vzorov (general central pattern - CPG), ktorý zasiela informácie o pohybe do miechy. CPG je teda určitý nervový obvod, ktorý môže produkovať vzorce správania sa. V pojmoch informatiky, je CPG vlastne neurónová sieť, ktorá vykonáva výpočty riešením rovníc. CPG sa hodí práve pri riešení problémov, ako je robustná kontrola kĺbov, jednoduché nastavenie rýchlosti chôdze a dĺžky kroku, zatiaľ čo na rozdiel od iných metód nie je potrebný dynamický model robota alebo prostredia, v ktorom sa agent nachádza. Je však aj niekoľko problémov, resp. nevýhod, ktoré sa musia pri použití CPG riešiť. Jednou z týchto nevýhod je vysoký počet parametrov, ktoré musia byť presne určené. To je značne náročná úloha, pretože musíme vedieť nastaviť sieť tak, aby generovala vhodné vzory pre kontrolu chôdze. Pre nastavenie týchto parametrov neexistuje žiadna tabuľka a ani žiadna metóda. Sieť sa musí natrénovať. Spomenutý tím použil pre modelovanie centrálného generátora vzorov tzv. Matsuokovú neurónovú model oscilátora. Tento model sa skladá z dvoch vzájomne inhibičných neurónov tzv. extor a extensor neurónov (Obrázok 2.2 - 15).



Obrázok 2.1-15 – Model neurónovej siete pre použitie s CPG [8]

Tím pôvodne uvažoval k natrénovaniu siete techniky z umelej inteligencie ako sú evolučné algoritmy, ale pre ich zložitosť siahli po metóde CLPSO (spomenutá vyššie) – výhodou je jej ľahká implementácia a jednoduché pochopenie. Samotná metóda CLPSO je rozšírením všeobecnej metódy Swarm Optimatization. Rozšírenie odstraňuje problémy s lokálnym minimom.

Kopy do lopty

Tím sa v súčasnosti zaoberá automatickým generovaním náročnejších pohybov, ako je napríklad samotná chôdza. K tomu využívajú pokročilé metódy, tak aby mohli generovať nové pohyby v reálnom čase. Tieto metódy sú založené na adaptívnych trajektóriách a rovnako aj strojovom učení sa. Tím k návrhu kopov využil vedomosti z pozorovania ľudí, pri vykonávaní týchto akcií a rovnako aj zo štatistického vyhodnocovania ich hráča v simulovanom 3D prostredí. Následne si navrhli neurónovú sieť k odhadu funkcie pre učiaci sa algoritmus.

2.1.1.4 Analýza fungovania systému a pravidiel Robocup

SimSpark je simulačné prostredie pre simuláciu viacerých agentov v trojrozmernom prostredí. SimSpark je určený pre simuláciu robotického futbalu a využíva sa ako oficiálne prostredie pre simuláciu robotickej ligy RoboCup [1]. V tejto analýze vychádzame z opisu servera z dokumentácie tímu Robokopya používateľského manuálu k prostrediu SimSpark.

SimpSpark architektúra

Server SimSpark je vybudovaný použitím aplikačného rámca Zeitgeist. Ten poskytuje základnú funkcionálnu podporu archívu, logovanie, zdieľané knižnice a pod. Jednotlivé časti servera sú implementované najmä v programovacích jazykoch C++ a Ruby. Dôležitou súčasťou servera je knižnica Oxygen, ktorá je zodpovedná za manažment agentov a ich monitorovanie. Táto vrstva udržiava pripojenie agentov - hráčov a monitoruje jednotlivé procesy. V neposlednom rade umožňuje beh simulácie pomocou simulačných cyklov s nastaviteľnými atribútmi. Vizualizácie sú vykonávané za podpory knižnice Kerosin. Tá používa k svojmu behu knižnicu OpenGL a SDL, no umožňuje aj použitie iných knižníc.

SimSpark server pracuje sekvenčne a je zodpovedný za simulačné procesy. V každom simulačnom cykle zbiera informácie zo všetkých senzorov agentov a takisto vyhodnotí všetky akcie vykonávané ich efektormi. Objekty v simulácii menia počas simulácie svoj stav, napr. pozíciu, rýchlosť pohybu, uhlovú rýchlosť. V každom novom cykle server vypočíta hodnoty nového stavu objektu, ktorý sa zmení pôsobením rôznych faktorov ako gravitácie, kolízií a pod.

Monitor a logplayer

SimSpark monitor slúži na vizualizáciu diania na serveri. Server posiela monitoru dáta, transformované do špeciálneho prispôsobiteľného formátu a slúži pre opis daného stavu

simulácie. Obsahuje informácie o jednotlivých objektoch, aktuálnom móde hry, skóre a pod. Monitor môže čítať dáta zo súboru - logu, čím sa dá vizualizovať zaznamenaná hra. V tomto kontexte sa monitoru hovorí logplayer. Monitor je v tomto móde spúšťaný s prepínačom --logfile, argumentom ktorého je cesta k logu, v ktorom je uložený záznam hry. Aktuálna verzia 0.6.x simuluje modely agentov ako humanoidných robotov (namiesto jednoduchého pohybu sfér v skorších verziách).

Komunikácia agenta so serverom

Agent komunikuje so serverom prostredníctvom tzv. S-výrazov, ktorý je definovaný ako reťazec resp. zoznam ďalších S - reťazcov). S-výrazy sú používané napríklad programovacím jazykom Lisp na uloženie kódu aj dát. Na kódovanie správ je použité ASCII kódovanie, takže jeden znak má dĺžku 1 byte.

Perceptory

Perceptory v simulácii sú prostriedkami pre vnímanie okolia agentov. Server vďaka nim posiela hráčom informácie o ich pozícii v prostredí, čo pomáha hráčovi pri orientácii a následnom rozhodovaní. Perceptory sa delia do dvoch skupín a to na základné a futbalové perceptory.

Základné perceptory

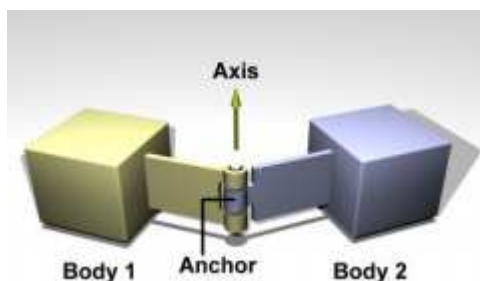
Základné perceptory sú dostupné pre všetky typy simulácii, a teda nie sú špecifické pre RoboCup 3D. Patria sem nasledujúce typy perceptorov:

1. GyroRate – slúži na doručuje informácie o orientácii tela agenta. Momentálne je v modeli hráča umiestnený len jeden GyroRate receptor a to v trupe agenta. Správa obsahuje názov tela, ku ktorému patrí a tri hodnoty rotačných uhlov, určujúce celkovú polohu vzhľadom k súradnicovej sústave.

