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 Vasiľ

Kontakt: fiit-tim-04@googlegroups.com

Obsah

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 súťažiace 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 odstránime 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. Našim 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

V tejto kapitole sa nachádza analýza zadaných úloh pre prvý šprint.

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 projekt. 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, prideľovanie 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.

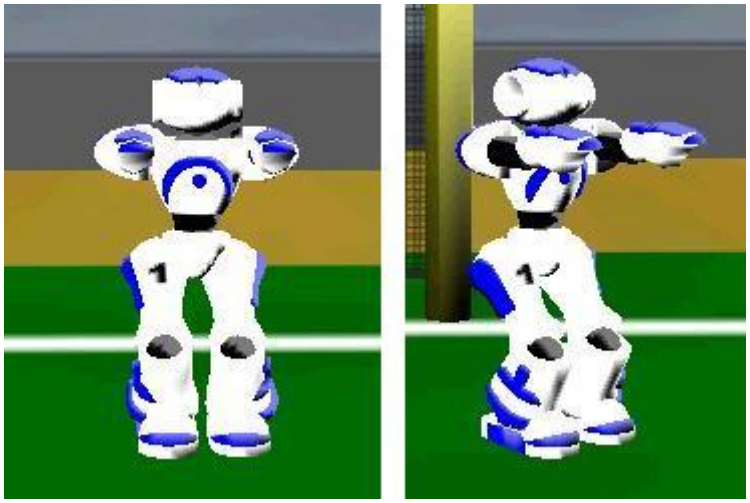
2.1.1.2 Analýza fakultných hráčov

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 prenesie svoje ťažisko na pravú nohu, zodvihne ľavú, nakloní sa doľava, prenesie 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, čim prenesie 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 chodila 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 prenesie 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 prenesie po osi bedrového kĺbu a loptu trafí 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
- Vybiehanie 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 funkcionalita 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 vykonávané 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.

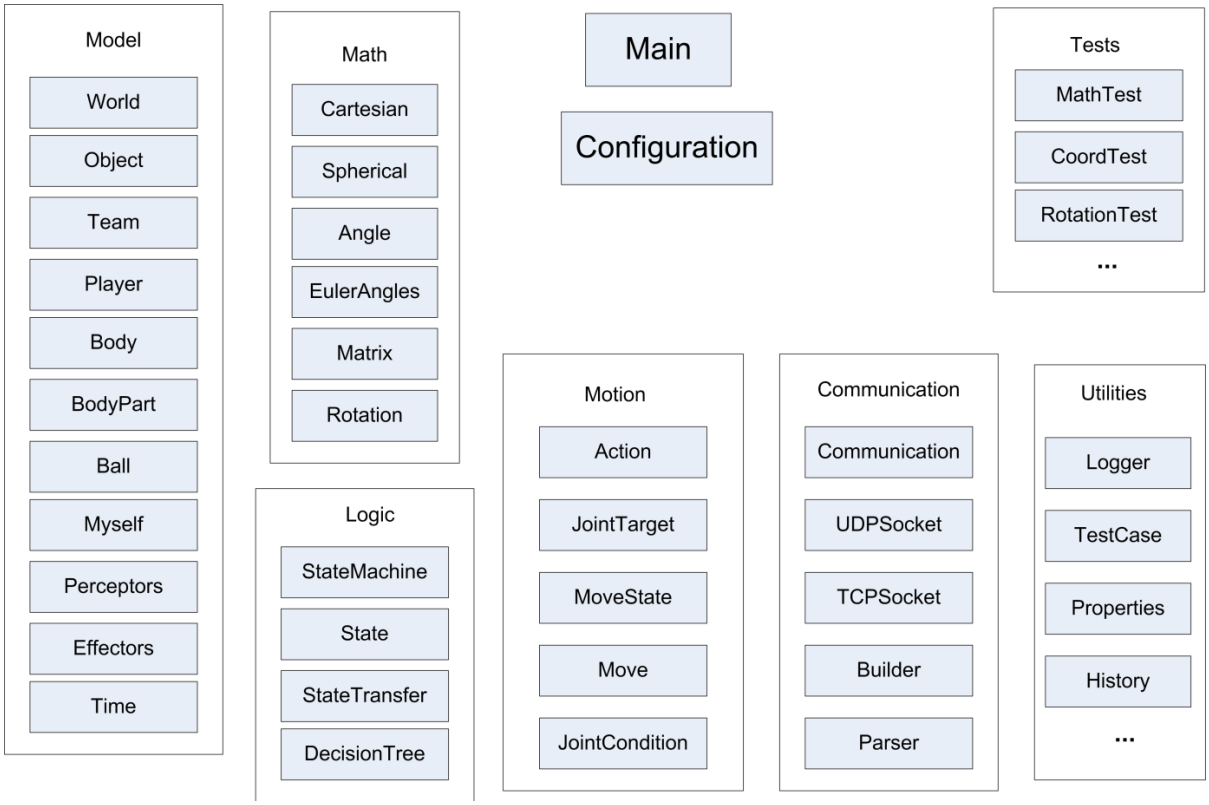
Tento hráč slúži ako dobrý základ, avšak je vo veľmi skorej fáze vývoja. Na tohto hráča však nadväzovali ďalšie tímové projekty a preto je pôvodný hráč tímu Agenty 007 pre náš projekt pomerne neperspektívny.

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 svety a objektov v ňom,
- Math - pomocné matematické operácie a štruktúry,
- Logic - stavový automat a prechodové funkcie,
- Motion - zabezpečenie pohybu,
- Comunication – komunikácia so serverom,
- Utilities – pomocné triedy,
- Test – obsahuje TestCase niektorých tried a algoritmov.



Obrázok 2.1-2 – Moduly projektu tímu Robokopy

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 tímom 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 Singapure, ale aj ďalšie zaujímavé tímy so zaujímavými technikami.

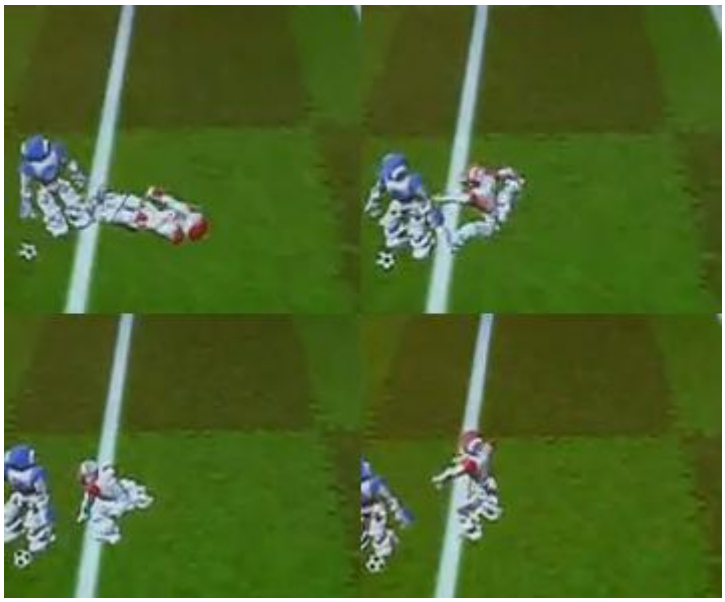
Apollo 3D

Tento tím sa stal víťazom turnaja Robocup 3D 2010 v Singapure, keď vo finále zvíťazil nad tímom NAO-Team Humboldt v predlžení 1:0. Tím Apollo 3D [3] bol založený študentmi na Univerzite pôšt 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 kombinácií 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 situácia s ostatnými hráčmi padá. Hráč má definované veľmi rýchle pohyby vstavenia, ktoré je možné vidieť na Obrázkoch 2.1 - 3., 2.1 - 4. Kopy hráča sú plynulé a hráč plynule prechádza z pohybu chôdze do pohybu kopu.

Správanie hráča z taktického hľadiska je také, že rieši útočné akcie najčastejšie individuálne. V minimálnom množstve sú útočné akcie riešené aj nahrávkami. V individuálnych útočných akciách hráč obchádza ostatných hráč za účelom eliminácie konfliktných situácií. Z dôvodu urýchlenia pohybu hráča k lopte, nedochádza v istých situáciách k otočeniu hráča k lopte a následnom pohybe chôdze k lopte, ale okamžitým pohybom chôdze vzad smerom k lopte.

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 Singapure. Výskum tímu je orientovaný hlavne na techniku hráča a strojové učenie s aplikáciou v kognitívnej robotike.

Hráč tímu NAO-Team Humbolt 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.

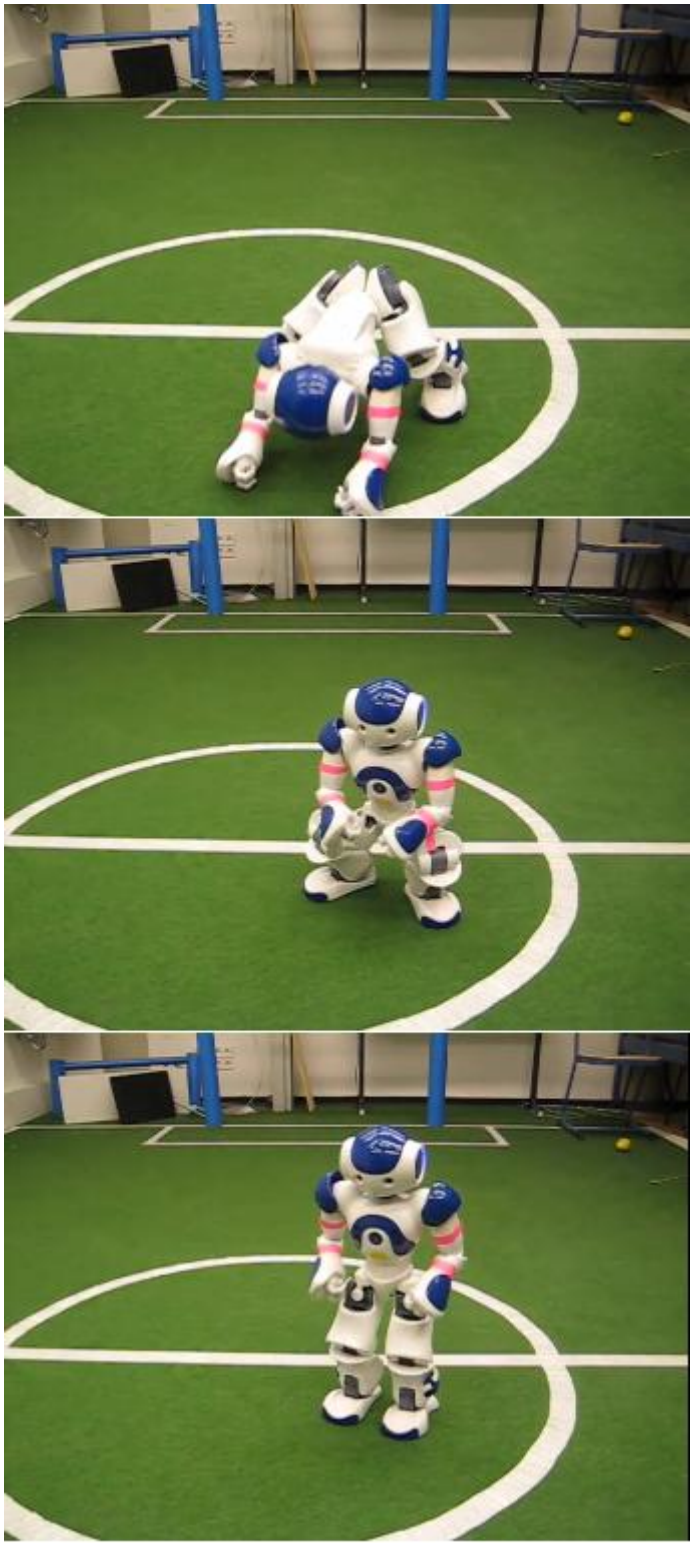
Hráč tohto tímu využíva na spracovanie obrazu rozoznávanie farieb na mriežke o veľkosti osemdesiat na šesťdesiat bodov. Objektmi spracovania sú hlavne lopta, čiary a ostatní hráči. Pomocou rôznych filtrov získava informácie o relatívnych a absolútnych pozíciách. Napríklad informácia o globálnej pozícií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

Pohyby hráča sú vytvárané pomocou nástroja vytvoreného týmto tímom. Dynamické pohyby sú vytvárane z predefinovaných kopov a pohybov do rôznych strán s využitým inverznej kinetiky a výpočtov na základe údajov zo snímačov. Správanie hráča je vykonávané pomocou stavové stromu, 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 sústredí 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 rýchlosti vykonania pohybov a kontrolou podkladu hráča alebo na dynamické kopy s prihliadaným na stav lopty a plynulý prechod od chôdze ku kopu.

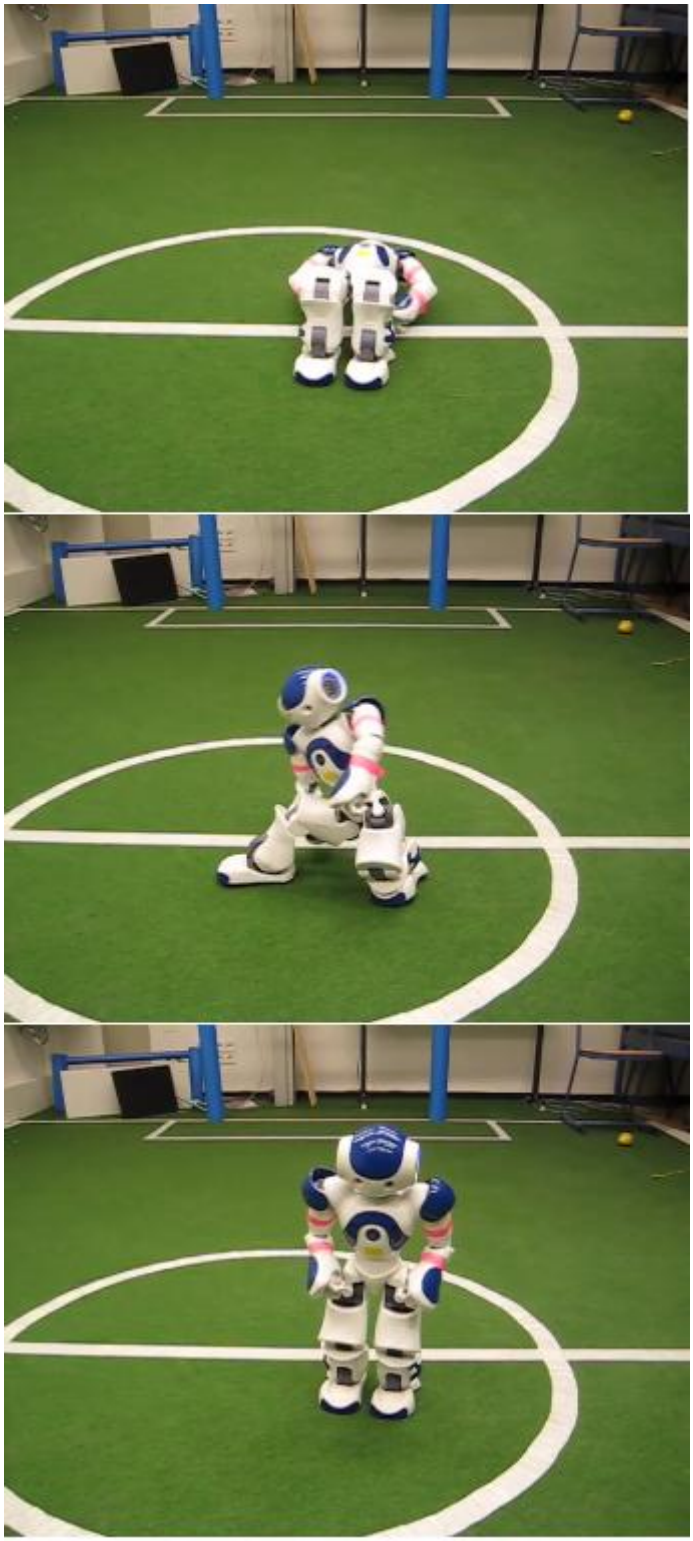
Na stránke tímu NAO-Team Humboldt sú prístupné aj videá zobrazujúce najrozvinutejšie pohyby hráča ako je kop, bočný kop, hľadanie lopty, nasledovanie lopty, vstávanie z brucha, vstávanie z chrbta a iné. Obrázky 2.1 - 5, 2.1 - 6, 2.1 - 7 zobrazujú niektoré z pohybov hráča. Tento tím dokonca poskytol aj 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 komunikácií 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]

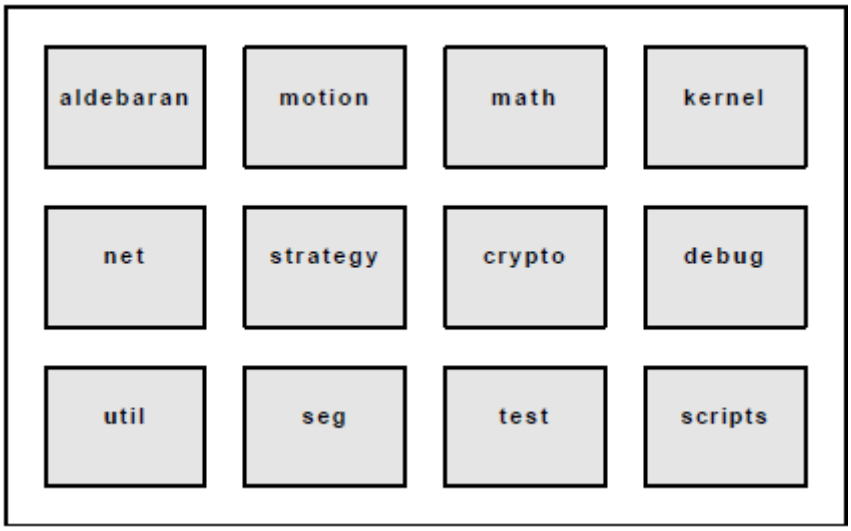
Nao-Team HTWK

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 frameworku, ktorý je v samotných Nao robotoch. Rozširuje však NaoQI 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 2.1 - 8 zobrazuje architektúru 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 kĺbov 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.

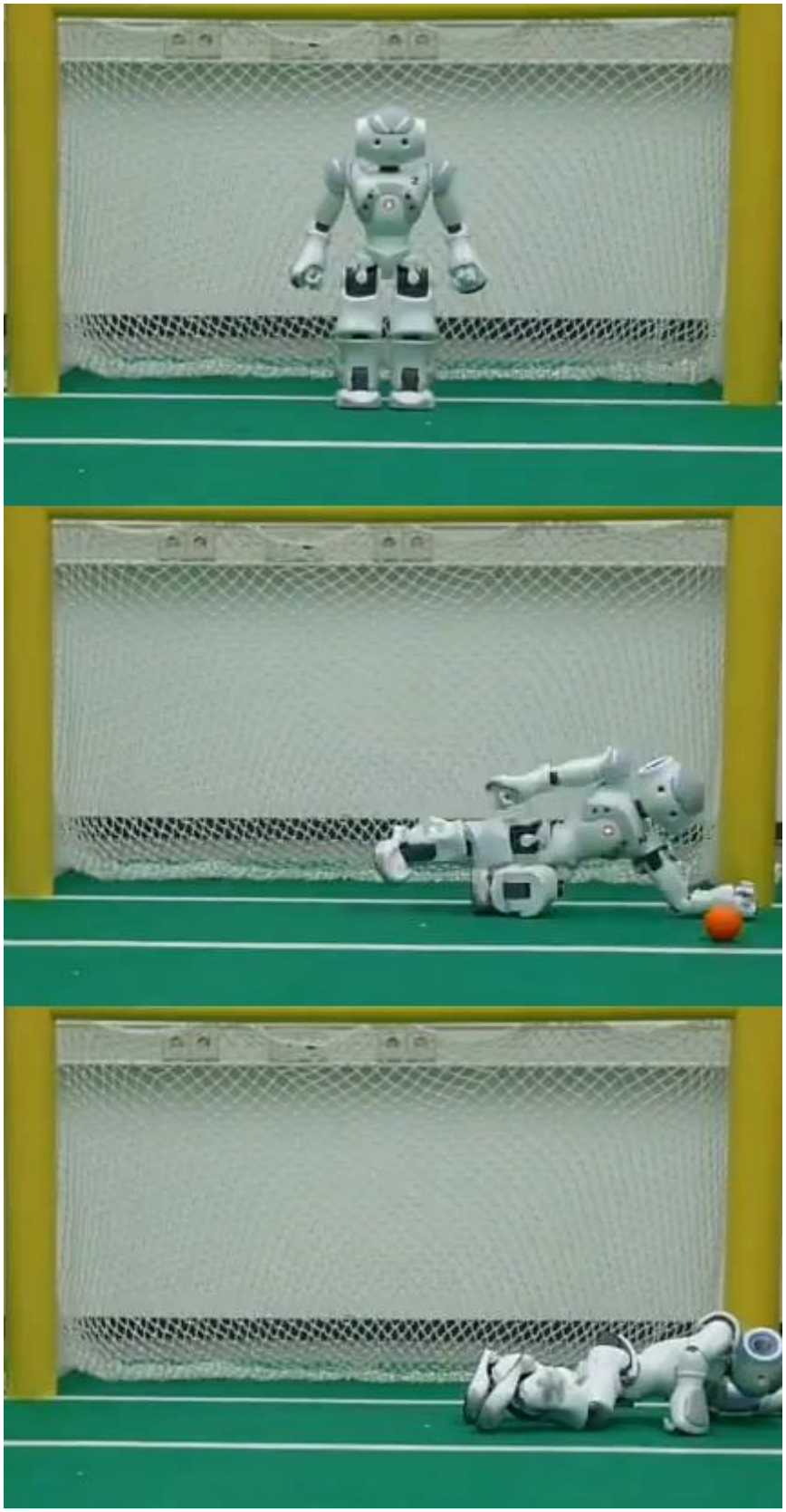
Pohyby

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 rozpoznávane 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 2.1 - 9.



