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

Tímový projekt - Robocup 3D

Dokumentácia k inžinierskemu dielu

Študijný odbor: Softvérové inžinierstvo, Informačné systémy
Predmet: Tímový projekt
Vedúci projektu: Ing. Ivan Kapustík
Tím: High5 (tím číslo 5)
Členovia tímu: Baranček Karol, Bc.
Bimbo Miroslav, Bc.
Boleček Tomáš, Bc.
Jurčák Ondrej, Bc.
Sedláček Andrej, Bc.
Šimko Ivan, Bc.

Obsah

1 Šprint 1

1.1 Analýza slovenských robocup projektov

1.1.1 Hviezdna jedenástka

Tím Hviezdna Jedenástka[1] pôsobil na Fakulte informatiky a informačných technológií v ročníku 2007 / 2008. Tento tím sa ako prvý na fakulte zaoberal 3D futbalom humanoidného typu, hoci v tej dobe nemali ani oficiálnu dokumentáciu. Hlavnou príčinou tohto výberu bolo pravdepodobné ukončenie vývoja iných typov k dohľadnej budúcnosti.

Použité technológie

Tím použil pre implementáciu hráča jazyk C++. Svoje rozhodnutie odôvodnil tým, že je v ňom jednak napísaný kód simulačného servera, jednak všetky analyzované tímy.

Jednotliví členovia tímu boli zvyknutí pracovať v rôznych vývojových prostrediach, pod rôznymi operačnými systémami. Vytvárali preto svoj projekt pre čo najväčšiu prenositeľnosť – oddelili zdrojové kódy od súborov súvisiacich s vývojovým prostredím a používali iba štandardné knižnice dostupné pre všetky operačné systémy.

Z dôvodu prenositeľnosti boli použité pre paralelizáciu vlákna POSIX Threads, ktoré sú štandardizované pre všetky bežné operačné systémy.

Architektúra hráča

Ako je vidieť na obrázku 1, architektúra hráča sa skladá z viacerých modulov:

- Hlavný cyklus: *Agent*
- Model hráča – informácie o hráčovi: *PlayerModel*
- Model sveta – informácie o okolitých objektoch: *WorldModel*
- Správanie a schopnosti agenta: *HighSkill, MiddleSkill, LowSkill*
- Posielanie a získavanie informácií zo sprav posielaných serverom: *Communication, Parser*

Agent

Obsahuje hlavný cyklus (v 20ms intervale): čítanie správ -> aktualizácia modelu sveta -> rozhodovanie -> posielanie správ späť na server.

PlayerModel

Obsahuje čas simulácie, informácie o hráčovi, jeho kĺboch, gyroskope a sile pôsobiacej na chodidlá. Vychádza z modelu soccerbot056, ktorý má 14 kĺbov, z ktorých sú dva typy:

- Hinge s jednuou osou otáčania
- Universal s dvomi osami otáčania

Každý kĺb má dve metódy na otočenie:

- metóda moveTo – posunúť kĺb na daný uhol
- metóda moveBy – relatívne posunutie kĺbu o daný uhol

1 http://labss2.fiit.stuba.sk/TeamProject/2007/team11is-si/

A dva spôsoby otáčania

- CONSTANT – kĺb sa počas celého pohybu otáča konštantnou rýchlosťou
- APPROXIMATING – kĺb spomaľuje rýchlosť otáčania s približovaním sa ku koncovému uhlu

WorldModel

Táto trieda uchováva informácie o všetkých objektoch, ktoré je hráč schopný vidieť a pomocou ktorých by sa mal vedieť rozhodovať - informácie o hracej lopte, o pozícií rohových zástavok a bránok na ihrisku, simulačný čas, čas hry (od výkopu) a mód hry (voľná hra, štandardná situácia, atď). Každý objekt má určenú polohu a čas, kedy bol videný naposledy. Dynamické objekty (napr. lopta) majú aj atribúty zrýchlenia, hmotnosti a rýchlosti – pre budúce možnosti predikcie situácií.

Zaujímavosťami bolo, že

- bránky boli reprezentované iba dvomi tyčkami (horná žrď nebola braná v úvahu)
- reprezentácia polohy objektov bola udávaná dvomi spôsobmi
  - sférickými súradnicami – (dva uhly a jedna vzdialenosť dvoch bodov)
  - karteziánskymi súradnicami (tri usporiadané body)

Skills

Zručnosti hráča sú obsiahnuté v troch úrovniach:

- low skill – kop do lopty, otočenie sa, krok, ...
- middle skill – chôdza, beh, otočenie sa k lopte, ...
- high skill – útočenie, …

Vyššie zručnosti sú zložené z nižších zručností. Ich plánovanie je prioritizované podľa daných úrovní – zručnosť vyššej úrovne čaká, kým je ukončená zručnosť nižšej úrovne.

Impelementované boli nasledovné zručnosti:

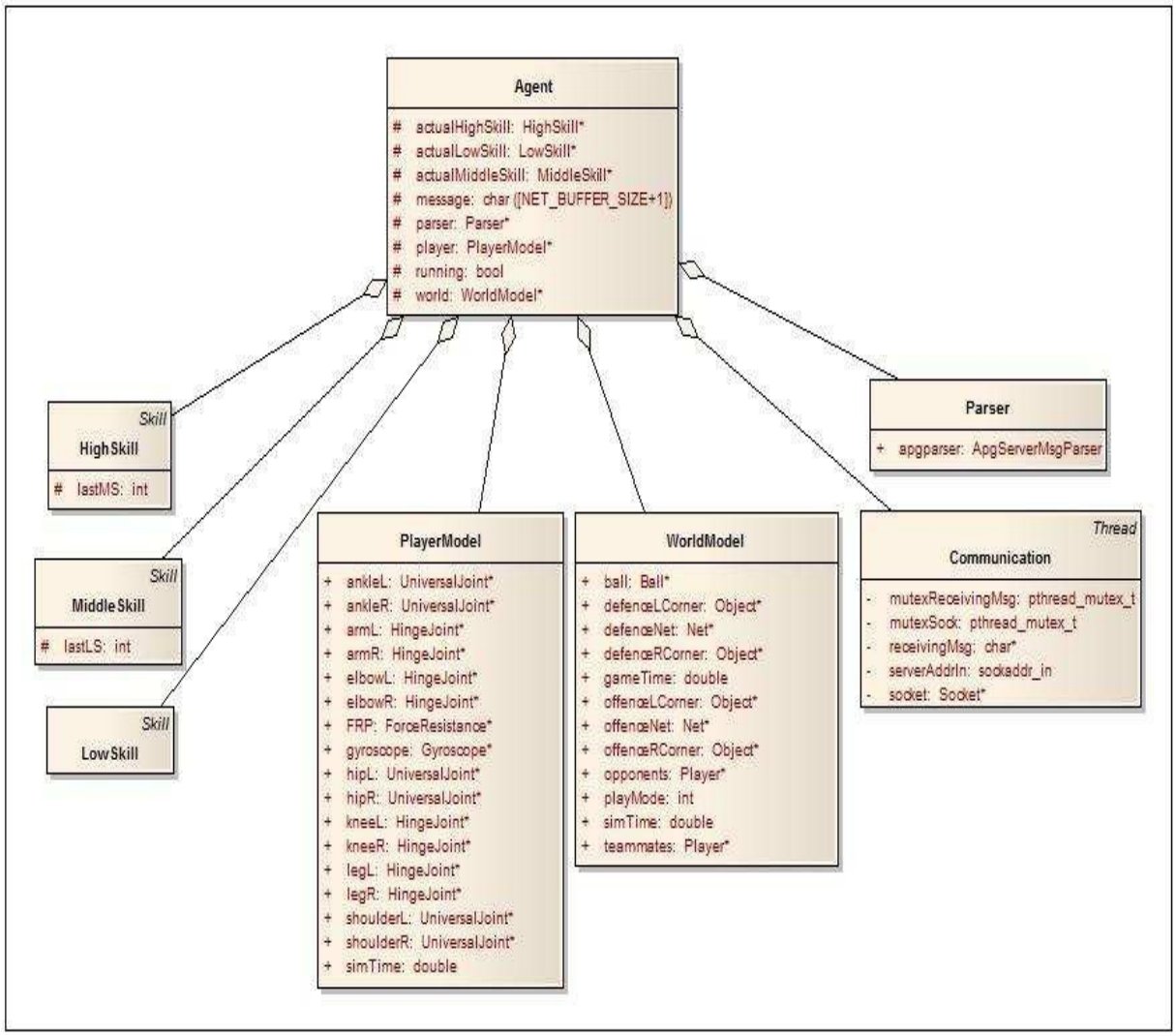
- vstávanie zo zeme
- chôdza
- otáčanie sa na mieste

Communication

Tento modul má na starosti komunikáciu so serverom, teda prijímanie a odosielanie správ obsahujúcich s-výrazy. V rámci tohto modulu boli implementované dva spôsoby komunikácie – TCP z dôvodu požiadavky na stabilitu komunikácie a UDP z dôvodu rýchlosti komunikácie. Samotný výber spôsobu komunikácie je tak na používateľovi.

Parser

Slúži na parsovanie prijatej správy od servera. Pri vytváraní tohto modulu bol použitý open-source program s názvom APG – ten je schopný na základe vloženej gramatiky vygenerovať parser – syntaktický strom s definovanými akciami.



Obrázok 1: Architektúra hráča tímu Hviezdna jedenástka

Zhodnotenie

Vzhľadom na to, že sa jednalo o prvý projekt na fakulte zaoberajúci sa humanoidným typom 3D Robocupu, ťažiskom práce bol návrh dobrej architektúry, obsiahnutie všetkých jej základných modulov v implementácii, jej prenositeľnosť a vytvorenie základných pohybov ako overenie funkčnosti hráča.

Hra zápasu pomocou tohto hráča nebola možná, keď ten nemal implementované mnohé základné pohyby, vnímanie iných hráčov a pod. Z pochopiteľných dôvodov ďalšími neriešenými problémami boli dynamické pohyby, podporné prostriedky, strojové učenie, rozhodovanie na vyššej úrovni.

Je pomerne ťažké určiť prínosy tejto práce vzhľadom na náš projekt, keďže ten vychádza z pokročilejšieho hráča v inom programovacom jazyku. Problémami vyriešenými týmto tímom sa už nie je potrebné ďalej zaoberať, neriešené problémy sú práve tie, na ktorých budeme v priebehu semestrov pracovať. Dokumentácia projektu je však celkom zrozumiteľne popísaná a tak môže pomôcť pri počiatočnej analýze problematiky a pochopeniu základnej idei architektúr hráčov.

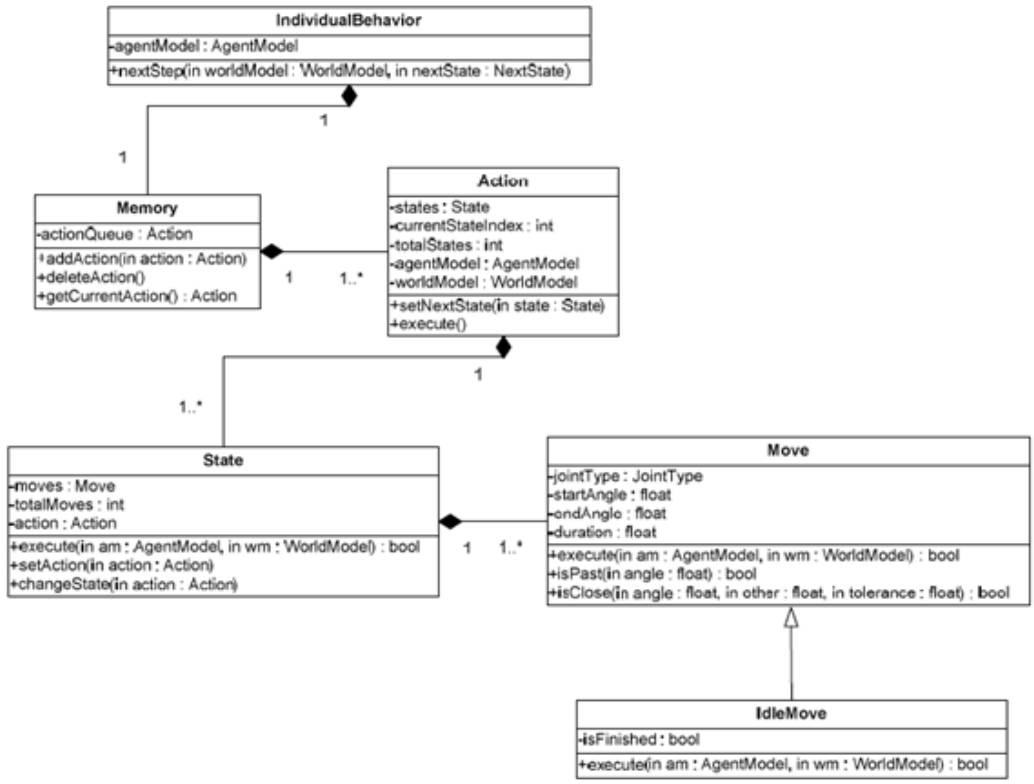
1.1.2 Agenty 007[2]

Je to tím, ktorý pracoval na RoboCupe v rámci tímového projektu na FIIT v akademickom roku 2008/2009. Vychádzali z tímových projektov z predchádzajúceho roka, od ktorého prevzali hlavne skompilované simulačné prostredie simspark, a tiež základ hráča. Inšpirovali sa zahraničným tímom Little Green bats, od ktoré prebrali štruktúru súborov, v ktorých sú uložené definované pohyby kĺbov hráča. Zamerali sa hlavne na tieto časti:

- Editor pohybov
- Stabilizačný modulu na udržiavanie rovnováhy hráča
- Vytvorenie pohybov pomocou editora
- Zachytávanie pohybov zo servera

Architektúra agenta

Aby bolo možné ľahko meniť správanie sa agenta, navrhli novú architektúru agenta (Obrázok 2). Pre väčšiu flexibilitu agenta sú pohyby načítané zo špeciálneho súboru. Správanie agenta pozostáva z akcií. Každá akcia pozostáva z podmienky, ktorá musí platiť aby sa akcia spustila a z kontinuálne nasledujúcich stavov. Samotné vykonávanie jednotlivých akcií riadi pamäť a taktiež je tu možnosť riadenia akcií pomocou vyššej logiky správania (plán do budúcnosti).

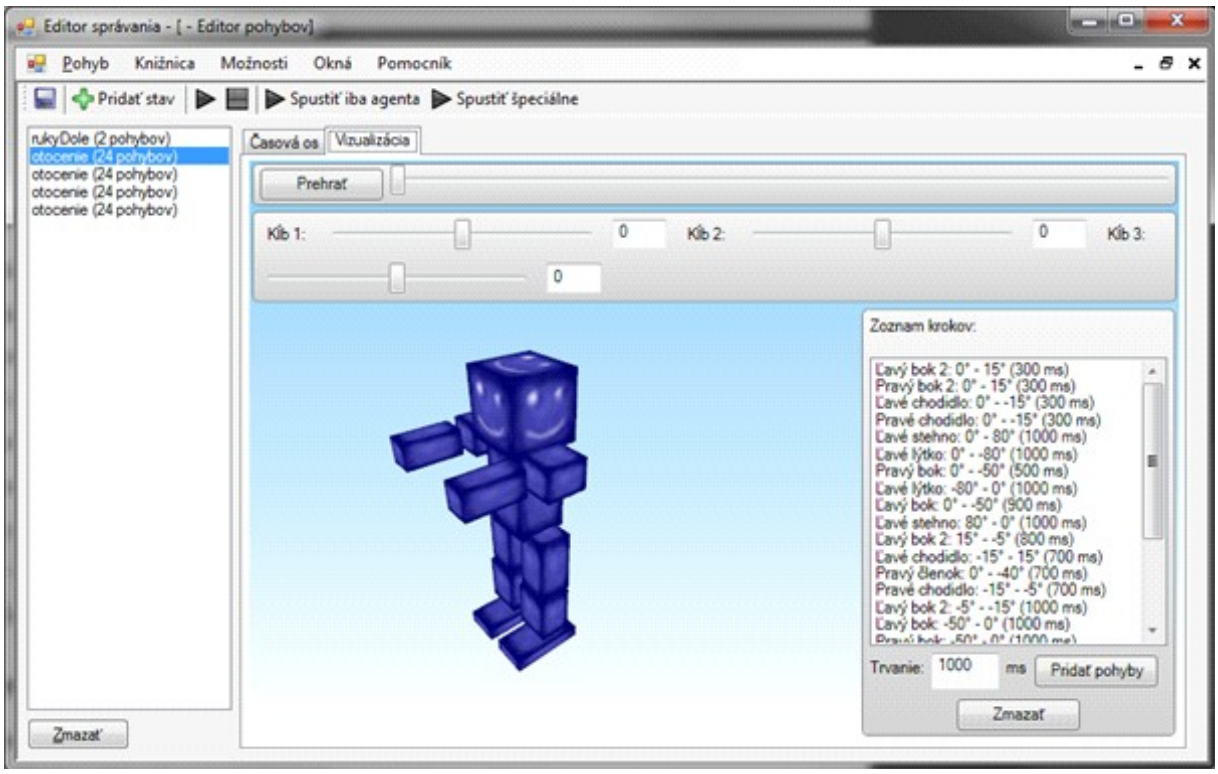


Obrázok 2: Diagram tried týkajúci sa pohybov agenta

2 Zdroje: http://labss2.fiit.stuba.sk/TeamProject/2008/team07is-si/

Editor pohybov

Editor pohybov je nástroj na vytváranie pohybov. Pohyby je možné ukladať do špeciálneho súboru, alebo je možné odtiaľ spustiť server aj s agentom, pričom agent vykonáva modelované pohyby. Je možné priamo importovať súbor s pohybmi alebo vytvoriť tieto pohyby priamo v editore. Pri vytváraní pohybov je možné vidieť časovú os, na ktorej je zobrazená časová následnosť elementárnych pohybov kĺbov alebo je možné celú sériu pohybov prehrať priamo v editore. Sú dva spôsoby zadávania pohybov a to nastaviť priamo parametre pre príslušný kĺb alebo priamo na modeli robota nastaviť kĺby a ich natočenia (Obrázok 3).



Obrázok 3: Vytváranie pohybov pomocou editora

Message Parser

Na zachytávanie pohybov hráča zo servera vytvorili mechanizmus, ktorý implementovali v agentovi. Tento mechanizmus umožňuje zaznamenávanie pohybu robota na serveri, v podobe vybraných správ, pomocou ktorých agent komunikoval so serverom. V editore sú potom tieto správy parsované a sú z nich vytvorené príslušné pohyby, ktoré je možné v editore zobraziť a editovať.

Rovnovážny modul

Agenti007 vytvorili prototyp rovnovážneho modulu, ktorý vychádza z pozícií jednotlivých častí tela a vypočítava celkové ťažisko hráča. Z oporných bodov(najnižšie- na zemi) agenta vypočítajú tzv. plochu stability. Pokiaľ sa ťažisko nachádza nad touto plochou je agent stabilný, pokiaľ je pod hrozí pád. Tento modul nebol reálne použitý priamo v agentovi ale mohol by byť používaný ako pomôcka pri vytváraní pohybov.

Ďalšie plány

Plánom do budúcna bolo vytvorenie vyššej vrstvy riadenie, na vytvorenie ktorej však bolo v ich prípade potrebná bohatá knižnica pohybov, s ktorými by táto vrstva pracovala. Táto vrstva by používala pravidlový systém na riadenie agenta. Pravidlá by mali byť rovnako ako pri pohyboch zapísané v konfiguračných súboroch, s možnosťou ich vytvorenia a editovania pomocou editora.