Formát správy : *(GYR (n <názov_tela>) (rt <x> <y> <z>))*

Príklad: *(GYR (n torso) (rt 0.01 0.07 0.46))*

2. HingeJoint – udáva hodnotu veľkosti uhla, o ktorý sa ohne daný kĺb agenta. Obrázku 2.2 - 16. zobrazuje ohnutý kĺb s označenou osou.

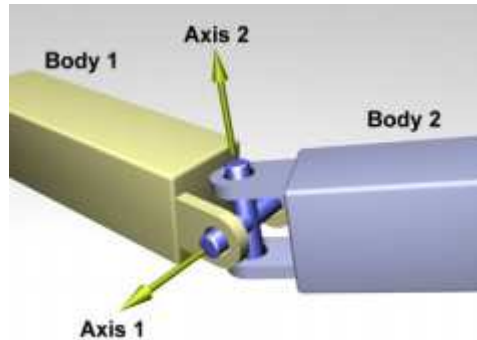


Obrázok 2.1-16 – HingeJoint kĺb [1]

Formát správy : $(HJ(n \text{ <názov_kĺbu>})(ax \text{ <veľkosť_uhla>}))$

Príklad: $(GYR(n \text{ torso})(rt \ 0.01 \ 0.07 \ 0.46))$

3. UniversalJoint - prijíma informácie z kĺbov, schopných ohnutia v dvoch smeroch. Tvorí ho meno kĺbu a hodnoty dvoch uhlov. Takýto kĺb je zobrazený na Obrázku 2.2 - 17.



Obrázok 2.1-17 – UniversalJoint kĺb [1]

Formát správy : $(UJ(n \text{ <name>})(ax1 \text{ <ax1>})(ax2 \text{ <ax2>}))$

Príklad: $(UJ(n \text{ laj1_2})(ax1 \text{ -1.32})(ax2 \text{ 2.00}))$

4. Touch – zachytáva kolízne udalosti agentov. Nadobúda hodnoty 1 alebo 0. 1 znamená, že nastala kolízia, 0 naopak, že je agent v kludovom stave.

Formát správy : $(TCH \ n \text{ <name> val } 0/1)$

Príklad: $(TCH \ n \text{ bumper val } 1)$

5. ForceResistant – slúži na detekciu pôsobenia sily na určitú časť tela agenta. V súčasnosti je možné jeho využitie pre ľavé (lf) a pravé (rf) chodidlo hráča. Definuje ho bod, na ktorý sila pôsobí a vektor sily.

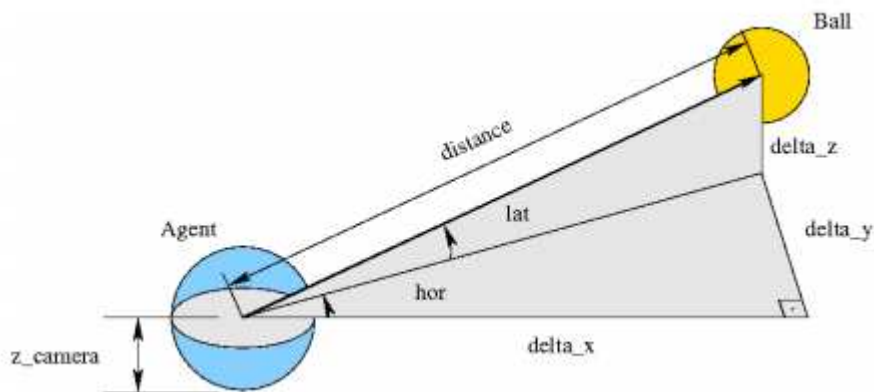
Formát správy : $(FRP(n \text{ <name>})(c \text{ <px> <py> <pz>})(f \text{ <fx> <fy> <fz>}))$

Príklad: $(FRP(n \text{ lf})(c \text{ -0.14 } 0.08 \text{ -0.05})(f \text{ 1.12 } -0.26 \text{ 13.07}))$

Futbalové perceptory

Nasledujúce perceptory sú špecifické pre RoboCup simuláciu:

1. Vision – posiela informácie o objektoch, ktorých hráč vidí, to znamená informácie o pozícii lopty, spoluhráčov, protihráčov a osem záchytných bodov v modeli hracej plochy. Perceptor zachytáva uhol o šírke 90 stupňov pred agentom (Obrázok 2.2 - 18). S každým zachytením správy prichádza takisto informácia o:
 - vzdialenosti medzi agentom a objektom
 - uhle medzi agentom a objektom v horizontálnej rovine
 - šírkovom uhle



Obrázok 2.1-18 – Model polárnych súradníc používaných Vision perceptorom [1]

Na rozdiel od simulovaného futbalu 2D tento perceptor nezachytáva informácie o rýchlosti objektov. Všetky vzdialenosti a veľkosti uhlov sú relatívne k pozícii kamery, ktorá je momentálne lokalizovaná v strede trupu hráča. Do všetkých správ je umelo vnášaná chyba resp. šum, ktorý pozostáva z nasledujúcich častí:

- kalibračná odchýlka z intervalu $(-0.005, 0.005)$ pre každú os pozície kamery. Táto odchýlka je vypočítaná len raz pre každý zápas.
- Dynamický šum je normálne distribuovaný okolo 0.0 + vzdialenostná odchýlka: $\sigma = 0.0965 + \text{uhlová odchýlka}$ (x - y): $\sigma = 0.1225 + \text{uhlová odchýlka}$ (šírková): $\sigma = 0.1480$

Formát správy : *(See (<name> (pol <distance> <angle1> <angle2>)) (P (team <teamname>) (id <playerID>) (pol <distance> <angle1> <angle2>)))*

Príklad: *(See (F1L (pol 19.11 111.69 -9.57)) (F2L (pol 16.41 -115.88 -11.15))(F1R (pol 46.53 22.04 -3.92)) (F2R (pol 45.49 -18.74 -4.00)) (G1L (pol 9.88 139.29 -21.07)) (G2L (pol 8.40 -156.91 -25.00)) (G1R (pol 43.56 7.84 -4.68)) (G2R (pol 43.25 -4.10 -4.71)) (B (pol 18.34 4.66 -9.90)) (P (team RoboLog) (id 1) (pol 37.50 16.15 -0.00)))*

Meno objektu sa vyberá z hodnôt:

- F1L, F1R, F2L, F2R značia rohy hracej plochy
- G1L, G1R, G2L, G2R značia žrde brán
- B značí loptu
- P značí hráča s ďalšími informáciami (team <názov tímu>) (id <playerID>)

2. GameState – posiela hráčom informácie o stave simulácie zápasu. Takisto ním na začiatku hry získajú informácie týkajúce sa veľkosti hracej plochy a lopty.