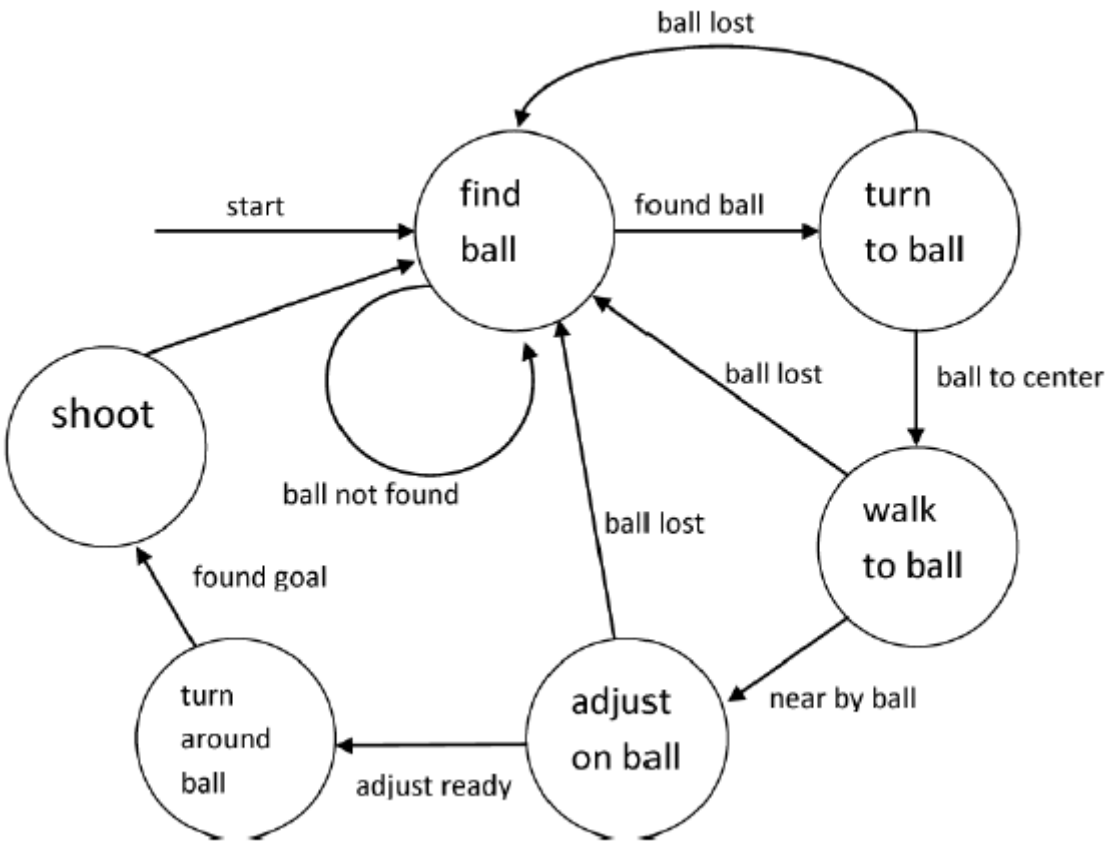
Obrázok 2.1-9 – Pohyb brankára [5]

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 2.1 - 10. Je zložený zo šiestich stavov. K rýchlejšiemu vyhodnocovaniu stratégie sa využíva metóda, pri ktoré 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 ma za následok rýchlejšie reakčné doby a menej pohybov hráča.



Obrázok 2.1-10 – Stavový diagram stratégie [5]

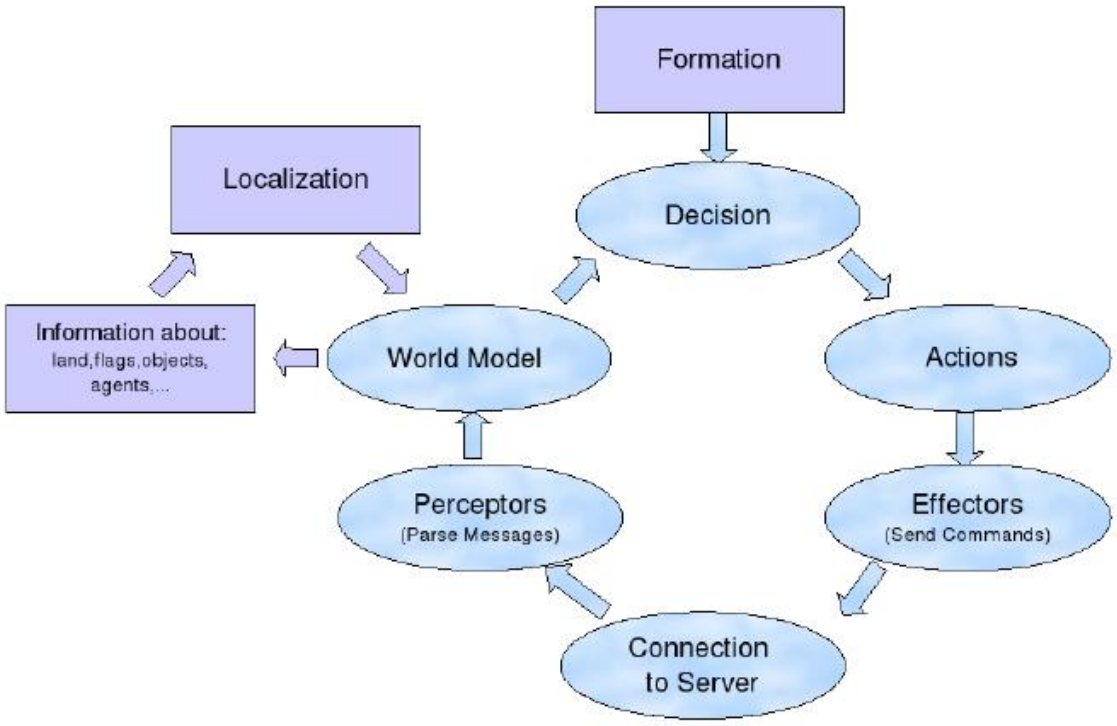
Alzahra

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 Azahra 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 počiatkoch sa tím Alzahra zameral 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 2.1 - 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

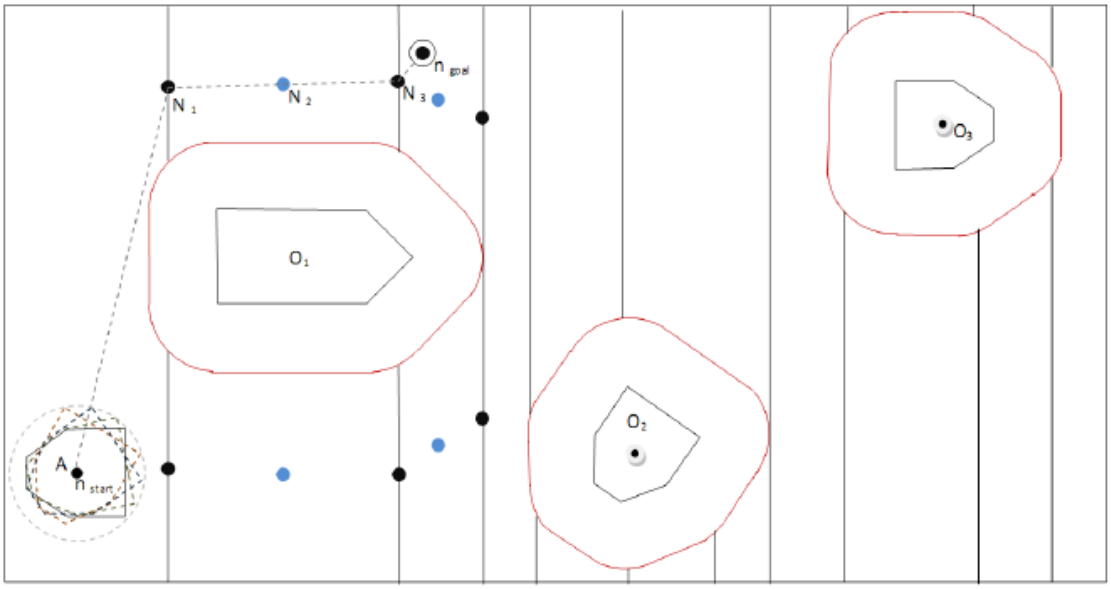
Za určovanie pozície je zodpovedná časť **Localization,** ktorá je jedna z najdôležitejších častí robota. Keďže v novšej verzií servera mal robot 120° zorné pole v každej dimenzií a zároveň boli zavedené určité obmedzenia k určovaniu polohy (robot nie vždy vidí tri body, podľa ktorých môže zistiť svoju 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 2.1 - 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 zabránili 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 2.1 - 12 sú jednotlivé body pohybu znázornené vrcholmi N_i.



Obrázok 2.1-12 – Ukážka nájdenia cesty [6]

Riadenie pohybu

Tím Alzahra 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:

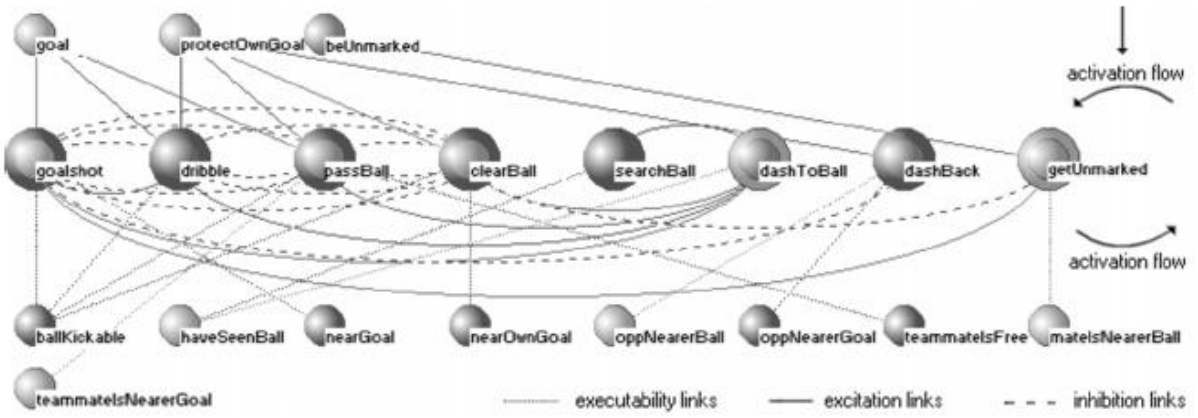
1. Metódy založené na trajektórií pohybu
2. Mentálne metódy (bližšie ich nešpecifikujú)
3. Metódy založené na CPG (založené na neurónových sieťach)

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.

Rozdhodovanie 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 reprezentácií cieľov s dynamickými funkciami (teda takými, ktoré závisia od situácie). Ciele, ktoré sa agenty 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 situácií (Obrázok 2.1 - 13).



Obrázok 2.1-13 – Rozšírená sieť správania [8]

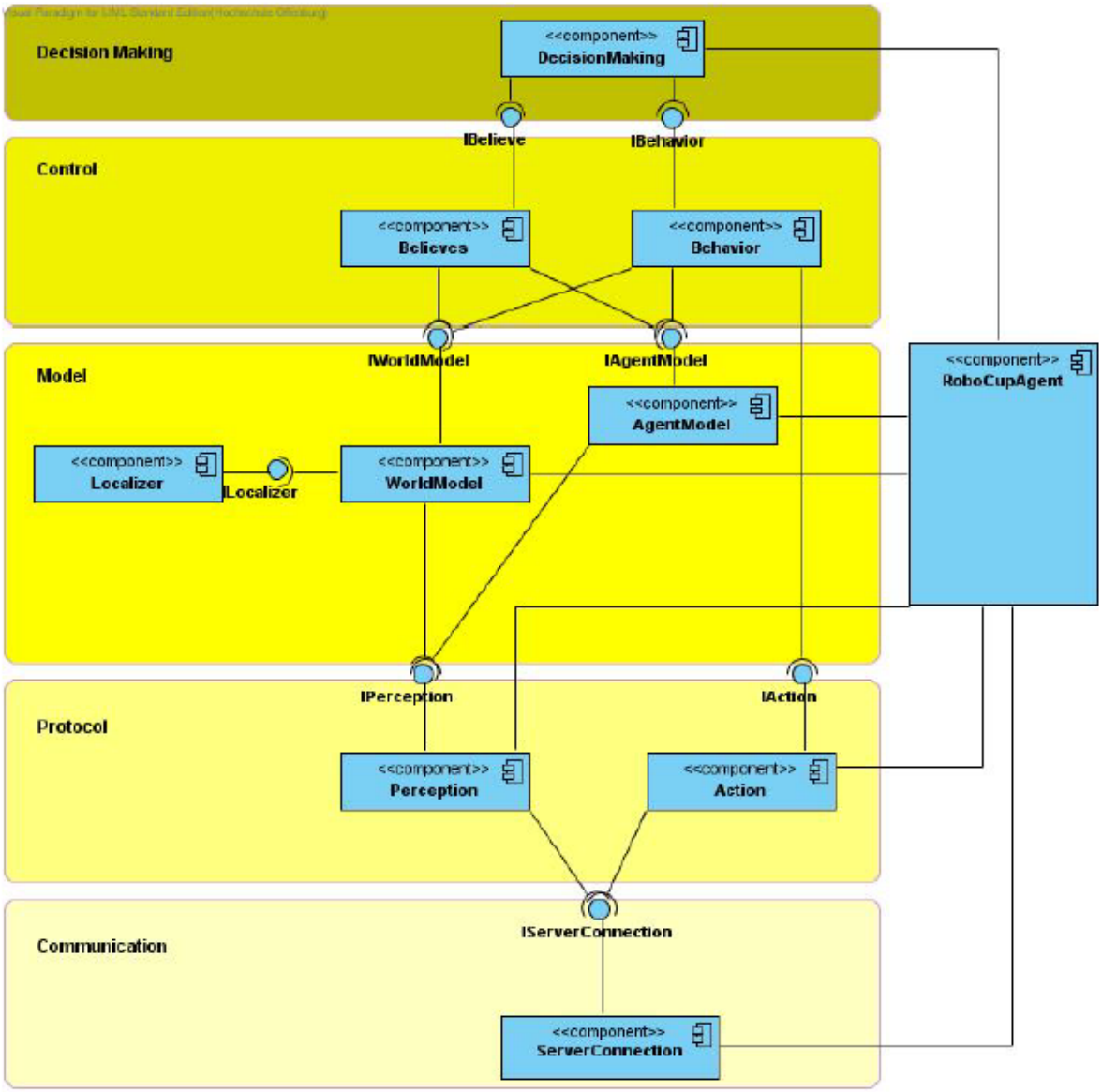
Implementácia

Z pohľadu implementácie robí tento tím priekopnícku prácu, keďže sa zameral na programovací jazyk Java, pretože väčšina iných hráčov je implementovaná v jazuku C++. Implementácia zahŕňa aj JUnit testy, ktoré zabezpečujú dynamické testovanie hráča.

Architektúra

Tento tím uverejnil svoj kód, ktorý bol použitý v súťaži RoboCup 2010. Hráč je založený na vrstvovej architektúre, ktorá je znázornená na Obrázku 2.1 - 14. Momentálne sa ich architektúra 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 tím a začať stavať na niektorej z vrstiev.

Tok informácií z jednej vrstvy do druhej je realizované 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 2.1 - 14), to znamená, že ľubovoľný komponent, môže byť nahradený 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 sieti 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.

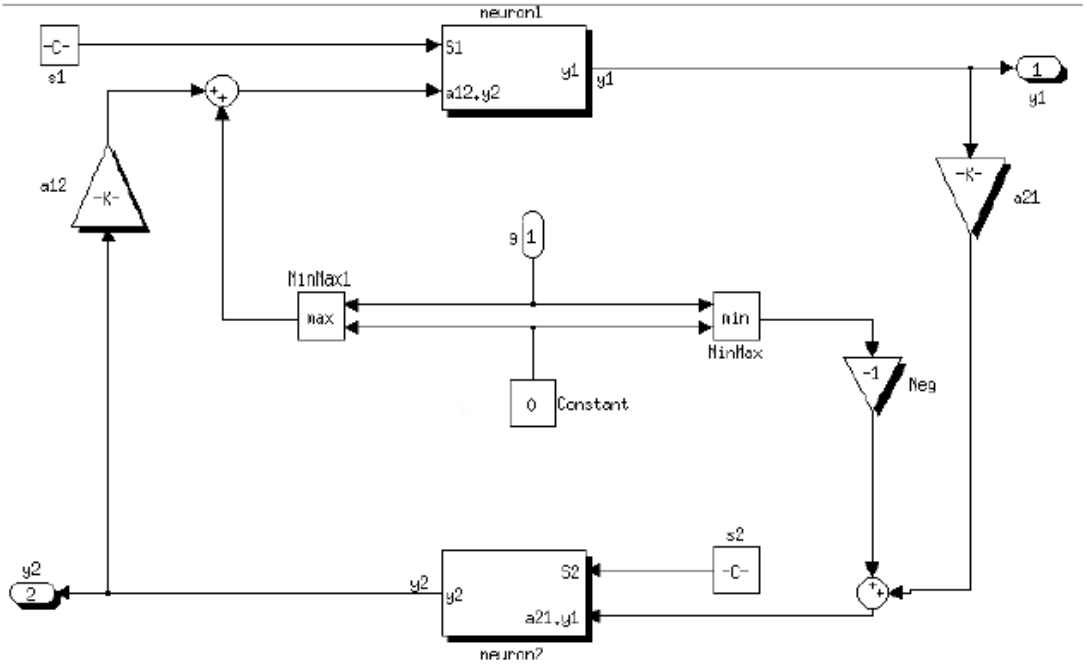
Riešenie zložitých problémov

Keďže pohyb robota je náročná vec, vo svojom riešení tento tím vyberá množstvo metód, ktoré sa snažia prispôsobiť 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 Optimalization (CLPSO) – nastavovanie parametrov systému, ktoré by sa inak muselo vykonávať experimentálne
- Reinforcement Learining – pre akcie v súvislostí s kopmi, ktoré musia rátať so zmenou v prostredí

Generovanie akcií

Tím pristúpil k riešeniu návrhu hráča podobne, ako ku simulácií 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 centrálneho generátora vzorov tzv. Matsuokové neurónový model oscilátora. Tento model sa skladá z dvoch vzájomne inhibičných neurónov tzv. exor a extensor neurónov (Obrázok 2.1 - 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 Robokopy používateľského manuálu k prostrediu SimSpark.

SimSpark architektúra

Server SimSpark je vybudovaný použitím aplikačného rámca Zeitgeist. Ten poskytuje základnú funkcionalitu ako 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 udržuje 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 knižnice 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.

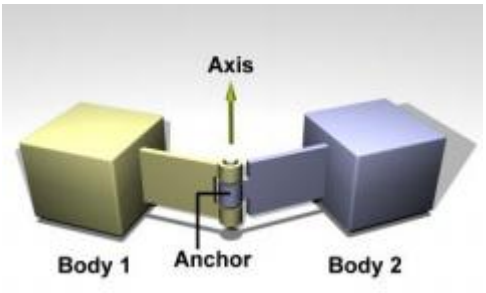
Základné perceptory

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 polohy 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))*

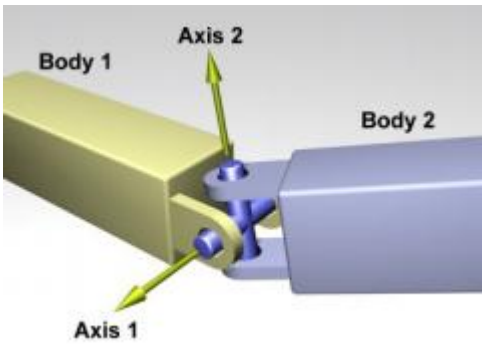
2. HingeJoint – udáva hodnotu veľkosti uhla, o ktorý sa ohne daný kĺb agenta. Obrázok 2.1 – 16 zobrazuje ohnutý kĺb s označenou osou.



Obrázok 2.1-16 – HingeJoint kĺb [1]

   Formát správy : *(HJ (n <názov_kĺbu>) (ax <veľkosť_uhla>))*
   Príklad: *(GYR (n 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 2.1 - 17.



Obrázok 2.1-17 – UniversalJoint kĺb [1]

Formát správy : *(UJ(n <name>)(ax1 <ax1>)(ax2 <ax2>))*
Príklad: *(UJ(n laj1_2)(ax1 -1.32)(ax2 2.00))*

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

Formát správy : *(TCH n <name> val 0|1)*
Príklad: *(TCH n bumper val 1)*

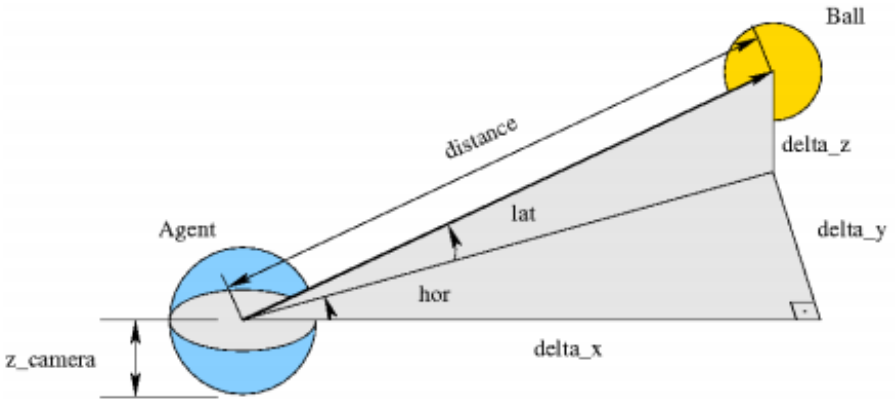
5. ForceResistant – slúži na detekciu pôsobenia sily na určitú časť tela agenta. V súčasnosti je možné jeho využitie pre ľavé (lf) a pravé (rf) chodidlo hráča. Definuje ho bod, na ktorý sila pôsobí a vektor sily.

Formát správy : *(FRP(n <name>)(c <px> <py> <pz>)(f <fx> <fy> <fz>))*
Príklad: *(FRP(n lf)(c -0.14 0.08 -0.05)(f 1.12 -0.26 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 2.1 - 18). S každým zachytením správy prichádza takisto informácia o:
   - vzdialenosti medzi agentom a objektom
   - uhle medzi agentom a objektom v horizontálnej rovine
   - šírkovom uhol



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 + uhlová odchýlka (x - y): sigma = 0.1225 + 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 efektory

V nasledujúcej časti opíšeme efektory, 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

V tejto kapitole sa nachádza návrh riešenia úloh v prvom šprinte.

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 je 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 prideľovanie 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 ostaných. Samotná komunikáciami 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 dokumentácií 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

V tejto kapitole sa nachádza opis implementácie úloh v prvom šprinte.

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 kompletné vývojové prostredie MS Visual Studio 2010, ktoré daný problém taktiež nevyriešilo. Problém bol vyriešení 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 zahraničné 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 inštalácií 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

V tejto kapitole sa nachádza opis testovania implementovaných úloh v prvom šprinte.