Zhrnutie

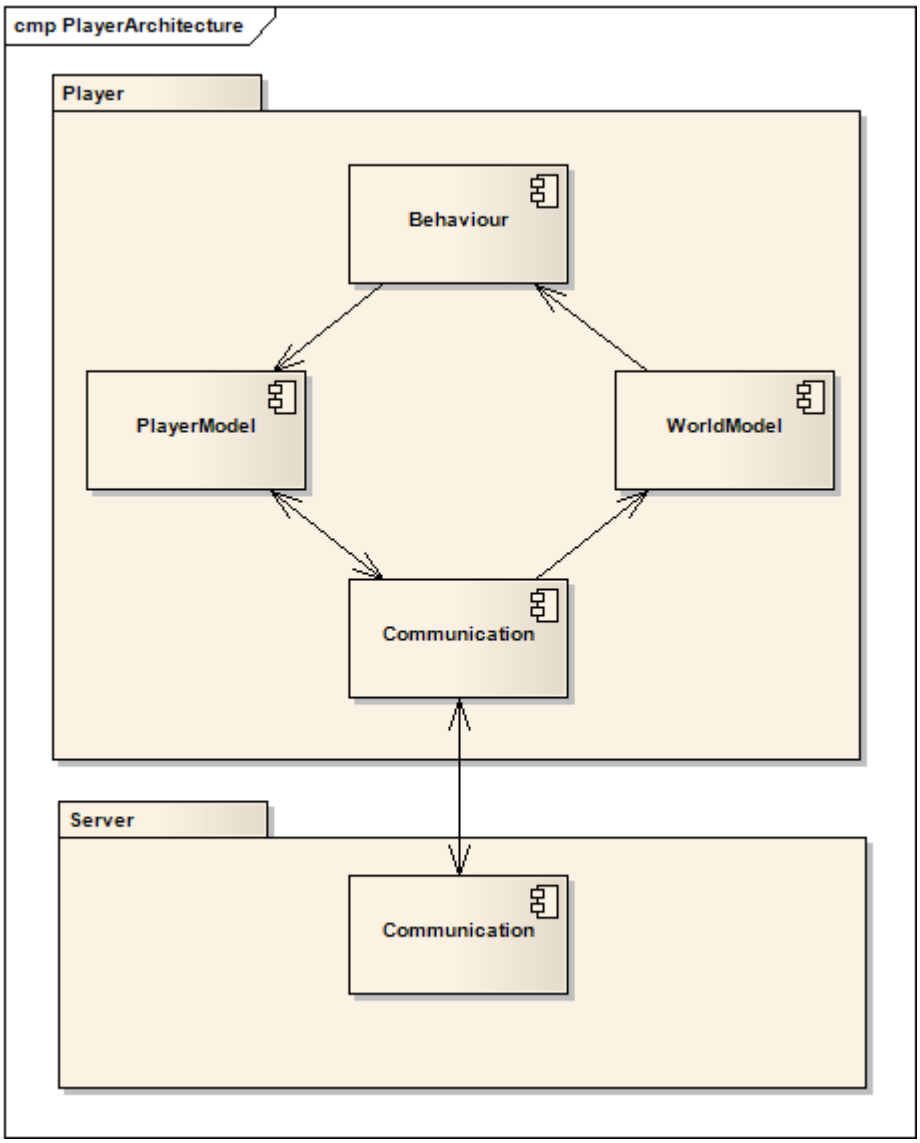
Pri agentoch 007 ide o začiatky čo sa týka 3d robotického futbalu. Editor pohybov je výborná pomôcka na vytváranie pohybov, má však ešte svoje nedostatky. Nie je tam implementovaná možnosť tvorby preconditions pre akcie. Taktiež stabilizačný modul je len v začiatkoch a bolo by vhodné jeho použitie buď priamo v agentovi alebo prinajmenšom pri tvorbe pohybov v editore. Agent je navrhnutý takým spôsobom aby ho bolo možné rozšíriť o riadiacu vrstvu, ktorá by vyberala pohyby.

1.1.3 Dream team

Dream team je univerzitný tím Fakulty informatiky a informačných technológií STU, ktorý vznikol vďaka predmetu Tímový projekt v roku 2008/2009. Hráč tohto tímu vychádza z predošlého hráča tímu Hviezdna Jedenástka.

Architektúra hráča

Pri vytváraní architektúry hráča vychádzali z predošlých implementácií hráčov. Architektonicky je hráč tímu rozdelený na štyri celky, znázornené na obrázku 4.



Obrázok 4: Architektúra hráča Dream Team

Communication

Hráč komunikuje so simulačným serverom prostredníctvom modulu *Communication.* Žiadne iné moduly so serverom nekomunikujú. Pri prijatí správy je začatý nový cyklus iterácie hráča. Dôležitou súčasnou modulu je parser, ktorý prekladá správy zo simulačného servera a vytvára správy pre na odoslanie.

WorldModel

Modul obsahuje všetky informácie o najaktuálnejšom stave okolitého sveta . Na základe týchto informácií sa môže hráč ľahšie rozhodnúť akú akciu vykonať. Model zahŕňa informácie o polohe lopty, bránky, rohov ihriska, spoluhráčov a protihráčov.

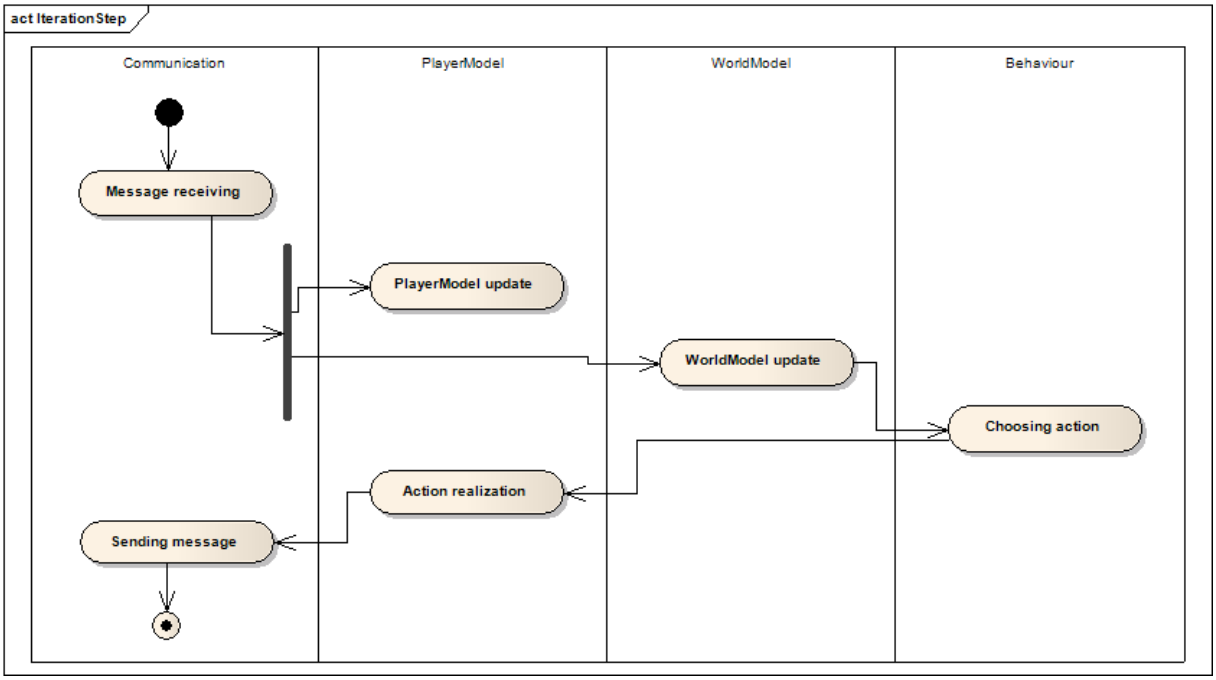
Behaviour

Komponent využíva informácie v predchádzajúcom module na rozhodnutie akej akcie vykonať. Pri rozhodovaní je vytvorená inštancia objektu opisujúceho hráčov pohľad na svet (Situation). Následne je tento objekt posúvaný rôznym objektom, ktoré predstavujú akcie robota. Každá akcia sa na základe situácie ohodnotí, čím udá svoju vhodnosť na vykonanie za daných podmienok okolitého prostredia. Rozhodovací modul (Dispatcher) vyberie akciu s najlepšou prioritou a zabezpečí jej vykonanie.

PlayerModel

Predstavuje aktuálneho hráča. Konkrétne obsahuje informácie o stave všetkých jeho kĺboch a vykonávaných akciách (rýchlostiach). Svoju funkcionalitu sprístupňuje práve modulu Behaviour a jeho hlavným výsledkom je posielanie správ modulu Communication na komunikáciu so simulačným serverom.

Celkové správne fungovanie hráča je založené na kooperácii znázornených modulov v cykle. Tento cyklus sa začína prijatím správy zo strany simulačného servera, pokračuje upravením modelu sveta, modelu hráča, výberom akcie a jej realizovaniu, teda vyslaním správy na simulačný server. Jedna iterácia opisovaného cyklu je znázornená na nasledujúcom obrázku (Obrázok 5).



Obrázok 5: Cyklus pracovania hráča Dream Team

Vykonávanie akcií so spätnou väzbou

Jedným z hlavných bodov implementácie tohto tímu bolo dynamické vykonávanie činností, konkrétne časti chôdze. Na základe zahrnutia modelu hráča do výberu akcie je možné porovnávať očakávaný stav so stavom reálnym a na základe tohto porovnania zistiť prípadné riziko pádu robota. Pri nastatí tejto situácie je možné naplánovať vykonanie inej akcie a teda predísť pádu robota.

Každá akcia (High skill) je rozdelená na viacero menších akcií (Low skill), ktoré obsahujú sériu postupne sa vykonávajúcich fáz. Každá fáza obsahuje informáciu ďalšej fáze na vykonanie ako aj informáciu o takzvanej „save“ akcie, teda akcie ktorú je potrebné vykonať v prípade neočakávaného stavu hráča (začína padať).

Opis schopností (vykonateľných akcií) pomocou XML

Tím umožňuje definíciu vykonateľných akcií robota vo formáte XML. Tento spôsob definovania umožňuje veľmi rýchle upravovanie a ladenie akcií robota. Vyššie (High Skills) a nižšie (Low skills) sú opisované zvlášť, pričom sa na seba odkazujú. Nižšie schopnosť sa ďalej odkazujú na jednotlivé fázy opisujúce presné pohyby kĺbov. Ukážka definovania aktivity pomocou súboru XML je znázornená na obrázku 6. K jednotlivým akciám definovaným týmto spôsobom je taktiež nutné vytvoriť zodpovedajúce triedy v jazyku Java, ktoré implementujú jednotlivé metódy na uskutočnenie pohybu.

Celkovo bol :

```
<robot>
  <high_skills>
    <high_skill name="walk_to_ball">
      <use_low_skill skill="walking"/>
    </high_skill>
  </high_skills>

  <low_skills>
    <low_skill name="walking">
      <initial_phase name="chodzapriprava"/>
    </low_skill>
  </low_skills>

  <phases>
  <phase name="chodzapriprava" next="wageRight">
    <efectors>
      <efector name="lle2">
        <start>0</start>
        <end>-5</end>
      </efector>
      <efector name="rle2">
        <start>0</start>
        <end>5</end>
      </efector>
    </efectors>
    <finalization_phase>ROLLBACK</finalization_phase>
    <rescue_movement>PROCEED</rescue_movement>
    <speed_constant>1</speed_constant>
  </phase>
  </phases>
</robot>
```

Obrázok 6: Dream Team definovanie akcií pomocou XML

1. Mávanie rukami - dva spôsoby
2. Otočenie o 180 stupňov
3. Chodenie dopredu – dva spôsoby
4. Robenie drepov
5. Chodenie doľava

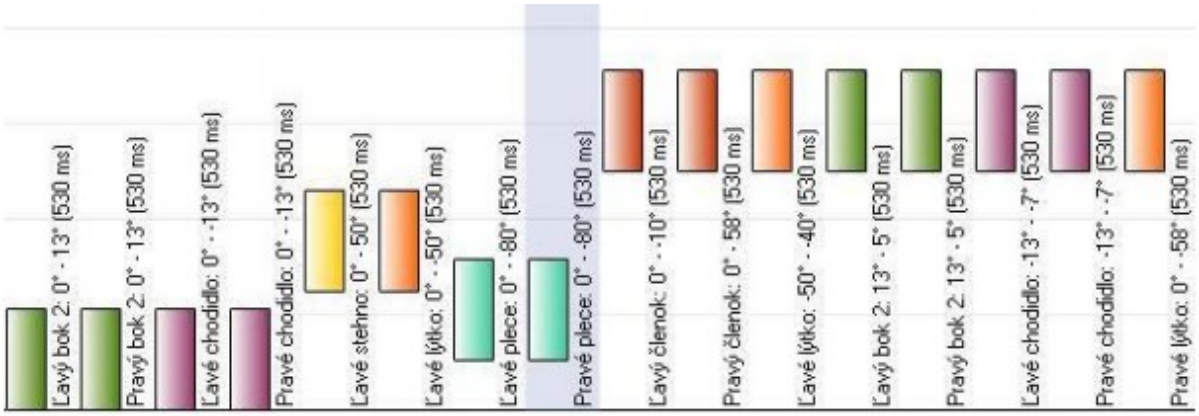
1.1.4 RoboKopy

Tím RoboKopy je univerzitný tím z Fakulty informatiky a informačných technológií, ktorý vznikol v roku 2009 vďaka predmetu Tímový projekt. Pri práci na projekte vychádzali najmä z výstupov tímu Agenty007, ktorý v roku 2009 vyhral TP Cup.

Tvorba základných a špeciálnych pohybov

Jednou z hlavných úloh tímu bolo vytvorenie základných pohybov robota, ako napríklad chôdza, otáčanie, vstávanie a kopnutie do lopty. Vo všeobecnosti vychádzali najmä z reálnych pohybov človeka a väčší dôraz bol kladený na stabilitu hráča pri vykonávaní pohybov ako na rýchlosť ich vykonania. Najmä preto, že vo vtedajšej verzii hráča nebola implementovaná fyzická stabilizácia. Vo veľkej miere bol použitý editor pohybov, ktorý vytvoril tím Agenty007.

Pri realizácii chôdze bolo nutné zapojiť množstvo kĺbov z dolnej časti hráča. Pre zabezpečenie vyššej stability boli zapojené aj pohyby rúk. Pri testovaní sa zistilo, že agent zvláda chôdzu lepšie na počítačoch s vyššou konfiguráciou. Na nasledujúcom obrázku (Obrázok 7) je znázornené vykročenie ľavou nohou.



Obrázok 7: Vykročenie ľavou nohou

Pohyb kopu do lopty vychádza zo samotného pohybu chôdze upraveného tak, aby bola lopta zasiahnutá s čo najväčšou hybnosťou. Vytvorený pohyb fungoval podľa očakávaní, avšak obsahoval citeľné nedostatky v podobe rýchlosti vykonania pohybu a veľkosti hybnej sily.

Taktiež boli vytvorené špeciálne pohyby, ktoré slúžia iba exhibičné účely. Pohyb v bok a kývanie rukami boli značne jednoduché. Väčšia pozornosť sa venovala klikom, ktoré pozostávali z viacerých pohybov.

Úprava kódu prevzatého projektu

Po analýze zdrojového kódu predošlého tímu sa členovia tímu rozhodli, že poopravujú viaceré chyby a niektoré časti projektu prepíšu do flexibilnejšej podoby. Pozmenené bolo logovacie API, testovacie API a matematické API. Ďalšie úpravy sa týkali opravy chýb v kóde a nevhodných použití programátorských techník. Taktiež došlo k zmenám pri používaní niektorých knižníc.

Stredná logika hráča

Nevyhnutnou schopnosťou hráčov pri hre je skladanie elementárnych pohybov tak, aby bolo možné uplatňovať takzvanú strednú logiku. Tá je reprezentovaná abstraktným stavovým automatom. Rozhodovanie sa následne riadi zisťovaním aktuálneho stavu, prechod do iného stavu a pohybov,

ktoré sa v nich majú vykonať. Prechod medzi stavmi vyžaduje aj prechodovú funkciu, ktorá je realizovaná pomocou rozhodovacieho stromu. Pre potreby strednej logiky boli tiež z elementárnych pohybov vytvorených niekoľko zložených pohybov. V ďalších fázach projektu bol napríklad vytvorený pohyb chôdze s postavením. To umožnilo plynulú sekvenciu pohybov, pretože ak hráč pri niektorom z nich spadol, jednoducho sa postavil a pokračoval.

Tak ako elementárne pohyby, aj stredná logika sa nachádza v textovom súbore. Jej tvorba alebo úprava bola teda prácna a neefektívna. Tím sa preto rozhodol doplniť editor pohybov o editor strednej logiky. Editor umožňuje vytvárať strednú logiku hráča alebo upravovať už existujúcu logiku.

Taktiež boli vytvorené vyhodnocovacie funkcie, ktoré poskytujú modulu strednej logiky podporné prostriedky pre rozhodovanie. Agent totiž potrebuje vedieť údaje o svete, ktoré mu tieto funkcie poskytovali. Využívajú tiež vnútornú reprezentáciu agenta o stave na ihrisku a všetky informácie z perceptorov. Dokážu preto poskytovať informácie o vzdialenosti od bránok, stabilite a polohe hráča, či informácii o možnosti streľby na bránu.

Ďalšou dôležitou funkciou bolo umožnenie výpočtu hodnôt nezávisle od hráčovej domácej polovici ihriska. Výpočty boli totiž realizované za predpokladu, že domáca polovica sa vždy nachádza naľavo. Nová funkcia indikovala, ktorá polovica hracej plochy je domáca a ktorá súperova.

Elementárna logika

Jednou z ďalších úloh tohto tímu bol návrh elementárnej logiky hráča, ktorá je dôležitým krokom k samotnej hre futbalu. Stavový automat bol týmto návrhom doplnený o subautomaty, ktoré sa nachádzali v jednotlivých stavoch. Umožní to v akomkoľvek stave vykonať nielen jeden pohyb ale celú skupinu pohybov. Medzi stavmi v subautomate sú taktiež definované prechodové funkcie. Takáto elementárna logika umožňuje hráčovi prísť k lopte, viesť ju smerom k bráne a vystreliť. Boli vytvorené jednotlivé rozhodovacie stromy.

Úprava hráča pre komunikáciu so serverom verzie 0.6.3

V priebehu projektu bola vydaná nová verzia simulačného servera, ktorá v sebe prinášala niekoľko zmien oproti predchádzajúcej verzii. Pribudol napríklad nový preceptor akcelerometer, možnosť posielať dlhšie správy efektorom Say, väčšie ihrisko a optimalizácia pre hru viacerých hráčov.

1.1.5 Critical Error

Tím Critical Error pôsobil na fakulte FIIT v rokoch 2009/2010. Nižšie spomenuté informácie vychádzajú z finálnej dokumentácie tímu[3].

Tím primárne nadviazal na prácu tímu Dream Team, ktorý pôsobil na fakulte FIIT v rokoch 2008/2009. Pri realizovaní projektu použili aj technológiu tímu Agenty 007, konkrétne editor pohybov.