Formát správy : *(GS (t <time>) (pm <playmode>))*

Príklad: *(GS (t 0.00) (pm BeforeKickOff))*

3. AgentState – udáva informácie o internom stave agenta, konkrétne o stave batérie a výške jeho teploty.

Formát správy : *(AgentState (temp <degree>) (battery <percentile>))*

Príklad: *(AgentState (temp 48) (battery 75))*

4. Hear – agentom nie je dovolené komunikovať medzi sebou priamo, ale len prostredníctvom simulačného servera. Z tohto dôvodu sa v agentoch nachádza Hear perceptor, ktorým hráč prijíma správy od ostatných hráčov.

Formát správy : *(hear <time> 'self' <direction> <message>)*

Príklad: *(hear 12.3 self ``helloworld'')*

Efektory

Efektory sa používajú vtedy, ak chceme, aby agent vykonal určitú akciu. V tom prípade agent pošle serveru správu identifikujúcu jeho zámer.

Základné efektory

Efektory opísané v tejto časti s sú základné efektory. Podobne ako perceptory, využívajú vo všetkých typoch simulácií. Patria sem nasledujúce efektory:

1. Create – slúži na vytvorenie hráča. Jeho argumentom je súbor, obsahujúci opis hráča. Dostupný je ako náhle je agent pripojený k serveru a po ňom sa očakáva volanie efektora Init, ktorý hráča priradí k tímu, meno ktorého zadáme ako argument.

Formát správy : *(scene <filename>)*

Príklad: *(scene rsg/agent/soccerbot056.rsg)*

2. HingeJoint – je potrebný k pohybu jednotlivých kĺbov hráča. Jeho atribútmi sú meno kĺbu a veľkosť uhla, o ktorý chceme daný kĺb ohnúť.

Formát správy : *(<name> <ax>)*

Príklad: *(lae3 5.4)*

3. UniversalJoint – k pohybu kĺbov v smere dvoch osí sa využíva efektov UniversalJoint.

Formát správy : *(<name> <ax1> <ax2>)*

Príklad: *(lae1 2 -2.3 1.2)*

Futbalové efekty

V nasledujúcej časti opíšeme efekty, ktoré sú špecifické pre RoboCup simuláciu.

1. Init - slúži na priradenie hráča k určenému tímu.

Formát správy : *(init (unum <playernumber>)(teamname <yourteamname>))*

Príklad: *(init (unum 1)(teamname FHO))*

2. Beam – slúži na teleportovanie hráča na určitú pozíciu, no zavolaný môže byť len pred začiatkom hry, t.j. po inicializácii hráča.

Formát správy : *(beam <x> <y> <rot>)*

Príklad: *(beam 10.0 -10.0 0.0)*

3. Say – spolu s Hear perceptorom umožňuje komunikáciu medzi hráčmi.

Formát správy : *(say <message>)*

Príklad: *(say ``helloworld"')*

2.1.2 Návrh

2.1.2.1 Vytvorenie stránky tímu

Budeme vytvárať statickú webovú prezentáciu, tak aby zdrojové súbory mohli byť prenosné a nezávislé od súborovej štruktúry. Použijeme technológie HTML, PHP a CSS. Dizajn bude vytvorený za pomoci profesionálneho grafického nástroja Adobe Photoshop verzie CS3. Dizajn bude tvorený na mieru, tak aby zodpovedal potrebám pre tímový projekt a reprezentoval náš tím vizuálne. Stránka bude obsahovať všetky predpísané sekcie tak, aby spĺňala požiadavky pre webovú prezentáciu tímového projektu.

2.1.2.2 Nástroj pre podporu projektu

Pre manažment verzií sme navrhli systém Dot Project, ktorý umožňuje zaznamenávať pridelovanie úloh jednotlivým členom tímu, ich plánovanie v závislosti od času a sledovanie plánu. Systém bude nainštalovaný na server hostingu, ktorý má jeden z členov tímu k dispozícii.

Na internú komunikáciu navrhujeme použiť systém Google groups. Táto služba nám umožňuje vytvoriť globálny alias, ktorý zahŕňa všetkých členov tímu a tak jednoducho môže jeden člen tímu kontaktovať ostatných. Samotná komunikácia medzi jednotlivcami bude riešená pomocou elektronickej pošty, prípadne prostredníctvom niektorého z nástrojov instant messaging.

2.1.2.3 Vytvorenie dokumentácie

Dokumentácia bude vytvorená a udržiavaná v elektronickej podobe. Na stránke tímu bude sprístupnená vo formáte doc alebo pdf. Dokumentácia bude mať predefinovanú štruktúru, aby opisovala jednotlivé šprinty a k nim prislúchajúce príbehy. Opis každého príbehu bude obsahovať analýzu, návrh, implementáciu a testovanie. V dokumentácii sa budú využívať predefinované štýly z dôvodu vizuálnej konzistencie dokumentu.

2.1.2.4 Výber hráča k ďalšiemu vývoju

Výber hráča k ďalšiemu vývoju bude konzultovaný pri vyhodnotení šprintu.

2.1.3 Implementácia

2.1.3.1 Vytvorenie stránky tímu

Webová stránka bola vytvorená pomocou HTML, CSS a PHP a je umiestnená na školskom servery LABSS2, je možné ju nájsť na adrese:

<http://labss2.fiit.stuba.sk/TeamProject/2010/team04is-si/>.

Webová stránka obsahuje nasledujúce sekcie:

- Domov – vizuálna reprezentácia tímu a témy projektu, aktuálne novinky
- O nás – predstavenie jednotlivých členov tímu
- Plán – plán projektu je členený do šprintov
- Na stiahnutie – dokumenty určené na stiahnutie (zápisnice, dokumentácia, ponuka)
- Linky – užitočné odkazy súvisiace s projektom

2.1.3.2 Nástroj pre podporu projektu

Systém Dot Project sa do ukončenia 1. Šprintu nepodarilo nainštalovať. Nastali problémy (komunikácia systému s databázou), ktoré sa nepodarili odstrániť a preto sa táto úloha presunula na začiatok druhého šprintu.

Pre komunikáciu členov tímu sme vytvorili účet na službe Google Groups. Vytvorili sme tak novú skupinu s názvom fiit-tim-04. Následne sme prostredníctvom pozvánky pozvali do skupiny všetkých členov tímu.