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

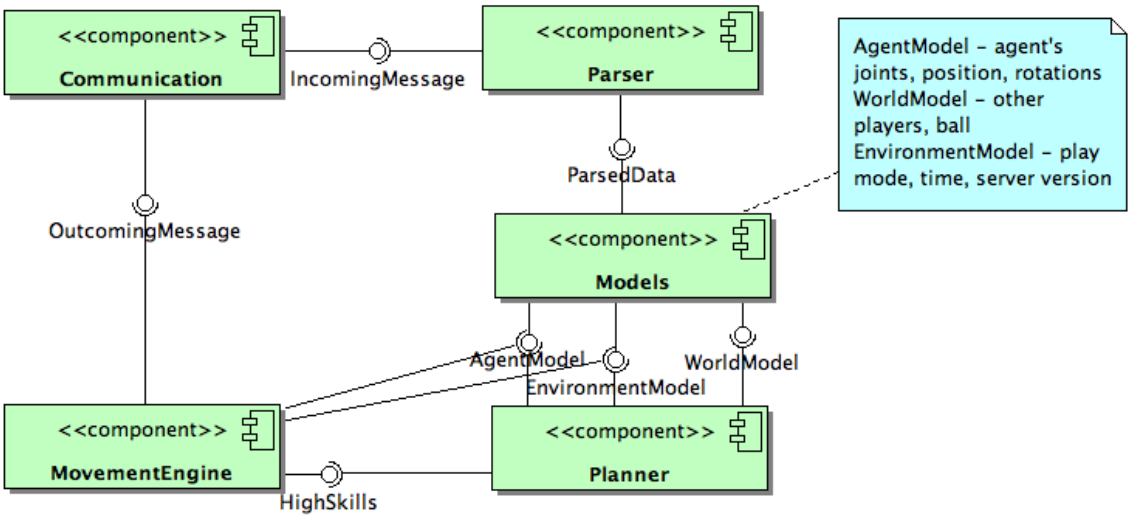
Vytvoriť základ pre dokumentáciu riadenia projektu
Zriadenie repozitára pre manažment verzií
Nainštalovanie klienta manažmentu verzií a vykonanie skúšobného commitu
Oboznámenie sa so zdrojovými kódmi vybraného hráča, jeho kompilácia a spustenie
Analýza kódu hráča - časť zodpovedná za prijímanie správ - 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

V tejto kapitole sa nachádza analýza úloh riešených v druhom šprinte.

2.2.1.1 Analýza prevzatých kódov hráča JIM

Táto úloha má za cieľ oboznámiť sa so zdrojovými kódmi hráča, odhalenie 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.



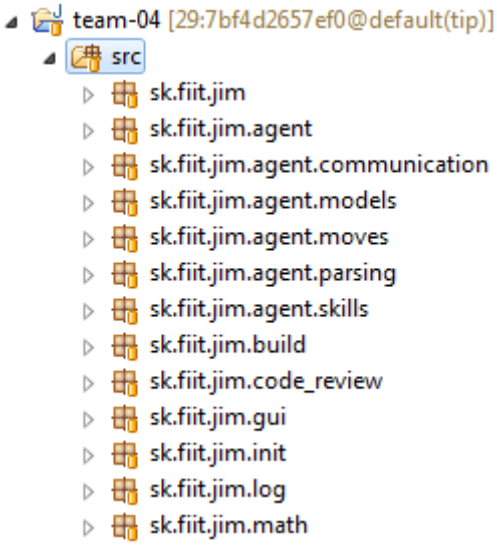
Obrázok 2.2-1 – Komponenty hráča JIM

Hráč JIM sa skladá z piatich komponentov, ktoré sú navzájom poprepájané rozhraniami. Dodržiava sa tak pravidlo slabej 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.

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.

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 čí 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 záväznosti 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 predpoveď 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 Agenta (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

```
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          assertThat(one.add(zero), is(equalTo(one)));
24          assertThat(one.add(cartesian(1, 0, 0)), is(equalTo(cartesian(2, 1, 1))));
25
26          assertThat(one.multiply(5.0), is(cartesian(5, 5, 5)));
27          assertThat(one.multiply(0.0), is(cartesian(0, 0, 0)));
28
29          assertThat(one.divide(0.2), is(equalTo(cartesian(5, 5, 5))));
30          assertThat(one.divide(0.2), is(equalTo(cartesian(5, 5, 5))));
31
32          assertThat(one.subtract(cartesian(0, 1, 0)), is(equalTo(cartesian(1, 0, 1))));
33
34          assertThat(one.negate(), is(equalTo(cartesian(-1, -1, -1))));
35          assertThat(zero.negate(), is(equalTo(cartesian(0, 0, 0))));
36
37          assertThat(one.addX(1.0), is(equalTo(cartesian(2, 1, 1))));
38          assertThat(one.addY(-1.0), is(equalTo(cartesian(1, 0, 1))));
39          assertThat(one.addZ(0.0), is(one));
40
41  //      original vector should remain the same - add, multiply, divide,
42  //      subtract, negate and other operations are immutable
43          assertThat(one, is(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

V tejto kapitole uvádzame návrh úloh realizovaných v druhom šprinte.

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 časti dokumentácie, zápisnice, metodiky, manažment projektu a preberacie protokoly. Formálna úprava bude riešená podľa štábnej kultúry opísanej v dokumentácií k riadeniu.

2.2.2.5 Aktualizácia webovej stránky

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

2.2.3 Implementácia

V tejto kapitole uvádzame opis implementácie úloh v druhom šprinte.

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
LowSkillTest	Pridanie metódy na testovanie finálneho stavu
LowSkill	Úprava zdrojového kódu pre zvýšenie čitateľnosti
LowSkills	Úprava zdrojového kódu pre zvýšenie čitateľnosti
Communication	Úprava zdrojového kódu

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 ošetrujúce 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 ošetrujúce podmienky, inak Ok
DynamicObject	Ok
EnviromentalModel	Dopísané ošetrujúce 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 triedy neexistoval, bolo treba vytvoriť nový test.

Časť pre odosielanie správ

Trieda	Testovacia trieda	Poznámky
LowSkill	LowSkillTest	Pridanie metódy na testovanie finálneho stavu
Parser	ParserTest	Ok
EffectorData	EffectorDataTest	Pridaná nová trieda
Communication	CommunicationTest	Pridaná nová trieda

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

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

V tejto kapitole opisujeme testovanie implementovaných úloh v druhom šprinte.

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

Skompilovateľnosť 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 centrálneho 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.

2.3 Šprint č. 03

Číslo šprintu	03
Počet príbehov	10
Začiatok šprintu	04.11.2010
Koniec šprintu	18.11.2010

Príbehy

Generovanie XML pre hráča JIM z editora pohybov
Analýza plánovača pohybov
Akcelerometer
Pohyby hráča – úprava a vylepšenie vykonávania základných pohybov
Návrh modulov a vyššieho správania
Dopísanie testov tried
Pridanie user stories a akceptačných testov k úlohám – “how to“ pre ich písanie
Guideline ako písať úlohy do JTracku
Úprava a aktualizácia web stránky
Dokumentácia, user stories, plánovanie

2.3.1 Analýza

V tejto kapitole uvádzame analýzu úloh v treťom šprinte.

2.3.1.1 Generovanie XML pre hráča JIM z editora pohybov

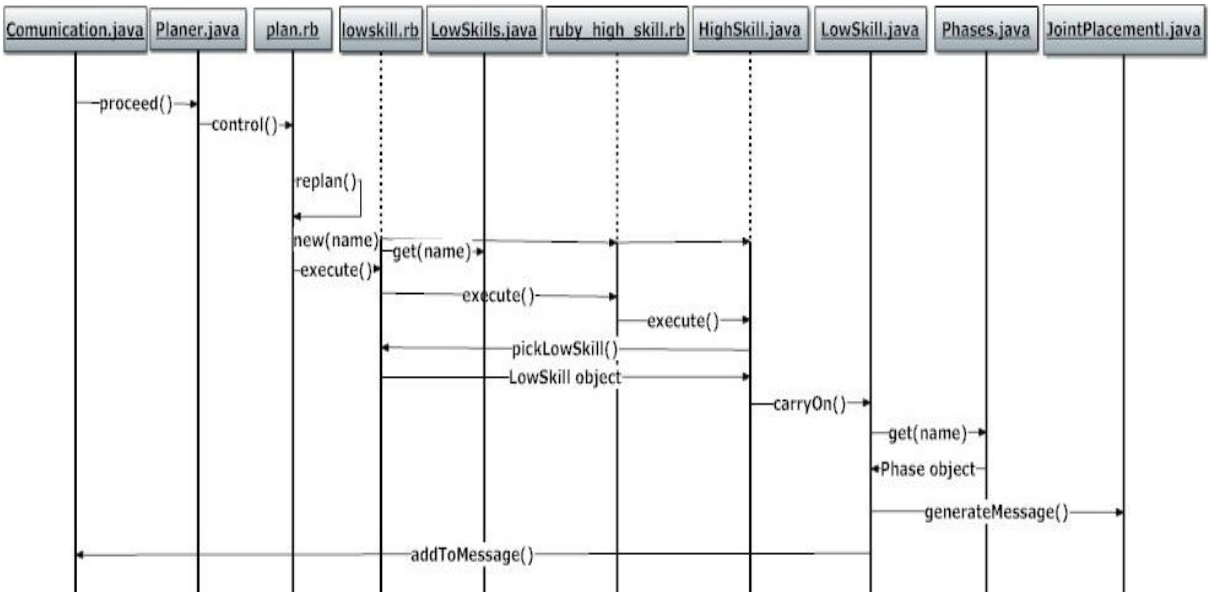
Cieľom tejto úlohy je exportovanie súboru typu XML z editora pohybov pre hráča JIM tímu The A Team. Vzhľadom na to, že hráč JIM disponuje vlastnou schémou XML, je potrebná práva nie úprava používateľského rozhrania editora pohybov, ale aj vytvorenie nových metód na exportovanie súboru typu XML. Problémom je hlavne namapovanie pohybov v editore pohybov do elementov a atribútov v súbore typu XML pre hráča JIM.

V súčasnej verzií editora pohybov je ukladanie súboru s pohybmi resp. exportovanie súboru typu XML s pohybmi hráča zabezpečované triedami *Commands*, *MotionEditorForm* a *XmlExporter*. V triede *Commands* je pomocou metódy *ExportXmlMessageFile* riešené vyvolanie dialógového okna pre uloženie resp. exportovanie súboru. Táto metóda volá metódu *ExportMovesToXml* triedy *MotionEditorForm*, ktorá zabezpečuje zoradenie všetkých pohybov podľa začiatočného času vykonávania a volanie metódy *exportToXMLFile* triedy *XmlExporter*, ktorej argumentom konštruktora sú zoradené pohyby. Metóda

exportToXMLFile zabezpečuje rozdelenie pohybov do viacerých fáz a vytvorenie súboru typu XML na exportovanie.

2.3.1.2 Analýza plánovača pohybov

Cieľom bolo overiť v akom stave sa plánovač nachádza a čo bude potrebne zmeniť na dosiahnutie nami želaného efektu. Momentálne je hráč v stave, keď vykonáva jeden pohyb dookola. Tento cyklus však nezabezpečuje plánovač, ale definícia pohybu, ktorého fázy sú v cykle. Plánovač momentálne spúšťa len jeden pohyb, ktorého meno sa nachádza v skripte. V plánovači je implementovaná kontrola skončenia pohybu aj keď iba na základe sledovania ubehnutého času. Vďaka tomu je teda možné spúšťať niekoľko pohybov po sebe. Diagram zobrazený na Obrázku 2.3 - 1. približne opisuje volanie metód a prepojenie objektov v plánovači.



Obrázok 2.3-1 – Sekvenčný diagram plánovača

Trieda, ktorá momentálne inicializuje celý proces vykonania pohybu je *Comunication.java*. Táto vo svojej metóde *mainLoop*, ktorá beží v hráčovi v nekonečnej slučke, volá metódu proceed, ktorá spúšťa *plan* skript. V ňom sa vytvára objekt triedy *LowSkill.rb*, ktorý v sebe nesie objekt triedy *LowSkill.java* obsahujúci pohyb, ktorý sa bude vykonávať. Následne sa nad objektom *LowSkill.rb* volá metóda *execute*. Táto je nadefinovaná v *HighSkill.java*, od ktorej ju *LowSkill.rb* zdedil. V tejto metóde sa kontroluje existencia aktuálneho pohybu a nastavuje a spúšťa sa nasledujúci pohyb po jeho ukončení. Po overení existencie sa pohyb nastaví, aby začal od prvej fázy a spustí sa. Kontrola správneho behu fáz je zabezpečená v objekte triedy LowSkill.java. O neustále volanie týchto metód sa stará metóda *mainLoop* v triede *Comunication.java*.

2.3.1.3 Akcelerometer

Pri analýze fungovania akcelerometra hráča sme natrafili na chybu, kedy hráč správne nespracovával správy držiace informáciu stavu akcelerometra, nakoľko ich prijímal v nesprávnom tvare:

```
"^ACC \\(n torso\\) \\(rt ([-.0-9]+) ([-.0-9]+) ([-.0-9]+)"
```

Nakoľko pôvodný hráč týmto vzorom správy filtruje všetky prijaté správy akcelerometra, nie je možné ich ďalej spracovávať. Je potrebné implementovať metódy, pomocou ktorých bude hráč schopný určiť svoju aktuálnu polohu. Pre zistenie stavu agenta bude postačovať implementácia nasledovných funkcií:

- `boolean isStanding();` - zistí, či agent stojí
- `public boolean isOnGround();` - zistí, či agent spadol (leží na zemi)
- `boolean isLyingOnBack();` - zistí, či leží na chrbte
- `boolean isLyingOnBelly();` - zistí, či leží na bruchu

2.3.1.4 Pohyby hráča – úprava a vylepšenie vykonávania základných pohybov.

Po analýzach vykonaných v predchádzajúcich šprintoch sme zistili, že hráč Jim má implementované len štyri pohyby a tie sú boli v takomto stave:

- Chôdza – pohyb nebol dokončený a hráč momentálne iba krokuje na mieste
- Otočenie vpravo a vľavo – tieto pohyby fungovali bezchybne
- Postavenie sa z chrbta – pohyb bol začatý, ale nedokončený hráč sa nikdy nepostavil

2.3.1.5 Návrh modulov a vyššieho správania

Autor hráča JIM sa pri jeho implementácií zameral najmä na nižšie časti logiky. Hráč sa tak skladá z kódu, ktorý je potrebný na správne načítanie vykonávanie základných pohybov zo súboru, parsovanie správ zo servera a ich zasielanie späť serveru. Implementovaný je rovnako jednoduchý model agenta, sveta a objektov a začiatok ich predikcie. Agent si zaznamenáva informácie z prostredia, ktoré ale v súčasnosti nevyužíva rozhodovanie, prípadne vylepšenie pohybu. V agentovi je tiež implementovaný výpočet niektorých hodnôt o tele agenta a okolitom svete, ktoré budú slúžiť ako základ pre rozhodovacie funkcie. Agent nemá implementovaný akcelerometer.

Autor vytvoril len niekoľko základných pohybov, bez ktorých by nebolo možné skladať nové pohyby. Prevzaté kódy hráča teda neobsahuje žiadnu implementovanú strednú alebo vyššiu logiku. Základná architektúra bola navrhnutá tak, aby bolo na nej možné stavať a rozšíriť ju (bližšie informácie o architektúre prevzatého hráča sa nachádzajú v časti 2.2.1.1 Analýza prevzatých kódov hráča JIM).

2.3.1.6 Guideline ako písať úlohy do JTracku

Po nainštalovaní, nastavení a sprevádzkovaní nástroja na sledovanie úloh bolo potrebné tímovým kolegom vytvoriť prezentáciu daného nástroja, jeho možností a spôsobu používania. Pre tieto potreby bola vypracovaná metodika, ktorá vysvetľuje prácu a možnosti práce s týmto nástrojom. Jej kompletné znenie sa nachádza na:
http://www.tonyb.sk/_my/jtrack/metodika10-01-04-xbaluchaa.doc

2.3.2 Návrh

V tejto kapitole uvádzame návrh riešenia úloh v treťom šprinte.

2.3.2.1 Generovanie XML pre hráča JIM z editora pohybov

Za účelom exportovania súboru typy XML pre hráča typu JIM je potrebná aj samotná úprava používateľského rozhrania editora pohybov. Pre rozšírenie o ďalší súbor typu XML je najvýhodnejšie rozšírenie filtra dialógového okna uloženia resp. exportovania súboru. To riešenie je najvýhodnejšie aj z prípadného rozširovania editora pohybov o exportovanie ďalších súborov typu XML. Ďalším dôvodom je koncentrácia funkcionality exportovania do jedného miesta editora pohybov.

Pre rozšírenie editora pohybov je potrebná aj úprava triedy *MotionEditorForm*, ktorá musí byť pri zvolení jednotlivých typov súborov typu XML zavolať správnu metódu triedy *XmlExporter*. Trieda *XmlExporter* musí byť rozšírená o ďalšiu metódu *exportToXMLFileForJim*, ktorá zabezpečí vytvorenie súboru typu XML so štruktúrou súboru typu XML hráča Jim. Pri vytváraní súboru sa využije existujúce rozdelenie pohybov do jednotlivých fáz s upraveným časovaním pre jednotlivé fázy. Z dôvodu situácie načítavania viacerých pohybov naraz hráčom Jim, je potrebné jednoznačné odlíšenie názvov fáz aj medzi jednotlivými pohybmi.

Pre zdrojové kódy editora bolo vytvorené úložisko. Úložisko sa nachádza na adrese http://bitbucket.org/marosss/robotbehavioreditor.

2.3.2.2 Analýza plánovača pohybov

Ako ukazuje analýza, plánovač pohybov využíva pri svojej práci RUBY skripty. Keďže ide o plánovač najnižšej vrstvy logiky a to je spájanie jednoduchých pohybov ako je krok či otočenie toto prepojenie sme sa rozhodli v najbližšej budúcnosti odstrániť. Skripty budú využívané iba vyššou logikou. Z tohto dôvodu bola v rámci tejto úlohy navrhnutá jednoduchá metóda ako overiť spúšťanie pohybov priamo z triedy *Plane.java*.

2.3.2.3 Pohyby hráča – úprava a vylepšenie vykonávania základných pohybov.

Pri návrhu pohybov sme sa inšpirovali tímami z predchádzajúcich rokov. Konkrétne tímom Agenty 007. Pohyby sme zadefinovali do XML súborov.

2.3.2.4 Návrh modulov a vyššieho správania

Návrh architektúry hráča je uvedený na diagrame zobrazenom na Obrázku 2.3 - 2. Popis jednotlivých komponentov je nasledovný:

Komunikácia

Komponent pre komunikáciu zabezpečuje prijímanie správ zo servera a ich správne parsovanie. Rovnako zabezpečuje odosielanie správ serveru smerom od hráča. Neustála komunikácia so serverom zabezpečuje, že hráč má vždy aktuálne informácie o okolitom svete – polohe a pozícií jednotlivých objektoch (iní hráči, lopta), stave hry a podobne.

Správanie

Základný komponent reprezentujúci správanie hráča. Charakterizuje individuálne správanie hráča, ktoré sa využíva v tímovom správaní (hráči pri hre spolupracujú). Správanie sa rozhoduje na základe bázy znalostí, ktorá obsahuje možných akcie. Akcie, ktoré sa vykonajú sú naplánované a vložené do plánovača, ktorého úlohou je správne usporiadanie akcií, zmena ich poradia, zrušenie niektorej akcie, podľa pokynov z individuálneho a tímového správania hráča. Vstupmi pre rozhodovanie sú informácie o svete z komponentu Model.

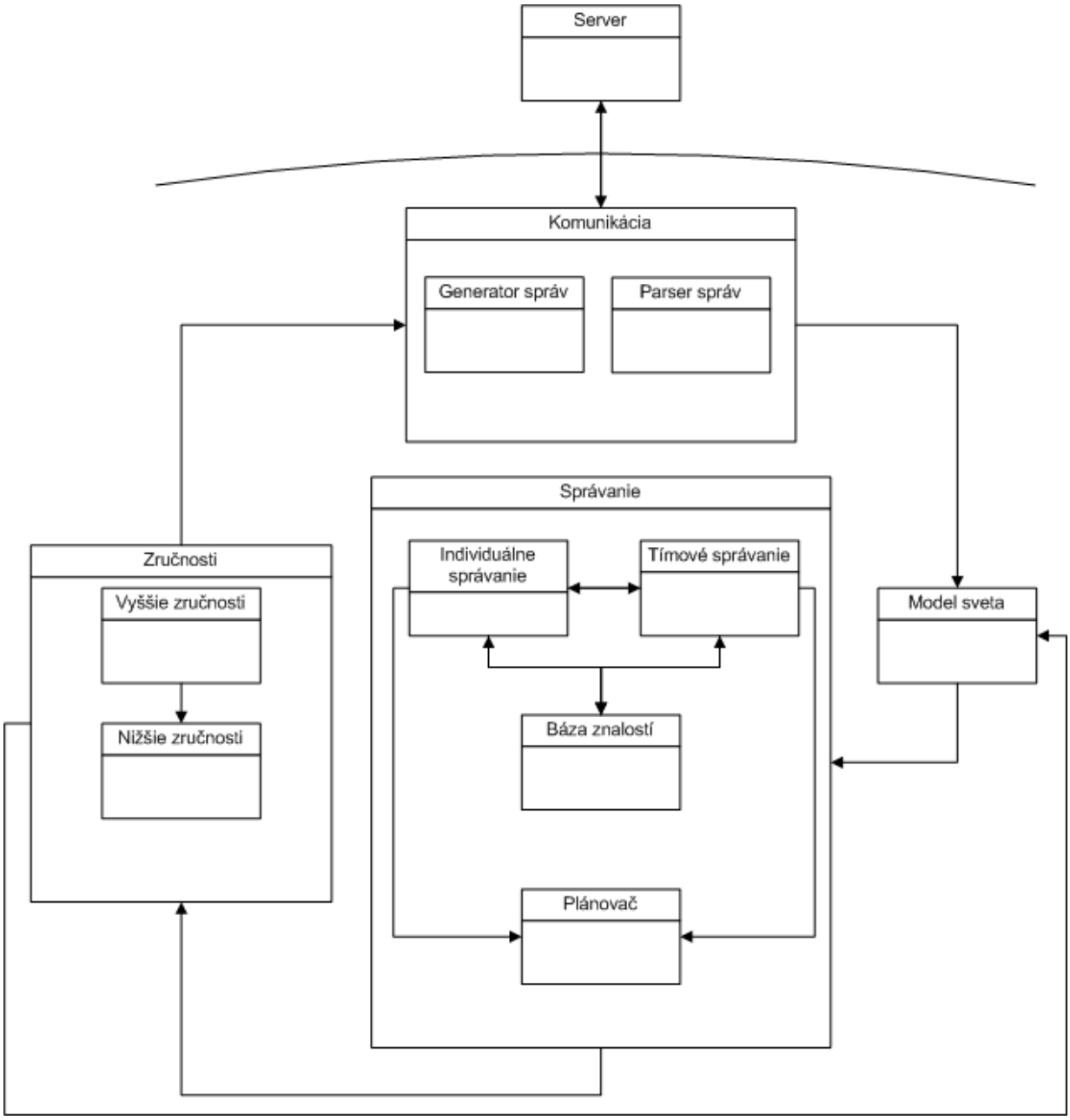
Zručnosti

Zručnosti predstavujú akcie, ktoré dokáže hráč vykonávať. Rozdeľujeme ich na nižšie – základné pohyby a vyššie zručnosti, ktoré sa skladajú z nižších a umožňujú vytvárať komplikované pohyby. Príkladom základného pohybu je jeden krok. Príkladom vyššej zručnosti je kopnutie do lopty s rozbehom. Ak hráč vykoná akciu, ovplyvní tak okolitý svet.

Model sveta

Model sveta je celkový model hráča a sveta, v ktorom sa hráč nachádza. Tento komponent zabezpečuje reprezentáciu modelu ihriska, objektov, ich predikciu, história sveta a všetky potrebne informácie o svete, ktoré hráč môže využiť. Je zdrojom informácií o interakciách s ostatnými hráčmi. Tento komponent zaobaľuje spomenuté modely do jednej časti – nie je monolitický.

Vzhľadom na to, že komponent pre komunikáciu a model sveta je už implementovaný, nižšie je uvedený konkrétny návrh komponentov pre správanie a zručnosti.



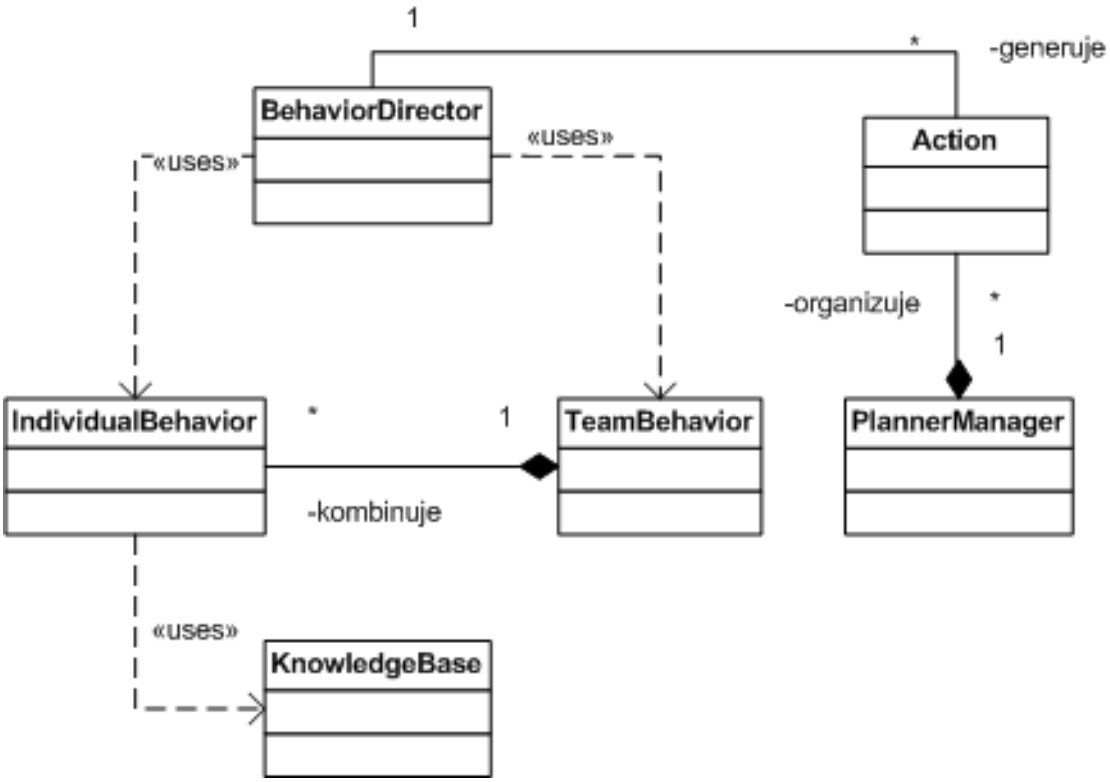
Obrázok 2.3-2 – Diagram architektúry hráča

Správanie

Pri návrhu správania vychádzame zo štyroch základných princípov [10]:

1. agent musí byť schopný dokončiť svoju úlohu,
2. agent sa musí vedieť rozhodovať na základe aktuálnej situácie,
3. rôzne typy správania musia byť rozdelené do skupín tak, aby si navzájom neprekážali,
4. niektoré typy správania sú použiteľné iba ak sú súčasťou skupiny, alebo sú použité až po vykonaní iných typov.

Preto sme sa navrhli architektúru znázornenú na diagrame zobrazenom na Obrázku 2.3 - 3.

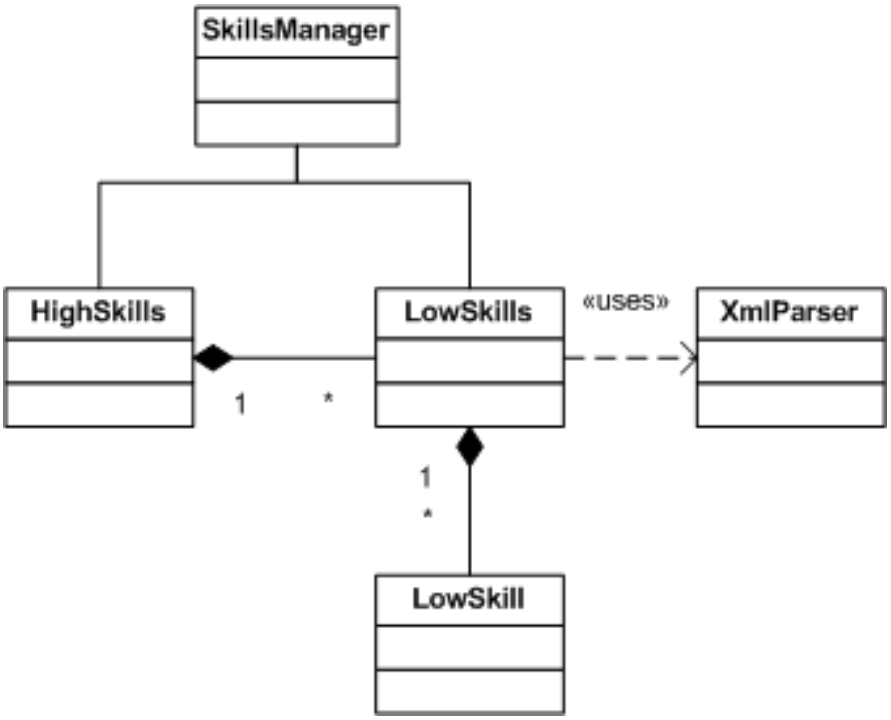


Obrázok 2.3-3 - Diagram tried komponentu pre správanie

Hlavnou triedou návrhu komponentu pre správanie je *BehaviorDirector*. Táto trieda riadi hráča, vyberá akciu, ktorá sa bude vykonávať. K svojmu rozhodnutiu využíva triedy *IndividualBehavior* a *TeamBehavior*. Akcia, ktorú vygeneruje *BehaviorDirector* je vložená do plánovača *PlannerManager*, ktorý sa stará o správne usporiadanie akcií, zmenu ich poradia, či zrušenie niektorej akcie na základe rozhodnutí, ktoré vykoná *BehaviorDirector*. *BehaviorDirector* sa rozhoduje dynamicky, na základe aktuálnych informácií z modelu sveta.

Zručnosti

Návrh zručností je znázornení na diagrame zobrazenom na Obrázku 2.3 - 4. Zručnosti sa rozdeľujú na nižšie – trieda *LowSkills* a vyššie – trieda *HighSkills*. Ešte menšiu granularitu predstavuje trieda *LowSkill*, ktorej typickým objektom môže byť otočenie jedného kĺbu. Vyššie zručnosti sa skladajú z nižších a umožňujú vytvárať komplikované pohyby. Hlavnou triedou je *SkillsManager*, ktorá slúži na získavanie zručností a na ktorú smerujú všetky požiadavky zo správania. Poskytuje informácie o jednotlivých zručnostiach, koordinuje a vyberá z nich tie, ktoré sú potrebné.



Obrázok 2.3-4 - Diagram tried komponentu zručností

2.3.3 Implementácia

V tejto kapitole uvádzame opis implementácie úloh v treťom šprinte.

2.3.3.1 Generovanie XML pre hráča JIM z editora pohybov

Dialogové okno uloženia resp. exportovania súboru bolo rožírené o nový filter,

dlg.Filter = "Xml files for Jim (.xml)|*.xml|Xml files (*.xml)|*.xml";*

ktorý umožňuje pri exportovaní zvolenie súboru typu XML pre hráča Critical Error a pre hráča Jim.

Pre zavolanie správnej metódy pri zvolení jednotlivých typov súborov typu XML bola trieda *MotionEditorForm* rozšírená o konštanty *XML_CRITICAL_ERROR_PLAYER* a *XML_JIM_PLAYER*. Tieto konštanty slúžia ako hodnoty nového argumentu metódy *ExportMovesToXml* triedy *MotionEditorForm*, na základe ktorého je zavolaná správna metóda.

V novej metóde *exportToXMLFileForJim* triedy *XmlExporter* bola využitá časť metódy *exportToXMLFile*, ktorá zabezpečuje rozdelenie pohybov do jednotlivých fáz. Časovanie fázy bolo upravené a v súbore typu XML je reprezentované hodnotou elementu duration. Hodnota je vypočítaná ako *minEndTime – minStartTime*, pričom *minEndTime* a *minStartTime* je čas konca resp. začiatku fázy získaný po rozdelení pohybov fázy z editora pohybov.
Jednoznačné odlíšenie názvov fáz aj medzi jednotlivými pohybmi je riešené skladaným názvu fázy z názvu *lowSkill*, čiže názvu súboru typu XML a poradového čísla fázy.

[lowSkillName]_phase[numberPhase]

Napr. *walk_phase1*

2.3.3.2 Pohyby hráča – úprava a vylepšenie vykonávania základných pohybov.

Pri písaní XML súborov pre Jim-a nastal problém, keďže pracuje na odlišnom princípe ako hráč tímu Agenty 007 a tento princíp neumožňuje súčasne bežať dvom fázam pohybu. S tým bola spojená potreba vykonania niekoľkých úprav pôvodného pohybu, aby sa dosiahli nami požadovaný efekt.

2.3.3.3 Analýza plánovača pohybov

Implementácia spočívala z napísania analýzy do dokumentácie, preskúšaní zmeny pohybov prepisovaním RUBY skriptu ovládajúceho spúšťaný pohyb a vyskúšaní spúšťania pohybov bez využitia skriptov upravením tried *Planer.java* a *FakeHighSkill.java*.

2.3.3.4 Akcelerometer

Vzor správ akcelerometra bol prispôsobený prijímaným správam. Agent teraz prijíma správy správne vo formáte:

```
"^ACC \\(n torso\\) \\(a ([-.0-9]+) ([-.0-9]+) ([-.0-9]+)"
```

Boli pridané dve testovacie správy a dopísané JUnit testy kontrolujúce správnu funkcionalitu spracovávania správ akcelerometra.

Taktiež boli implementované resp. upravené funkcie, slúžiace na detekciu polohy agenta isStanding(), isOnGround(), isLyingOnBelly(), isLyingOnBack(). Pôvodný hráč používal na detekciu polohy hráča informácie z gyro perceptora - ako je trup agenta naklonený vo vzťahu k súradnicovej sústave. V našom hráčovi bola implementovaná táto detekcia na základe informácií z akcelerometra. V metódach zisťovania stavu agenta je kontrolované pôsobenie gravitačnej sily na jednotlivé osy hráča. Ak hodnota presiahne určitú hraničnú hodnotu, stav sa zmení. Boli implementované nasledujúce metódy:

```
public boolean isStanding()
{
	return accelerometer.getZ() > 7.0;
}

public boolean isOnGround()
{
	return Math.abs(accelerometer.getY()) > 9.3;
}

public boolean isLyingOnBack()
{
	return isOnGround() && ( accelerometer.getY() > 9.3 );
}

public boolean isLyingOnBelly()
{
	return isOnGround() && !isLyingOnBack();
}
```

2.3.3.5 Dokumentácia, user stories, plánovanie

Dokumentácia bola vytvorená integrovaním dokumentácií od jednotlivých členov tímu. Bola vytvorená podľa návrhu softvérom Microsoft Word. Export do formátu pdf bol realizovaný pomocou softvéru PDF Creator.

2.3.4 Testovanie

V tejto kapitole uvádzame testovanie implementovaných úloh v treťom šprinte.

2.3.4.1 Generovanie XML pre hráča JIM z editora pohybov

Testovanie prebiehalo exportovaním pohybov definovaných v knižnici editora pohybov a vytvorených pohybov, a ich porovnávaním s pohybmi hráča Jim spúšťaného priamo z vývojového prostredia. Pri testovaní už počas vývoja sa odhalila potreba jednoznačného odlíšenia názvov fáz aj medzi jednotlivými pohybmi. Po vyriešení tohto problému boli pohyby vykonané hráčom Jim totožné s pohybmi definovanými v knižnici editora pohybov a vytvorenými pohybmi.

2.3.4.2 Pohyby hráča – úprava a vylepšenie vykonávania základných pohybov.

Každý novovytvorený pohyb bol niekoľkokrát spustený a ponechaný bežať 2 minúty, aby sa overilo jeho fungovanie a stabilita. Nasledujúce tabuľky ukazujú výsledky testov.

Chôdza

Číslo testu	Výsledok
1	Hráč kráčal celý čas dopredu a nedošlo k pádu
2	Hráč kráčal celý čas dopredu a nedošlo k pádu
3	Hráč kráčal celý čas dopredu a nedošlo k pádu
4	Hráč kráčal celý čas dopredu a nedošlo k pádu
5	Hráč kráčal celý čas dopredu a nedošlo k pádu

Otočenie vpravo

Číslo testu	Výsledok
1	Hráč sa otáčal správnym smerom a nespadol.
2	Hráč sa otáčal správnym smerom a nespadol.
3	Hráč sa otáčal správnym smerom a nespadol.
4	Hráč sa otáčal správnym smerom a nespadol.
5	Hráč sa otáčal správnym smerom a nespadol.

Otočenie vľavo

Číslo testu	Výsledok
1	Hráč sa otáčal správnym smerom a nespadol.
2	Hráč sa otáčal správnym smerom a nespadol.
3	Hráč sa otáčal správnym smerom a nespadol.
4	Hráč sa otáčal správnym smerom a nespadol.
5	Hráč sa otáčal správnym smerom a nespadol.

Pád brankára vľavo

Číslo testu	Výsledok
1	Brankár spadol správnym smerom
2	Brankár spadol správnym smerom
3	Brankár spadol správnym smerom
4	Brankár spadol správnym smerom
5	Brankár spadol správnym smerom

Pád brankára vpravo

Číslo testu	Výsledok
1	Brankár spadol správnym smerom
2	Brankár spadol správnym smerom
3	Brankár spadol správnym smerom
4	Brankár spadol správnym smerom
5	Brankár spadol správnym smerom

Vstanie spredu:

Číslo testu	Výsledok
1	Hráč spadol smerom dopredu a postavil sa.
2	Hráč spadol smerom dopredu a postavil sa.
3	Hráč spadol smerom dopredu a postavil sa.
4	Hráč spadol smerom dopredu a postavil sa.
5	Hráč spadol smerom dopredu a postavil sa.

Vstanie z chrbta:

Číslo testu	Výsledok
1	Hráč spadol smerom dozadu a postavil sa.
2	Hráč spadol smerom dozadu a postavil sa.
3	Hráč spadol smerom dozadu a postavil sa.
4	Hráč spadol smerom dozadu a postavil sa.
5	Hráč spadol smerom dozadu a postavil sa.

2.3.4.3 Analýza plánovača pohybov

Keďže neboli implementované žiadne nové funkcie, testovanie v tejto úlohe spočívalo iba z vyskúšania možností aktuálneho plánovača, vyskúšania spúšťania pohybov bez využitia RUBY skriptu a overenia existencie analýzy v dokumentácii.

2.3.4.4 Dokumentácia, user stories, plánovanie

Dokumentácia bola opravovaná na štylistické a gramatické chyby členmi tímu.

2.4 Šprint č. 04

Číslo šprintu	04
Počet príbehov	11
Začiatok šprintu	18.11.2010
Koniec šprintu	02.12.2010

Príbehy

Editor pohybov, dorobenie importu Xml **User story:** Ako člen tímu viem importovať XML pre hráča JIM do editora pohybov.
Spájanie pohybov **User story:** Ako developer potrebujem aby hráč vedel spájať základné pohyby podľa potreby situácie, aby bolo možné vykonávať úlohy zadané vyššou logikou hráča.
Akcelerometer **User story:** Hráč vie vyhodnotiť v akej polohe sa nachádzajú jeho jednotlivé časti a celkový jeho stav (stojím, ležím).
Stabilná chôdza **User story:** Ako owner potrebujem, aby sa hráč vedel hýbať.
Rýchla chôdza **User story:** Ako owner potrebujem, aby sa hráč vedel hýbať.
Vstanie z chrbta **User story:** Ako owner potrebujem, aby sa hráč vedel hýbať.
Otáčanie vrpavo, vlavo **User story:** Ako owner potrebujem, aby sa hráč vedel hýbať.
Kopnutie dopredu **User story:** Ako owner potrebujem aby sa hráč vedel hýbať.

Ľahnutie na brucho, chrbát po páde brankára **User story:** Ako owner potrebujem aby sa hráč vedel hýbať.
Úprava a aktualizácia web stránky **User story:** Ako product owner potrebujem mať k dispozícii aktuálny prehľad stavu projektu, aby som mal dohľad nad jeho progresom.
Tvorba a publikovanie projekt backlogu **User story:** Ako developer potrebujem mať k dispozícii celkový a podrobný plán projektu.

2.4.1 Analýza

V tejto kapitole uvádzame analýzu úloh riešených v štvrtom šprinte.

2.4.1.1 Editor pohybov, dorobenie importu xml

Cieľom tejto úlohy je možnosť importovania súborov typu XML do editora pohybov pre hráča JIM tímu The A Team. Toto na jednej strane umožní, aby sme mohli editovať súbory, ktoré sme vytvorili a je ich potrebné znovu modifikovať. Na druhej strane v spolupráci s exportom XML umožňuje vygenerovať nové XML, ktoré je v editore možné veľmi pohodlne upraviť.

Vzhľadom na to, že hráč JIM disponuje vlastnou schémou XML, je potrebné vykonať úpravu v existujúcom importe súborov XML, ktoré nie je kompatibilné s XML pre hráča JIM. Problém, ktorý sa tu objavuje je najmä v chýbajúcom tagu pre štartovaciu pozíciu kĺbku, resp. pozíciu kĺbu, v ktorej pohyb začína.

V súčasnej verzii editora pohybov je importovanie súboru s pohybmi vo formáte XML zabezpečované triedami *Commands*, *MotionEditorForm* a *XmlImporter*. V triede *Commands* je pomocou metódy *ImportXmlMessageFile* riešené vyvolanie dialógového okna pre importovanie súboru. Táto metóda volá metódu *ImportMovesFromXml* triedy *Library*, ktorá zabezpečuje správne načítanie všetkých fáz, stavov a príslušných pohybov zo súboru XML. Pre samotné parsovanie XML sa používajú triedy *EffectorNode, PhaseNode* a trieda *XmlImporter.*

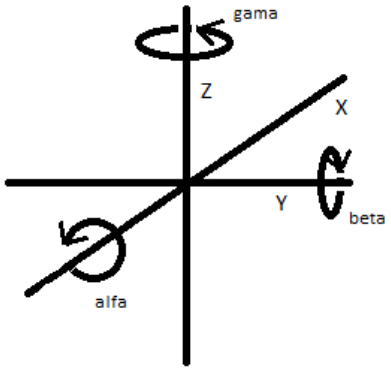
2.4.1.2 Spájanie pohybov

Analýza pre túto úlohu bola vykonaná ako samostatná úloha v predchádzajúcom šprinte.

2.4.1.3 Akcelerometer

Pôvodný hráč sa nesprávne pokúša o výpočet budúcej pozície agenta pomocou správ z akcelerometra, ktorých prijímanie na jednej strane fungovalo nesprávne a na strane druhej je tento výpočet zle implementovaný.

Dáta z akcelerometra uvádzajú aké zrýchlenia pôsobia na jednotlivé osy agenta. V tých je prirátané aj simulované gravitačné zrýchlenie. Pre výpočet rýchlosti hráča je potrebné poznať jeho zrýchlenie bez pôsobenia gravitácie. Aby sme boli schopní odpočítať vektor gravitačného zrýchlenia od jednotlivých osí agenta, potrebujeme mať informáciu o jednotlivých uhloch odklonenia jeho tela v priestore od osí karteziánskej sústavy. Na to nám poslúžia dáta z gyrorate perceptora, ktorý udáva naklonenie trupu agenta, ktoré je zobrazené na Obrázku 2.4 - 1.



Obrázok 2.4-1 - Zmena naklonenia osí podľa gyroskopu

Gyrorate perceptor udáva natočenia trupu agenta v súradnicovej sústave. Uhol *alfa* prislúcha natočeniu x-ovej súradnice trupu, uhol *beta* y-ovej a uhol *gama* z-ovej. Pre odpočítanie gravitačného zrýchlenia od jednotlivých osí je potrebné poznať ich samotné odklonenie od pôvodnej súradnicovej sústavy. To vypočítame ak jednotkové vektory ležiace na osiach otáčame v priestore podľa uhlov prislúchajúcim zvyšným dvom dimenziám. Pre os x sú uhlami otáčania uhly *beta* a *gama*, pre os y uhly *alfa* a *gama* a nakoniec pre os z uhly *alfa* a *beta*. Výpočet bude realizovaný použitím rotačných matíc [12].

Po výpočte akcelerácie hráča bez pôsobenia gravitačnej sily budeme schopní vypočítať zmenu rýchlosti oproti záznamu z predchádzajúceho cyklu podľa vzorca

$$\Delta v = a.t$$

Zmenu polohy v jednom simulačnom cykle vypočitame pripočítaním vektora zmeny polohy, ktorý vypočítame nasledovne:

$$\Delta p = v.t$$

kde t = 0.02 s, čo je dĺžka jedného simulačného cyklu.

2.4.1.4 Pohyby hráča

- Otočenie z boku na brucho – pohyb je potrebný v prípade, že hráč padne na bok. Hráč sa musí dostať do takej polohy, aby z nej mohol ihneď vstať.
- Otočenie z chrbta na brucho – v prípade, že hráč padne na chrbát, je v niektorých prípadoch potrebné aby sa dostal na brucho. Je to z toho dôvodu, že hráč dokáže niektoré pohyby vykonať iba pri pozícií z brucha.
- Otočenie z brucha na chrbát – v prípade, že hráč padne na brucho, je v niektorých prípadoch potrebné aby sa dostal na chrbát. Je to z rovnakého dôvodu, ako v predchádzajúcom prípade - hráč dokáže niektoré pohyby vykonať iba pri pozícií z chrbta.

2.4.1.5 Pohyby hráča – úprava a vylepšenie vykonávania základných pohybov.

V predošlom šprinte bolo vytvorených niekoľko základných pohybov avšak niektoré z nich neboli v požadovanej kvalite. Jednalo sa o pohyby:

- Vstávanie z chrbta – vstávanie nebolo úplne stabilné, po viacerých opakovaniach pádu sa stalo, že sa hráč dostával do nestabilnej fázy.
- Stabilná chôdza – chôdza, ktorá bola prebratá od tímu Agenty007 bola veľmi labilná a hráč nechodil úplne rovno.
- Stabilná chôdza od autorov JIMa – chôdza bola síce stabilná, ale hráč sa vychyľoval z pôvodného smeru, teda nešiel priamo dopredu.

2.4.1.6 Tvorba a publikovanie projekt backlogu

Je potrebné vytvoriť projektový backlog, ktorý bude slúžiť ako TODO list pre náš projekt. Projekt backlog bude potrebné spracovať pomocou tabuľkového procesora.

2.4.2 Návrh

V tejto kapitole uvádzame návrh riešenia úloh v štvrtom šprinte.

2.4.2.1 Editor pohybov, dorobenie importu xml

Za účelom importovania súborov typu XML pre hráča typu JIM je potrebná samotná úprava používateľského rozhrania editora pohybov a rovnako aj úprava používateľských tried. Keďže riešenie, ktoré vytvoríme musí byť generické a musí poskytovať priestor na rozšírenie, budeme využívať štandardné praktiky objektovo-orientovanej analýzy a návrhu.

Keďže v aplikácií sa už jeden import súborov XML nachádza, je potrebné túto funkcionalitu ponechať a pridať novú. Pre rozšírenie o ďalší import súborov XML bude najvýhodnejšie ak zavedieme rozhranie. Rozhranie bude vytvárať dohodu resp. požiadavky kladené na triedu, ktorá bude zabezpečovať import. To riešenie je najvýhodnejšie aj z prípadného rozširovania editora pohybov o importovanie ďalších typov XML.

Pre rozšírenie editora pohybov je preto potrebné zaviesť nové rozhranie, navrhnúť novú triedu, zabezpečujúcu import a rovnako aj triedy, ktoré trieda pre import využíva (reprezentácia efektorov a fáz). Rovnako bude potrebné upraviť aj triedu *MotionEditorForm*, ktorá musí, pri zvolení jednotlivých typov importovaných súborov, zavolať správnu metódu triedy import.

Pri importovaní súboru sa využije existujúca infraštruktúra pre reprezentáciu pohybov vo vnútri aplikácie. Z dôvodu situácie načítavania viacerých pohybov naraz hráčom Jim, je potrebné jednoznačné odlíšenie názvov fáz aj medzi jednotlivými pohybmi.

Rovnako ako v úlohe pre export súborov XML je využívané úložisko, ktoré sa nachádza na adrese: http://bitbucket.org/marosss/robotbehavioreditor.

2.4.2.2 Spájanie pohybov

Pri návrhu plánovača sme sa rozhodli nadviazať na existujúci systém fáz, nižších a vyšších schopností a iba doplniť nami požadovanú logiku úlohou, ktorej bude uchovávať postupnosť naplánovaných nižších schopností. Rovnako bol navrhnutý mechanizmus postupného vykonávania jednotlivých schopností zo zoznamu a čakania na ďalšie v prípade že je zoznam prázdny a ich okamžité vykonanie ak sú opäť zadané nové schopnosti do zoznamu. Zároveň bola navrhnutá možnosť prerušenia tohto plánu a vyprázdnenia zoznamu v prípade potreby zmeny alebo nastania nečakanej udalosti.

2.4.2.3 Pohyby hráča

Pohyby otočenie z boku na brucho, otočenie z chrbta na brucho, otočenie z brucha na chrbát boli navrhnuté tak, aby sa hráč dokázal dostať do stabilnej polohy čo najrýchlejšie. Je to z toho dôvodu, že pri páde sa hráč musí opäť dostať do vzpriamenej polohy.

2.4.2.4 Pohyby hráča – úprava a vylepšenie vykonávania základných pohybov.

Jednotlivé pohyby sme sa snažili implementovať tak, aby sa hráč nachádzal čo najdlhšie v stabilnej fáze. V predchádzajúcom šprinte bolo vytvorené vstávanie hráča z chrbta, pri ktorom sa hráč po viacerých opakovaniach pádu dostával do labilnej polohy. V hre by sa zrejme táto malá chyba neprejavila, keďže hráč by musel padnúť viackrát po sebe na zem, ale napriek tomu sme sa ju rozhodli odstrániť.

K dispozícii sme mali dve chôdze. Prvá bola prepísaná od tímu Agenty007, tá druhá bola priamo od autorov JIMa. Po dôkladnejšej analýze týchto pohybov sme zistili, že tieto chôdze sa nenachádzajú v nami želanom stave. Chôdza od tímu Agenty007 vyzerala príliš

labilne (aj keď hráč nepadol) a pri menšom dotyku iného hráča by mohol hneď spadnúť. Ďalej chôdza nebola priama, hráč sa po čase vychýlil zo smeru a šiel doprava alebo doľava.
Druhá chôdza, priamo od autorov JIMa vyzerala na prvý pohľad veľmi dobre implementovaná. Hráč bol po celú dobu chôdze stabilný avšak podobne ako pri predchádzajúcej chôdzi sa hráč vychyľoval z priameho smeru do strán.

Cieľom bolo teda upraviť predchádzajúce chôdze tak, aby sa hráč nevychyľoval do strán, resp. aby bol hráč stabilný.

2.4.2.5 Tvorba a publikovanie projekt backlogu

Výber úloh z projektového backlogu bude fungovať podľa princípu LIFO („last in first out“), a teda bude to chronologické zoradenie úloh, pričom úloha na vrchu bude mať najvyššiu prioritu. Na vytvorenie backlogu bude vhodná aplikácia MS Excel.

2.4.3 Implementácia

V tejto kapitole uvádzame implementáciu riešenia úloh v štvrtom šprinte.

2.4.3.1 Editor pohybov, dorobenie importu xml

Dialogové okno importu súboru XML bolo rožírené o nový filter:
dlg.Filter = "Xml files of Jim (.xml)|*.xml|Xml files of Dream Team derivates (*.xml)|*.xml";*
ktorý umožňuje okrem XML od hráča JIM importovať aj XML od iných hráčov. To znamená, že pôvodná funkcionalita zostala zachovaná.

Pre zavolanie správnej metódy pri zvolení jednotlivých typov súborov typu XML bola trieda *MotionEditorForm* rozšírená tak, že sa v triede *ImportXmlMessageFile*, bola implementovaná logika, ktorá trieda sa použije na import. Trieda pre import sa vyberá na základe zvoleného filtra v dialogovom okne pre import správy.

Dôležitím rozhraním je rozhranie *IXmlImport*, ktoré musia implementovať všetky triedy zabezpečujúce import do aplikácie. Rozhranie predpisuje implementáciu dvoch metód: *lowLevelSkillsInFile()* a *movesOf(string lowLevelSkill)*, ktoré sú využívané v aplikácií pre získanie dát zo súborov XML.

Bola vytvorená nová trieda *XmlImportForJim*, ktorá implementuje rozhranie *IXmlImport* a zabezpečuje import dát zo súboru XML určeného pre hráča JIMa. Okrem toho boli vytvorené triedy *EffectorsForJim* a *PhaseForJim*, ktoré slúžia na parsovanie fragmentov zo súboru XML.

2.4.3.2 Spájanie pohybov

Úloha bola implementovaná v jazykoch Java a Ruby. Okrem samotnej logiky opísanej v návrhu bolo implementované aj automatické postavenie hráča ak nastane pád hráča.

2.4.3.3 Akcelerometer

Výpočet akcelerácie bez pôsobenia simulovanej gravitačnej sily nebolo implementované do takej miery, aby bolo prospešné. Z tohto dôvodu túto úlohu považujeme za *nesplnenú* a bola presunutá do jedného z nasledujúcich šprintov. Jedným z možných dôvodov je nesprávne spracovávanie správ gyroskopu, čo bude potrebné otestovať.

Boli implementované nové metódy pre prácu s vektormi a to:

- rotácia vektora okolo osy súradnicovej sústavy
- rotácia vektora okolo iného vektora

Bola potrebná mierna refaktorizácia triedy *AgentModel.java* a presunutie pôvodných funkcií pre prácu s vektormi do triedy *Vector3D.java*.

Kód dodržiava normy určené štábnou kultúrou, je funkčný, refaktorovaný a dostatočne okomentovaný.

2.4.3.4 Pohyby hráča