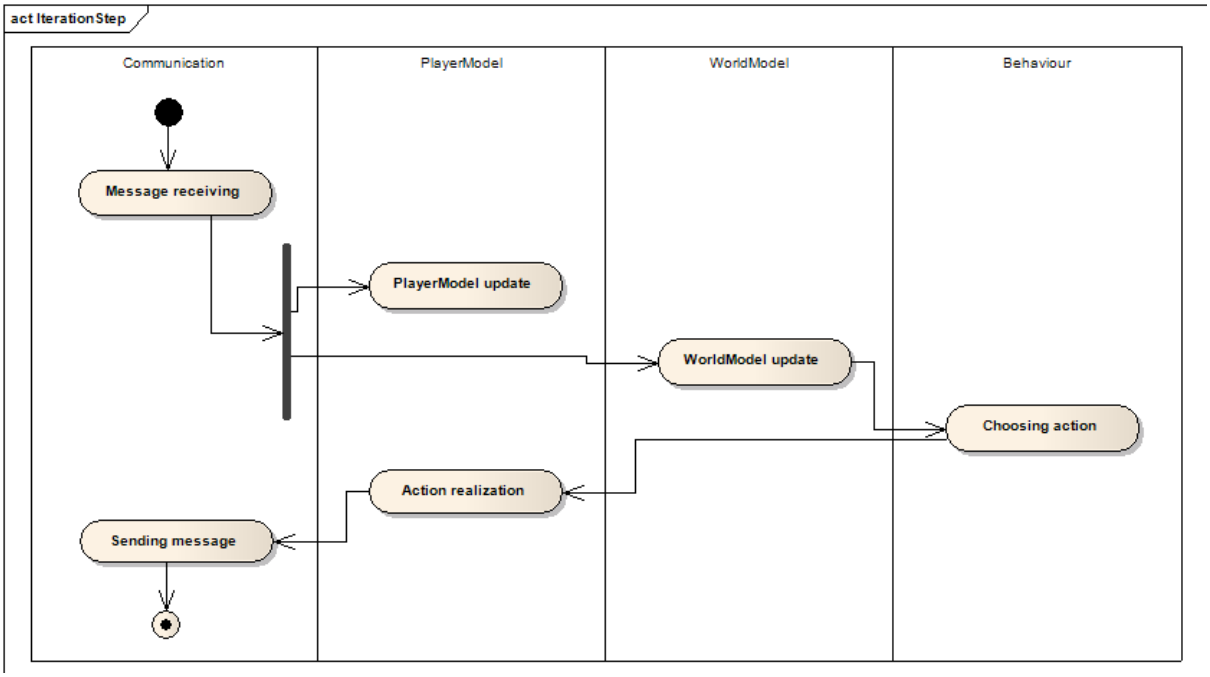
Podľa oficiálnej dokumentácie sa tím zaoberal nasledovnými problematikami:

- Úprava editora pohybu tímu Agenty 007. Rozšírenie editora o funkcionalitu importovania a exportovania pohybov agenta z a do XML súboru formátu tímu Dream Team. Editor je vytvorený v jazyku C#. Editor umožňuje modelovanie pohybov agentov, pričom pri každom modelovaní sa vychádza zo základnej polohy agenta.
- Úprava parsera správ zo servera tímu Dream Team tak aby bol kompatibilný zo serverom 0.6.3.
- Úprava logiky logovania tak aby logovanie bolo zapínané podmienečne a aby logická časť logovania bola architektonicky oddelená od iných častí agenta. Bol implementovaný preprocesor AspectC++ vďaka ktorému bolo navyše zaistené, že logovanie je konfigurovateľné.
- Úprava servera. Agent Sirius nemal k dispozícií žiadny testovací framework a server samotný je nepoužiteľný na testovanie pohybov agentov. Server je určený primárne na hranie zápasu a obsahuje v sebe kontrolné mechanizmy, ktoré zakazujú a resetujú akékoľvek nezvyčajné správanie. Napríklad ak by sme chceli agenta premiestniť priamo pri bránu aby sme mohli otestovať kop tak nám to server nedovolí. Z tohto dôvodu tím upravil serverovú časť konkrétne súbor soccer.dll aby bolo možné spúšťať testovanie.
- Zameranie sa na analýzu záznamu zápasov z majstrovstiev z roku 2008, za účelom inšpirácie pre tvorbu efektívnej chôdze agenta. Konkrétne sa zameranie týkalo tímu SEU Red Sun. Avšak pri analýze sa vyskytli problémy s tím, že záznamy obsahujú veľa šumu a neloguje sa úplne všetko čo viedlo k tomu, že záznam sa dal použiť iba čiastočne a nebolo možné rekonštruovať pohyb.
- Implementácia algoritmu na určenie statických a dynamických objektov. Pri dynamických objektoch sa tím zaoberal zlepšovaním výpočtu budúcich pozícií na low-level úrovni na základe pamätania si 2 posledných pozícií pri lopte a posledných 10 pozícií pri agentovi.
- Implementácia algoritmu na rozhodovanie v reálnom čase. Riešenie je realizované za pomoci dynamických knižníc a pre zmenu správania robota je postačujúce zmeniť knižnicu.

Architektúra agenta

Architektúra agenta tímu vychádza priamo z architektúry tímu Dream Team. Pozostáva zo štyroch komponent a to Communication, WorldModel, Behaviour a PlayerModel. Komponenta Communication zapuzdruje všetky potrebné funkcie a zastrešuje komunikáciu zo serverom. Modul WorldModel reprezentuje model ihriska, PlayerModel obsahuje aktuálny model agenta a zároveň slúži na realizáciu akcií po rozhodovacom procese. Komponent Behaviour sa stará o výber nasledujúcej akcie na základe podkladov, ktoré dostáva na zhodnotenie. Komunikácia medzi jednotlivými komponentmi je znázornená na obrázku 8.

3 http://labss2.fiit.stuba.sk/TeamProject/2009/team17is-si/index.html



Obrázok 8: Graf komunikácie

Implementovaný hráč dokázal vykonávať nasledujúce pohyby:

- Pohyb do boku - Pri kroku doľava hráč prenesie svoje ťažisko na pravú nohu. Následne zodvihne ľavú nohu, 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. Pre pohyb do opačnej strany sa použije symetricky opačný pohyb.
- Otáčanie – Otočenie sa smerom doľava je realizované tak, že agent mierne od seba vytočí nohy a následne sa nakloní doľava. Týmto prenesie svoje ťažisko na ľavú nohu. Pokračuje zodvihnutím pravej nohy a otočením sa na ľavej nohe. Potom sa agent vráti do základnej polohy a cyklus sa opakuje, až kým sa neotočí do želanej polohy. V jednom cykle sa agent otočí približne o 30°. Pri otáčaní sa využíva to, že ide o simulované prostredie, kde nie je žiadny odpor agentovej nohy voči podložke.
- 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
- Finalizačné fázy – sú potrebné pre korektné ukončenie daného pohybu za účelom vykonania iného pohybu. Implementované sú nasledovné finalizačné fázy:
- 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

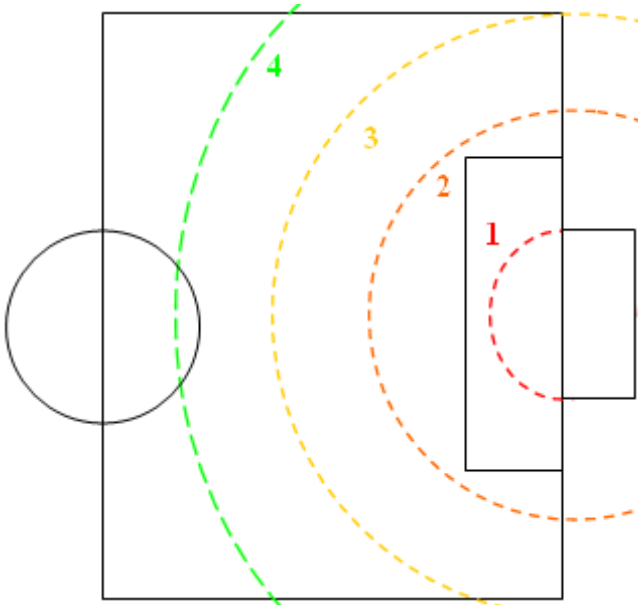
Brankár je rozdelený do troch aspektov správania sa:

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

Pohyby využívané brankárom:

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

Brankár mení svoje správanie na základe toho, kde sa v ihrisku nachádza lopta. Zóny nachádzania sa lopty a k tomu patričného rozličného správania sú znázornené na obrázku 9.



Obrázok 9: Zóny odlišného správania sa brankára

Tím zároveň vytvoril agenta JIM prekódovaním agenta Sirius do jazyka Java. Veľké plus je, že agent JIM má svoj vlastný testovací framework. S tohto dôvodu nie je potrebné upravovať serverovú časť. Problémom pri oboch agentoch je skutočnosť, že dokumentácia je na nízkej úrovni dalo by sa povedať skoro žiadnej. S tohto dôvodu je potrebné hľadať niektoré súvislosti v zdrojových kódoch agentov. Zdrojové kódy sú organizované prehľadne avšak je tu opäť absencia komentárov.

1.1.6 Neurotics (3D simulation video)

Tím Neurotics vznikol na našej Fakulte informatiky a informačných technológií STU v roku 2007 a jeho vedúcim bol Ing. Marián Lekavý PhD.. Tím Neurotics nadviazal na minuloročné tímy 6th Sense (2006) a Hazard Team (2005) a na ich prístupy a agentov.

Tím 6th Sense a ich agent Hazard (nadviazanie na Hazard Team) bol skvelým základom pre ich agenta. Podľa ich názoru mal daný agent výbornú architektúru správania, pretože bola veľmi flexibilná a jednoducho rozšíriteľná. Ide o model vo vrstvách, ktorý je následne organizovaný do modulov. Nevýhodou ale bolo, že daný agent bol postavený na staršom modely servera, ktorý používal na zobrazenie hráčov iba tvar trojrozmernej gule a nie najnovších hráčov humanoidného tvaru. Agent Hazard mal 3-vrstvový model správania, ktorý ale Neurotics zjednodušili iba na dve vrstvy – vrchná vrstva obsahuje primitívne modely (postav sa, …) a spodná vrstva pracuje so základnými pohybmi jednotlivých častí tela

Druhým agentom na ktoré Neurotics nadviazali bol agent Zigorat, ktorý dopĺňal predchádzajúceho agenta o výborný komunikačný modul. Ten modul bol pozmenený pre ich potreby, ale funkčné rozhranie nechali pôvodne.

Ako hlavnú úlohu si tím Neurotics zvolili zmenu statických metód riadenia pohybových funkcií agentov, na dynamické metódy riadenia. Túto zmenu uskutočnili pomocou evolučných algoritmov, a ich hlavným cieľom bolo pomocou týchto algoritmov naučiť ich hráča pomerne inteligentne vstávať po páde z ležiacej polohy do polohy pred pádom. Samozrejme sa tieto úlohy snažili robiť tak, aby budúcoročné tímy mohli na ich prácu nadväzovať a pokračovať v rozvoji daného agenta.

Tím Neurotics použili typ robota „soccerbot056“ na serveri verzie 0.5.6, kde bol implementovaný systém SPARK, ale bez použitia SPADES. Komunikácia so serverom prebiehala pomocou protokolov TCP a UDP. Daný robot (agent) dostával z perceptorov správy v časových intervaloch 20ms. Stred sférickej sústavy bol v ťažisku hornej časti agenta. Agentov zrakový perceptor videl v uhle 360° a objetky boli jednoduché hmotné body a poloha daného objektu bola relatívna, vyjadrená v sférických súradniciach od robota.

Základná štruktúra agenta Neurotic:

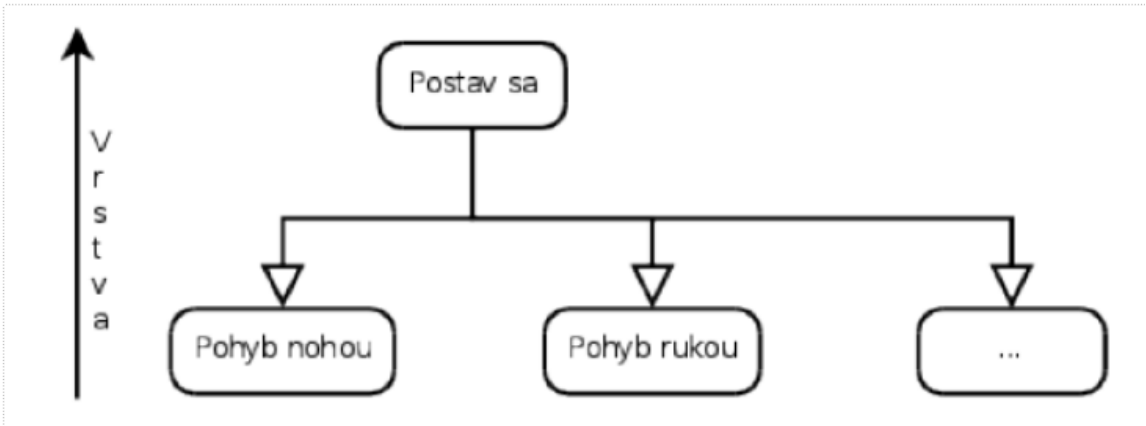
1. Hlavný vykonávací cyklus
2. Komunikačný modul
3. Pohľad na svet (agent Zigorat)
4. Modely správania (agent Hazard)

Hlavný vykonávací cyklus bolo samotné jadro agenta. V tejto časti agenta sa vykonáva volanie jednotlivých podmodulov ktoré zaručia danú funkčnosť.

Komunikačný modul agenta je, ako už bolo spomenuté, prevzatý z agenta Zigorata. Modul sa stará o posielanie a prijímanie správ zo servera.

Pohľad na svet je takisto prevzatý z agenta Zigorata. Zmeny v tejto časti neboli takmer žiadne.

Agent Neurotik využíva štruktúru modelov správania sa (behaviors) z agenta Hazarda. Štruktúra je založená na modularite a jednoduchosti jednotlivých modelov. Každý model je reprezentovaný triedou, ktorá je zdedená z abstraktného rozhrania BaseBehaviorModule. Toto rozhranie obsahuje metódy, ktoré sa v konkrétnych modeloch implementujú. Pomocou makier REGISTER_MODULE (meno) sa jednotlivé moduly zaregistrujú do systému. Makrom USE_MODULE (meno, cesta) sa modul prihlási ako aktívny na použitie. Následne sú moduly rozdelené do viacerých vrstiev.



Obrázok 10: Vrstvy modelov udalosti

Popis správaní:

- AdjustJointBehavior – Podľa zadaných hodnôt priamo nastaví uhlovú rýchlosť kĺbu.
- AnkleBehavior – Nastaví uhol otočenia členku.
- BasicBehavior – Základná trieda hráča, z ktorej dedia všetky ostatné triedy, nemá žiadnu špeciálnu funkcionalitu.
- BeamBehavior – Vysiela beam príkaz serveru. Beam príkaz prenesie hráča na požadovanú pozíciu.
- ElbowBehavior – Nastaví uhol otočenia lakťa.
- HipBehavior - Nastaví uhol otočenia bedrového kĺbu.
- InitPosBehavior – Inicalizuje hráča a nastaví všetko potrebné pre jeho štart.
- KneeBehavior - Nastaví uhol otočenia kolena.
- MainBehavior – Trieda rozhodujúca na základe zložitejších procedúr o nasledujúcej postupnosti akcií. V prototype táto trieda vykonáva samotné chodenie – postupnosť jednotlivých menších elementárnych akcií súvisiacich s chodením.
- RotateArmBehavior - Nastaví uhol otočenia ramena.
- RotateLegBehavior - Nastaví uhol otočenia nohy.
- ShoulderBehavior - Nastaví uhol otočenia plecom.
- TurnToBehavior – Otočí hráča na požadovaný uhol.
- TurnToLeftBehavior – Otočí hráča na požadovaný uhol smerom doľava.
- TurnToRightBehavior - Otočí hráča na požadovaný uhol smerom doprava.
- WalkBehavior – Hráč kráča rovno v smere, v ktorom je otočený.

Postavenie sa pomocou evolučného algoritmu

Pre realizáciu simulovanej evolúcie bolo potrebné zadefinovať nasledovné:

- jedinca a jeho reprezentáciu
- spôsob ohodnocovania schopnosti jedincov prežiť (fitness)
- spôsob výberu rodičov

- operácie kríženia a mutácie
- spôsob výberu jedincov do novej generácie

Jedinec je definovaný genetickou informáciou. Tá musí (pre potreby evolúcie pohybu) definovať pohyb robota. Tvorí ju chronologická postupnosť základných pohybov robota (pohybov jednotlivých kĺbov), indexovaných časom, kedy sa tento pohyb začne vykonávať. Dĺžka chromozómu nie je dopredu známa. Preto použili tzv. „Messy"-chromozómy (mchromozómy). Tie majú premenlivú dĺžku. Samotný chromozóm je tvorený postupnosťou usporiadaných dvojíc, zložených z indexu a hodnoty génu. Ak sa v chromozóme nachádza viac génov s rovnakým indexom, použije sa len jeden z nich, a to prvý v poradí. V tomto konkrétnom prípade by index tvoril čas a hodnotu génu zoznam vstupov pre metódy zabezpečujúce pohyb jednotlivých kĺbov.

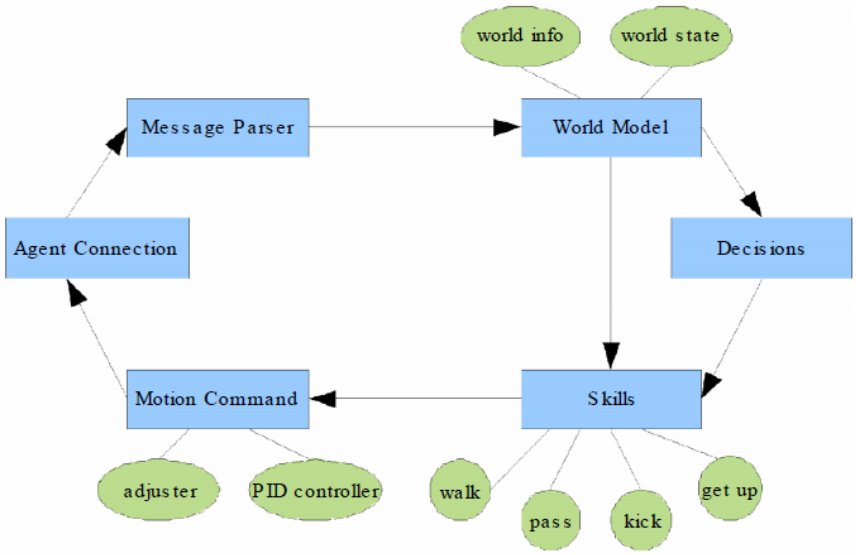
Fitness funkcia na ohodnotenie jedincov obsahuje päť položiek, ktoré sa podieľajú na výslednej hodnote normovanou hodnotou z intervalu <0;1> prenásobenou váhou položky. Na výber jedincov na reprodukciu je použitá stratégia ruletového výberu, ktorá imituje náhodnosť udalostí v živote jedincov. Dvaja jedinci sa krížia na úrovni chromozómov, aj na úrovni génov. Na úrovni chromozómov dochádza k jednoduchému jednobodovému kríženiu. Mutácia na úrovni chromozómu spočíva v pridaní, alebo odobratí náhodného génu. Na úrovni génov sa krížia stromy s rovnakým indexom a to tak, že si vymenia podstromy. Mutácia na úrovni génov je zmena času génu a vymenenie náhodného podstromu za iný náhodne vygenerovaný podstrom. Do novej generácie vstupujú všetci potomkovia rodičov z predchádzajúcej generácie a najlepší jedinci z predchádzajúcej generácie.

1.2 Analýza zahraničných robocup projektov

1.2.1 Analýza tímu CIT3D

Tím CIT3D pochádza z technologického inštitútu v Changzhou v Číne. Tím bol založený v roku 2005 pod názvom CZU3D. Počas svojho pôsobenia na medzinárodných súťažiach sa umiestnili v roku 2006 na 2 mieste, v roku 2008 na 4. mieste a v roku 2011 sa im podarilo umiestniť na 2.mieste.

Architektúra[4] agenta je navrhnutá tak, že agent je tvorený šiestimi samostatnými navzájom nezávislými modulmi, ktoré medzi sebou komunikujú. Takáto architektúra zabezpečuje jednoduchší vývoj agenta a zároveň umožňuje flexibilitu pri vývoji, možnosť diferenciácie agentov a ľahšiu koordináciu medzi agentmi. Architektúra je zobrazená na obrázku 11.