2.1.3.3 Vytvorenie dokumentácie

Dokumentácia bola vytvorená podľa návrhu softvérom Microsoft Word. Export do formátu pdf bol realizovaný pomocou softvéru PDF Creator.

2.1.3.4 Inštalácia serveru, editora pohybov a hráčov

Inštalácia serveru

Boli stiahnuté a nainštalované nasledujúce inštalácie:

1. MS Visual C++2008 Redistributable Package (x86).
2. Simulačný server Simspark v0.2
3. Samotný server pre RoboCup 3D rcssserver3d v0.6.3

Spustenie servera bolo zabezpečené skriptom:

C:\Program Files\rcssserver3d 0.6.3\bin\rcssserver3d.cmd.

RoboKopy

Spustenie hráča tímu zahŕňalo stiahnutie a rozbalenie súborov s hráčom. Následne bol hráč spustený z príkazového riadku pomocou súboru RoboKopy.exe, ktorého argument bol konfiguračný súbor. Po spustení monitoru nasledujúcim skriptom sa hráč spolu s loptou zobrazil na monitore.

C:\Program Files\rcssserver3d 0.6.3\bin\rcssmonitor3d.cmd,

Agenty 007

Ďalšou snahou bolo spustiť hráča tímu Agenty 007. Tento tím mal vytvorený editor súborov, ktorý už automaticky spúšťa server aj monitor, a tak si už len stačilo vybrať jeden z pohybov, ktorý mal byť realizovaný a spustiť hráča. Po spustení sa však vyskytla chyba s chýbajúcou knižnicou. Neskôr bolo zistené, že knižnica je súčasťou MS Visual C++2008 Redistributable Package (x86), a preto boli inštalované rôzne verzie, čo však neskončilo úspechom. Ďalším odporúčaním bolo nainštalovať kompletne vývojové prostredie MS Visual Studio 2010, ktoré daný problém taktiež nevyriešilo. Problém bol vyriešený až nainštalovaním nižšej verzie MS Visual Studio, konkrétne MS Visual Studio 2008. Následne bolo možné otestovanie hráča a vytvorenie a editovanie pohybu.

JIM

Pri spúšťaní fakultného hráča implementovaného v Jave s názvom JIM nastali mierne komplikácie. Po spustení servera sa hráč nechcel pripojiť na server. Hráč vyhadzoval výnimky ohľadom parsovania vstupného XML súboru. Problém bol identifikovaný a odstránený. Problém bol v XML súbore, ktorý mal síce správnu štruktúru, ale názov jedného z elementov bol nesprávny. V XML bol definovaný pohyb pre kĺb lae5, ktorý neexistuje, pretože ruka každého robota má len 4 kĺby. Po zmene názvu elementu na lae1 prebehlo pripojenie hráča v poriadku a po spustení monitoru sa hráč zobrazil.

NAO-Team Humboldt

Pri spúšťaní tohto zahraničného hráča nenastali žiadne komplikácie. Po spustení servera, hráča a následne monitora sa hráč zobrazil na monitore a predviedol plynulú a stabilnú chôdzu. S pripojením aj viacerých hráčov nebol žiaden problém. Hráči vykonávali pohyb chôdze po celej ploche.

2.1.3.5 Vytvorenie a editácia pohybov hráča

Po odstránení problémov pri inštalácii hráča tímu Agenty 007 a spustení editora pohybov si každý člen tímu vyskúšal vytvorenie a editovanie pohybu hráča a jeho následné spustenie.

2.1.4 Testovanie

2.1.4.1 Vytvorenie stránky tímu

Stránka bola spustená a odskúšaná každým členom tímu. Stránka je funkčná a jej fungovanie je bezproblémové. Nenastali žiadne komplikácie pri implementácii a spúšťaní.

2.1.4.2 Nástroj pre podporu projektu

Náš komunikačný nástroj Googlegroups sme otestovali zaslaním testovacích správ. Každý člen tímu poslal na globálny alias testovací mail. Tento testovací mail slúžil na kontrolu, či je každý člen v skupine zaregistrovaný korektne a či nám prichádzajú maily od jednotlivých členov. Toto testovanie skončilo úspešne – každý mail od člena tímu, prišiel ostatným kolegom. Druhý typ testovania, ktoré sme vykonali bolo zaslanie mailu z externej adresy – teda takej, ktorá nie je priradená v našej skupine. Tento mail poslal jeden z členov tímu z adresy inej ako bola zaregistrovaná v skupine. Test však neprebehol správne, pretože správa sa vrátila ako nedoručená. Identifikovali sme problém, že nám nechodia správy z externého prostredia. Problém sme vyriešili v nastaveniach služby, kde sme museli explicitne povoliť možnosť prijímať správy z externého zdroja. Následne sme test zopakovali a správa poslaná z externej emailovej adresy bola doručená všetkým členom tímu. Test tak prebehol úspešne.

2.1.4.3 Vytvorenie dokumentácie

Dokumentácia bola opravovaná na štylistické a gramatické chyby členmi tímu.

2.2 Šprint č. 02

Číslo šprintu	02
Počet príbehov	10
Začiatok šprintu	21.10.2010
Koniec šprintu	04.11.2010