Pohyby otočenie z boku na brucho a otočenie z chrbta na brucho boli implementované od znovu. Implementácia týchto pohybov bola na základe reálneho správania človeka – boli zapojené tie kĺby, ktoré využíva človek pri vykonaní týchto pohybov. Pri otočení z chrbtu na brucho sme použili pohyb na vstávanie z chrbtu. Pri implementácií sme teda zobrali časť pohybu kĺbov z pohybu pre vstávanie z chrbta a pohyby zvyšných kĺbov sme doplnili analogicky, ako sme to urobili pri predchádzajúcich dvoch pohyboch.

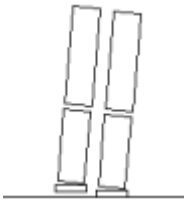
2.4.3.5 Pohyby hráča – úprava a vylepšenie vykonávania základných pohybov.

Pri pohybe vstávanie z chrbta sme museli doladiť len niekoľko drobných detailov ako jemné doladenie uhlov kĺbov a ich rýchlosť.

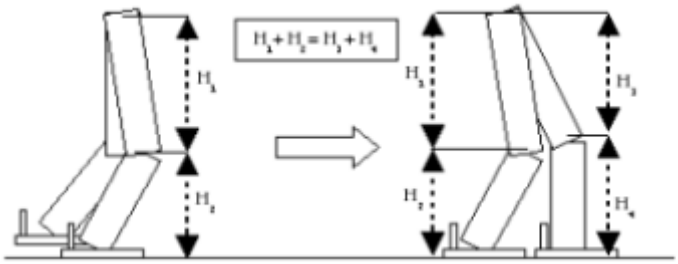
Chôdzu prebranú od tímu Agenty007 sme sa snažili postupne odlaďovať a rozdeliť do dvoch rovnomerných fáz. Rozdelenie na tieto dve fázy je prirodzenejšie, keďže reálne človek pri chôdzi vykonáva tie isté pohyby pravou aj ľavou nohou. Po dlhších úpravách sme však dospeli k tomu, že s touto chôdzou to nebude možné pretože by si vyžadovala kompletné prepísanie. Rozhodli sme sa preto analyzovať druhú chôdzu priamo od autorov JIMa. Táto chôdza spočíva v prenesení ťažiska nižšie k zemi (hráč je mierne skrčený) a v robení malých krokov (drobčenie). Fázy boli pekne rozdelené pre ľavú aj pravá nohu a pri oboch sa vykonávalo presne to isté. Pri tomto drobčení však hráč šúcha nohy o zem, čo pravdepodobne spôsobuje práve to vychyľovanie do strán.

Z dôvodu, že by bolo veľmi namáhavé prerábať tieto chôdze, sme sa rozhodli napísať vlastnú, pri ktorej hráč nebude šúchať o zem a svoju váhu bude prenášať len v stabilnej fáze. Týmto sa vyhneme tomu, že by sa mohol hráč dostať do nestabilnej fázy.
Implementácia stabilnej chôdze pochádza z troch krokov:

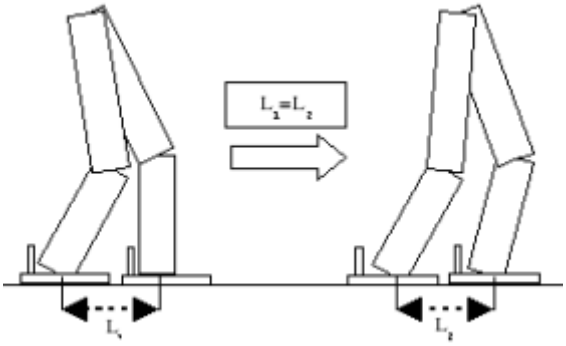
1) Prenesenie váhy na jednu nohu (druhou tak budeme môcť vykročiť) Obrázok 2.4 - 1
2) Vykročenie druhou nohou vpred tak, aby sa položila celá na zem. Obrázok 2.4 - 2
3) Prenesenie váhy na druhú nohu. Obrázok 2.4 - 3



Obrázok 2.4-2 - Prenesenie váhy [11]



Obrázok 2.4-3 - Vykročenie druhou nohou vpred [11]



Obrázok 2.4-4 - Prenesenie na druhú nohu [11]

Boli vytvorené dve stabilné chôdze. Pri prvej chôdzi hráč vždy položí celú nohu na zem, čím nemá ako stratiť stabilitu. Druhá spočíva v robení väčších krokov ako v prvej. Pri tejto chôdzi sa však na chvíľu dostáva do nestabilnej polohy keďže nedokáže položiť celé chodilo na zem. Toto sme však vyriešili ďalším krokom, kde sa hráč nakloní mierne dopredu a došliapne na celé chodidlo.

2.4.3.6 Tvorba a publikovanie projekt backlogu

Pri tvorbe backlogu sa nevyskytli žiadne závažnejšie prekážky. Štruktúra tabuľky bola počas prác niekoľko krát pozmenená, tak aby bola docielená čo najväčšia prehľadnosť.

2.4.4 Testovanie

V tejto kapitole uvádzame opis testovania implementovaných úloh v štvrtom šprinte.

2.4.4.1 Editor pohybov - dorobenie importu xml

Testovanie konečnej funkcionality prebiehalo importovaním pohybov definovaných v XML určených pre hráča JIM tímu The A Team. Importované pohyby boli dôkladne porovnané s importovaným XML. Následne boli údaje znovu exportované a porovnané s pôvodným XML. Pri tomto testovaní nebola objavená žiadna chýba a testy prebehli úspešne.

Importovanie súborov XML sa využívalo už počas vývoja a vďaka nemu sa našli drobné chyby, ktoré zapríčinili nestabilitu celej aplikácie (preklepy pri parsovaní XML, čítanie neexistujúcich elementov a pod.). Všetky odhalené chyby boli odstránené.

2.4.4.2 Spájanie pohybov

Testovanie bolo vykonané nasledovne najprv boli vložené zoznamu v prípade dva otočenia vpravo jeden chybný pohyb, ktorý mal za úlohu hráča zhodiť a tak overiť prerušovanie plánovania, za týmto pohybom boli ešte vložené dva pohyby, ktoré by sa kvôli prerušeniu nemali vykonať. Následne po automatickom postavení boli ešte do zoznamu poslané dva pohyby ktoré mali za úlohu simulovať nový plán úloh pre hráča po prerušení.

Spojený pohyb: Dvojité otočenie, spadnutie, prerušenie vykonávania, postavenie, otočenie a chôdza

Číslo testu	Výsledok
1	Hráč vykonal pohyby tak ako mal dvakrát sa otočil, spadol, prerušil vykonávanie a automaticky sa postavil a potom sa ešte otočil a začal kráčať.
2	Hráč vykonal pohyby tak ako mal dvakrát sa otočil, spadol, prerušil vykonávanie a automaticky sa postavil a potom sa ešte otočil a začal kráčať.
3	Hráč po páde neprerušil pohyby hneď ale až po vykonaní jedného z pohybov, ktoré mali byť vyhodené.
4	Hráč vykonal pohyby tak ako mal dvakrát sa otočil, spadol, prerušil vykonávanie a automaticky sa postavil a potom sa ešte otočil a začal kráčať.
5	Hráč vykonal pohyby tak ako mal dvakrát sa otočil, spadol, prerušil vykonávanie a automaticky sa postavil a potom sa ešte otočil a začal kráčať.

2.4.4.3 Pohyby hráča

Každý novovytvorený pohyb bol niekoľko krát spustený a ponechaný bežať 2 minúty, aby sa overilo jeho fungovanie a stabilita. Nasledujúce tabuľky ukazujú výsledky testov.

Otočenie z boku na brucho:

Číslo testu	Výsledok
1	Hráč spadol na bok a obrátil sa na brucho.
2	Hráč spadol na bok a obrátil sa na brucho.
3	Hráč spadol na bok a obrátil sa na brucho.
4	Hráč spadol na bok a obrátil sa na brucho.
5	Hráč spadol na bok a obrátil sa na brucho.

Otočenie z chrbta na brucho:

Číslo testu	Výsledok
1	Hráč padol na chrbát a obrátil sa na brucho.
2	Hráč padol na chrbát a obrátil sa na brucho.
3	Hráč padol na chrbát a obrátil sa na brucho.
4	Hráč padol na chrbát a obrátil sa na brucho.
5	Hráč padol na chrbát a obrátil sa na bok.

Otočenie z brucha na chrbát:

Číslo testu	Výsledok
1	Hráč padol na brucho a obrátil sa na chrbát.
2	Hráč padol na brucho a obrátil sa na chrbát.
3	Hráč padol na brucho a obrátil sa na chrbát.
4	Hráč padol na brucho a obrátil sa na chrbát.
5	Hráč padol na brucho a obrátil sa na chrbát.

2.4.4.4 Pohyby hráča – úprava a vylepšenie vykonávania základných pohybov.

Každý novovytvorený pohyb bol niekoľko krát spustený a ponechaný bežať 2 minúty, aby sa overilo jeho fungovanie a stabilita. Nasledujúce tabuľky ukazujú výsledky testov.

Vstávanie z chrbta:

Číslo testu	Výsledok
1	Hráč spadol smerom dozadu a postavil sa.
2	Hráč spadol smerom dozadu a postavil sa.
3	Hráč spadol smerom dozadu a postavil sa.
4	Hráč spadol smerom dozadu a postavil sa.
5	Hráč spadol smerom dozadu a postavil sa.

Prvá chôdza:

Číslo testu	Výsledok
1	Hráč kráčal celý čas dopredu a nedošlo k pádu
2	Hráč kráčal celý čas dopredu a nedošlo k pádu
3	Hráč kráčal celý čas dopredu a nedošlo k pádu
4	Hráč kráčal celý čas dopredu a nedošlo k pádu
5	Hráč kráčal celý čas dopredu a nedošlo k pádu

Druhá chôdza:

Číslo testu	Výsledok
1	Hráč kráčal celý čas dopredu a nedošlo k pádu
2	Hráč kráčal celý čas dopredu a nedošlo k pádu
3	Hráč kráčal celý čas dopredu a nedošlo k pádu
4	Hráč kráčal celý čas dopredu a nedošlo k pádu
5	Hráč kráčal celý čas dopredu a nedošlo k pádu

2.4.4.5 Tvorba a publikovanie projekt backlogu

Projektový backlog bol vytvorený úspešne a jeho napĺňanie preukázalo jeho dobrú použiteľnosť. Otvorenie dokumentu prebehlo úspešne aj po jeho stiahnutí z webovej stránky tímu.

2.5 Šprint č. 05

Číslo šprintu	05
Počet príbehov	11
Začiatok šprintu	02.12.2010
Koniec šprintu	09.12.2010

Príbehy

Doplnenie dokumentácie k inžinierskemu dielu **User story:** žiadne
Doplnenie dokumentácie k riadeniu projektu **User story:** žiadne
Finalizácia dokumentácie, formátovanie a vytlačenie **User story:**
Doplnenie komentárov do zdrojových kódov, refaktoring a ich formátovanie **User story:** žiadne
Dohodnutie prezentácie s tímom 05 pre porovnanie dosiahnutých výsledkov **User story:** Ako členovia tímu budeme vedieť kedy sa koná prezentácia výsledkov s druhý tímom a rovnako budeme vedieť aj rámcový obsah prezentácie.
Prezentácia - technické zabezpečenie **User story:** Ako prezentujúci potrebujem mať pripravené materiály na prezentáciu, aby bolo možné predviesť dosiahnuté výsledky.
Prezentácia - pripraviť powerpointovskú prezentáciu **User story:** Ako prezentujúci potrebujem mať pripravenú štruktúru prezentácie, aby bolo možné plynule predviesť dosiahnuté výsledky.
Úprava a aktualizácia web stránky **User story:** Ako product owner potrebujem mať k dispozícii aktuálny prehľad stavu projektu, aby som mal dohľad nad jeho progresom.

2.5.1 Implementácia

V tejto kapitole uvádzame implementáciu riešenia úloh v piatom šprinte.

2.5.1.1 Doplnenie dokumentácie k inžinierskemu dielu

Dokumentácia k inžinierskemu dielu bola doplnená a vytvorená podľa návrhu softvérom Microsoft Word. Export do formátu pdf bol realizovaný pomocou softvéru PDF Creator.

2.5.1.2 Doplnenie dokumentácie k riadeniu projektu

Dokumentácia k riadeniu projektu bola doplnená a vytvorená podľa návrhu softvérom Microsoft Word. Export do formátu pdf bol realizovaný pomocou softvéru PDF Creator.

2.5.1.3 Finalizácia dokumentácie a vytlačenie

Dokumentácia k inžinierskemu dielu a dokumentácia k riadeniu projektu bola formálne upravená do finálnej podoby a vytlačená.

2.5.1.4 Dohodnutie prezentácie

Termín prezentácie s tímom č. 05 bol dohodnutý na 16.12.2010 o 10:00. S termínom boli oboznámení vedúci a členovia oboch tímov.

2.5.1.5 Prezentácia - technické zabezpečenie

Obsah praktickej prezentácie prototypu bol navrhnutý a prezentovaný členmi tímu.

2.5.1.6 Prezentácia - pripraviť obsah prezentácie

Obsah prezentácie prototypu bol navrhnutý a prezentácia bola vytvorená softvérom Microsoft PowerPoint. Export do formátu pdf bol realizovaný pomocou softvéru PDF Creator.

2.5.1.7 Úprava a aktualizácia web stránky

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

2.5.2 Testovanie

V tejto kapitole uvádzame opis testovania implementovaných úloh v piatom šprinte.

2.5.2.1 Doplnenie dokumentácie k inžinierskemu dielu

Dokumentácia k inžinierskemu dielu bola opravená členmi tímu na štylistické a gramatické chyby.

2.5.2.2 Doplnenie dokumentácie k riadeniu projektu

Dokumentácia k riadeniu projektu bola opravená členmi tímu na štylistické a gramatické chyby.

2.5.2.3 Finalizácia dokumentácie a vytlačenie

Dokumentácia k inžinierskemu dielu a dokumentácia k riadeniu projektu bola skontrolovaná po formálne stránke.

2.5.2.4 Dohodnutie prezentácie

Termín dohodnutia prezentácie bude overený uskutočnením prezentácie v dohodnutom termíne.

2.5.2.5 Prezentácia - technické zabezpečenie

Obsah praktickej prezentácie prototypu bol upravený a doplnený členmi tímu.

2.5.2.6 Prezentácia - pripraviť obsah prezentácie

Prezentácia prototypu bola opravená členmi tímu po obsahovej stránke a na štylistické a gramatické chyby.

2.5.2.7 Úprava a aktualizácia web stránky

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

2.6 Šprint č. 06

Číslo šprintu	06
Počet príbehov	10
Začiatok šprintu	14.02.2011
Koniec šprintu	28.02.2011

Príbehy

Aktualizácia dokumentácie **User story:** žiadne
Otestovanie validity dát akcelerometra **User story:** žiadne
Výpočet rýchlosti a polohy hráča na ihrisku **User story:** žiadne
Vytvorenie pohybov rýchla chôdza **User story:** žiadne
Vytvorenie pohybov otočenie vľavo a vpravo **User story:** žiadne
Vytvorenie hrubého plánu letného semestra pre jednotlivé šprinty **User story:** žiadne
Návrh komponentu správania **User story:** žiadne
Dokončenie spájania pohybov **User story:** žiadne
Analýza a návrh predikcie polohy časti tela agenta User story: žiadne
Úprava a aktualizácia web stránky

User story:
žiadne

2.6.1 Analýza

2.6.1.1 Dokončenie spájania pohybov

Úloha bola pokračovaním úlohy z predchádzajúceho šprintu a preto zdieľa aj analýzu vykonanú pre túto úlohu.

2.6.1.2 Akcelerometer

Úloha správneho fungovania akcelerometra nebola v predchádzajúcom šprinte splnená. Dôvodom mohol byť zlý výpočet rotácie vektorov v priestore resp. nesprávne spracovanie správ gyrorate perceptora.

Pre otestovanie správneho spracovanie správ gyroskopu je potrebné dopísať JUnit testy s viacerými testovacími prípadmi a vstupmi. Pri nezistení chyby je nutné otestovať samotnú funkcionalitu výpočtu rotácie vektorov v priestore.

2.6.1.3 Otestovanie validity dát akcelerometra

Pôvodný hráč správne počítal polohu svoju polohu z dát zo See perceptora. Z dôvodu občasného zlého výpočtu rýchlosti a polohy tela agenta použitím dát akcelerometra je potrebné tieto veličiny kalibrovať. To sa deje použitím dát zo See perceptora, u ktorého sme si ich správnosťou istejší. Preto má tento výpočet vždy prednosť pred výpočtom akcelerometra a prepisuje veličiny hráča.

2.6.2 Návrh

2.6.2.1 Dokončenie spájania pohybov

Rovnako táto úloha zdieľa aj návrh s úlohou, na ktorú nadväzuje. Boli spravené len mierne úpravy kvôli rozhraniu ktoré má ponúkať vyšším vrstvám správania.

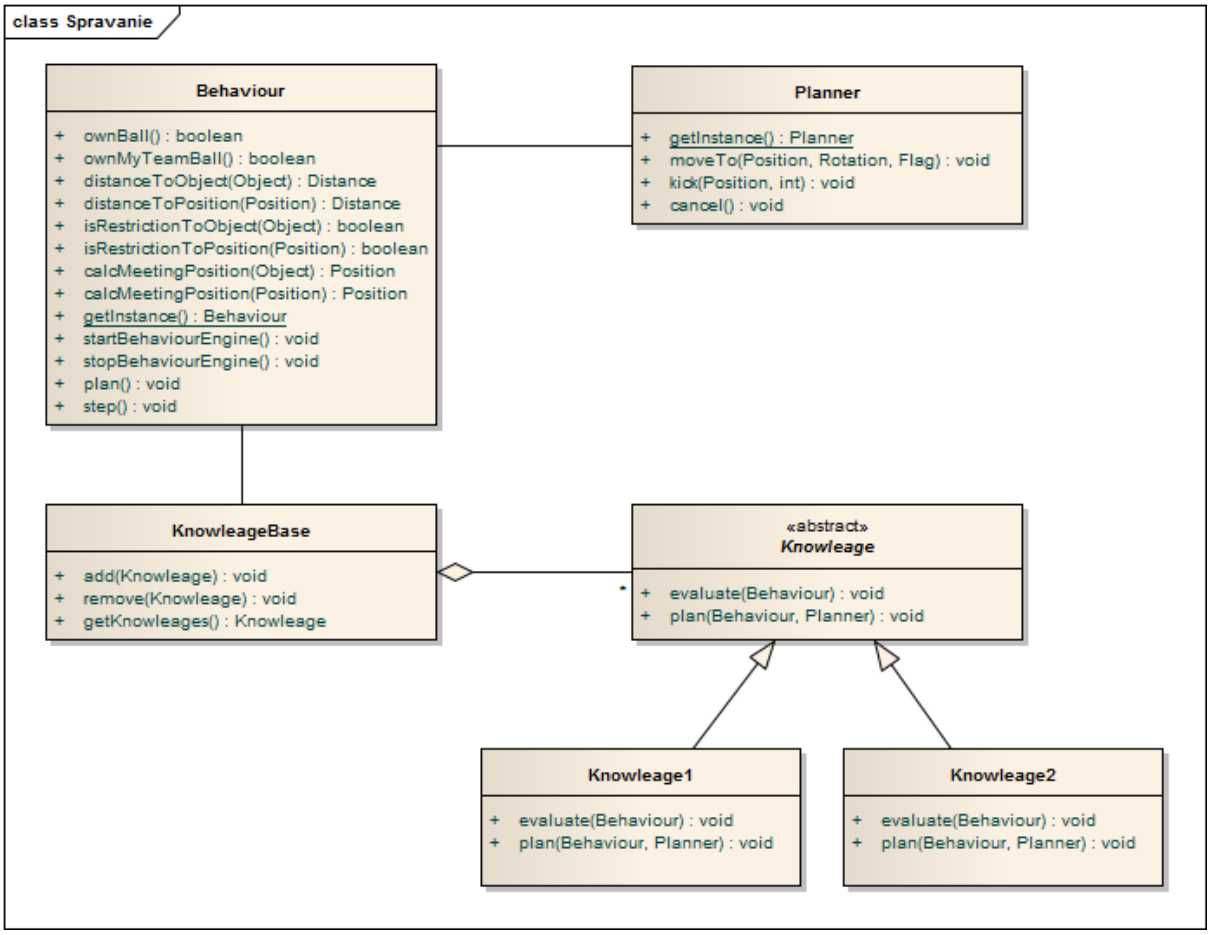
2.6.2.2 Návrh komponentu správania

Konkrétny návrh komponentu správania

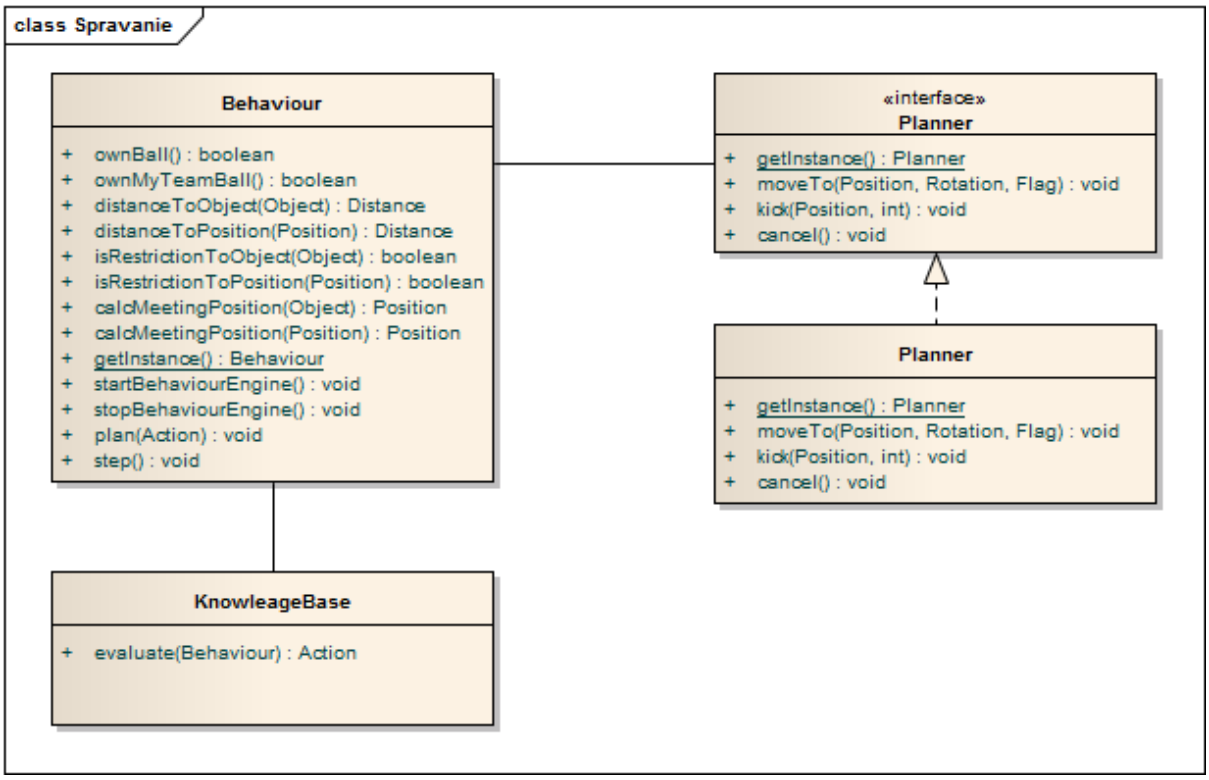
Komponent správania je najdôležitejším komponentom autonómneho agenta, pretože zaručuje jeho správanie sa v čase. Agent pomocou tohto komponentu vyhodnocuje aktuálnu situáciu a na základe nich plánuje akcie, ktoré vykoná. V našom prvom návrhu sme v rámci tohto komponentu identifikovali štyri dôležité časti - Individuálne správanie, Tímové

správanie, Báza znalostí a Plánovač. V tomto návrhu individuálne správanie charakterizuje správanie hráča ako jednotlivca a využíva sa v tímovom správaní (hráči pri hre spolupracujú). Správanie sa rozhoduje na základe bázy znalostí, ktorá obsahuje možných akcie. Akcie, ktoré sa vykonajú sú naplánované a vložené do plánovača, ktorého úlohou je správne usporiadanie akcií, zmena ich poradia, zrušenie niektorej akcie, podľa pokynov z individuálneho a tímového správania hráča. Vstupmi pre rozhodovanie sú informácie o svete z modelu sveta.

Z tohto návrhu sme vychádzali pri konkrétnom návrhu komponentu správania. V konkrétnom návrhu uvažujeme dve rôzne riešenia, ktoré sú znázornené na diagramoch X a Y. Sú to dva rôzne prístupy a preto opíšeme čím sa odlišujú. Oba prístupy majú svoje výhody a nevýhody. Najprv opíšeme ich spoločné vlastnosti a potom vysvetlíme v čom sa odlišujú.



Obrázok 2.4-5 – Prvý návrh



Obrázok 2.4-6 – Druhý návrh

V konkrétnom návrhu uvažujeme 3 resp. 4 a viac tried (v závislosti od prístupu) a jedno rozhranie. Individuálne správanie a tímové správanie v konkrétnom návrhu predstavuje trieda *Behaviour*. Trieda *Planner* predstavuje plánovač, bázu znalostí trieda *KnowledgeBase*. Okrem toho uvažujeme rozhranie, ktoré implementuje trieda *Planner*. Najdôležitejšie triedy sú triedy *Behaviour* a *Planner*. Implementácia triedy *KnowledgeBase* závisí od použitého prístupu (viď nižšie).

Trieda *Behaviour* má nasledovné metódy:

- *ownBall*() - vracia true/false, či sa hráč nachádza v dostatočnej blízkosti lopty,
- *ownMyTeamBall*() - vracia true/false, či hráčov tím má loptu,
- *distanceToObject(object)* - vzdialenosť k bráne, lopte alebo inému objektu,
- *distanceToPosition(position)* - vzdialenosť k danej pozícií,
- *isRestrictionToObject(object)* - na priamke k danému objektu je prekážka,
- *isRestrictionToPosition(position)* - na priamke k danej pozícií je prekážka,
- *calcMeetingPosition(object)* - vypočíta pozíciu stretnutia hráča s objektom (lopta, iný hráč...),
- *getInstance*() - vráti inštanciu triedy *Behaviour* (návrhový vzor *Singleton*),
- *startBehaviourEngine*() - spustí vyhodnocovanie správania,
- *stopBehaviourEngine*() - zastaví vyhodnocovanie správania.

Trieda *Planner* má nasledovné metódy:

- *getInstance*() - vráti inštanciu triedy *Behaviour* (návrhový vzor *Singleton*),
- *moveTo(position, rotation, flag)* - naplánuje posunutie hráča na zadanú pozíciu s určenou rotáciou,
- *kick(position, power)* - naplánuje kop na danú pozíciu s danou silou.,
- *cancel()* - zruší aktuálny pohyb.

Jednotlivé komponenty v tomto návrhu boli navrhnuté tak, aby ich bolo možné nahradiť inými a aby medzi nimi bola voľná viazanosť. Preto je možné aj reálne ľubovoľný z komponentov nahradiť, bez redundantnej úpravy niektorého iného komponentu.

Prístup k báze znalostí pomocou dedenia

Na diagrame na obrázku 2.6.2 1 je znázornený prvý prístup - prístup pomocou dedenia. V tomto prípade sú jednotlivé pravidlá (znalosti) implementované pomocou tried, ktoré dedia od triedy *Knowledge*. *Knowledge* predpisuje dve metódy, ktoré musia potomkovia implementovať - metódy *evaluate*() a *plan*(). Pri na tomto prístupe sme využili návrhový vzor *Strategy*.

Metóda *evaluate* slúži na overenie, či sa dané pravidlo môže použiť za daných podmienok. Podmienky definuje trieda *Behaviour* pomocou svojich *public* metód. Ak je metóda *evaluate*() vyhodnotená na *true*, zavolá sa metóda plan(), ktorá zavolá jednu z metód plánovača s konkrétnou akciou. Tak sa zabezpečí naplánovanie tejto akcie v plánovači a plánovač sa postará o jej splnenie.