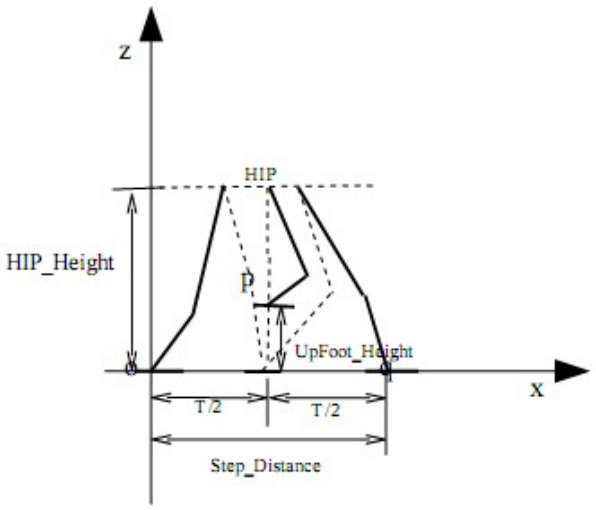


Obrázok 11: Architektúra agenta tímu CIT3D

Zručnosti agenta sú chôdza, otočenie, kop, vstávanie, pád, rozohrávanie. Dobre prepracovaná je chôdza robota.

Pri návrhu algoritmu chôdze agenta tím vychádza z prototypu reálneho človeka. Agent však pri chodení nepoužíva ruky. Tím zanalyzoval a rozdelil ľudskú chôdzu do dvoch štádií. Štádium kedy sme jednou nohou na zemi a štádiu kedy sme oboma nohami na zemi, ktoré sa dookola opakujú. Každú nohu rozdelili na 3 časti: stehno, lýtko a chodidlo. Matematickými rovnicami popísali pohyb, ktorý sa následne snažili vyladiť. Model nohy je znázornený na obrázku 12. Chôdzu sa im podarilo dostať do veľmi kvalitnej úrovne. Robot robí rýchle malé kroky pričom sa mu darí dobre držať stabilitu.

4 http://hedayat.fedorapeople.org/misc/2010tdps/cit3d_tdp.pdf



Obrázok 12: Model nohy agenta CIT3D

Pri procese vstávania sa tím takisto snažil vychádzať z prototypu reálneho človeka. Agent vstáva za pomoci rúk a nôh, pričom vstávanie dokázali odladiť a vstávanie je na časovej úrovni 1.6 sekundy. Proces vstávania je znázornený na obrázku 13.

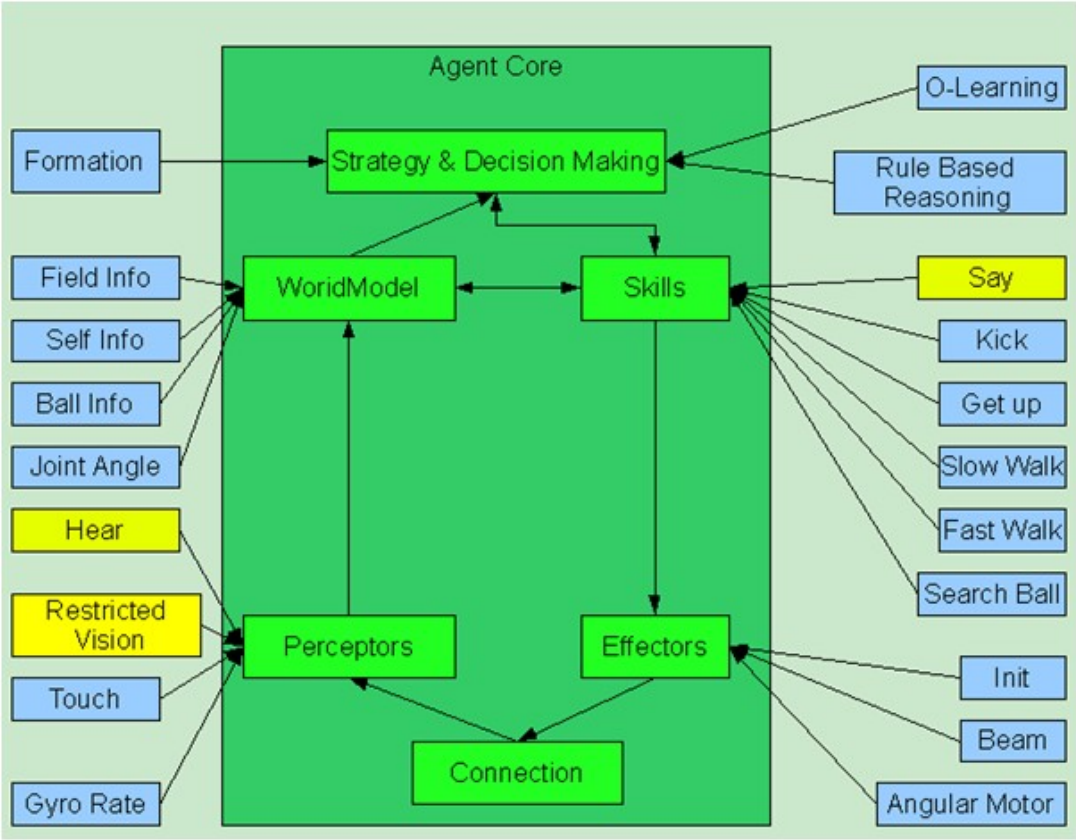
Obrázok 13: Proces vstávania CIT3D agenta

1.2.2 Analýza tímu SEU-RedSun

Tento čínsky tím vznikol v roku 2005 na Southeast university Nanjing. Už krátko po svojom vzniku sa zúčastňovali medzinárodných súťaží v RoboCupe 3D a v roku 2009 dokonca vyhrali turnaje Iran Open a Robocup Graz. Ich cieľom je vyvinúť agenta s dôrazom na znovupoužitie a na správanie čo najviac podobné ľudskému.

Architektúra agenta

Vzhľadom na časté zmeny a aktualizácie serveru SimSpark sa členovia tímu rozhodli, že svojho hráča spravia čo najflexibilnejšieho. Model hráča vytvorili na základe architektúry prídavných modulov čo znamená, striktne vrstvenú štruktúru hráča so singleton modulmi. Takto vedia na zmeny reagovať jednoducho zmenou alebo nahradením niektorého modulu.



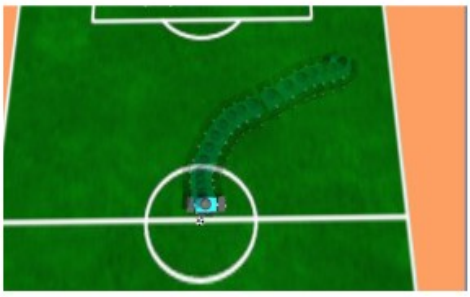
Obrázok 14: Architektúra agenta

Na obrázku 14 je zobrazená architektúra agenta tohto tímu. Zelenou farbou sú označené moduly patriace do jadra agenta. Žltou a modrou farbou zas prídavné moduly.

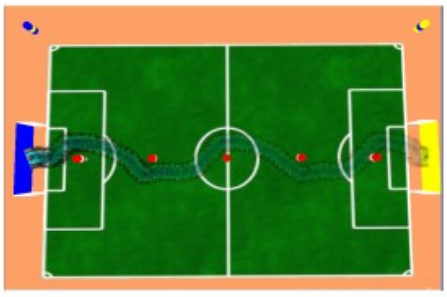
Chôdza a kopanie do lopty

Aby bola hra robota čo najviac podobná hre človeka, je nutné zdokonaliť niektoré jeho pohyby ako chôdza, vstávanie alebo kopanie do lopty.

Napríklad samotná chôdza by mala byť plynulá a nezávislá od momentálnej orientácie hráča. Toto sa čínskemu tímu podarilo dosiahnuť implementovaním takzvanej viacsmerovej chôdze. Jedná sa o schopnosť agenta meniť smer chôdze bez nutnosti zastavenia a natočenia. Vzniká tak plynulý pohyb, ktorý vie v reálnom čase reagovať na vynútené zmeny smeru.



(a) The robot approaches the ball before using its right leg kicking the ball. It can be observed that the robot walks to the desired position smoothly and precisely.



(b) The robot walks through flags which placed on the pitch. The robot bypasses all the flags successfully.

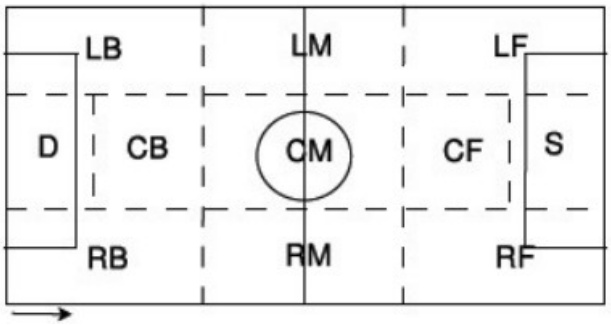
Obrázok 15: Viacsmerová chôdza

Tímu sa tiež podarilo implementovať vylepšený spôsob kopania do lopty. Ich agent je schopný kopnúť do lopty bez ohľadu na jej pozíciu vzhľadom na hráča. Vie teda z každej časti ihriska vystreliť na bránu.

Neoddeliteľnou súčasťou schopností agenta je tiež vstávanie zo zeme. Tímu sa podarilo nasadiť veľmi rýchle vstávanie na úrovni dvoch sekúnd.

Racionálna nahrávka do určitej oblasti

Veľmi zaujímavý je spôsob, akým sa pristupuje k nahrávkam. Namiesto implementácie systému, ktorý pošle loptu presne na určité miesto, bol vytvorený systém, ktorý pošle loptu do určitej oblasti. Znamená to rapídne zníženie množstva výpočtov a teda zvýšenie efektivity. Tím rozdelil ihrisko na 11 oblastí (Obrázok 16).

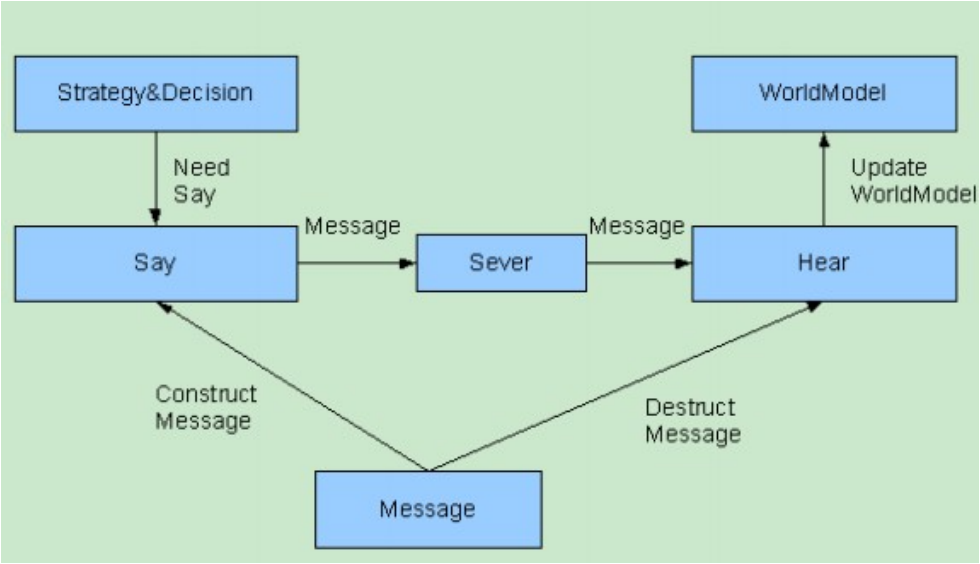


Obrázok 16: Ihrisko rozdelené na oblasti

Schopnosti pri hre

Aby bol agent schopný lokalizovať vlastnú polohu pri hre potrebuje na to vidieť tri rohové zástavky. Počas zápasu však môže nastať situácia, že zástavky nie sú vidieť, preto musí agent otáčať hlavou. To ale trvá veľmi dlho. Preto sa používa rozdelenie ihriska na oblasti, popísane v kapitole vyššie, vďaka ktorému agent vie kam má otočiť hlavu, aby čo najrýchlejšie lokalizoval zástavky a loptu. Implementovaný je tiež algoritmus na neustále sledovanie dôležitých bodov na ihrisku, vďaka čomu vie agent oveľa efektívnejšie reagovať na hru. Sleduje takto napríklad loptu a rohové zástavky.

Pri hre šiestich proti šiestim sa komunikácia medzi agentmi stáva čoraz dôležitejšou. Preto bol vytvorený komunikačný model (Obrázok 17), pomocou ktorého sa agenti medzi sebou dorozumievajú. Oficiálny limit na jednu správu je 20 bytov.



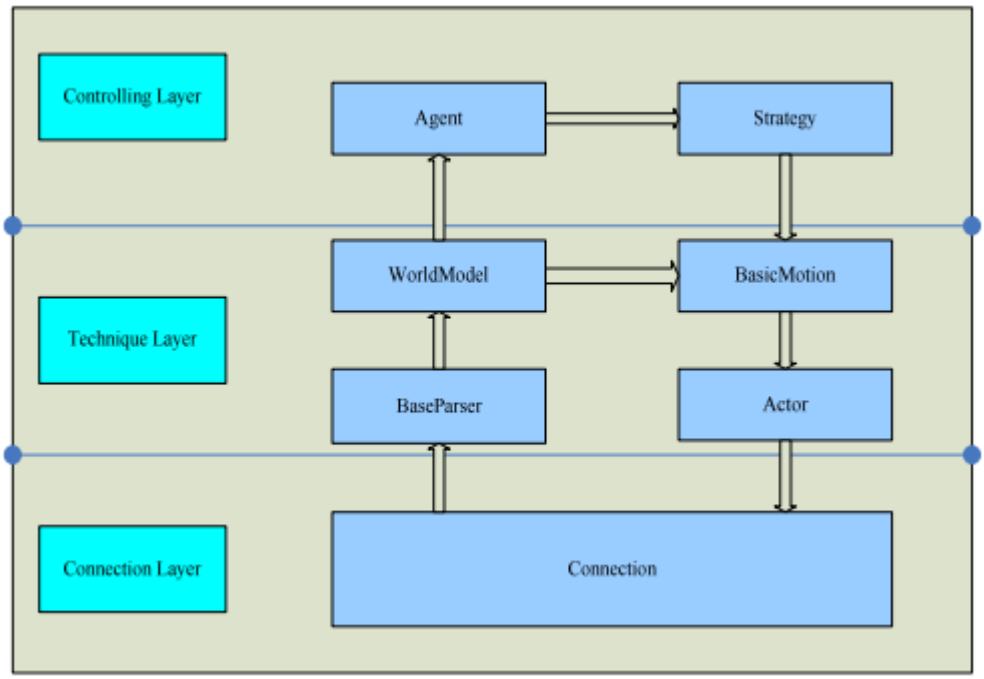
Obrázok 17: Model komunikácie

1.2.3 Analýza tímu KylinSky3D[5]

Tento tím vznikol na Čínskej univerzite Hohai. Ich cieľom je vyvinúť výkonný a rozšíriteľný zdrojový kód agenta. Taktiež si dali za úlohu dosiahnuť stabilnú a autonómnu chôdzu robota.

Architektúra agenta

Achitektúra agenta (Obrázok 18) tímu KylinSky3D sa skladá z troch hlavných vrstiev, ktoré spolu navzájom spolupracujú. Ide o vrstvu pripojenia, technickú vrstvu a kontrolnú vrstvu. Vrstva pripojenia sa stará o komunikáciu agenta so serverom. Technická vrstva tvorí akési jadro a skladá sa zo štyroch častí: WorldModel, BasicMotion, BaseParser a Actor. Kontrolná vrstva obsahuje stratégie agenta pri hre.

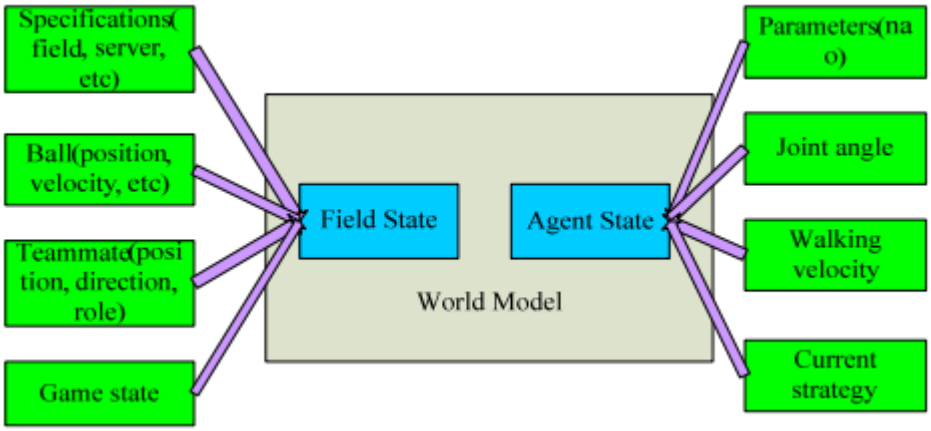


Obrázok 18: Architektúra agenta

Model sveta

Model sveta (Obrázok 19) obsahuje statické a dynamické informácie o hre, stavoch ihriska, správach od iných agentov a strategické informácie. Tím rozdelil tento model na dve základné logické časti: stav ihriska a stav hráča. Stav ihriska obsahuje napríklad informácie o jeho veľkosti, polohy rohových zástaviek, polohy lopty, polohy spoluhráčov alebo stav hry. Stav agenta obsahuje informácie o parametroch robota, uhle kĺbov, rýchlosti chôdze a aktuálnej hernej stratégii.

5 http://hedayat.fedorapeople.org/misc/2010tdps/kylinsky_TDP.pdf



Obrázok 19: Model sveta

Chôdza agenta

Tím KylinSky 3D sa vo veľkej miere zameral na stabilnú a zároveň dynamickú chôdzu. Vytvorili preto prístup, ktorý nazvali offlline plánovanie a online revízia. Znamená to toľko, že pred začatím pohybu si agent cestu naplánuje a počas neho ju už iba upravuje vzhľadom na vzniknuté okolnosti. Určili si dokonca štyri predpoklady, pre chôdzu:

1. trup robota je orientovaný vždy zvislo vzhľadom na zem
2. chodidlá sú orientované vždy vodorovne vzhľadom na zem
3. pri prvom kroku je ľavá noha podporná a pravá vykračujúca
4. trajektórie ľavej a pravej nohy sú rovnaké, keďže chôdza je periodická činnosť

Pre potreby chôdze boli použité viaceré zložité kinematické rovnice. Robot tohto tímu pri nej dokonca používa aj ruky, čím zvyšuje stabilitu pohybu.

1.3 Analýza fyzikálneho modelu robota[6]

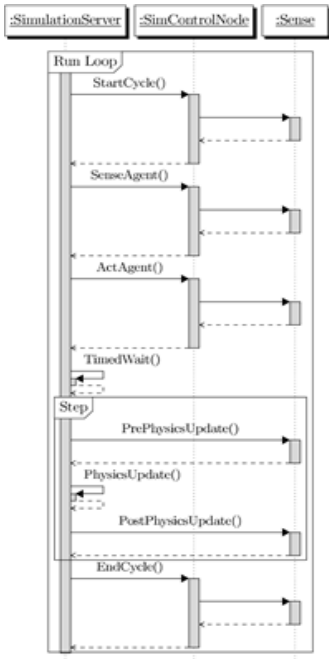
1.3.1 Simspark

Simspark je fyzikálny multiagentový simulačný systém určený pre agentov v trojdimenzionálnom priestore. Je postavený na simulačnom systéme Spark. Keďže jadro simulátora nebolo nikdy prispôsobené pre simuláciu futbalu, všetky služby simulátora prispôsobené futbalu boli implementované ako pluginy.