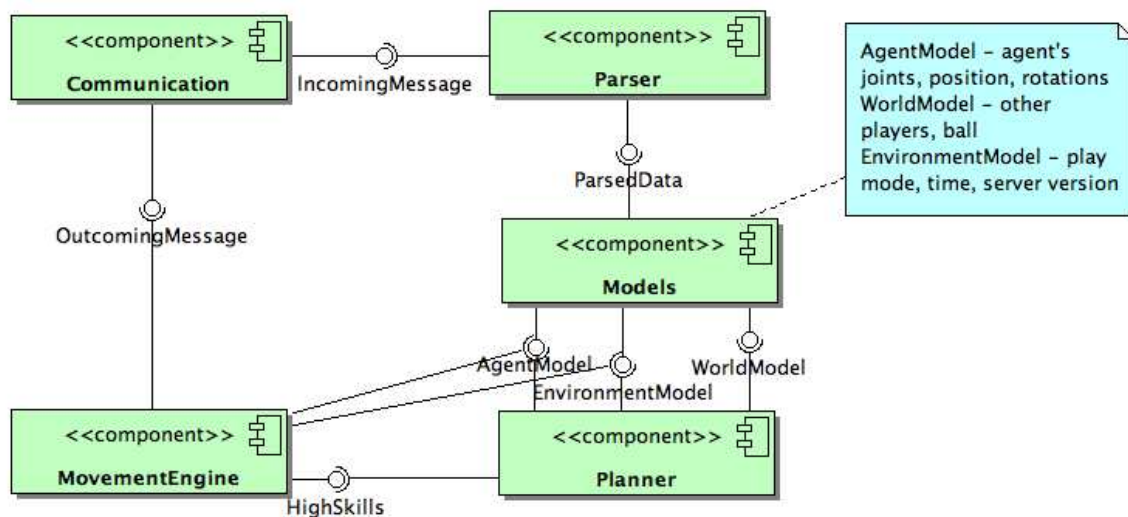
Príbehy

Vytvoriť základ pre dokumentáciu riadenia projektu
Zriadenie repozitára pre manažment verzií
Nainštalovanie klienta manažmentu verzií a vykonanie skúšobného commitu
Oboznámenie sa so zdrojovými kódmi vybraného hráča, jeho kompilácia a spustenie
Analýza kódu hráča - časť zodpovedná za prijímanie správ - napísanie unit testov a prípadný refaktoring
Analýza kódu hráča - časť zodpovednú za odosielanie správ – napísanie unit testov a prípadný refaktoring
Analýza kódu hráča - časť zodpovednú fyziku - napísanie unit testov a prípadný refaktoring
Analýza kódu hráča - časť zodpovedná za riadiacu logiku - napísanie unit testov a prípadný refaktoring
Celková analýza kódu, zamyslieť sa ako kód rozšíriť – vytvoriť hrubý návrh
Nástroje pre podporu projektu

2.2.1 Analýza

2.2.1.1 Analýza prevzatých kódov hráča JIM

Táto úloha má za cieľ oboznámiť sa so zdrojovými kódmi hráča, odhaliť potencionálnych problémov a prípadný refaktoring na nižšej úrovni (odstrániť zjavné chyby v neefektívnosti, skomentovať nejasné veci a podobne). Po analýze prevzatého zdrojového kódu sme dospeli k niekoľkým poznatkom. Hráč JIM sa odvíja od hráča tímu Robokopy. Rozdiel je v implementačnom jazyku. Pri hráčovi JIM je tomu Java. Jadro hráča je implementované v Jave, nižšia strednú logiku je čiastočne možné riadiť pomocou Ruby skriptov. Hlavné komponenty hráča JIM sú znázornené na Obrázku 2.2 - 1.



Obrázok 2.2-1 – Komponenty hráča JIM

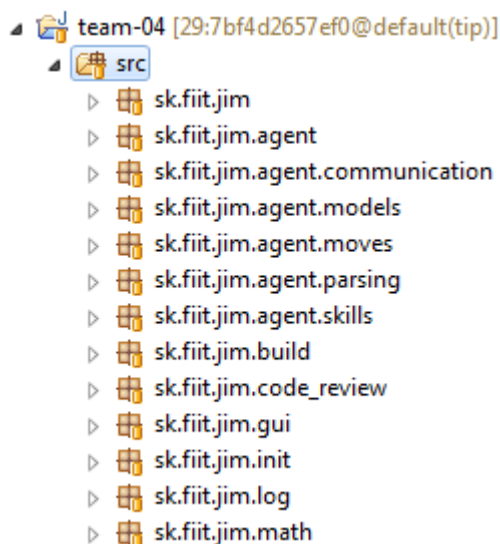
Hráč JIM sa skladá z piatich komponentov, ktoré sú navzájom poprepájané rozhraniami. Dodržiava sa tak pravidlo slabšej závislosti medzi komponentmi. Popis jednotlivých komponentov je nasledujúci:

- **Communication** – táto časť samotná slúži na komunikáciu so serverom, implementuje sieťový protokol a analyzuje správy.
- **Parser** – slúži na parsovanie dát zo serveru.
- **Models** – Obsahuje model agenta, sveta (objektov v nom) a samotného hracieho prostredia – aktuálny čas, mód, v ktorom sa nachádza hráč, verzia serveru.
- **Planner** – plánuje, ktoré akcie sa vykonajú v čase.
- **MovementEngine** – zodpovedá za pohyb hráča po ihrisku.

Komponenty sa skladajú z viacerých tried, ktoré sú úplne. Kód je funkčný.

Štruktúra kódu

Štruktúra balíkov zdrojových kódov sa nachádza na Obrázku 2.2 - 2. Kód je členený prehľadne do jednotlivých balíčkov. Nevýhodou je však, že kód je zároveň slabo komentovaný, na niektorých miestach chýbajú komentáre úplne. Preto je na niektorých miestach orientácia priamo v kóde zhoršená.



Obrázok 2.2-2 - Balíčky prevzatých zdrojových kódov

Každý z balíčkov obsahuje len niekoľko tried, ktoré skutočne logicky patria do daného balíka. V každom balíku sa okrem tried nachádzajú aj ich testovacie triedy.

2.2.1.2 Analýza najdôležitejších tried

V nasledujúcej časti uvádzame analýzu najdôležitejších častí hráča v tomto šprinte.

Analýza riadiacej logiky

Riadiaca logika hráča Jim má niekoľko častí. Prvou je balík „sk.fiit.jim.init“ tento má na starosti samotné spustenie hráča a procesov so spustením spojených ako načítanie pohybov či spojenie so serverom. Obsahuje triedy:

- Main – vstupná trieda, zabezpečuje spustenie hráča
- Script – spúšťa a interpretuje skripty
- ScriptBoot – určuje ktoré skripty sa využijú
- SkillsFromXmlLoader – načíta pohyby(schopnosti) z XML súborov

Ďalšou časťou je balík „sk.fiit.jim.agent.move“ v tomto balíku sú umiestnené dátové a obslužné triedy pohybov a to konkrétne tieto:

- EffectorData – dátová trieda, ktorá obsahuje kĺb a koncový uhol
- Joint – dátová trieda(Enumeration), obsahuje mená kĺbov a rozmedzia uhlov v ktorých sa môžu pohybovať, zároveň vie prekladať mená kĺbov do jazyka serveru
- JointPlacement – dátová a obslužná trieda obsahujúca dáta o požadovanej pozícii kĺbov v závislosti na vykonávanú fázu
- LowSkill – informácie o schopnosti
- LowSkills – zoznam načítaných schopností nadefinovaných v XML súbore(súboroch), pravdepodobne začiatok podpory pre strednú a vyššiu logiku
- Phase – dátová a obslužná trieda pre fázy
- Phases – fázy a ich údaje pre jednotlivé schopnosti(pohyby)

Treťou časťou je balík „sk.fiit.jim.agent.skills“, ktorý sa však ešte nepoužíva a je to iba začiatok implementácie pre vyššie logiky. Poslednou časťou zodpovednou za riadiacu logiku sú Ruby skripty ktoré majú na starosti výber a spúšťanie jednotlivých schopností z množiny načítanej do triedy LowSkills. Momentálne sú skripty aj pohyby nadefinované tak, že sa cyklicky vykonáva iba jeden, respektíve dva pričom prvý je teleport na pozíciu určenú v „rollback.xml“. Skript zodpovedný za výber pohybu a jeho vykonanie je „plan.rb“.