Jednotlivé pravidlá sú agregované v triede *KnowledgeBase*. V tejto triede je v každom kroku prechádzaná celá agregácia, sú vyhodnocované podmienky jednotlivých pravidiel a v prípade, že je pri niektorom pravidle splnená podmienka, je dané pravidlo uplatnené pomocou metódy *plan*(). Výhodou je ľahšia implementácia, ktorá ale prináša vyššiu zložitosť - logika sa bude nachádzať na viacerých miestach. Vyhodnocovanie bude ale pravdepodobne rýchle.

Prístup k báze znalostí pomocou logických klauzúl (Prolog)

Druhý prístup je založený na princípoch jazyka Prolog (diagram na obrázku 2.6.2-2). Pri tomto prístupe bude treba použiť niektorú z implementácie jazyka Prolog pomocou Javy (existujú na to dostupné knižnice), alebo naprogramovaním jednoduchého produkčného systému. V tomto prístupe budú zapísané všetky pravidlá v textovom súbore pomocou Hornových klauzúl. Následne sa bude vyhodnocovať dopyt z množiny pravidiel, pomocou funkcie *evaluate*() v triede *KnowledgeBase*, ktorá vráti akciu, ktorá sa má vykonať. Tá bude spracovaná pomocou metódy *plan*() a naplánovaná v plánovači. Výhodou tejto metódy je jednoduchá reprezentácia a úprava pravidiel. Správanie hráča je teda veľmi jednoduché zmeniť na jednom mieste a je ho dokonca možné meniť dynamicky. Nevýhodou je, že pri tomto spôsobe bude zrejme potrebné viac implementovať (produkčný systém) na rozdiel od prvého spôsobu. Taktiež samotné vyhodnocovanie pravidiel môže byť pomalšie.

Návrh pravidiel v báze znalostí

1. Ak nikto z môjho tímu nemá loptu, tak idem za loptou.
2. Ak mám loptu, idem k bráne.
3. Ak mám loptu a som pri bráne, tak kopnem.
4. Ak nemám loptu a niekto z tímu má loptu, bežím k bráne po druhej strane.
5. Ak nevidím loptu, otočím sa o X stupňov.
   -Ak som príliš vľavo, tak sa otáčam doprava.
   -Ak som príliš vpravo, tak sa otáčam doľava.
   -Ak som nevidel loptu a vidím loptu, otočím sa presne smerom k lopte.
6. Ak som pri bránke a nemôžem strieľať, tak prihrám loptu.
7. Ak som pri bránke a nemôžem strieľať a nemôžem prihrať, tak kopnem šikmo
8. Ak mám loptu a som pri bráne v strede, tak kopnem loptu vysoko.
9. Ak som brankár, som len na bránkovej čiare.
10. Ak som brankár idem na pozíciu stretnutia s loptou
11. Ak som brankár a nestíham na pozíciu stretnutia s loptou, tak sa hodím.
12. Ak som brankár a mám loptu, tak prihrám hráčov, ktorý je najvoľnejší.

Uvedené pravidlá sú tie najzákladnejšie, ktoré by náš hráč mal poznať. Pri implementácií sa budú musieť ešte zjemňovať. Ďalšie zjemňovanie a ladenie pravidiel predpokladáme pri testovaní hráča.

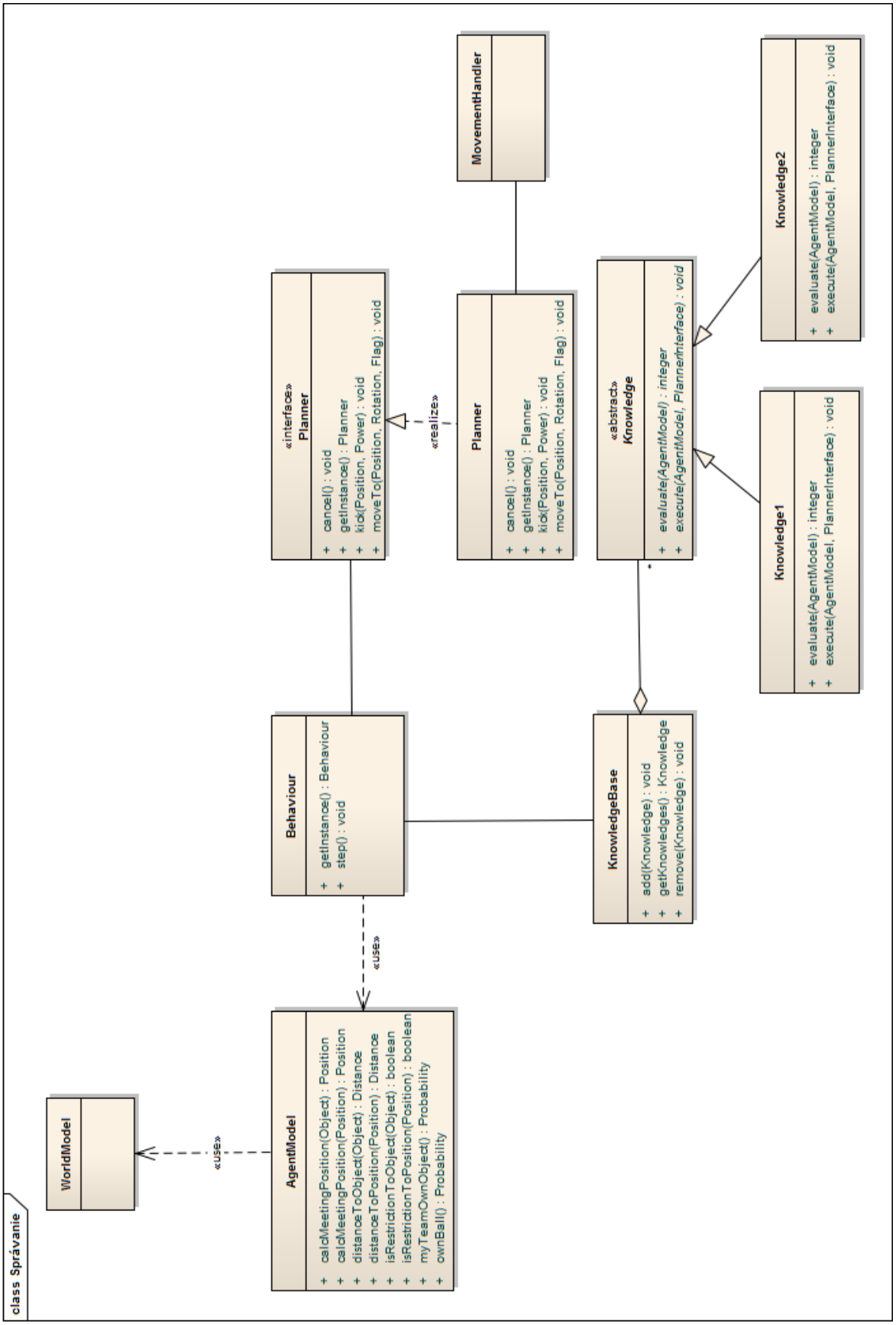
Pravidlá sú zapísané v tvare ak podmienka, potom akcia, čo umožňuje ich ľahké pochopenie, a zároveň ľahkú implementáciu (či už v pomocou prvého alebo druhého prístupu, ktoré boli spomenuté v kapitole Konkrétny návrh komponentu správania). Podmienky v hlave pravidla je možné zreťazovať. Rovnako predpokladáme prioritu pravidiel (či už pozíciou v súbore - podľa pravidiel prologu sa použije prvé pravidlo, pre ktoré platia dané podmienky, alebo zoradením objektov triedy *Knowledge* v agregácií triedy *KnowledgeBase*)

Vybraný spôsob pre model správania

V ďalšom postupe sme sa rozhodli postupovať prvým riešením, teda hlavný princíp pozostáva z myšlienky, že každé pravidlo je implementované ako samostatné trieda. Triedy dedia od abstraktnej triedy *Knowledge.*

Toto riešenie sa konzultovalo v rámci nášho tímu a rovanko aj s product ownerom a ďalej sa optimalizovalo. Na obrázku 2.1 je uvedený finálny návrh správania. Hlavné zmeny, ktorými tento návrh prešiel, sú nasledovné:

1. Jednotlivé metódy z triedy *Behavior*, ako napríklad metódy *ownBall, myTeamOwnBall, calcMeetPosition* boli presunuté priamo do modelu. Je to z toho dôvodu, že tieto metódy zodpovedajú skôr metódam, ktoré by mal poskytovať priamo model agenta.
2. Z triedy *Behavior* bola odstránená metóda *plan.* Táto metóda nie je priamo využívané v triede *Behavior*, pretože plánovanie prebieha v plánovači.



Obrázok. 2.1. Dátový model komponentu správania.

2.6.3 Implementácia

2.6.3.1 Dokončenie spájania pohybov

Úloha bola nadviazala na predošlú implementáciu v jazyku Java.

2.6.3.2 Akcelerometer

Bola opravená funkcionalita výpočtu rotácií priamok v priestore a implementovaná funkčná verzia akcelerometra. Z jeho dát vypočítaná rýchlosť hráča a z rýchlosti jeho poloha. Občas vznikajú situácie, kedy je korektnosť výstupov obmedzená, z toho dôvodu, že dáta prichádzajúce z perceptora akcelerometra sú zaokrúhľované na dve desatinné miesta. To spôsobuje občasné "uletenie" vypočítanej rýchlosti a polohy do vysokých hodnôt.

Pri implementácii samotného testovania spracovávania správ gyroskopu sme zistili, že simulačný server v prvých cykloch simulácie posiela nekorektné informácie z akcelerometra (rádovo prvých 500 cyklov) – nekorektne vysoké hodnoty zrýchlenia pre jednotlivé osy tela hráča. Gyroskop funguje správne.

Z týchto dvoch zistení vznikla pre ďalší šprint potreba kalibrácie výpočtu rýchlosti a polohy hráča použitím dát zo *See* perceptora.

2.6.3.3 Otestovanie validity dát akcelerometra

Pri implementácii kalibrácie výpočtov z dát akcelerometra sme zistili, že data zo See perceptora neposiela simulačný server každý herný cyklus, ale až každý tretí. Preto kalibrácia rýchlosti prebehne vždy vtedy, ak má hráč k dispozícii informácie o svojej polohe vypočítané z toho, čo vidí, štyri cykly po sebe. Výsledná rýchlosť sa potom vypočíta ako:

$$VP – ZP / 3 \times DC$$

Kde VP – ZP je dĺžka vektora rozdielu výslednej a začiatočnej polohy, ktorý je následne vydelený dĺžkou troch cyklov, čo predstavuje 0,06 sekúnd.

2.6.4 Testovanie

2.6.4.1 Dokončenie spájania pohybov

Počas testovania boli odhalené tieto nedostatky:

1. Nedostatok pohybov, ktoré nie sú nadefinované ako cykly. Čo obmedzilo dostupnosť niektorých miest, pretože hráč mal obmedzené pohybové schopnosti.

2. Niektoré funkcie modelu agenta, ktoré sa využívajú podávajú chybné údaje. Najviac to bolo vidno na funkcii getPosition(), ktorá vracala rozdielne údaje pri rôznych počiatočných otočeniach hráča. Rovnako celá súradnicová sústav modelu vyzerala byť posunutá o jeden meter smerom k súperovej bráne.

3. Problém s začiatočnou hodnotou gyroskopu. Nie vždy gyroskop stihol vypočítať aktuálny stav skôr ako sa začalo plánovať spájanie pohybov a teda vrátilo volanie jeho funkcií hodnoty, ktoré spôsobili nesprávne plánovanie.

4. Problematický sa ukázal aj jednoduchý greedy plánovač.

2.7 Šprint č. 07

Číslo šprintu	07
Počet príbehov	08
Začiatok šprintu	28.02.2011
Koniec šprintu	14.03.2011

Príbehy

Plánovač - odstránenie bugov, ktoré nastali **User story:** žiadne
Navrhnúť vrstvu nad plánovačom, analýza iných tímov, začiatok implementácie **User story:** žiadne
Kalibrácia akcelerometra a doplnenie modelu sveta **User story:** žiadne
Implementácia frameworku správania **User story:** žiadne
Pohyby - kopnutie zľava, kopnutie sprava, rôzne úrovne sily kopnutia, dokončiť chôdzu **User story:** žiadne
Dokumentácia **User story:** žiadne
Analýza, návrh a implementácia predikcie polohy časti tela agenta **User story:**

žiadne
Úprava a aktualizácia web stránky **User story:** žiadne

2.7.1 Analýza

2.7.1.1 Predikcia polohy časti tela agenta

Cieľom tejto úlohy je predigovanie polohy jednotlivých častí tela agenta. Informácie o agentovi dostávame prostredníctvom informácií o jednotlivých kĺboch jeho tela. Jednotlivé kĺby sú v správach zo servera jednoznačne identifikované kľúčovými skratkami a aktuálnym uhlom daného kĺbu agenta. Uhly kĺbov sú v stupňoch a preto práca s nimi bude jednoduchá. Predikcia bude spočívať v zapamätaní si polôh kĺbov v čase a ich stavu, a teda či kĺb je v pohybe a teda nemôže byť použitý v inom pohybe alebo nie je v pohybe a je možné ho použiť pre iný pohyb.

2.7.1.2 Plánovač - odstránenie bugov, ktoré nastali

Úloha bola pokračovaním úlohy z predchádzajúceho šprintu „Dokončenie spájania pohybov“ a analýza sa zakladala na výsledkoch testov tejto úlohy.

2.7.1.3 Navrhnúť vrstvu nad plánovačom, analýza iných tímov, začiatok implementácie

Analýza úlohy spočívala v prejdení analýzy minuloročných tímov a hľadaní inšpirácie pre naše riešenie. Konkrétne sme sa zamerali na tím Neurotics z roku 2007, ktorý nám bol odporučený vedúcim tímu.

2.7.2 Návrh

2.7.2.1 Predikcia polohy časti tela agenta

Na základe zistených informácií sme sa rozhodli definovať polohu častí tela prostredníctvom polohy jednotlivých kĺbov. Kĺby a ich stavy budeme ukladať do tabuľky. Rozmermi tabuľky budú časová línia (reprezentovaná bodmi kedy príde odozva zo servera) a jednotlivé kĺby agenta. Využijeme existujúce triedy pre spracovanie prijatých dát zo servera a naparsujeme dáta, ktoré budeme pri práci potrebovať. V našom prípade ide o kĺby a ich polohu. Ďalej budeme využívať triedy pracujúce na zadávaní pohybov agentovi, z ktorých budeme získavať požadované uhly kĺbov, ktoré má agent dosiahnuť, tak aby sme následným porovnaním

požadovaného a aktuálneho stavu kĺbu vedeli explicitne označiť kĺb za pohybujúci či nečinný. Samozrejmosťou bude aj odkladanie všetkých polôh jednotlivých kĺbov v navrhnutej tabuľke, čo by neskôr mohlo napomôcť pri odlaďovaní pohybov agenta. Výstupom modulu by mali byť informácie o stave každého kĺbu (v pohybe/nečinný).

2.7.2.2 Plánovač - odstránenie bugov, ktoré nastali

Úloha bola navrhnutá tak, že sa oddelilo plánovanie z triedy Planner. V tejto triede sa vytvorila premenná typu IMovementPlanner , ktorej jediná metóda planMovements(actualDestination) zabezpečí naplánovanie pohybov pre dosiahnutie danej destinácie. Tým sa umožnilo vytváranie viacerých plánovačov. A oddelilo sa manažovanie destinácií od plánovania samotných pohybov.

2.7.2.3 Navrhnúť vrstvu nad plánovačom, analýza iných tímov, začiatok implementácie

Pre túto vrstvu správania bolo potrebne navrhnúť systém tried ktoré by umožnili vytvorenie vrstiev úloh kde by umožňovali vyšším vrstvám využívať funkcionalitu nižších typov správaní. Objektovo orientované jazyky sú pre túto úlohu obzvlášť vhodné vďaka dedeniu. Pre túto úlohu sme sa rozhodli využiť architektonický štýl Command, ktorý sme postavili na skupine tried implementujúcich rozhranie ICommand. Tieto triedy skupinu konštruktorov, pre inicializáciu vnútorných premenných a metódu execute, ktorá zabezpečí vykonanie úlohy pre ktorú je daná trieda určená. Jednotlivé triedy pritom môžu jedna od druhej dediť alebo sa využívať priamym volaním.

2.7.3 Implementácia

2.7.3.1 Plánovač - odstránenie bugov, ktoré nastali

Úloha implementovaná v jazyku Java. Implementácia spočívala vo vytvorení triedy SimpleMovementPlanner, ktorej plánovanie bolo postavené síce tiež na greedy algoritme avšak s upravenou heuristikou výberu pohybov.

2.7.3.2 Navrhnúť vrstvu nad plánovačom, analýza iných tímov, začiatok implementácie

Kvôli nedostatku času sa nám nepodarilo začať implementáciu.

2.7.3.3 Kalibrácia akcelerometra a doplnenie modelu sveta

V rámci implementácie vyššieho správania sa hráča bolo potrebné, aby bol schopný určiť stav časti sveta, o ktorej má informácie. Boli definované nasledovné stavové funkcie.

`Vector3D calcMeetingPosition(DynamicObject object)` – vypočíta pozíciu stretu hráča s pohybujúcim sa objektom, ktorý je argumentom funkcie, v určitom cykle.
`Vector3D calcMeetingPosition(Vector3D position)` – vypočíta čas stretu hráča so statickým bodom, ktorý je argumentom funkcie, v určitom cykle.
`double getDistanceToObject(DynamicObject object)` – vypočíta vzdialenosť od objektu
`boolean isRestrictionToObject(DynamicObject object)` – zistí, či má agent voľnú cestu k objektu, ktorý vystupuje ako argument (zatiaľ neimplementované)
`boolean isRestrictionToObject(Vector3D position)` – zistí, či má agent voľnú cestu k statickému bodu, ktorý vystupuje ako argument (zatiaľ neimplementované)
`boolean myTeamHasBall()` – zistí, či niektorý zo spoluhráčov agenta má loptu
`boolean haveBall()` – zistí, či má agent loptu

Tvorba funkcií, ktoré ešte neboli implementované sa prenáša do ďalšieho šprintu. Je nutné skontrolovať fungovanie parsovanie dát z perceptoru See, ktorý počíta polohy spolohráčov a protihráčov.

2.7.3.4 Implementácia komponentu správania

Implementovanie modulu správania prebehlo podľa uvedeného diagramu. Bolo vytvorených niekoľko pravidiel, podľa ktorých sa mal hráč správať. Problém ale nastal s integráciou modulu správania s plánovačom. Plánovač obsahuje momentálne niekoľko chýb, pre ktoré sa nepodarilo funkčnosť správania plne zintegrovať a otestovať.

2.7.4 Testovanie

2.7.4.1 Plánovač - odstránenie bugov, ktoré nastali

Testovanie dopadlo o lepšie ako v úlohe „Dokončenie spájania pohybov“ avšak opakovali sa problémy 1. – 3. .

2.7.4.2 Navrhnúť vrstvu nad plánovačom, analýza iných tímov, začiatok implementácie

Keďže sa nám nepodarilo začať implementáciu žiadne testy neboli vykonané.

2.8 Šprint č. 08

Číslo šprintu	08
Počet príbehov	10
Začiatok šprintu	14.03.2011
Koniec šprintu	28.03.2011

Príbehy

Odstránenie skriptov z plánovača **User story:** žiadne
Odstránenie chýb z modelu hráča a sveta a doplnenie funkcii pre vyššie správanie a medzivrstvu plánovača **User story:** žiadne
Vyššie správanie ďalšie pravidlá **User story:** žiadne
Dokončiť kopnutia - kopnutie zľava, kopnutie sprava, silný kop, lob **User story:** žiadne
Predikcia pozícií predmetov vo svete **User story:** žiadne
Vytvoriť otáčajúce chôdze **User story:** žiadne
Implementácia vrstvy nad plánovačom **User story:** žiadne
Dokumentácia **User story:** žiadne
Implementácia predikcie polohy časti tela agenta

User story: žiadne
Úprava a aktualizácia web stránky User story: žiadne

2.8.1 Analýza

2.8.1.1 Odstránenie skriptov z plánovača

Analýza práce plánovača bola vykonaná v prvom semestri.

2.8.1.2 Predikcia polohy predmetov vo svete

Z potreby implementácie vrstvy zabezpečujúcej správanie hráča vznikla požiadavka na implementáciu metódy zisťujúcej viditeľnosť lopty hráčom a predpokladanie jej polohy v určitom čase.

Z dokumentácie Simspark bolo zistené, že z Vision Perceptorov, perceptorov informujúcich o objektoch, ktoré sú videné hráčom, nie sú posielané správy zo servera v každom cykle, ale až v každom tretom cykle. Tento fakt spôsobuje v súčasnej implementácií chýbajúce informácie v modeli sveta.

K predpokladaniu polohy lopty je v súčasnom hráčovi využívaná trieda *DynamicObject*, ktorej inštanciou je aj samotný objekt lopta. K predpokladaniu polohy lopty je využívaný nasledujúci postup. Z relatívnej polohy lopty získanej zo správy servera je vypočítaná absolútna poloha lopty a táto poloha je uchovávaná spolu s časom videnia. Pri ďalšom videní lopty je už možné predpokladať polohu lopty v určitom čase, pretože získaním ďalšej absolútnej polohy lopty je vypočítaná rýchlosť lopty. Na základe rýchlosti a poslednej známej pozície lopty je vypočítaná predpokladaná poloha lopty v určitom čase.

Problém v súčasnej implementácii je vznik výnimky v situácii, keď je volaná metóda výpočtu predpokladanej polohy lopty, ale nebola získaná druhá absolútna poloha lopty, čiže nenastavená hodnota rýchlosti lopty.

2.8.2 Návrh

2.8.2.1 Odstránenie skriptov z plánovača

Úloha si nevyžadovala žiadny zložitý návrh spočívala hlavne vo vytvorení Java triedy, ktorá by nahradila Ruby triedy ktoré plánovač využíval.

2.8.2.2 Predikcia polohy predmetov vo svete

Zo zistené faktu posielania správ o objektoch, ktoré sú videné hráčom až v každom treťom cykle, vznikol nasledujúci návrh metódy zisťujúce viditeľnosť lopty. Ak čas posledného videnia lopty, čiže získania správy zo servera je väčší alebo rovný ako 0.06, tak je metóda vracia logickú hodnotu o nevidení lopty hráčom. V opačnom prípade je vracia logickú hodnotu o videní lopty. Hodnota 0.06 je určená na základe trvania jedného cyklu, čo je 0.02. Hodnota 0.06 spôsobuje využitie informácií o lopte aj v cykloch, ktorých nie sú posielané správy o videných objektoch.
Problém vzniku výnimky v súčasnej implementácií predikcie lopty je možné odstrániť nastaveným počiatočnej hodnoty rýchlosti nulovým vektorom pomocou statickej metódy *cartesian*.

2.8.3 Implementácia

2.8.3.1 Odstránenie skriptov z plánovača

Úloha implementovaná v jazyku Java. Implementácia spočívala vo vytvorení triedy MoveExecutor, a jednej jej vnorenej triedy Skill, ktoré nahradili celú funkcionalitu dovtedy vykonávanú Ruby triedami.

2.8.3.2 Predikcia polohy časti tela agenta

Triedy boli implementované podľa navrhnutého modelu. Nové triedy boli vytvorené v balíčku sk.fiit.jim.agent.prediction. Volanie týchto tried (Prediction.processJoint(jointName, finalAngle, actualAngle)) prebieha v metóde calculateAngularSpeedFor, ktorá sa nachádza v triede JointPlacement v balíčku sk.fiit.jim.agent.moves.

Metóda zabezpečujúca uchovávanie dát o kĺboch:

```
public static void processJoint(String jointName, double jointValueDesired, double jointValueActual) {

        Memory.addToMemory(jointName + SENT, jointValueDesired);
        Memory.addToMemory(jointName + REVEIVED, jointValueActual);

        // is final == true
        if(jointValueActual == jointValueDesired) {
                Memory.addToMemory(jointName + STATUS, true);
        }

        // is moving == false
        else {
                Memory.addToMemory(jointName + STATUS, false);
```

```
            }

            System.out.println(jointName + ": \tfinal angle: " + jointValueDesired +
"\tactual angle: " + jointValueActual + "\tstatus: " +
Memory.getLastObjectFromMemory(jointName + STATUS));
        }
```

Poskytovanie údajov o kĺboch, pre neskoršie použitie pri plánovaní pohybov.

```
        /**
         * Method, which check if joint is stable.
         *
         * @param jointName
         * @return
         * true - joint is stable. <br>
         * false - joint is on the move. <br>
         */
        public static boolean isStable(String jointName) {
            return (Boolean) Memory.getLastObjectFromMemory(jointName + STATUS);
        }

        /**
         * Method, which check if joint is moving.
         *
         * @param jointName
         * @return
         * true - joint is moving. <br>
         * false - joint is stable. <br>
         */
        public static boolean isMoving(String jointName) {
            return ! (Boolean) Memory.getLastObjectFromMemory(jointName + STATUS);
        }
```

2.8.4 Testovanie

2.8.4.1 Odstránenie skriptov z plánovač

Testovanie spočívalo v otestovaní vykonávania pohybov rovnakým spôsobom ako to bolo v prípade vyžívania Ruby skriptu. Tieto boli z projektu vymazané. Vykonanie všetkých pohybov bez využívania skriptu prebehlo v poriadku.

2.8.4.2 Predikcia polohy časti tela agenta

Pri testovaní sme nenarazili na väčšie problémy, ktoré by nejako obmedzili či upravili požadovaný stav výsledného modulu. Odhalili sme niekoľko chýb pri parsovaní dát zo servera, ktoré boli po ich identifikovaní odstránené. Ďalej prišlo ku zmene dátovej štruktúry tabuľky oproti pôvodnej, kvoli efektívnejšiemu behu modulu. Modul bol kompletne otestovaný a je plne funkčný.

2.8.4.3 Predikcia polohy predmetov vo svete

Testovanie prebiehalo pomocou testovacích logov, ktorými bolo otestovanie zistenie viditeľnosti lopty aj počas cyklov bez správ z Vision perceptorov.
Testovanie predikcie predmetov prebiehalo taktiež pomocou testovacích logov a to porovnávaním predikovanej polohy v jednom cykle s posunom 0.06 a skutočnej polohy získanej z správy zo servera. Predikované hodnoty a skutočné hodnoty sa určitých prípadoch odličovali, čo bolo spôsobené chybami vnáranými do správ, ktoré mierne skresľovali hodnotu rýchlosti, čo malo celkovo vplyv na predikované hodnoty. JUnit testy predikcie polohy predmetov boli úspešne vykonané.

2.9 Šprint č. 09

Číslo šprintu	09
Počet príbehov	08
Začiatok šprintu	28.03.2011
Koniec šprintu	11.04.2011

Príbehy

Odstránenie chýb z modelu hráča a sveta a doplnenie funkcii pre vyššie správanie a medzivrstvu plánovača **User story:** žiadne
Vyššie správanie ďalšie pravidlá **User story:** žiadne
Vytvoriť bočné kopy **User story:** žiadne
Vytvoriť kopy nezávislé na postavení hráča **User story:** žiadne
Implementácia vrstvy nad plánovačom **User story:** žiadne
Dokumentácia **User story:** žiadne
Implementacia predikcie polohy časti tela agenta **User story:** žiadne
Úprava a aktualizácia web stránky **User story:** žiadne

2.9.1 Analýza

2.9.1.1 Vytvorenie pohybov a kopov

Pre lepšie zvládnutie herných situácií bolo potrebné vytvoriť niekoľko ďalších pohybov:

- Otočenie vľavo – otáčanie vľavo od autorov JIMa bolo značne pomalé a bolo potrebné vytvoriť rýchlejšiu variantu.
- Otočenie vpravo – otáčanie vpravo od autorov JIMa bolo značne pomalé a bolo potrebné vytvoriť rýchlejšiu variantu.
- Šikmá chôdza doľava – v niektorých prípadoch môže byť zastavenie hráča a jeho následné otočenie zbytočné pomalé. Preto bolo potrebné vytvoriť pohyb, ktorý umožní hráčovi dostať sa k určitému objektu šikmo doprava.
- Šikmá chôdza doprava – v niektorých prípadoch môže byť zastavenie hráča a jeho následné otočenie zbytočné pomalé. Preto bolo potrebné vytvoriť pohyb, ktorý umožní hráčovi dostať sa k určitému objektu šikmo doľava.
- Kopnutie do lopty – kopnutie do lopty nebolo implementované a bolo potrebné ho vytvoriť.
- Kopnutie do ľavej strany – kopnutie lopty do ľavej strany nebolo implementované a bolo potrebné ho vytvoriť.
- Kopnutie do pravej strany – kopnutie lopty do pravej nebolo implementované a bolo potrebné ho vytvoriť.

2.9.1.2 Implementácia vrstvy nad plánovačom

Analýza úlohy bola vykonaná v šprinte číslo 7.

2.9.2 Návrh

2.9.2.1 Vytvorenie pohybov a kopov

Všetky pohyby sme sa snažili implementovať na základe ľudských pohybov. Pri návrhu otáčaní vľavo, vpravo a kopoch sme sa zamerali hlavne na rýchlosť. Pri chôdzi bol kladený dôraz na stabilitu.

2.9.2.2 Implementácia vrstvy nad plánovačom

Návrh úlohy bol vykonaný v šprinte číslo 7.

2.9.3 Implementácia

2.9.3.1 Vytvorenie pohybov a kopov

Okrem otáčaní vľavo a vpravo boli všetky pohyby implementované na základe reálnych pohybov človeka. Pri otáčaniach sme zistili, že hráč vykoná pohyb rýchlejšie, keď nebude musieť ohýbať nohy v kolenách. Pri implementácii šikmej chôdze bola ako základ použitá stabilná chôdza, ktorá bola vytvorená minulý semester. V nej sme museli vytvoriť vychýlenie nohy o určitý uhol a následne doladiť ostatné kĺby, ktoré po tomto vychýlení spôsobovali nestabilitu hráča. Šikmú chôdzu sme implementovali tak, aby sa hráč natočil šikmo doprava/doľava o určitý uhol a následne pokračoval stabilnou chôdzou. Kopnutia sme implementovali tak, aby neboli závislé na presnom postavení hráča od lopty. Vzdialenosť od lopty sa však určitým spôsobom podieľa na sile kopnutia.

2.9.3.2 Implementácia vrstvy nad plánovačom

Bolo implementovaných niekoľko základných tried tejto vrstvy správania, ktoré zabezpečovali úlohy ako:

1. pohyb na pozíciu
2. pohyb k objektu sveta
3. pozretie sa na objekt sveta
4. nájdenie objektu sveta(ak ho hráč nevidí)

2.9.4 Testovanie

2.9.4.1 Vytvorenie pohybov a kopov

Otočenie vľavo + otočenie vpravo

Číslo testu	Výsledok
1	Hráč sa otočil doprava/doľava a nedošlo k pádu
2	Hráč sa otočil doprava/doľava a nedošlo k pádu
3	Hráč sa otočil doprava/doľava a nedošlo k pádu
4	Hráč sa otočil doprava/doľava a nedošlo k pádu
5	Hráč sa otočil doprava/doľava a nedošlo k pádu

Šikmá chôdza doľava + šikmá chôdza doprava

Číslo testu	Výsledok
1	Hráč sa natočil šikmo doprava/doľava a nedošlo k pádu
2	Hráč sa natočil šikmo doprava/doľava a nedošlo k pádu
3	Hráč sa natočil šikmo doprava/doľava a nedošlo k pádu
4	Hráč sa natočil šikmo doprava/doľava a nedošlo k pádu
5	Hráč sa natočil šikmo doprava/doľava a nedošlo k pádu

Kopnutie do lopty

Číslo testu	Výsledok
1	Hráč kopol do lopty a nedošlo k pádu
2	Hráč kopol do lopty a nedošlo k pádu
3	Hráč kopol do lopty a nedošlo k pádu
4	Hráč kopol do lopty a došlo k pádu
5	Hráč kopol do lopty a nedošlo k pádu

Kopnutie do ľavej strany + kopnutie do pravej strany

Číslo testu	Výsledok
1	Hráč kopol do lopty a nedošlo k pádu
2	Hráč kopol do lopty a nedošlo k pádu
3	Hráč kopol do lopty a došlo k pádu
4	Hráč kopol do lopty a nedošlo k pádu
5	Hráč kopol do lopty a došlo k pádu

2.9.4.2 Implementácia vrstvy nad plánovačom

Testovanie prebehlo v poriadku a ukázalo, že všetky úlohy sa vykonávajú správne. Bude však potrebné dodatočné testovanie v spolupráci s vyššou logikou.

2.10 Šprint č. 10

Číslo šprintu	10
Počet príbehov	09
Začiatok šprintu	11.04.2011
Koniec šprintu	02.05.2011

Príbehy

Odstránenie chýb z modelu hráča a sveta a doplnenie funkcii pre vyššie správanie a medzivrstvu plánovača **User story:** žiadne
Vyššie správanie ďalšie pravidlá **User story:** žiadne
Doplnenie ukončovacích fáz pre cyklické pohyby **User story:** žiadne
Drobčivá chôdza **User story:** žiadne
Predikcia pozícií predmetov vo svete – predikcia lopty **User story:** žiadne
Revízia implementácie predikcie polohy časti tela agenta, úprava logovania a overenie zápisov **User story:** žiadne
Aktualizácia stránky **User story:** žiadne
Revízia dokumentácie **User story:** žiadne

Doplnenie parsera pre ostatných hráčov **User story:** žiadne
Oprava plánovača **User story:** žiadne

2.10.1 Analýza

2.10.1.1 Revízia implementácie predikcie polohy časti tela agenta, úprava logovania a overenie zápisov

Cieľom tejto úlohy je úprava aktuálneho kódu pre predikciu častí tela agenta. Pôvodná implementácia neumožňuje konfiguráciu pamäti, ani možnosť vypnutia tvorby logov, čo môže byť pri reálnom nasadení agenta neefektívne. Ďalšou úlohou je kontrola frekvencie zapisovania do pamäte agenta, a teda overenie, či ide o zápis vždy aktuálnych dát, s ktorými pracuje server.

2.10.1.2 Drobčívá chôdza a ďalšie pohyby

Pre lepšie zvládnutie herných situácií bolo potrebné vytvoriť niekoľko ďalších pohybov:

- pohyb doľava – môže slúžiť pre presnejšie nastavanie hráča k lopte.
- pohyb doprava – pohyb môže slúžiť pre presnejšie nastavanie hráča k lopte.
- drobčenie – pohyb môže slúžiť pre presnejšie nastavanie hráča k lopte.

2.10.2 Návrh

2.10.2.1 Predikcia polohy časti tela agenta

Rozhodli sme sa vytvoriť nový balík, ktorý bude obsahovať implementáciu konfigurátora pre pamäť agenta (polohy kĺbov) s možnosťou dynamicky rozšíriť tento konfigurátor. Pre tento účel bude navrhnuté špeciálne rozhranie. Správnosť (aktuálnosť) údajov zapisovaných do pamäte hráča overíme výpisom časovej pečiatky, ktorá by sa pri jednotlivých cykloch mala meniť v intervale 20ms, čo je interval odozvy zo servera. Výstupom týchto modulov by mal byť konfiguračný súbor, virtuálna pamäť hráča a pamäťový súbor, ktorého tvorba bude voliteľná v konfiguračnom súbore.

2.10.2.2 Drobčívá chôdza a ďalšie pohyby

Drobčenie sme sa snažili implementovať na základe reálnych pohybov človeka. Pri jej implementácii sme ako základ použili skoršie vytvorenú stabilnú chôdzu. Pohyby doprava a doľava sme najskôr implementovali na základe ľudského pohybu, kedy hráč ohýbal nohy v kolenách. Tento pohyb sme sa však rozhodli odstrániť z dôvodu rýchlosti vykonania. Ohýbanie kolien znamenalo fázu navyše, ktorá sa musela vykonávať určitý čas. Odstránením tejto fázy sa hráč začal trieť o hraciu plochu, pričom vznikajú menšie straty. Pohyb je však vykonávaný rýchlejšie.

2.10.3 Implementácia

2.10.3.1 Predikcia polohy časti tela agenta

Triedy boli implementované podľa navrhnutého modelu. Nové triedy boli vytvorené v balíčku sk.fiit.jim.agent.configuration. Konfiguračný súbor bol umiestnený do koreňového adresára projektu a momentálne obsahuje tieto dáta:

```
# ------------------------------------------------------------------------------------------
# nastavenie parametrov pre pracu s pamatou

# nastavenie vypisov pamate - nastavenie vypisov na konzolu - ak je hodnota true => bude sa vypisovat na konzolu. ak je false, nebude sa vypisovat na konzolu.
memoryLogConsole: false

# nastavenie vypisov pamate - nastavenie vypisov do suboru - ak je hodnota true => bude sa vypisovat do suboru. ak je false, nebude sa vypisovat do suboru.
memoryLogFile: false

# nastavenie vypisov pamate - nastavenie mena suboru, do ktoreho sa bude zapisovat ak hodnota parametra "memoryLogFile" bude true
memoryPathToFile: C:\\memory.log

# ------------------------------------------------------------------------------------------
# nastavenie parametrov pre pomocne vypisy

# nastavenie pomocnych vypisov - nastavenie vypisov na konzolu - ak je hodnota true => bude sa vypisovat na konzolu. ak je false, nebude sa vypisovat na konzolu.
assistantLogConsole: false

# nastavenie pomocnych vypisov - nastavenie vypisov do suboru - ak je hodnota true => bude sa vypisovat do suboru. ak je false, nebude sa vypisovat do suboru.
assistantLogFile: false
```

```
# nastavenie pomocnych vypisov - nastavenie mena suboru, do ktoreho sa bude zapisovat ak hodnota parametra "assistantLogFile" bude true
assistantPathToFile: C:\\assistant.log

# ----------------------------------------------------------------------------------------------------
# nastavenie sposobu prace s pamatou

# ak je nastavena hodnota all - budu sa pamatat vsetky hodnoty
# ak je nastavena hodnota last - bude sa pamatat iba posledna hodnota
# defaultna hodnota: all
memoryRememberValues: last
# ----------------------------------------------------------------------------------------------------
```

Balík „configuration“ obsahuje triedy pre dynamické definovanie premenných, ako aj ich typovú kontrolu v konfiguračnom súbore, preto je možné tento súbor využiť v rámci celého projektu.

2.10.4 Testovanie

2.10.4.1 Predikcia polohy časti tela agenta

Pri testovaní sme nenarazili na žiadne problémy. Modul bol kompletne otestovaný a je plne funkčný. Testovanie prebehlo na všetkých parametroch konfiguračného súboru úspešne. Otestovaná bola aj frekvencia zápisov do pamäti hráča, pričom sa táto zhodovala s frekvenciou prijímania dát zo servera.

2.10.4.2 Drobčívá chôdza a ďalšie pohyby

Pohyb doľava a doprava

Číslo testu	Výsledok
1	Hráč sa presunul doprava/doľava a nedošlo k pádu
2	Hráč sa presunul doprava/doľava a nedošlo k pádu
3	Hráč sa presunul doprava/doľava a nedošlo k pádu
4	Hráč sa presunul doprava/doľava a nedošlo k pádu
5	Hráč sa presunul doprava/doľava a nedošlo k pádu

Drobčivá chôdza

Číslo testu	Výsledok
1	Hráč sa pohyboval smerom dopredu a nedošlo k pádu
2	Hráč sa pohyboval smerom dopredu a nedošlo k pádu
3	Hráč sa pohyboval smerom dopredu a nedošlo k pádu
4	Hráč sa pohyboval smerom dopredu a nedošlo k pádu
5	Hráč sa pohyboval smerom dopredu a nedošlo k pádu

2.11 Šprint č. 11

Číslo šprintu	11
Počet príbehov	01
Začiatok šprintu	02.05.2011
Koniec šprintu	09.05.2011

Príbehy

Dokončenie dokumentácie, odstránenie prípadných chýb, nedostatkov, ktoré zostali z predchádzajúcich úloh

User story:

Product owner dostane hotový prototyp, ktorý je výsledkom práce tímového projektu v letnom semestri 2011 vrátane dokumentácie.

2.11.1 Analýza

2.11.1.1 Dokončenie dokumentácie

Je potrebné doplniť do projektovej dokumentácie záznam o šprinte 10 a 11. Do riadiacej dokumentácie je potrebné doplniť zápisnice za šprint 10 a 11, záznamy o plánovaní, ktoré boli vykonané počas mesiaca apríl a rovnako aj autorstvo jednotlivých častí.

2.11.2 Testovanie

2.11.2.1 Dokončenie dokumentácie

Jednotliví členovia tímu si prečítali vytvorené dokumentácie, odstránili prípadné chyby (preklepy, pravopis a pod), ktoré sa v dokumentácií objavili.

3 Zhrnutie celkového výsledku v akademickom roku 2010/11

V tejto kapitole uvádzame konkrétne výsledky práce na výslednom produkte, ktorý je predmetom odovzdávania v letnom semestri akademického roku 2010/2011.

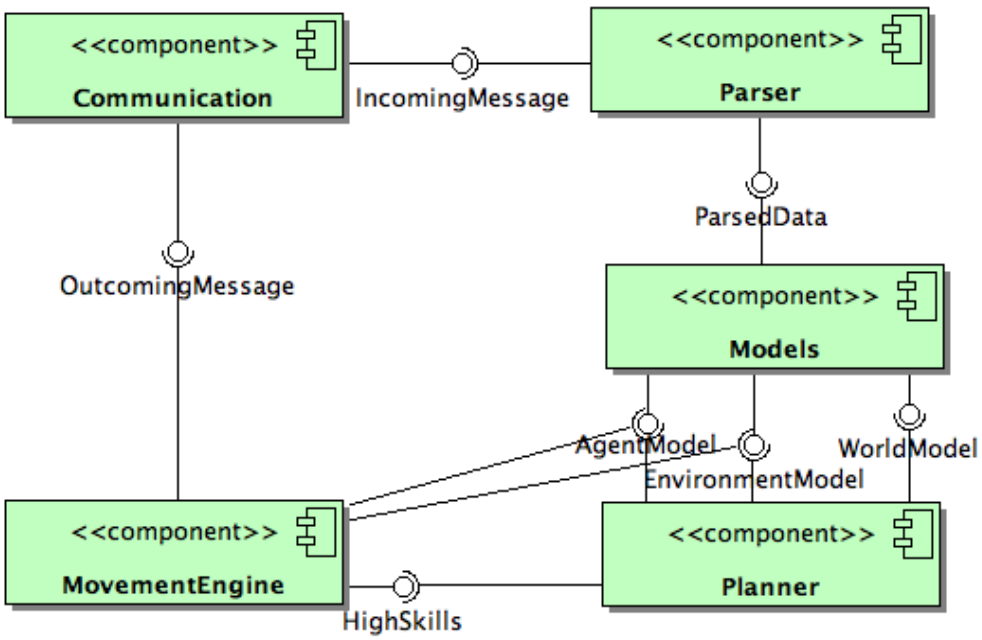
1. Doplnili sme model sveta, predikciu kĺbov hráča a pohybov objektov v modeli sveta
    - Doplnili sme model sveta o spoluhráčov a protihráčov
    - Doplnili sme predikciu pozícií jednotlivých kĺbov hráča
    - Doplnili sme predikciu lopty
2. Doplnili a opravili sme parsovanie správ zo servera
    - Doplnili sme parsovanie spoluhráčov a protihráčov
    - Opravili sme parsovanie akcelerometra a gyroskopu
3. Navrhli a implementovali sme pohyby hráča
    - Pohyby pre vstávanie
    - Pohyby pre otočenie
    - Pohyby pre chôdzu
    - Pohyby pre kopnutie
    - Pohyby pre brankára
4. Navrhli a implementovali sme plánovanie pohybov
    - Odstránili sme plánovanie pomocou skriptov a implementovali vylepšené plánovanie priamo v hráčov
5. Navrhli a implementovali sme správanie hráča
    - Vytvorili sme základný rámec pre vývoj správania hráča
6. Doplnili sme editor pohybov
    - Doplnili sme importovania a exportovania pohybov vo forme XML hráča JIM
7. Integrovali sme parciálne vylepšenia
    - Vytvorili sme hráča, ktorý realizuje individuálne formy správania.

3.1 Výsledný dátový model hráča

Výsledný dátový model hráča JIM je zobrazený na Obr. 3.1. Zdrojové kódy hráča JIM sa skladajú z piatich komponentov, ktoré sú navzájom poprepájané rozhraniami. Dodržiava sa tak pravidlo slabej 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.



Obrázok 3.1. Dátový model hráča.

3.2 Výsledný dátový model správania

Základný komponent reprezentujúci správanie hráča (obrázok 3.2-1). Charakterizuje individuálne správanie hráča, ktoré sa využíva v tímovom správaní (hráči pri hre spolupracujú). Správanie sa rozhoduje na základe bázy znalostí, ktorá obsahuje možných akcie. Akcie, ktoré sa vykonajú sú naplánované a vložené do plánovača, ktorého úlohou je správne usporiadanie akcií, zmena ich poradia, zrušenie niektorej akcie, podľa pokynov z individuálneho a tímového správania hráča. Vstupmi pre rozhodovanie sú informácie o svete z komponentu Model.

Pri návrhu správania vychádzame zo štyroch základných princípov:

1. agent musí byť schopný dokončiť svoju úlohu,
2. agent sa musí vedieť rozhodovať na základe aktuálnej situácie,
3. rôzne typy správania musia byť rozdelené do skupín tak, aby si navzájom neprekážali,
4. niektoré typy správania sú použiteľné iba ak sú súčasťou skupiny, alebo sú použité až po vykonaní iných typov.

Trieda *Behaviour* má nasledovné metódy:

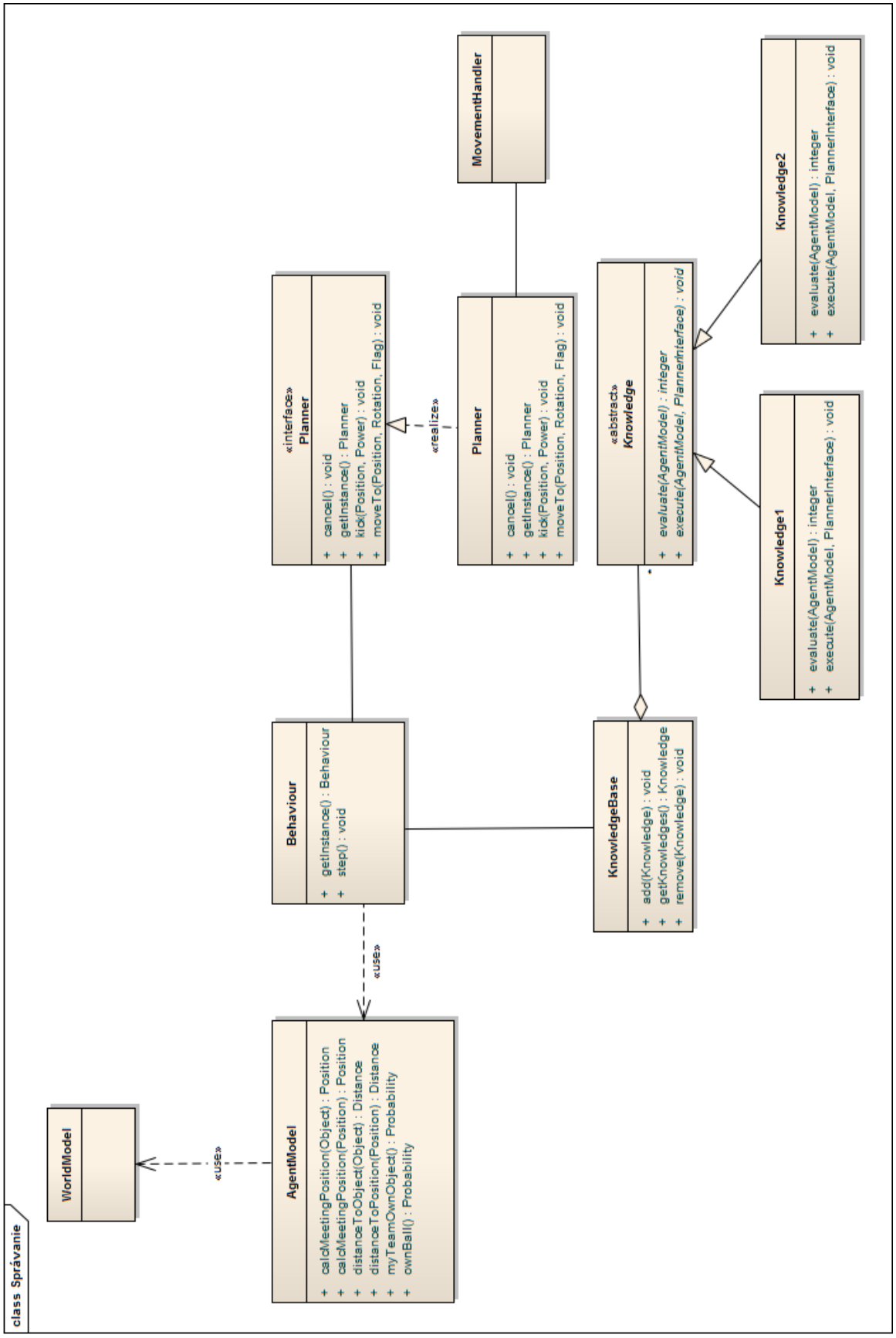
- getInstance() - vráti inštanciu triedy Behaviour (návrhový vzor Singleton),
- step() – jeden krok v správaní, vyhodnotí jednotlivé znalosti a vyberie tú najlepšiu

Trieda *Planner* má nasledovné metódy:

- getInstance() - vráti inštanciu triedy Behaviour (návrhový vzor Singleton),
- moveTo(position, rotation, flag) - naplánuje posunutie hráča na zadanú pozíciu s určenou rotáciou,
- kick(position, power) - naplánuje kop na danú pozíciu s danou silou.,
- cancel() - zruší aktuálny pohyb.

Do triedy *AgentModel* sme pridali funkcie pre analýzu modelu sveta:

- ownBall() - vracia true/false, či sa hráč nachádza v dostatočnej blízkosti lopty,
- ownMyTeamBall() - vracia true/false, či hráčov tím má loptu,
- distanceToObject(object) - vzdialenosť k bráne, lopte alebo inému objektu,
- distanceToPosition(position) - vzdialenosť k danej pozícií,
- calcMeetingPosition(object) - vypočíta pozíciu stretnutia hráča s objektom (lopta, iný hráč...),
- calcMeetingPosition(object) - vypočíta pozíciu stretnutia s danou pozíciou



Obrázok 3.2-1. Dátový model modlu správania.

4 Spustenie hráča a editora pohybov

4.1 Hráč

Spustenie hráča sa skladá z troch krokov:

1. Nainštalovanie servera
2. Spustenie hráča
3. Spustenie serveru pre simuláciu

Jednotlivé kroky spustenia hráča sú popísané v podkapitolách 4.1.1.1, 4.1.1.2 a 4.1.1.3.

4.1.1 Nainštalovanie servera a stiahnutie hráča

1. Najskôr je potrebné nainštalovať Microsoft Visual C++ 2008 Redistributable Package (x86) http://robocup.isvet.sk/data/vcredist_x86.exe
2. Simulačný server a server pre robocup v krokoch 3. a 4. Je potrebné nainštalovať na jeden disk
3. Potom nainštalujte simulačný server stiahnuteľný http://robocup.isvet.sk/data/simspark-0.2-win32.exe
4. Samotný inštalačný balík serveru pre RoboCup 3D verzie 0.6.3, ktorý inštalujte ako posledný http://robocup.isvet.sk/data/rcssserver3d-0.6.3-win32.exe
5. Následne je potrebné si stiahnuť hráča JIM z repozitára https://bitbucket.org/xpagaca/team-04/get/tip.zip alebo z priloženého CD v priečinku *Zdrojove kody/JIM* a importovať do Eclipsu ako projekt

4.1.2 Spustenie hráča

1. V súbore settings.rb skontrolujte adresu servera (mala by byť 127.0.0.1) riadok: Communication.instance.server_ip = "127.0.0.1", súbor sa nachádza v adresári scripts/config/
2. spustite server otvorením c:\Program Files\rcssserver3d 0.6.3\bin\rcssserver3d.cmd

3. spustite monitor otvorením c:\Program Files\rcssserver3d 0.6.3\bin\rcssmonitor3d.cmd
4. spustite projekt, o chvíľu by ste mali vidieť ihrisko s hráčom a loptou

4.1.3 Spustenie serveru pre simuláciu

V súbore C:\Program Files\rcssserver3d 0.6.3\share\rcssserver3d\simspark.rb upravte riadok $enableInternalMonitor = false na $enableInternalMonitor = true;
Pozn. Pod operačným systémom windows 7 je potrebné tento súbor skopírovať na iné miesto ako inštalačný adresár, upraviť a následne nakopírovať naspať.

4.2 Editor pohybov

Editor pohybov sa spúšťa priamo prostredníctvom spustenia súbor RobotBehaviourEditor.exe v priečinku:

*Zdrojove kody**Editor pohybov*\ *robotBehaviorEditor-Release*\

V prípade editácie zdrojových kódov je potrebné otvoriť súbor *RobotBehaviourEditor.suo* vo Visual Studiu 2008 alebo novšom, ktorý sa nachádza v priečinku:

*Zdrojove kody**Editor pohybov**robotBehaviourEditor-Source*

5 Zdroje

Táto kapitola obsahuje zoznam zdrojov použitých pri vypracovaní projektu.

[1] Gelányi, Ľ., a kol.: *Projektová dokumentácia tímu 17*, STU, FIIT, (2009).
dostupné elektronicky:
http://labss2.fiit.stuba.sk/TeamProject/2009/team17is-si/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_robocup2009_Symposium/tdps/3d-simulation/3d_simulation_Apollo3D.pdf

[3] Stránka tímu Apollo http://robocup.njupt.edu.cn/

[4] Stránka tímu Nao-Team Humboldt http://www.naoteamhumboldt.de/

[5] Stránka tímu Nao-Team HTWK 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] Jahangiri, S., Khalifeh, T., M., et al.: *Polytechnic-Parsian German Open 2010 Team Description Paper*, Amirkabir University of Technology, Tehran, Iran (2010).
dostupné elektronicky:
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/robocup/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-offenburg.de/prof/kdorer/robocup/magmaOffenburg/downloads/magmaOffenburg_TDP2009.pdf

[10] Little Green Bats: *RoboCup 3D Simulation HowTo,* (2009).
dostupné elektronicky:
http://sourceforge.net/projects/littlegreenbats/files/Documentation/Howtorobocop/littlegreenbats_howtorobocop-0.1.tar.gz/download

[11] Team UIAI, UI-AI3D 2007 Team Description, (2007).
dostupné elektronicky:
www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/uiai2007TDP.pdf

[12] Rotation matrix
dostupné elektronicky:
http://en.wikipedia.org/wiki/Rotation_matrix