Simulačný cyklus

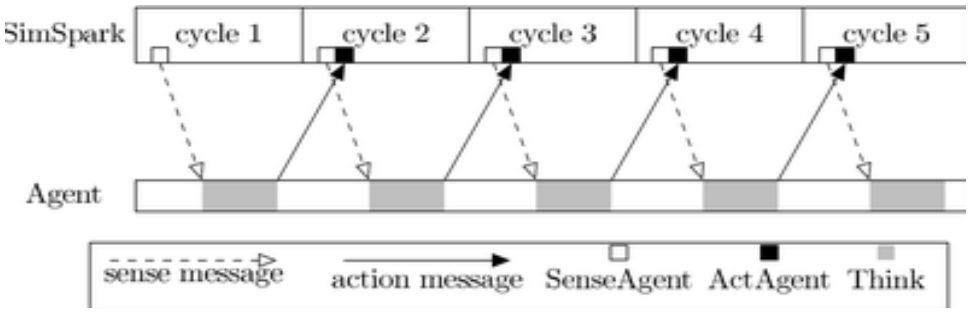
SimsSpark implementuje jednoduchý model na ošetrovanie eventov simulácie a na ošetrovanie všetkých akcií, ktoré prichádzajú od agenta. Keďže server sa nesnaží vyrovnávať oneskorenie v sieti alebo výpočtové zdroje agentov, opakujúce sa simulácie môžu mať rôzny výstup.

Hlavná slučka simulácie(Simulation update loop) v simsparku je plne modifikovateľná, pretože je vytvorená z pluginov nazývaných „simcontrol nodes“. Pri štarte servera je vyvolaný init event a pri ukončovaní jeho behu done event. Simulačný server taktiež zaznamenáva údaj o dĺžke trvania posledného cyklu. Simcontrol nodes sa stará o meranie času a synchronizáciu simulačného času s reálnym časom použitým na renderovanie scény. Simulačná slučka sa snaží, aby každý cyklus trval rovnako (20ms). Pokiaľ je simulácia rýchlejšia, zvyšný čas server čaká. Naopak ak je pomalšia, aktualizuje sa len fyzika a server nekomunikuje s agentmi.



Obrázok 20: Sekvenčný diagram hlavnej slučky

6 Zdroje: http://simspark.sourceforge.net/wiki/index.php/Main_Page
SimSpark User's Manual Version 1.2



Obrázok 21: Následnosť správ od agenta a ich spracovanie serverom

V každom cykle je najskôr vyvolaný „sense event“ a potom „action event“, dôsledkom čoho akcie, ktoré agent poslal v ntom cykle, budú vykonané v nasledujúcom (n+1)cykle (Obrázok 21).

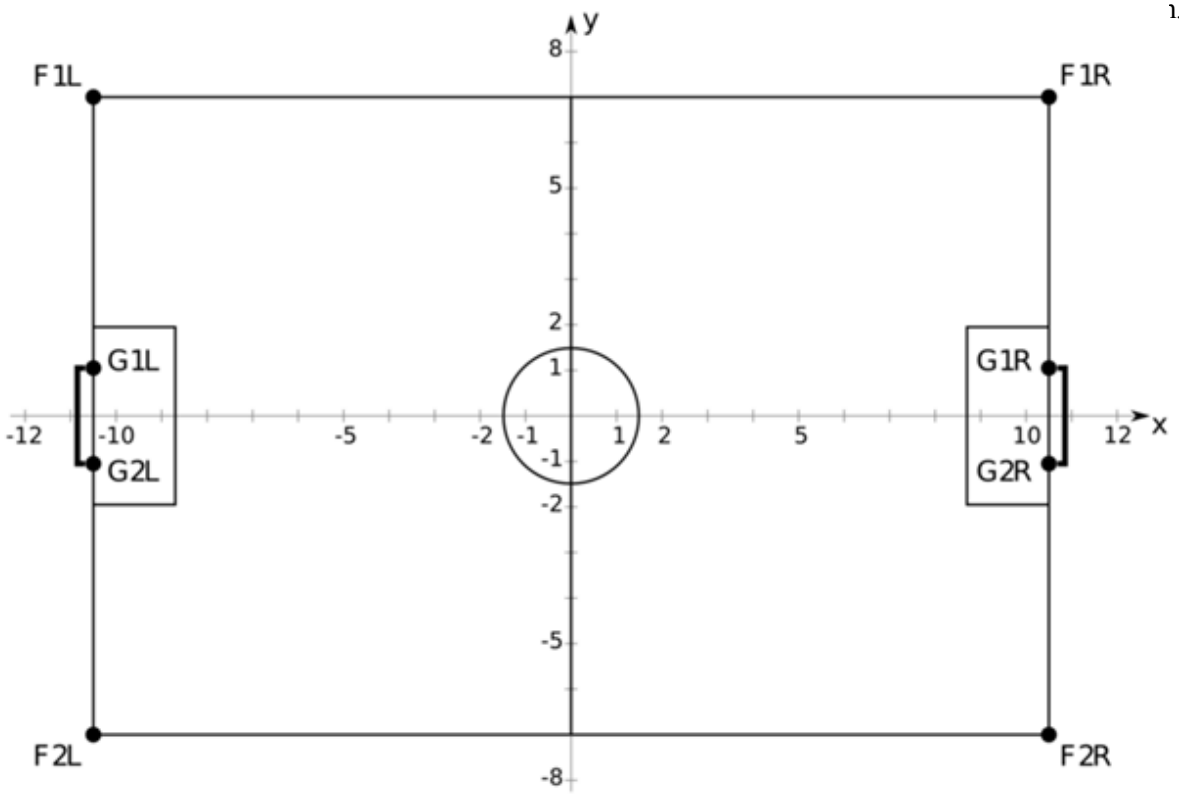
Keďže dnešné počítače umožňujú paralelné spracovávanie, je možné aby server bežal v multi-thread režime. V tomto režime „simcontrol nodes“ sú spúšťané paralelne, a aj výpočet fyziky je pustený v samostatnom vlákne.

1.3.2 Rcssserver3d

Rcssserver3d je oficiálne prostredie pre 3D Soccer Simulation League v RoboCupe. Implementuje simuláciu futbalového zápasu, v ktorom dva tímy až jedenástich robotov hrajú proti sebe. Súčasným modelom robota je robot nazývaný Nao. Množstvo pravidiel zápasu je možné automaticky kontrolovať, aj keď veľa situácií ako je napríklad neférové správanie hráčov musí byť kontrolovaných ľudským rozhodcom cez Monitor.

Ihrisko

R ı.



Obrázok 22: Schéma futbalového ihriska rssserver3d simulačného prostredia

Automatický rozhodca

- Sleduje čas každej polovice
- Sleduje, ktorý hráč sa posledný dotkol lopty
- Sleduje skóre
- Sleduje či je lopta mimo ihriska a rozhoduje komu má patriť lopta pri výkope, vyhadzovaní.
- Počas voľného kopu musí byť protivník minimálne 1,3m od lopty.
- Počas výkopu brankára musí byť protivník minimálne 1,0m od lopty.
- Postavenie mimo hry je implementované len experimentálne.
- Snaží sa zabraňovať nechcenému zablokovaniu hry nahromadením robotov okolo lopty, a taktiež deteguje ležiacich robotov na ihrisku.
- Sleduje, či tím neblokuje svoju bránku viacerými hráčmi.

Ľudský rozhodca

Rozhoduje cez pripojený monitor. Je zodpovedný za začatie hry na začiatku každej polovice zápasu. Rieši situácie, keď sa hra uviazne v mŕtvom bode, napríklad keď sa navzájom blokujú hráči a nemôžu sa dostať k lopte. Taktiež rozhoduje fauly (používanie rúk a nevhodné správanie).

Komunikačný protokol

Komunikácia so serverom prebieha pomocou TCP/IP protokol u. Správy posielané a prijímané serverom sú vo formáte S-výrazov. Správy sú vytvárané z ASCII znakov, čo umožňuje zakódovanie jedného znaku do jedného bytu. Správa má 32bitovú hlavičku(big endian), v ktorej je uložené poradové číslo správy.

Nao robot

Nao je humanoidný robot používaný pri RoboCupe s reálnymi hráčmi. Jeho výška je 57cm a váha 4,5kg. Podľa neho je vytvorený model Nao robota na simulačnom serveri rcssserver3d. Je vybavený množstvom perceptorov a efektorov, pomocou ktorých agent získava informácie o stave robota na serveri(perceptory) a manipuluje s ním(efektory). Obsahuje 22 kĺbov, pričom každý kĺb je možné natočiť len jedným smerov(na rozdiel od starších verzií, ktoré obsahovali aj univerzálny kĺb). Zložitejšie kĺby ako je napríklad bedrový kĺb alebo ramenný je zložený z viacerých jednoduchých kĺbov.

1.3.3 Perceptory

Perceptory sú zmysly agenta cez, ktoré vníma prostredie. Server posiela agentovi cez komunikačný protokol správy s perceptorov.

GyroRate perceptor

Poskytuje informácie o zmene orientácie tela od pozície v poslednom cykle. Správa obsahuje tri čísla, ktoré predstavujú uhlové rýchlosti vzhľadom na x-ovú, y-ovú, z-ovú os v stupňoch za sekundu. Perceptor sa nachádza v hornej časti trupu Nao robota.

Formát správy:	`(GYR (n <name>) (rt <x> <y> <z>))`
<name>	Meno časti, v ktorej sa nachádza tento perceptor
`<x> <y> <z>`	uhlové rýchlosti vzhľadom na x-ovú, y-ovú, z-ovú os v stupňoch za sekundu, zaokrúhlené na dve desatinné miesta
Príklad správy:	`(GYR (n torso) (rt 0.01 0.07 0.46))`
Frekvencia správy:	Každý cyklus servera
Šum:	Server nevnáša žiaden šum do tejto správy

Tabuľka 1: Formát správy GyroRate perceptora

HingeJoint perceptor

Poskytuje informácie o natočení kĺbu. Nao robot obsahuje 22 kĺbov. Nulový uhol predstavuje vyrovnaný kĺb. Každý kĺb má taktiež minimálny a maximálny uhol, ktorý závisí od modelu Nao robota.

Formát správy:	`(HJ (n <name>) (ax <ax>))`
<name>	Identifikátor kĺbu
`<ax>`	Aktuálne natočenie uhlu, zaokrúhlené na dve desatinné miesta
Príklad správy:	`(HJ (n laj3) (ax -1.02))`
Frekvencia správy:	Každý cyklus servera
Šum:	Server nevnáša žiaden šum do tejto správy

Tabuľka 2: Formát správy HingeJoint perceptora

ForceResistant perceptor

Tlakový perceptor hovorí , hovorí aká veľká sila pôsobí na telo. Telo správy obsahuje dva vektory. Prvý vektor je počiatočný bod a druhý vektor je výsledná sila pôsobiaca v tomto bode. Tieto dva vektory sú len aproximácia reálnej sily. Počiatočný bod je vypočítaný ako váhový priemer všetkých bodov, na ktoré sila pôsobí. Nao robot má dva tieto perceptory na spodnej strane chodidla s označením lf a rf.

Formát správy:	`(FRP (n <name>) (c <px> <py> <pz>) (f <fx> <fy> <fz>))`
`<name>`	lf ak prichádza z ľavého chodidla alebo rf ak z pravého
`<px> <py> <pz>`	Súradnice počiatočného bodu, zaokrúhlené na dve desatinné miesta
`<fx> <fy> <fz>`	Zložky vektora sily, zaokrúhlené na dve desatinné miesta. Dĺžka vektora hovorí o výslednej sile v newtonoch(kg.m/s2)
Príklad správy:	`(FRP (n lf) (c -0.14 0.08 -0.05) (f 1.12 -0.26 13.07))`
Frekvencia správy:	Každý cyklus servera, ale len v prípade, že nastane kolízia
Šum:	Server nevnáša žiaden šum do tejto správy

Tabuľka 3: Formát správy ForceResistance perceptora

Accelerometer

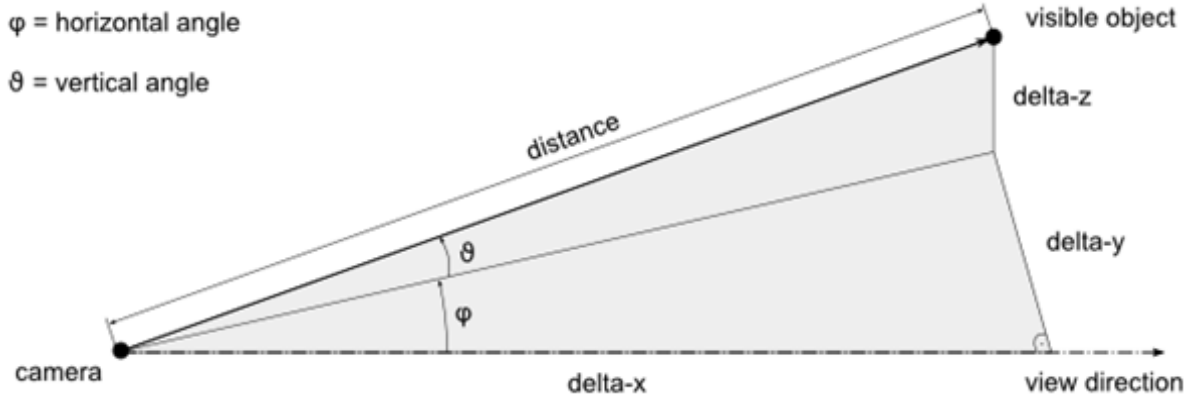
Meria zrýchlenie relatívne k voľnému pádu, čo znamená, že v pokoji indikuje zrýchlenie 9,81m/s^2(gravitačné zrýchlenie). Nao má accelerometer umiestnený v hornej časti trupu.

Formát správy:	`(GYR (n <name>) (a <x> <y> <z>))`
`<name>`	Meno časti, v ktorej sa nachádza tento perceptor
`<x> <y> <z>`	Zrýchlenie vzhľadom na x-ovú, y-ovú, z-ovú os v m/s^2, zaokrúhlené na dve desatinné miesta
Príklad správy:	`(ACC (n torso) (a 0.00 0.00 9.81))`
Frekvencia správy:	Každý cyklus servera
Šum:	Server nevnáša žiaden šum do tejto správy

Tabuľka 4: Formát správy accelerometra

Vision perceptor

Podáva informácie o objektoch, ktoré agent vidí na ihrisku, ako sú iný hráči, lopta, čiary, značky. Nao robot ma tzv. Restricted Vision Perceptror, ktorý je ohraničený 120° zorným poľom.



Obrázok 23: Rozpoznávanie objektu Vision perceptorom

Do údajov tohto perceptora server automaticky vnáša chyby a to:

- Kalibračnú chybu na pozíciu kamery. Pre každú os, chyba predstavuje hodnotu vybranú z intervalu -0,005 až 0,005 pomocou rovnomerného rozloženia.
- Dynamický šum je normálne rozdelenie so základom $\sigma = 0$:
- Chyba vzdialenosti: $\mu = 0.0965$ a ešte je potom táto chyba vynásobená vzdialenosťou/100
- Chyba φ: $\mu = 0.1225$
- Chyba ϑ: $\mu = 0.1480$

Značky a lopta sú opísané len jedným vektorom. Čiary sú opísané dvomi vektormi. Prvý vektor je začiatok a druhý koniec časti čiary, ktorá je zornom poli hráča. Spoluhráči alebo protihráči sú opísaný množinou pozícií jednotlivých častí tela, ktoré sú viditeľné.

Formát správy:	`(See +(<name> (pol <distance> <angle1> <angle2>))`
	`+(P (team <teamname>) (id <playerID>) +(<bodypart>`
	`(pol <distance> <angle1> <angle2>)))`
	`+(L (pol <distance> <angle1> <angle2>) (pol <distance>`
	`<angle1> <angle2>)))`

`<name>`	Jednoduché objekty(Značky: *F1L, F1R, F2L, F2R, G1L, G1R, G2L, G2R, Lopta: B*)
`<distance>:`	Vzdialenosť od viditeľného objektu
`<angle1>`	Horizontal angle
`<angle2>`	Vertical angle
`<teamname>`	Meno tímu do ktorého patrí hráč
`<playerID>`	Číslo hráča
`<bodypart>`	Časť tela (head, rlowerarm, llowerarm, rfoot, lfoot)

Príklad správy:	`(See (G2R (pol 17.55 -3.33 4.31))`
	`(G1R (pol 17.52 3.27 4.07))`
	`(F1R (pol 18.52 18.94 1.54))`
	`(F2R (pol 18.52 -18.91 1.52))`
	`(B (pol 8.51 -0.21 -0.17))`
	`(P (team teamRed) (id 1)`
	`(head (pol 16.98 -0.21 3.19))`
	`(rlowerarm (pol 16.83 -0.06 2.80))`
	`(llowerarm (pol 16.86 -0.36 3.10))`
	`(rfoot (pol 17.00 0.29 1.68))`
	`(lfoot (pol 16.95 -0.51 1.32)))`
	`(P (team teamBlue) (id 3)`
	`(rlowerarm (pol 0.18 -33.55 -20.16))`
	`(llowerarm (pol 0.18 34.29 -19.80))))`
	`(L (pol 12.11 -40.77 -2.40) (pol 12.95 -37.76 -2.41))`
	`(L (pol 12.97 -37.56 -2.24) (pol 13.32 -32.98 -2.20))`

Frekvencia správy:	Každý tretí cyklus
Šum:	Kalibračná chyba a dynamický šum

Tabuľka 5: Formát správy Vision perceptora

GameState Perceptor

Informuje o aktuálnom stave hry.

Formát správy:	`(GS (t <time>) (pm <playmode>))`
`<time>`	Aktuálny čas hry
`<playmode>`	Aktuálny stav hry
Príklad správy:	`(GS (t 0.00) (pm BeforeKickOff))`
Frekvencia správy:	Každý cyklus servera

Tabuľka 6: Formát správy GameState perceptora

Hear perceptor

Keďže nieje dovolené komunikovať medzi agentmi priamo, je možné komunikovať medzi nimi pomocou zvukových správ. Tento perceptor zachytáva zvukové správy vyslovené inými hráčmi.

Maximálna dĺžka správy je 20 bytov. Správa môže obsahovať len ASCII znaky z množiny [0x21; 0x7E], okrem zátvoriek a to sú znaky 0x28, 0x29. Správa vyslovená vo vzdialenosti viac ako 50m nemôže byť zachytená. Každý hráč zachytáva len jednu správu počas dvoch simulačných cykloch. Aby družstvá neblokovali protivníkove senzory, iba jedna správa tímu počas dvoch simulačných cyklov bude vypočutá, a to tá ktorá sa prvá dostane na server. A taktiež sú vypočuté správy posielané samy sebe.

Formát správy:	`(hear <time> self/<direction> <message>)`
`<time>`	Simulačný čas, v ktorom bola správa počutá
`<direction>`	Horizontálny uhol v stupňoch, hovorí o tom odkiaľ správa prišla
`<messge>`	Samotná správa
Príklad správy:	`(hear 12.3 self helloworld)` `(hear 12.3 -12.7 helloyourself)`

Tabuľka 7: Formát správy Hear perceptora

1.3.4 Efektory

Pomocou efektorov je možné dávať príkazy agentovy v simulácii. Správy pre efektory sú posielané na server, ktorý podľa nich mení scénu.

Create efektor

Pokiaľ sa agent pripojí na server nie je viditeľný a nie je možné s ním pracovať. Tento efektor je použitý na fyzické vytvorenie agenta v simulácii a následne aj všetkých jeho ďalších efektorov a perceptrov. Agent je vytvorený na základe popisu v súbore, ktorý je predaný ako parameter v správe:

Formát správy:	`(scene <filename>)`
Príklad správy:	`(scene rsg/agent/nao/nao.rsg)`

Tabuľka 8: Formát správy Create efektora

HingeJoint efektors

Dáva príkaz kĺbu robota aby sa natočil. Prvý parameter je identifikátor kĺbu a druhý je veľkosť natočenia v stupňoch v jednom simulačnom cykle. Zmena natočenia pokračuje v každom ďalšom cykle(nielen v tom v ktorom pošleme správu) a to dovtedy, kým nepríde nová správa s novým parametrom. Pokiaľ aktuálne natočenie kĺbu natrafí na maximum, ostane v tomto maxime pokiaľ nepríde nová správa.