Analýza časti zodpovednej za fyziku

Hráč Jim má triedy a metódy spojené s fyzikou združené v dvoch balíkoch. Prvý je „sk.fiit.jim.agent.models“ a má na starosti správu sveta a všetkých objektov v ňom. Obsahuje nasledovné triedy:

- AgentModel – model hráča
- AgentPositionCalculator – určuje aktuálnu pozíciu hráča
- DynamicObject – slúži na detekciu predpoved' pozície pohybujúcich sa objektov
- EnviromentModel – model prostredia serveru(verzie)
- FixedObject – dátová trieda s pozíciami fixných objektov(rohy, bránky)
- KalmanAdjuster – Kalmanovým filtrom filtruje prijaté informácie zo servera o polohe lopty a vlajok aby odstránil šum
- WorldModel – model sveta a objektov v ňom, udržiava informácie o ich polohe, rýchlosti a pod.

Druhým balíkom je „sk.fiit.jim.math“ sem sa nachádzajú pomocné matematické funkcie. Tento balík obsahuje nasledovné triedy:

- Angles – výpočty súvisiace s uhlami
- KalmanForVariable – Kalmanov filter pre premennú
- KalmanForVector – Kalmanov filter pre vektor
- MathExpressionEvaluator – vyhodnocuje matematické výrazy zadané ako textový reťazec v syntaxe jazyka Ruby

2.2.1.3 Analýza miest pre rozšírenie

Pri analýze zdrojových kódov boli zistené nasledujúce časti, ktoré je potrebné lepšie zanalyzovať a doimplementovať:

- Akcelerometer v modele Agentu (hráč nevie či je vo vzduchu alebo na zemi)
- Vzorce pre výpočet vektorov v triede `3DVector` sú podľa testov správne, napriek tomu je potrebné bližšie preskúmať, či sú použité správne.
- Všetky triedy s balíka `sk.fiit.jim.agent.skills` – je to iba začiatok implementácie vyššej logiky

2.2.1.4 Analýza testovania

Pre testovanie v existujúcom projekte sa používa framework JUnit 4.0 a autor využíva štandardné metódy tohto frameworku. Framework JUnit 4.0 budeme využívať aj my pri ďalšom testovaní, pretože nám umožňuje jednoduché, veľmi flexibilné a zároveň úplne testovanie.

Prevzatý zdrojový kód je na vysokej úrovni otestovanosti. Takmer pre každú triedu v projekte existuje testovacia trieda, ktorá testuje metódy danej triedy. Testy pokrývajú a overujú väčšinou základné možnosti tried, v niektorých prípadoch je test naozaj detailný a pokrýva všetky relevantné možnosti, akými je možné testovať danú triedu. Ukážka testu je zobrazená na Obrázku 2.2 - 3.

Testy sa nachádzajú v balíku pri triede, ktorá je nimi testovaná. Ich pomenovanie sa skladá z názvu metódy, ktorú testujú a slovom `Test` na konci. Napríklad `SettingsTest`. Všetky testy vytvorené v projekte prejdú po spustení bez chyby a projekt je teda na základe týchto testov validný.

```
*Vector3DTest.java X
12  *
13  *@Title      Jim
14  *@author     $Author: marosurbanec $
15  */
16  public class Vector3DTest{
17
18      @Test
19      public void basicOperations(){
20          Vector3D one = cartesian(1, 1, 1);
21          Vector3D zero = cartesian(0, 0, 0);
22
23          assertEquals(one.add(zero), is(equalTo(one)));
24          assertEquals(one.add(cartesian(1, 0, 0)), is(equalTo(cartesian(2, 1, 1))));
25
26          assertEquals(one.multiply(5.0), is(equalTo(cartesian(5, 5, 5))));
27          assertEquals(one.multiply(0.0), is(equalTo(cartesian(0, 0, 0))));
28
29          assertEquals(one.divide(0.2), is(equalTo(cartesian(5, 5, 5))));
30          assertEquals(one.divide(0.2), is(equalTo(cartesian(5, 5, 5))));
31
32          assertEquals(one.subtract(cartesian(0, 1, 0)), is(equalTo(cartesian(1, 0, 1))));
33
34          assertEquals(one.negate(), is(equalTo(cartesian(-1, -1, -1))));
35          assertEquals(zero.negate(), is(equalTo(cartesian(0, 0, 0))));
36
37          assertEquals(one.addX(1.0), is(equalTo(cartesian(2, 1, 1))));
38          assertEquals(one.addY(-1.0), is(equalTo(cartesian(1, 0, 1))));
39          assertEquals(one.addZ(0.0), is(equalTo(one)));
40
41      // original vector should remain the same - add, multiply, divide,
42      // subtract, negate and other operations are immutable
43      assertEquals(one, is(equalTo(one)));
44  }
```

Obrázok 2.2-3 – Ukážka existujúceho testu

Niektoré testy nezahŕňajú všetky možnosti. Tieto testy je preto potrebné skontrolovať, refaktorovať a pridať nové testovacie prípady.

Zoznam testov, ktoré sa nachádzajú v projekte:

- AgentModelTest.java
- DynamicObjectTest.java
- KalmanAdjusterTest.java
- WorldModelTest.java
- LowSkillTest.java
- ParserTest.java
- Perceptors.java
- FakeHighSkill.java
- ParserToModelIntegrationTest.java
- ClassToPackTest.java
- DirectoryMoverTest.java
- JarFileBuilderTest.java
- ManifestBuilderTest.java
- ScriptTest.java

- SkillFromXmlLoaderTest.java
- LogTest.java
- AnglesTest.java
- KalmanTest.java
- MathExpressionEvaluatorTest.java
- Vector3DTest.java
- SettingsTest.java

2.2.1.5 Manažment verzií

Po vzájomnej dohode bol ako nástroj pre manažment verzií vybraný Mercurial. Mercurial je distribuovaný systém pre podporu revízií a verziovania zdrojových kódov a iných dokumentov. Pri verziovaní je možné využívať server a lokálne úložiská s kópiami zdrojových kódov, ktoré predstavujú klientov. Je preto potrebné nájsť a zriadiť vhodný server a vybrať vhodný klient pre verziovanie zdrojových súborov.