Formát správy:	`(<name> <ax>)`
Príklad správy:	`(lae3 5.3)`

Tabuľka 9: Formát správy HingeJoint efektora

Synchronize efektor

Server, môže byť taktiež používaný v “agent sync mode”, v ktorom predtým ako nastane nový cyklus musí server zozbierať od všetkých pripojených agentov synchronizačnú správu.

Formát správy:	(syn)

Tabuľka 10: Formát správy Synchronize efektora

Init efektor

Posiela sa len raz pre každého agenta po Create efektore. Zaregistruje agenta ako hráča príslušného tímu a priradí mu číslo. Pokiaľ je ako číslo hráča zadaná nula, číslo je priradené automaticky. Maximálne množstvo hráčov na každej strane je momentálne šesť.

Formát správy:	(init (unum <playernumber>)(teamname <yourteamname>))
<yourteamname>	Nesmie obsahovať medzery. Bezpečné je používanie znakov [A-Za-z_-]
Príklad správy:	(init (unum 1)(teamname FHO))

Tabuľka 11: Formát správy Init efektora

Beam efektor

Nastaví hráča na príslušnú pozíciu pred začatím každého polčasu.

Formát správy:	(beam <x> <y> <rot>)
Príklad správy:	(beam 10.0 -10.0 0.0)

Tabuľka 12: Formát správy Beam efektora

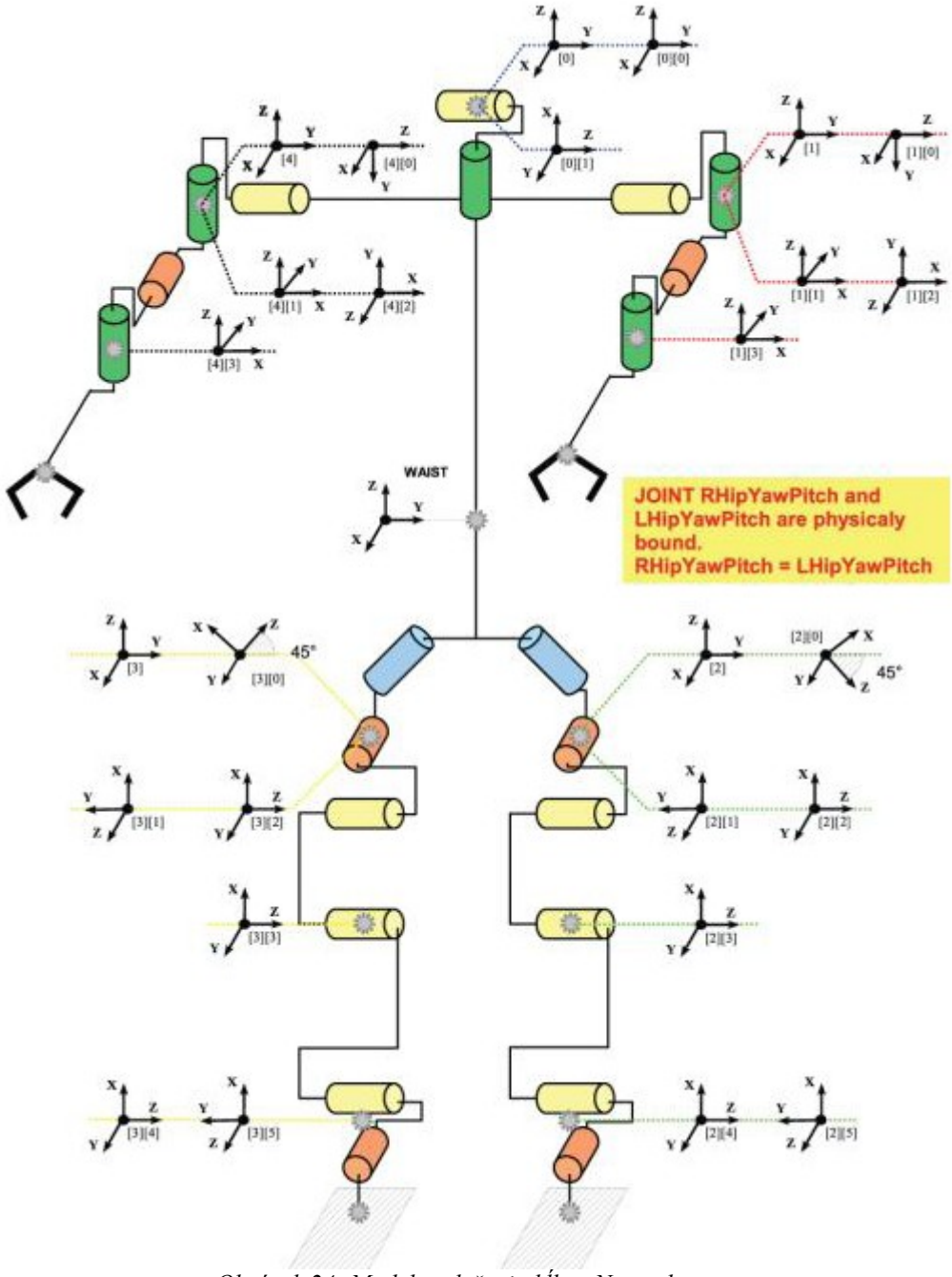
Say efektor

Slúži na vyslovenie zvukovej správy slúžiacej na komunikáciu medzi hráčmi. Platia tu rovnaké obmedzenia, čo sa týka formátu správy, ako pri Hear perceptore.

Formát správy:	(say <message>)
Príklad správy:	(say helloworld)

Tabuľka 13: Formát správy Say efektora

1.3.5 Model Nao robota



Obrázok 24: Model rozloženia kĺbov Nao robota

No.	Description	Hinge Joint	Perceptor name	Effector name
1	Neck Yaw	`[0][0]`	`hj1`	`he1`
2	Neck Pitch	`[0][1]`	`hj2`	`he2`
3	Left Shoulder Pitch	`[1][0]`	`laj1`	`lae1`
4	Left Shoulder Yaw	`[1][1]`	`laj2`	`lae2`
5	Left Arm Roll	`[1][2]`	`laj3`	`lae3`
6	Left Arm Yaw	`[1][3]`	`laj4`	`lae4`
7	Left Hip YawPitch	`[2][0]`	`llj1`	`lle1`
8	Left Hip Roll	`[2][1]`	`llj2`	`lle2`
9	Left Hip Pitch	`[2][2]`	`llj3`	`lle3`
10	Left Knee Pitch	`[2][3]`	`llj4`	`lle4`
11	Left Foot Pitch	`[2][4]`	`llj5`	`lle5`
12	Left Foot Roll	`[2][5]`	`llj6`	`lle6`
13	Right Hip YawPitch	`[3][0]`	`rlj1`	`rle1`
14	Right Hip Roll	`[3][1]`	`rlj2`	`rle2`
15	Right Hip Pitch	`[3][2]`	`rlj3`	`rle3`
16	Right Knee Pitch	`[3][3]`	`rlj4`	`rle4`
17	Right Foot Pitch	`[3][4]`	`rlj5`	`rle5`
18	Right Foot Roll	`[3][5]`	`rlj6`	`rle6`
19	Right Shoulder Pitch	`[4][0]`	`raj1`	`rae1`
20	Right Shoulder Yaw	`[4][1]`	`raj2`	`rae2`
21	Right Arm Roll	`[4][2]`	`raj3`	`rae3`
22	Right Arm Yaw	`[4][3]`	`raj4`	`rae4`

Table 14: Zoznam kĺbov Nao robota a ich identifikátorov

Body part					Hinge Joint				
Name	Parent	Translation	Mass	Geometry	Name	Anchor	Axis	Min	Max
torso			1.2171	Box 0.1, 0.1, 0.18					
neck	torso	0, 0, 0.09	0.05	Cylinder L: 0.08 R: 0.015	HJ1	0, 0, 0	0,0,1	-120	120
head	neck	0, 0, 0.065	0.35	Sphere 0.065	HJ2	0, 0,-0.005	1,0,0	-45	45
shoulder	torso	0.098, 0, 0.075(r) -0.098, 0, 0.075(l)	0.07	Sphere 0.01	AJ1	0, 0, 0	1,0,0	-120	120
upperarm	shoulder	0.01, 0.02, 0(r) -0.01, 0.02, 0(l)	0.150	Box 0.07, 0.08, 0.06	AJ2	-Translation	0,0,1	-95(r) -1(l)	1(r) 95(l)
elbow	upperarm	-0.01, 0.07, 0.009(r) 0.01, 0.07, 0.009(l)	0.035	Sphere 0.01	AJ3	0, 0, 0	0,1,0	-120	120
lowerarm	elbow	0, 0.05, 0	0.2	Box 0.05, 0.11, 0.05	AJ4	-Translation	0,0,1	-1(r) -90(l)	90(r) 1(l)
hip1	torso	0.055, -0.01, -0.115(r) -0.055, -0.01,-0.115(l)	0.09	Sphere 0.01	LJ1	0, 0, 0	-0.7071 ,0,0.7071(r) -0.7071 ,0,-0.7071 (l)	-90	1
hip2	hip1	0, 0, 0	0.125	Sphere 0.01	LJ2	0, 0, 0	0,1,0	-45(r) -25(l)	25(r) 45(l)
thigh	hip2	0, 0.01, -0.04	0.275	Box 0.07, 0.07, 0.14	LJ3	-Translation	1,0,0	-25	100
shank	thigh	0,0.005,-0.125	0.225	Box 0.08, 0.07, 0.11	LJ4	0,-0.01, 0.045	1,0,0	-130	1
ankle	shank	0, -0.01,-0.055	0.125	Sphere 0.01	LJ5	0, 0, 0	1,0,0	-45	75
foot	ankle	0, 0.03,-0.035	0.2	Box 0.08, 0.16, 0.03	LJ6	-Translation	0,1,0	-25(r) -45(l)	45(r) 25(l)

Table 15: Konfigurácia Nao robota

1.4 Podrobná analýza Androids

1.4.1 Opis analýzy

Tím Androids pod vedením Ing. Ivana Kapustíka v minulom akademickom roku (2010/2011) pracoval na tímovom projekte Robocup 3D.

Po dôkladnej analýze zahraničných a slovenských tímov sa tím Androids rozhodol nadviazať na fakultného agenta JIM, ktorý vznikol prenesením agenta Sirius (C++) tímu Critical Error do Javy. Pri ich začiatkoch práce sa agent JIM nachádzal v dvoch verziách:

1. Jim (hlavná línia vývoja)
2. JimTP (experimentálna línia, vhodná pre Tímové projekty)

Agent JIM sa v tejto fáze nachádzal približne v takomto stave:

Ako vybraté vývojové prostredie bol zvolený Eclipse a jazyk Java, pričom ale logika a konfigurácie boli v Ruby. Komunikácia, parser, model sveta a predpovedací modul bol implementovaný v Jave. Vývoj bol aspektovo-orientovane rozšírený pomocou AspectJ. Približne 90% kódu malo prejsť testami, keďže agent bol vyvíjaný technikou Test Driven Development. Bola uskutočnená zmena načítavania pohybov a správania sa za behu. Vypracovaná príslušná dokumentácia a implementovaných niekoľko základných pohybov (vstávanie z brucha, vstávanie z chrbta, prototyp chôdze).

1.4.2 Opis návrhu riešenia

Pri návrhu riešenia sa rozhodli zamerať na päť základných oblastí:

1. Samotné pohyby – v prvom rade sa rozhodli zaoberať pohybmi tela agenta, pretože hráči môžu plnohodnotne využiť potenciál až po dokonalom ovládaní vlastného tela
2. Vyššie schopnosti agenta a stratégiu hrania – vytvorenie vyšších schopností za pomoci nižších a následne ich kombináciou vytváranie stratégií ako plánovanie útoku či obrany
3. Rozšírenie editora pohybov tímu Agenty 007 – pridanie ďalších funkcionalít, prípadne pridanie novej časti zaoberajúcej sa vyššou logikou, teda plánovaním a stratégiou
4. Vytváranie vyššej logiky XABSL – založený na stavovom automate
5. Testovací framework – testovanie schopností agenta

1.4.3 Opis implementácie

Implementácia prebiehala v dvoch obdobiach a teda v letnom a zimnom semestri. Príčom v letnom semestri bol vytvorený nejaký návrh prototypu, v ktorom si definovali všetkých päť vyššie zmienených bodov, ktoré by radi implementovali.

Následne ešte v zimnom semestri stihli vypracovať základný prototyp. V tomto prototype išlo hlavne o implementáciu jednoduchého testovacieho frameworku, vytvorenie základných pohybov agenta a úpravy v editore pohybov, ktoré im pomohli aj pri ďalšej práci a vytváraní nových a zložitejších pohybov v letnom semestri. Z pôvodne plánovaných pohybov nakoniec implementovali iba niektoré, ktoré budú ďalej opísané. Ostatné neboli implementované či už kvôli zložitej implementácii so slabým výsledkom, dlhému trvaniu alebo celkovej nadbytočnosti.

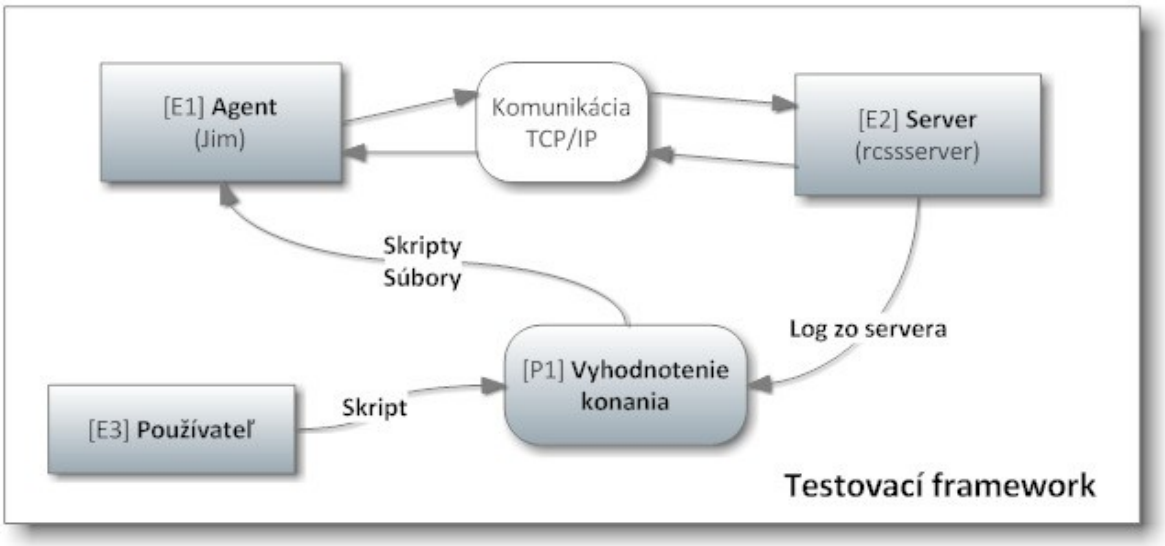
Letný semester

Implementovanie testovacieho frameworku

Testovací framework je aplikácia, ktorá slúži na posielanie príkazov agentom a ich plnenie vyhodnocuje na základe údajov získaných zo servera. Nakoľko testovanie má obsiahnuť viaceré udalosti na ihrisku, bolo nutné presne špecifikovať požiadavky na možnosti testovacieho frameworku.

Testovací framework musí umožňovať:

- Testovanie nižších aj vyšších schopností hráča
- Konfiguráciu udalostí na ihrisku
- Testovanie viacerých hráčov a ich kooperácie
- Umožniť optimalizáciu skrze hľadanie lepších parametrov



Obrázok 25: Diagram toku dát. Zaradenie Testovacieho frameworku v kontexte agentov a servera

Bolo potrebné upraviť štandardnú komunikáciu medzi agentom a serverom, aby agent prijímal príkazy z vonku. Bolo treba presne zadefinovať príkazy ako aj vyhodnocovanie daných príkazov. Tím vytvoril testovací framework za pomoci skriptovacieho jazyka Ruby s tým, že agent bude prijímať skripty v stanovenom tvare a spúšťať ich na svojej strane.

Na strane agenta bol implementovaný open-source TFTP server na strane agenta a TFTP klient na strane frameworku. TFTP server bol navyše upravený tak, aby pri prijatí súboru so špecifickým menom („ruby.exec“) tento súbor neprijímal štandardne, ale aby ho po prijatí vykonal ako Ruby skript. Na strane servera robotického futbalu nebolo potrebné vykonať žiadne zmeny. To je dôležité z dôvodu, že vývoj servera napreduje a zmeny by bolo nutné vykonávať pri každej novej verzii. Na druhú stranu je veľmi dôležité povedať, že server bol na zaradenie do aplikácií typu Testovací framework pripravený.

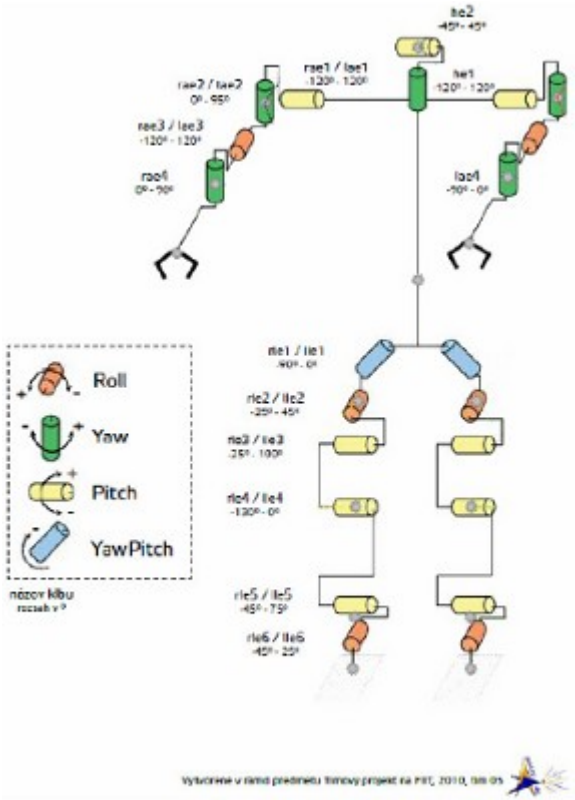
Používateľ do testovacieho frameworku zadá skript v jazyku Ruby, ktorý obsahuje:

- čo sa má vykonať
- časovo závislý test, ktorý určuje, či výsledný test môže dopadnúť pozitívne
- takýto typ testu je dôležité do skriptu zaradiť z hľadiska efektívneho využitia času, kedy je zrejmé, že test by v konečnom dôsledku dopadol neúspechom (ako príklad môžeme uviesť nahrávku – ak už má zlý smer, tak je zrejmé, že nahrávka nepríde do požadovanej pozície)
- test, ktorý hovorí, či výsledok akcie bol úspešný

Pomôcka pri vytváraní pohybov

Prvotne bol vytvorený podrobný prehľad anatómie agenta JIM, kvôli vývoju pohybov pomocou XML súborov. Cieľom bolo vytvoriť pomôcku, ktorá v začiatkoch tvorby pohybov umožní prekonať počiatočné ťažkosti pri identifikovaní kĺbov a zisťovaní ich možností. Bola vytvorená pomôcka, ktorá obsahuje tri základné informácie o každom kĺbe, ktoré sú potrebné pri vývoji pohybu:

- Názov kĺbu
- Orientácia otáčania (kladná / záporná, naznačená šípkami)
- Rozsah v stupňoch



Obrázok 26: Anatómia robota NAO – pomôcka pri vývoji pohybov

Základné pohyby v prototype

Vstávanie - Vstávanie z brucha

- najskôr hráč uvedie všetky kĺby do neutrálnej polohy, okrem ramenných, ktoré pripaží
- následne rozkročí a úplne skrčí nohy
- v ďalšej fáze postupne prinožuje a vystiera nohy, čím zároveň vstáva
- v koncovej fáze vystierania hráč predpaží ruky, ktoré až do tejto fázy boli pripažené

Daný pohyb dosiahol 100% úspešnosť v 30 pokusoch.