2.2.2 Návrh

2.2.2.1 Návrh v analýze zdrojových kódov

V tomto behu je potrebné prejsť a opraviť nasledujúce časti prebratých zdrojových kódov hráča, zodpovedných za:

- Odosielanie správ
- Prijatie správ
- Fyzika hráča
- Riadiacu logiku hráča

Pri oprave sa zameriavame na zjavné chyby, nesprávne pomenovania a nedostatočné komentáre pri zdrojových súboroch.

2.2.2.2 Návrh testovania

V tomto behu je potrebné prejsť, upraviť prípadne napísať nové testy, pre otestovanie časti prebratých zdrojových kódov hráča, zodpovedných za:

- Odosielanie správ
- Prijatie správ
- Fyzika a matematika hráča
- Riadiacu logiku hráča

Pri oprave sa zameriavame na zjavné chyby, nesprávne pomenovania a nedostatočné komentáre pri zdrojových súboroch.

2.2.2.3 Návrh manažmentu verzií

Bude potrebné zriadiť repozitár s podporou pre manažment verzií projektu. Repozitár musí byť prístupný online a každým členom tímu, s možnosťou jeho jednoznačnej identifikácie, tak aby bolo možné sledovať nielen zmeny, ale aj osobu, tieto zmeny vykonávajúcu.

2.2.2.4 Vytvorenie dokumentácie k riadeniu

Dokumentácia bude vytvorená a udržiavaná v elektronickej podobe. Na stránke tímu bude sprístupnená vo formáte doc alebo pdf. Dokumentácia bude mať predefinovanú štruktúru podľa štábnej kultúry., aby opisovala ponuku, plánovanie, úlohy členov tímu, autorstvo jednotlivých častí dokumentácie, zápisnice, metodiky, manažment projektu a preberacie protokoly. Formálna úprava bude riešená podľa štábnej kultúry opísanej v dokumentácii k riadeniu.

2.2.2.5 Aktualizácia webovej stránky

Bude nevyhnutné opätovné aktualizovať webovú prezentáciu nášho tímu, aby boli verejnosti prístupné aktuálne informácie o prograse v tímovom projekte.

2.2.3 Implementácia

2.2.3.1 Implementácia úpravy zdrojových kódov

Pre úpravu zdrojových kódov sme použili nástroj Eclipse. Zdrojové kódy bolo potrebné ručne prejsť a opraviť prípadne nedostatky. Triedy, ktoré zabezpečujú jednotlivé časti sú nasledovné:

Časť pre odosielanie správ

Trieda	Poznámky

Časť pre prijímanie správ

Trieda	Poznámky
ParsedData	Doplnené argumenty o perceptoroch typu hear a bumper.
Perceptors	Doplnená podpora spracovania správ perceptorov typu hear a bumper.
HearReceptor	Vytvorená nová dátová trieda HearReceptor pre perceptor typu hear
Communication	Prijímanie správ prebehlo OK

Časť pre fyziku hráča

Trieda	Poznámky
Angles	Malé úpravy formátovania, inak OK
KalmanForVariable	OK
KalmanForVector	OK
MathExpressionEvaluator	Dopísané ošetrojúce podmienky, inak OK
Vector3D	Nie je zatiaľ jasné, či používa správne vzorce, inak OK

Časť riadiacej logiky

Trieda	Poznámky
AgentModel	Nie je implementovaný akcelerometer.
AgentPositionCalculator	Dopísané ošetrojúce podmienky, inak OK
DynamicObject	Ok
EnviromentalModel	Dopísané ošetrojúce podmienky, inak OK
FixedObject	Ok
KalmanAdjuster	Malé úpravy formátovania, inak Ok
WorldModel	

2.2.3.2 Implementácia testovania

Pre úpravu zdrojových kódov testov sme použili nástroj Eclipse. Eclipse má v sebe priamo podporu pre písanie testov vo frameworku JUnit, ktoré je možné priamo spúšťať a vyhodnocovať. Zdrojové kódy testov bolo potrebné ručne prejsť a opraviť ich prípadne nedostatky. V prípade, že test pre danú triedu neexistoval, bolo treba vytvoriť nový test.

Časť pre odosielanie správ

Trieda	Testovacia trieda	Poznámky

Časť pre prijímanie správ

Trieda	Testovacia trieda	Poznámky
ParsedData	ParsedDataTest	Dopísané testovanie pre perceptory hear a bumper. Testy prebehli OK.
Perceptors	PerceptorsTest	Príjímanie správ prebehlo OK

Časť pre fyziku hráča

Trieda	Testovacia trieda	Poznámky
Angles	AnglesTest	Dopísané ďalšie možnosti, ktoré mohli nastať, test prebehol OK
KalmanForVariable	KalmanTest	OK
KalmanForVector	KalmanTest	OK
MathExpressionEvaluator	MathExpressionEvaluatorTest	Dopísané ďalšie možnosti, ktoré mohli nastať, test prebehol OK
3DVector	3DVectorTest	Dopísané ďalšie možnosti, ktoré mohli nastať, test prebehol OK

Časť riadiacej logiky

Trieda	Testovacia trieda	Poznámky
AgentModel	AgentModelTest	Ok
AgentPositionCalculator	AgentPositionCalculatorTest	Zmenené formátovanie, inak Ok.
DynamicObject	DynamicObjectTest	Dopísané ďalšie možnosti, ktoré mohli nastať, test prebehol OK
EnviromentalModel	EnviromentalModelTest	Upravené formátovanie, inak OK
FixedObject	FixedObject	Ok
KalmanAdjuster	KalmanAdjusterTest	Zmenené formátovanie, dopísaných zopár komentárov, test prebehol Ok.
WorldModel	WorldModelTest	Dopísané ďalšie možnosti, ktoré mohli nastať, test prebehol OK

2.2.3.3 Implementácia manažmentu verzií

Po vzájomnej dohode bol ako nástroj pre manažment verzií vybraný Mercurial. Mercurial je distribuovaný systém pre podporu revízií a verziovania zdrojových kódov a iných dokumentov. Pri verziovaní je možné využívať server a lokálne úložiská s kópiami zdrojových kódov, ktoré predstavujú klientov. Je preto potrebné nájsť a zriadiť vhodný server a vybrať vhodný klient pre verziovanie zdrojových súborov.

Server pre manažovanie verzií bol pomocou na webovej službe bitbucket.org. Služba ponúka vytvorenie centrálneho úložiska pre projekty, ktoré sú prístupné na stiahnutie pre všetkých používateľov služby. Vytvárať zmeny na serveri však môžu len schválení používatelia. Úložisko sa nachádza na adrese <https://bitbucket.org/xpagaca/team-04>. V úložisku sa nachádzajú zdrojové súbory, ktoré sme prebrali od minuloročného tímu a ďalšie dokumenty, ktoré sú predmetom verziovania.