Vstávanie - Vstávanie z chrbta

- vystretie tela a pripaženie rúk
- rýchle predpaženie a čiastočné rozkročenie, vďaka čomu sa hráč čiastočne posadí

- dokončenie rozkročovania a pokrčenia nôh, a zároveň opätovné pripaženie, čím sa hráč prevráti na brucho a skončí v polohe, ktorú opisuje druhá odrážka v opise vstávania z brucha
- následne pohyb prebieha ako vstávanie z brucha

Pohyb dosiahol 100% úspešnosť v 30 pokusoch.

Pád vpred a pád vzad

Tieto pomocné pohyby sú založené na jednoduchej rýchlej zmene uhla kĺbu členku, ktorý je zodpovedný napnutie a pritiahnutie nártu. Boli vyvinuté čisto pre potreby testovania vstávania.

Chôdza - Drobná pomalá chôdza dopredu

Táto chôdza je veľmi pomalá a jej účelom je predovšetkým presné priblíženie k lopte bez jej odkopnutia.

- hráč prenesie váhu na opornú nohu a druhú nohu zdvihne a mierne vystrie v kolene, čím ju predsunie pred opornú nohu, zároveň rukou na strane opornej nohy pohne v ramene dopredu
- dostúpi na predsunutú nohu a prenesie na ňu váhu kvôli ďalšiemu kroku

Počas 10 testovacích kôl po 3 minúty bol výsledok testu 100% a hráč nespadol.

Chôdza - Drobná pomalá chôdza dozadu

- mierne odrazenie sa z päty pri predkopnutí nohy, lebo päta sa jemne zachytí o zem
- mierne zanoženie pri vrátení predkopnutej nohy späť na zem

Počas 10 testovacích kôl po 3 minúty bol výsledok testu 100% a hráč nespadol.

Chôdza - Drobčivá rýchla chôdza dopredu

Táto chôdza má rovnaké fázy ako predošlé dve chôdze, odlišuje sa len v hodnotách otočení jednotlivých kĺbov a je pri nej výraznejšie prenášanie váhy na opornú nohu pri kroku. Je určená hlavne na prekonávanie väčších vzdialeností.

Otáčanie – Otočenie o 90°

- mierne pokrčenie nôh a rúk v inicializačnej fáze pre zvýšenie stability priblížením ťažiska k zemi
- prenesenie váhy na opornú nohu a otočenie druhej nohy v bedrovom kĺbe úplne na stranu
- prenesenie váhy na otočenú nohu a prinoženie pôvodnej opornej nohy
- uvedenie pohybovaných kĺbov do pozície, v ktorej boli na konci inicializačnej fázy pred ďalším pokračovaním

Hráč ani po 10 minútach opakovania nestratil rovnováhu, preto by mal byť pohyb stabilný.

Kop do lopty – Priamy kop do lopty

- Počiatočná fáza prikrčenia sa dostane robota do polohy, z ktorej môže kop vykonať čo najstabilnejšie. Prikrčenie posunie ťažisko robota nižšie a zvýši tak stabilitu. Do tejto fázy sa zapájajú kĺby bedier, kolien a členkov.
- Druhá fáza vychýlenia tela do strany získa pre agenta priestor na samotný kop a zabezpečí, že po zdvihnutí nohy (opačnej ako je smer naklonenia agenta) sa agent nepreváži na opačnú stranu. V tejto časti pohybu sa okrem kĺbov členkov a bedier zapájajú aj ramenné kĺby ruky.

- Treťou fázou je zodvihnutie nohy. Ide o spoluprácu bedrového, kolenného kĺbu a kĺbu členka s cieľom nezavadiť nohou počas zdvíhania o podložku.
- Štvrtá fáza je samotný výkop so zapojením troch kĺbov – členku, kolena a bedra. Cieľom bolo zapojiť väčší počet kĺbov a dosiahnuť tak sčítanie momentov.
- Posledná fáza je stabilizačná, slúži na návrat agenta do vzpriamenej polohy.

Z 30 pokusov hráč pri kope ani raz nespadol, ale presnosť kopu nebola meraná.

Kop do lopty – Kop do lopty hranou chodidla

- Prvá fáza je založená na prvej fáze priameho kopu – agent sa prikrčí do polosedu, aby tak získal lepšiu stabilitu.
- V druhej fáze sa agent naváži na jednu stranu, aby tak udržal rovnováhu po zodvihnutí opačnej nohy, ktoré nasleduje v ďalšej fáze. Zapája kĺby členkov, bedier, torzo sa mierne nakláňa dopredu kvôli stabilite a rovnováha sa vyvažuje aj rukou.
- V ďalšej fáze agent zapojením kĺbov bedra, kolena a členku vysunie nohu pred telo a zároveň mierne do boku, aby si tak prichystal vhodnú pozíciu na kop.
- Štvrtá fáza je samotný kop, ktorý realizujú bedrové kĺby a zároveň ho vyvažujú ruky (kĺby ramena a lakťa).
- Posledná fáza je stabilizačná, slúži na návrat agenta do vzpriamenej polohy.

Z 30 pokusov hráč pri kope ani raz nespadol, ale presnosť kopu nebola meraná.

Bránenie – Pád brankára na bok

- v prvej fáze hráč zdvihne ruky šikmo nad hlavu, čo spôsobí, že jeho ťažisko sa presunie vyššie nad zem a bude ľahšie docieliť samotný pád, a zároveň mu to pri dopade pomôže ostať ležať na boku miesto prevrátenia na chrbát alebo na brucho
- v ďalšej fáze pokrčí nohu na strane, na ktorú má spadnúť a zároveň ruku na tej strane vychýli mierne do boku, čím presunie ťažisko na stranu
- následne sa pohybom členku vystretej nohy odrazí dostatočne na to, aby vyvolal samotný pád a miernym napnutím členku na pokrčenej nohe zabezpečí, že bude padať priamo na bok a nepretočí sa počas pádu tvárou k zemi
- počas pádu vychýlenú ruku vráti do polohy rovnobežnej s druhou rukou a napne oba členky, čím ešte kúsok zväčší priestor, ktorý bude svojím telom kryť

Bránenie – Prevrátenie na chrbát

- v prvej fáze sa hráč presunie nezaťaženú ruku a nohu za telo a zaťaženú ruku smerom pred telo, čo spôsobí, že sa prevráti na chrbát
- v druhej fáze pripaží a vráti bedrové kĺby do neutrálneho uhla

Úprava editora pohybov

Hlavnou úpravou v editore pohybov bol XML export pohybov v tvare pre agenta JIM. Následne rozšírenie o funkcionalitu symetrických a previazaných kĺbov.

Editor pohybov je napísaný v jazyku C#, kde na generovanie XML súborov je použitá knižica System.Xml. Na zmenu od exportovania pohybov pre agenta Sirius na agenta Jim boli upravené niektoré metódy.

Formát XML

Hlavička - otvárací tag <robot>

Constants - obsahuje parametre, namiesto číselných hodnôt. Možno pomocou nich napr. hromadne meniť polohu kĺbov.

Low_skills – tag <low_skill> názov pohybu

Phases - Medzi tagmi <phases> a </phases> sú jednotlivé fázy pohybu tak, ako boli vytvorené v editore. Každá fáza má jedinečný názov. V novom XML exporte sú názvy fáz v tvare nazov_pohybu_phase<číslo>.

Nový tvar efektorov tak obsahuje už iba názov a koncovú hodnotu natočenia príslušného kĺbu.

Začiatok fázy mínus koniec fázy je uzavretý v tagoch <duration> a </duration>.

Symetria pohybov

Po inšpirácii tímom Kouretes, implementovali do editora pohybov políčko pre spoločný pohyb symetrických kĺbov. Políčko určuje či pohyby budú:

- Nezávislé – každý kĺb sa pohybuje nezávisle od svojho symetrického spoločníka
- Symetrické – dva symetrické kĺby sa musia pohybovať po symetrických trajektóriách
- Zrkadlové – dva symetrické kĺby sa musia pohybovať po zrkadlovej trajektórii

Zimný semester

Testovací framework – tréner

Testovací framework vznikol pre potreby aplikácie strojového učenia.

Funkcionalita daného frameworku:

- vytvorenie špecifickej udalosti na ihrisku (+ realizácia)
- spracovávanie priebehu udalosti
- vyhodnotenie udalosti
  - metrika úspešnosti
  - vzdialený podnet na úpravu rozhodovania agenta adresovaná priamo jemu

Plánovanie agenta

Plánovací modul agenta JIM pozostáva z dvoch komponentov – komponent správania a komponent plánovania.

Komponent plánovania agenta JIM

Na implementáciu plánovacieho modulu bol použitý produkčný systém. Všetky pravidlá sú zapísané a následne podľa dopytu je zvolená akcia, ktorá je naplánovaná a zaradená do plánovača. Nevýhodou tohto prístupu môže byť pomalšie vyhodnocovanie pravidiel.

Komponent správania agenta JIM

Pravidlá sú zapísané v jednoduchom tvare, if podmienok. Jednotlivé pravidlá boli navrhnuté nasledovne:

1. Ak nevidím loptu => otočím hlavu

2. Ak vidím loptu a mám otočenú hlavu => otočím sa smerom k lopte
3. Ak vidím loptu => idem k lopte
4. Ak mám loptu => kopnem k bráne súpera
5. Ak spadnem => vstanem
6. Ak som brankár => som len pri bráne
7. …

Toto sú najjednoduchšie a najzákladnejšie pravidlá spávania. Vďaka produkčnému systému je veľmi jednoducho pozmeňovať staré či pridávať nové. Toto patrí medzi pár odporúčaní pre ďalšie tímy.

1.5 Zoznam zahraničných tímov

1.5.1 CIT3D Soccer Simulation

Viac informácií je dostupných na: http://hedayat.fedorapeople.org/misc/2010tdps/cit3d_tdp.pdf

V roku 2011 na majstrovstvách v Istanbule sa tím umiestnili na druhom mieste.

Tento tím má dobre rozpracovanú chôdzu - walking model. Chôdza je implementovaná podľa ľudského vzoru. Pri pohybe sa nevyužívajú ruky robot robí malé kroky a drží si stabilitu.

Tím má dobre rozpracovanú techniku vstávania. Agent sa dokáže postaviť za 1.6 sekundy.

Obrázok 27: Vstávanie agenta tímu CIT3D

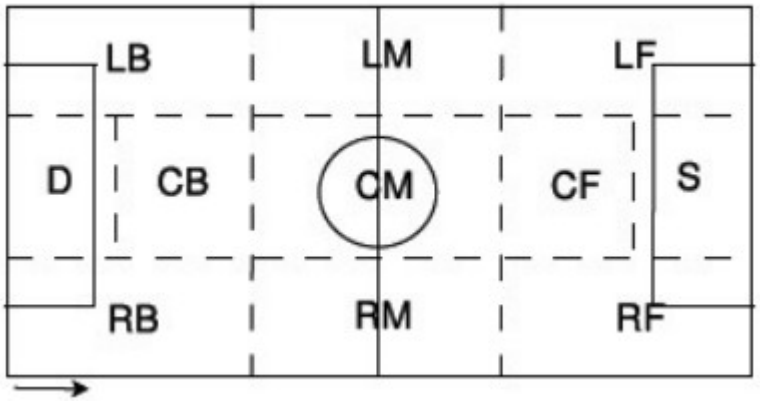
1.5.2 Sue Red Sun

Viac informácií je dostupných na: http://hedayat.fedorapeople.org/misc/2010tdps/seuredsun_TDP.pdf.

Tím má dobrú architektúru hráča. Skladá sa z jadra a prídavných modulov. Prídavne moduly sa dajú flexibilne meniť, čím vie tím jednoducho reagovať na zmeny.

Zaujímavé je, že tím má implementovanú viacrozmernú chôdzu. Čo znamená, že agent dokáže meniť smer chôdze bez potreby zastavenia. Zmena smeru je dynamická naproti agentom ktorý musia zastaviť pootočiť sa a pokračovať.

Tím sa zaoberal zjednodušením rozhodovacieho procesu pri prihrávke. Ihrosko je rozdelené na oblasti do ktorých môže agent poslať loptu. Napriti iným agentom agent jednoducho odkopne loptu do oblasti a nie na presne stanovenú pozíciu.



Obrázok 28: Oblasti na ktoré je rozdelené ihrisko

1.5.3 UT Austin Villa

Viac informácií je dostupných na http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/

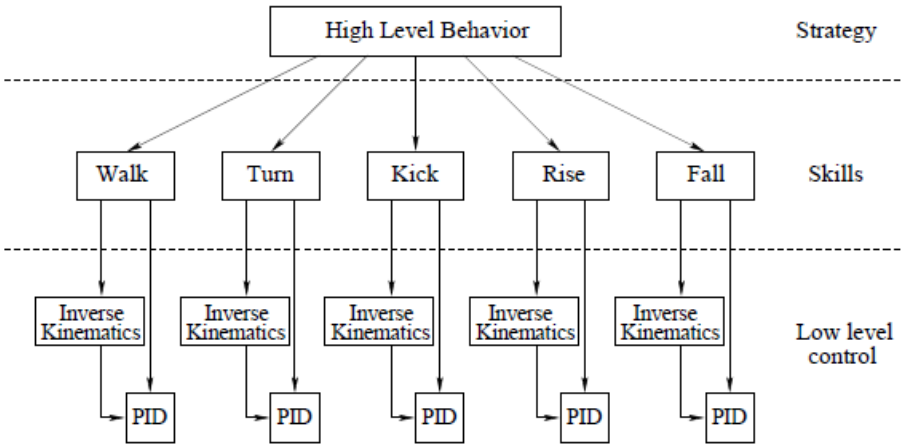
Tento tím vyhral ročník 2011 v Istanbule. Tím pochádza z Ameriky z Texaskej univerzity v Austine.

Tím vytvoril testovacie prostedie, ktoré simuluje reálny svet. Tím vytvoril a na ich stánke je dospný na stiahnutie agent typu:

- Coach – ide o agenta, ktorý je schopný poskytovať iným agentom strategické rady (o hre prostredníctvom ručne zadaných pravidiel a o súperovi prostredníctvom pravidiel naučených zo súperovej hry), Linux binary
- Agenta typu Coachable player – agenti, ktorí dokážu zapracovať rady od coach agenta do svojej hry, postavení na agentoch tímu UvA Trilearn

Zaujímavá je architektúra tímu, ktorá je rozdelená na:

- Stratégia
  - vysokoúrovòové správanie
- Zručnosti
  - chôdza
  - otočenie
  - kop
  - vstávanie
  - pád
- Nízkoúrovňové ovládanie
  - ovládanie kĺbov cez PID ovládače



Obrázok 29: Architektúra tímu Ut Austin Villa

Skill Statistic	Performance Value
Forward walking: linear velocity	(1.07 ± 0.00) m/sec
Side walking: linear velocity	(0.62 ± 0.01) m/sec
Backward walking: linear velocity	(1.03 ± 0.00) m/sec
Turning: angular velocity	(112.03 ± 0.24) deg/sec
Kicking: distance reached by ball after kick	(5.09 ± 0.07) m
Rising: Time to rise after falling forwards	(2.55 ± 0.10) sec
Rising: Time to rise after falling backwards	(2.10 ± 0.10) sec

Obrázok 30: Výkonostná tabuľka tímu

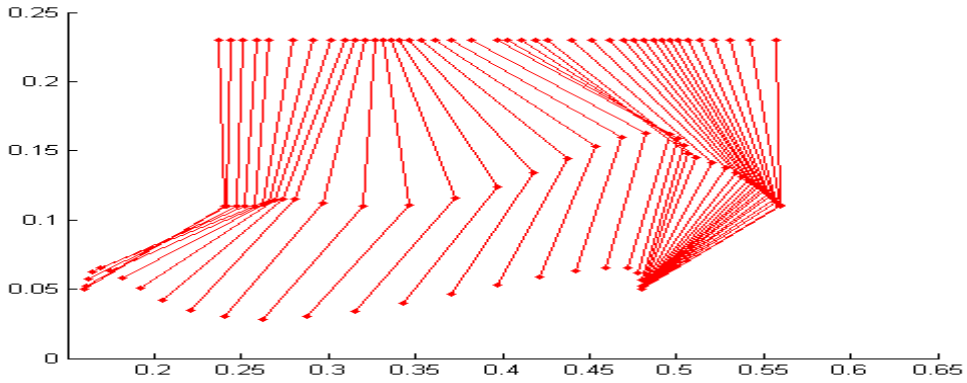
1.5.4 Apollo 3D

Viac informácii je dostupných na http://sourceforge.net/projects/apollo3d/

V roku 2011 sa dostali do finále. Robocup 3D vyhrali v roku 2010.

Tím má dobre rozpracovanú techniku na presné rozpoznanie pozície hráča.

Tím sa hĺbkovo zaoberal zlepšovaním chôdze. Z nižšie uvedeneného grafu je vidno ako sa im podarilo vyladiť proces chôdze.



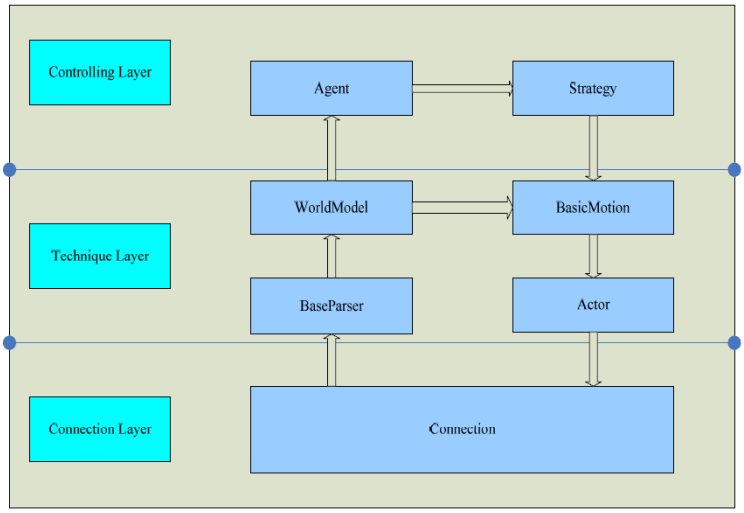
Obrázok 31: Graf popiseuje proces chôdze tímu Apollo 3D

1.5.5 KylinSky

Viacej informácii je dostupných na http://hedayat.fedorapeople.org/misc/2010tdps/kylinsky_TDP.pdf

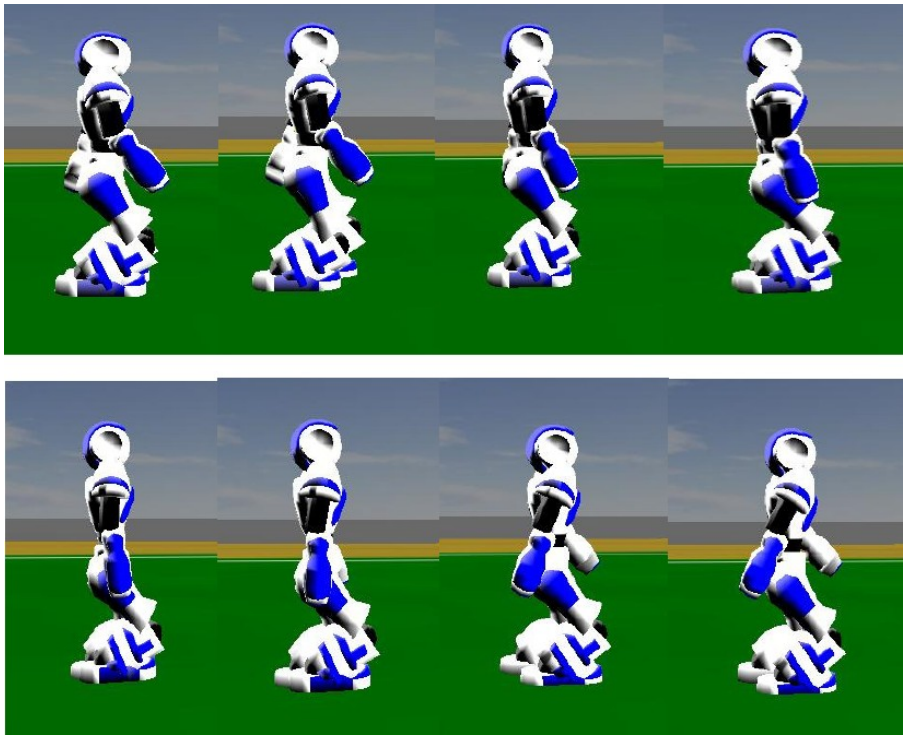
Hong Jiang, Chaohong Cai V roku 2011 sa dostali do finále.