Pre prístup do úložiska si musel každý člen tímu vytvoriť konto na portáli bitbucket.org a následne zaslať svoje prihlasovacie meno Adamovi Pagáčovi. Ten pridal členov tímu medzi osoby, ktoré môžu projekt upravovať.

Nástroje, ktoré sme zvolili na verziovanie prostredníctvom systému Mercurial:

- TortoiseHg, dostupný na adrese <http://tortoisehg.bitbucket.org/download/index.html>
- zásuvný modul do Eclipse – MercurialEclipse, ktorý je prístupný v Eclipse pomocou nasledujúceho postupu:
 1. Nainštalovať a spustiť Eclipse
 2. V ponuke menu vybrať „Help“ -> „Instal New Software“(„Software updates“, záleží od verzie Eclipse)
 3. V otvorenom dialógovom okne zadať do položky „Work with“ adresu „<http://cbes.javaforge.com/update>“
 4. Následne vybrať z ponuky MercurialEclipse a dať Instal
 5. Reštartovať Eclipse

2.2.3.4 Vytvorenie dokumentácie k riadeniu

Dokumentácia bola vytvorená podľa návrhu softvérom Microsoft Word. Export do formátu pdf bol realizovaný pomocou softvéru PDF Creator.

2.2.3.5 Aktualizácia webovej stránky

Web stránka bola aktualizovaná podľa stávajúcich požiadaviek. Boli upravené sekcie pre plán tímu, ako aj pridané nové dokumenty na stiahnutie.

2.2.4 Testovanie

2.2.4.1 Testovanie úpravy zdrojových kódov

Správnosť opravených súborov bola overená prostredníctvom kompilátora javac. Všetky zdrojové súbory podarilo skompilovať – sú tak syntakticky správne. Keďže sa v tejto fáze nerobili veľké zmeny v kóde (zmeny boli len na úrovni komentárov, alebo iných drobných zmien), nie je predpoklad, že by sa niektorá časť stala nevalidnou. Okrem toho sa spustili všetky testy a tie prebehli úspešne.

2.2.4.2 Testovanie správnosti testov

Správnosť opravených testovacích súborov bola podobne ako pri zdrojových súborov tried overená prostredníctvom kompilátora javac. Všetky zdrojové súbory podarilo skompilovať – sú tak syntakticky správne. Keďže sa v tejto fáze nerobili veľké zmeny v kóde (zmeny boli len na úrovni komentárov, prípadne sprísňovanie testov), nie je predpoklad, že by sa niektorá časť stala chybnou. Všetky testy po spustení prebehli úspešne.

2.2.4.3 Testovanie manažmentu verzií

Po inštalácii pluginu do Eclipse nasledoval skúšobný commit, ktorý po importovaní projektu do vývojového prostredia spočíval len v nasledujúcich činnostiach:

1. Vytvoriť prípadne zmeniť súboru, ktorý chceme commit-núť
2. Kliknúť na tento súbor, alebo ktorýkoľvek jeho rodičovský balík prípadne celý projekt, pravým tlačidlom v okne „Project Explorer“ a vybrať „Team“ -> „Commit“

Vyššie opísané činnosti uložili zmeny v súbore do lokálneho úložiska. Ak chceme zmenu preniesť do centrálného úložiska po tom ako spravíme „Commit“ vykonáme rovnakým spôsobom ešte „Push“. Tieto činnosti sa podarili splniť takmer všetkým členom tímu. Menšie problémy s prihlásením, ktoré sa vyskytli a s tým súvisiaci neúspešný commit, boli odkonzultované a odstránené. Úložisko beží bezproblémovo a je spoľahlivé.

2.2.4.4 Vytvorenie dokumentácie k riadeniu

Dokumentácia bola opravovaná na štylistické a gramatické chyby členmi tímu.

2.2.4.5 Testovanie aktualizácie webovej stránky

Pri aktualizácii nevznikli žiadne problémy a webová prezentácia funguje rovnako bezproblémovo, ako tomu bolo pred aktualizáciou.

3 Zdroje

Táto kapitola obsahuje zoznam zdrojov použitých pri vypracovaní projektu.

[1] Gelányi, L., a kol.: *Projektová dokumentácia tímu 17*, STU, FIIT, (2009).

dostupné elektronicky: http://labss2.fiit.stuba.sk/TeamProject/2009/team17isi/files/CE_Dokumentacia_final_LetnySemester.doc

[2] Ding, K., Zhao, Y., Huang, Y., Zhang, Z.: *Apollo 3D Humanoid Simulation Team Description*, College of Automation, Nanjing University of Posts and Telecommunications, (2009).

dostupné elektronicky:

http://kucitypic.kasetsart.org/kucity.com8/kucity8_roboocup2009_Symposium/tdps/3d-simulation/3d_simulation_Apollo3D.pdf

[3] <http://roboocup.njupt.edu.cn/>

[4] <http://www.naoteamhumboldt.de/>

[5] <http://naoteam.imn.htwk-leipzig.de/>

[6] Davari, M., et al.: *Alzahra Soccer 3D Simulation Team Team Description Paper for RoboCup 2010*, Alzahra University, Tehran, Iran, (2010).

dostupné elektronicky:

http://www.alzahrarobocup.com/upl/1008271282918375alzahra_TDP.pdf

[7] <http://sites.google.com/site/parsiansoccer3d/d>

[8] Dorer, K., et al.: *The magmaOfenburg 2010 RoboCup 3D Simulation Team*, Hochschule Ofenburg, Elektrotechnik-Informationstechnik, Germany, (2010).

dostupné elektronicky:

<http://www.et->

[it.fhoffenburg.de/prof/kdorer/roboocup/magmaOffenburg/downloads/magmaOffenburg_TDP2010.pdf](http://www.et-it.fhoffenburg.de/prof/kdorer/roboocup/magmaOffenburg/downloads/magmaOffenburg_TDP2010.pdf)

[9] Dorer, K., et al.: *The magmaOfenburg 2009 RoboCup 3D Simulation Team*, Hochschule Ofenburg, Elektrotechnik-Informationstechnik, Germany, (2009).

dostupné elektronicky:

<http://www.et-it.fh->

[hoffenburg.de/prof/kdorer/roboocup/magmaOffenburg/downloads/magmaOffenburg_TDP2009.pdf](http://www.et-it.fhoffenburg.de/prof/kdorer/roboocup/magmaOffenburg/downloads/magmaOffenburg_TDP2009.pdf)