Zaujímavá je architektúra agenta. Achitektúra agenta tímu KylinSky3D sa skladá z troch hlavných vrstiev, ktoré spolu navzájom spolupracujú. Ide o vrstvu pripojenia, technickú vrstvu a kontrolnú vrstvu. Vrstva pripojenia sa stará o komunikáciu agenta so serverom. Technická vrstva tvorí akési jadro a skladá sa zo štyroch častí: WorldModel, BasicMotion, BaseParser a Actor. Kontrolná vrstva obsahuje stratégie agenta pri hre.

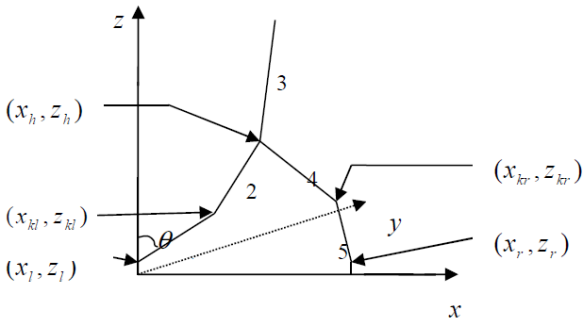


Obrázok 32: Architektúra agenta tímu KylinSky

Pri chôdzi agent tohto tímu používa aj ruky.



Obrázok 33: Fázy chôdze agenta tímu KylinSky



Obrázok 34: Model chodidiel agenta tímu KylinSky

1.5.6 Boldhearts

Viacej informácii je dostupných na http://hedayat.fedorapeople.org/misc/2010tdps/boldhearts_tdp.pdf

Tím sa zaoberal optimalizáciou chôdze za pomoci optimalizácie krokov.

1.5.7 RoboCanes

Viac informácii je dostupných na http://hedayat.fedorapeople.org/misc/2010tdps/robocanes_TDP.pdf

Agent tímu robocanes sa postupom časú učí. Na základe svojich znalostí robí rozhodovanie a kontrolu.

Tím má veľmi dobré spracovaný editor pohybov, ktorý má názov roboViz

Obrázok 35: Editor RoboViz tímu RoboCanes

2 Šprint 2

2.1 Reprezentácia pohybov – framework na tvorbu anotácie

2.1.1 Analýza

Aby sa vedel hráč rozhodovať, ktorý pohyb naplánovať v danej hernej situácii, bezpodmienečne musí byť informovaný o tom, čo robí ktorý pohyb (aký bude mať vplyv jeho naplánovanie na budúcnosť, ktorý z pohybov je vhodné vybrať v danej situácii). Preto je potrebné vyrobiť anotáciu pre pohyby, vďaka ktorej bude možné automatizovať ich plánovanie.

Požiadavky na anotácie:

1. strojová spracovateľnosť (počítač musí vedieť robiť rozhodnutia na základe anotácie)
2. obšírnosť (čím viac informácií hráč má, tým lepšie sa vie rozhodovať)
3. automatizovateľnosť (vlastnosti pohybov by malo byť možné vypĺňať automaticky, analýzou ich reálneho behu, s minimálnym prispením človeka)
4. zrozumiteľnosť (je možné, aby im porozumel aj človek – mohol ich vytvárať, čítať a editovať)
5. všeobecnosť (množina atribútov reprezentujúcich pohyby by mala byť pre všetky pohyby rovnaká – táto vlastnosť zaručí jednoduchosť práce s anotáciami)

Počas analýzy sme identifikovali niektoré základné atribúty, ktoré musí anotácia obsahovať:

- *typ pohybu* – hráč si na základe neho môže pre danú situáciu vyselektovať množinu teoreticky použiteľných pohybov (napr. ak sa potrebuje presunúť z jedného miesta na druhé, bude vyberať spomedzi pohybov typu „chôdza“, v inej situácii typ „kop“).
- *zmena pozície a natočenia hráča* počas trvania pohybu – táto znalosť je potrebná pri výbere pohybu, ak sa hráč potrebuje presunúť alebo otočiť. Môže byť rozhodujúca aj za okolností, že posunutie ši otočenie hráča v danej situácii nie je vítaná vlastnosť pohybu.
- *dĺžka trvania pohybu* – na jej základe si môže vybrať napr. najrýchlejší dostupný pohyb s potrebnými vlastnosťami.
- *pravdepodobnosť pádu* počas vykonávania pohybu - na jej základe si môže vybrať napr. najspoľahlivejší dostupný pohyb s potrebnými vlastnosťami.
- *predpodmienky* – hráč si vyberie taký pohyb, ktorý nadväzuje na jeho aktuálny stav.
- *stav po vykonaní pohybu* – umožňuje hráčovi plánovanie ďalších pohybov do budúcnosti, skladanie pohybov.

Je dôležité si uvedomiť, že rovnaký pohyb je vykonaný vždy trochu inak – preto je potrebné pre popísanie pohybu používať priemerné hodnoty z viacerých behov daného pohybu.

Pri počítaní zmeny pozície hráča v dôsledku vykonania pohybu je dôležité si uvedomiť, že hráč vie počítať zmenu svojej pozície iba z pohľadu ihriska, nás ale zaujíma zmena pozície z pohľadu hráča. Preto je potrebné túto hodnotu počítať na základe kombinácie informácií o zmene pozície na ihrisku a otočenia hráča pred vykonaním pohybu.

Na druhú stranu potom hráč bude musieť vedieť pri plánovaní pohybu prepočítať na základe anotácie a svojho otočenia, aký bude mať vplyv vykonanie daného pohybu na jeho pozíciu v rámci ihriska.

2.1.2 Návrh

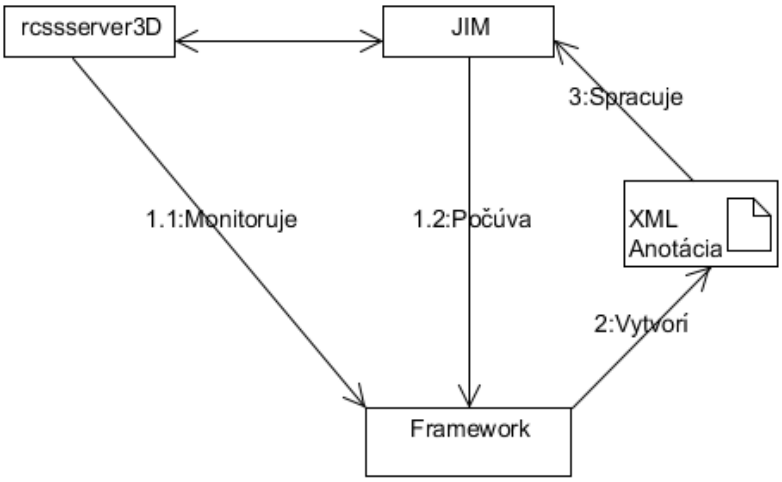
Reprezentácia údajov

Na základe uvedených požiadaviek sme sa rozhodli ukladať anotácie k pohybom v samostatných súboroch typu XML. Takto dostupné anotácie je možné pomerne jednoducho spracovávať a meniť tak človekom ako i strojom. V prípade potreby je možné takúto reprezentáciu anotácie vložiť priamo do XML súboru pohybu. V tabuľke 16 sú popísané všetky podstatné atribúty anotácie.

Proces automatizácie vytvárania anotácie

Napriek tomu, že človek musí vyplniť niektoré logické atribúty anotácie (typ pohybu, poznámka), väčšina je spracovateľná automaticky a naopak, pre človeka je náročné takého hodnoty zisťovať (napr. priemerné otočenie hráča počas pohybu). Preto navrhujeme nasledovný spôsob automatizácie vytvárania anotácie (Obrázok 36) zložený z 3 krokov:

1 *Zbieranie údajov:*
   1.1 Framework vie na základe monitorovania servera presné údaje o stave ihriska (kde je hráč, ako je otočený a pod).
   1.2 Hráč plánuje svoje pohyby, preto vie presne povedať, kedy začal a kedy skončil vykonávanie akého pohybu.
2 *Vyhodnocovanie údajov:* Framework takéto informácie z oboch zdrojov dokáže spájať v zmysle zisťovania vlastností jednotlivých pohybov a exportovať na základe nich súbory s XML anotáciou pohybu (priemerné hodnoty pre každý použitý pohyb).
3 *Spracovanie anotácie:* Hráč dokáže takto vytvorenú XML anotáciu využiť pri plánovaní pohybov.



Obrázok 36: Proces spracovania údajov

Vysvetlivky k XML reprezentácii anotácie

Značka	Význam
`<anotation>`	Koreňová značka
`<name>`	Meno pohybu, na ktorý sa anotácia vzťahuje
`<checksum>`	Kontrolná suma vzťahujúca sa na XML anotovaného pohybu, aby sa zabránilo nekonzistentnosti anotácie a pohybu v dôsledku jeho zmeny.
`<type>`	Typ pohybu, kategória. Môže nadobúdať hodnoty: `kick`, `walk`, `fall`, `look`, `stand_up`, `rotation`, `other`. Môže mať aj viacero z týchto hodnôt (napr. ak sa jedná o zložitejší pohyb).
`<duration>`	Priemerná dĺžka trvania pohybu v milisekundách.
`<rotation>`	Priemerná zmena natočenia hráča počas vykonania pohybu. Sú v ňom vnorené značky určujúce hodnoty otočenia po jednotlivých osiach v stupňoch. (Hodnoty otáčania okolo osi X a Y sú zatiaľ nepovinné, keďže ich využitie je momentálne nepravdepodobné)
`<move>`	Priemerná zmena polohy hráča počas vykonania pohybu. Sú v ňom vnorené značky určujúce hodnoty posunu po jednotlivých osiach z pohľadu hráča v metroch.(Hodnota posunu hráča po osi Z je zatiaľ nepovinná, keďže jej využitie je momentálne nepravdepodobné)
`<fall>`	Priemerný počet pádov hráča počas 100 vykonaní daného pohybu.
`<note>`	Poznámka autora stručne opisujúca daný pohyb v ľudskej reči.
`<preconditions>`	Obsahuje informácie o predpodmienkach pre vykonanie daného pohybu (v akom stave sú kĺby a aká je poloha hráča na začiatku pohybu). Obsahuje značky `<joints>` a `<lying>`
`<end_state>`	Obsahuje informácie o stave hráča po vykonaní daného pohybu. Obsahuje značky `<joints>` a `<lying>`
`<joints>`	Obsahuje značky označujúce jednotlivé kĺby a hodnotu ich natočenia. Prednastavená hodnota pre každý kĺb je 0. Značky pre jednotlivé kĺby a ich ohraničenia sú kompatibilné so značkami použitými pre pohyb hráča (v prípade potreby dostupné v XSD súbore)
`<lying>`	Informácia o polohe hráča. Môže nadobúdať hodnoty: `false`, `on_back`, `on_belly`, `on_side`.

Tabuľka 16: Vysvetlivky pre XML anotáciu pohybov

2.2 Analýza pohybu hráča. Otestovanie pohybov

V tejto kapitole je vykonaná analýza pohybov hráča, ktorého nám zanechal tím Androids. Ku každému z pohybov je napísaný krátky opis a zhodnotenie pohybu. Každý pohyb tiež obsahuje tabuľku testov, ktorá hovorí o stabilite jednotlivých iterácií testovaného pohybu. Vo všeobecnosti sú rýchlosti vykonania pohybov na dobrej úrovni.

Legenda pre testovaciu tabuľku:

- Č.: číslo testu
- Stabilita: 1 – pohyb je stabilný, 2 – pohyb nie je stabilný, 3 – hráč spadol

Pohyb *fall_left*

Hráč zámerne spadne na ľavý bok.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Tabuľka 17: Test pohybu fall_left

Pohyb je v poriadku a hráč vždy spadne na požadovaný bok. Nedokáže však vstať.

Pohyb *fall_right*

Hráč zámerne spadne na pravý bok.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 18: Test pohybu fall_right

Pohyb je v poriadku a hráč vždy spadne na požadovaný bok. Nedokáže však vstať.

Pohyb *head_left_120*

Hráč otočí hlavou o 120 stupňov napravo.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 19: Test pohybu head_left_120

Pohyb je veľmi stabilný a rýchlosť jeho vykonania je optimálna.

Pohyb *head_right_120*

Hráč otočí hlavou o 120 stupňov naľavo.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 20: Test pohybu head_right_120

Pohyb je veľmi stabilný a rýchlosť jeho vykonania je optimálna.

Pohyb *kick_left*

Hráč sa prikrčí a vykoná jednoduché kopnutie špičkou ľavej nohy pred seba.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 21: Test pohybu kick_left

Pohyb je veľmi stabilný a rýchlosť jeho vykonania je optimálna. Sila kopu je primeraná. Počas vykonávania tohto pohybu sa hráč vychýli o pár stupňov na pravú stranu.

Pohyb *kick_right*

Hráč sa prikrčí a vykoná jednoduché kopnutie špičkou pravej nohy pred seba.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 22: Test pohybu kick_right

Pohyb je veľmi stabilný a rýchlosť jeho vykonania je optimálna. Sila kopu je primeraná. Počas vykonávania tohto pohybu sa hráč vychýli o pár stupňov na ľavú stranu.

Pohyb *kick_straight_edge_l*

Hráč sa prikrčí a vykoná kopnutie pred seba vnútornou časťou chodidla ľavej nohy.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 23: Test pohybu kick_straight_edge_l

Pohyb je relatívne stabilný a rýchlosť jeho vykonania je optimálna. Kop je však veľmi slabý a lopta musí byť v bezprostrednej blízkosti hráča.

Pohyb *kick_straight_edge_r*

Hráč sa prikrčí a vykoná kopnutie pred seba vnútornou časťou chodidla pravej nohy.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 24: Test pohybu kick_straight_edge_r

Pohyb je relatívne stabilný a rýchlosť jeho vykonania je optimálna. Kop je však veľmi slabý a lopta musí byť v bezprostrednej blízkosti hráča.

Pohyb *nachrbat*

Hráč začne vystierať ruky pred seba, pomocou čoho sa jednoducho dostane na chrbát pokiaľ leží na boku.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 25: Test pohybu nachrbat

Toto je veľmi užitočný pohyb, pretože sa pomocou neho dokáže hráč efektívne prevaliť z boku na chrbát.

Pohyb *padvpred*

Hráč zámerne padne dopredu.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 26: Test pohybu padvpred

Pád aj postavenie sú efektívne.

Pohyb *padvzad*

Hráč zámerne padne dozadu.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 27: Test pohybu padvzad

Pád aj postavenie sú efektívne.

Pohyb *rollback*

Hráč sa nastaví do začiatočnej polohy

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 28: Test pohybu rollback

Pohyb *sidekick_left*

Hráč sa prikrčí a kopne vnútornou časťou ľavej nohy bokom do lopty tak, že ju pošle napravo.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 29: Test pohybu sidekick_left

Pohyb je stabilný ale relatívne pomalý. Sila kopnutia je však veľmi nízka, pohyb môže slúžiť jedine na akési posunutie lopty.

Pohyb *sidekick_right*

Hráč sa prikrčí a kopne vnútornou časťou pravej nohy bokom do lopty tak, že ju pošle naľavo.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 30: Test pohybu sidekick_right

Pohyb je stabilný ale relatívne pomalý. Sila kopnutia je však veľmi nízka, pohyb môže slúžiť jedine na akési posunutie lopty.

Pohyb *sit_down*

Hráč rozkročí nohy a pokúsi sa sadnúť na zem.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	3	3	3	3	3	3	3	3	3	3	3

Table 31: Test pohybu sit_down

Pohyb je extrémne nestabilný. Hráčovi sa počas testovania nepodarilo sadnúť ani jeden krát, vždy spadol na chrbát alebo na brucho. Pokiaľ sa prevážil na brucho, nevedel sa postaviť.

Pohyb *sit_stand*

Hráč rozkročí nohy a následne ich znovu spojí.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	2	2	3	2	1	3	1	2	2

Table 32: Test pohybu sit_stand

Tento pohyb je relatívne nestabilný. Koncová pozícia nôh nezodpovedá štartovacej.

Pohyb *stand_back*

Hráč sa postaví z polohy ležmo na chrbte.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 33: Test pohybu stand_back

Pohyb je stabilný a hlave rýchly.

Pohyb *stand_front*

Hráč sa postaví z polohy ležmo na bruchu.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 34: Test pohybu stand_front

Pohyb je stabilný a hlave rýchly.

Pohyb *stepleft*

Hráč vykoná ľavou nohou krok do boku.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	2	2	2	2	2	2	2	2	2	2	2

Table 35: Test pohybu stepleft

Pomalý a nie úplne stabilný pohyb. Pokiaľ pokračuje po inom pohybe, hráč väčšinou spadne.

Pohyb *stepleft1*

Hráč vykoná ľavou nohou krok do boku.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	2	2	2	2	2	2	2	2	2	2	2

Table 36: Test pohybu stepleft1

Pomalý a nie úplne stabilný pohyb. Pokiaľ pokračuje po inom pohybe, hráč väčšinou spadne.

Pohyb *stepright*

Hráč vykoná pravou nohou krok do boku.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	2	2	2	2	2	2	2	2	2	2	2

Table 37: Test pohybu stepright

Pomalý a nie úplne stabilný pohyb. Pokiaľ pokračuje po inom pohybe, hráč väčšinou spadne.

Pohyb *stepright1*

Hráč vykoná pravou nohou krok do boku.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	2	2	2	2	2	2	2	2	2	2	2

Table 38: Test pohybu stepright1

Pomalý a nie úplne stabilný pohyb. Pokiaľ pokračuje po inom pohybe, hráč väčšinou spadne.

Pohyb *turnleft90*

Hráč sa prikrčí a otočí o približne 90 supňov doľava. Najskôr natočí jednu nohu a následne k nej priloží druhú, čím sa otočí.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	2	1	2	2	3	1	2	2	2

Table 39: Test pohybu turnleft90

Pohyb je relatívne stabilný, ale otočenie trvá pomerne dlho. Po páde sa hráč nevedel postaviť.

Pohyb *turn_left_cont_20*

Hráč sa otočí o 20 supňov doľava podobným spôsobom ako u predchádzajúceho pohybu.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	2	2	2	2	2	3	2	2	3	2	2

Table 40: Test pohybu turn_left_cont_20

Pohyb nie je veľmi stabilný, hráč občas spadol.

Pohyb *turn_left_cont_5*

Hráč sa otočí o 5 supňov doľava.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 41: Test pohybu turn_left_cont_5

Pohyb je stabilný, otočenie relatívne rýchle.

Pohyb *turnright90*

Hráč sa prikrčí a otočí o približne 90 supňov doprava. Najskôr natočí jednu nohu a následne k nej priloží druhú, čím sa otočí.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	2	2	2	2	1	2	2	2	2

Table 42: Test pohybu turnright90

Pohyb je relatívne stabilný, ale otočenie trvá pomerne dlho.

Pohyb *turn_right_cont_20*

Hráč sa otočí o 20 supňov doprava podobným spôsobom ako u predchádzajúceho pohybu.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	2	2	2	2	2	2	2	2	2	2	2

Table 43: Test pohybu turn_right_cont_20

Pohyb nie je veľmi stabilný, hráč občas spadol.

Pohyb *turn_right_cont_5*

Hráč sa otočí o 5 supňov doprava.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 44: Test pohybu turn_right_cont_5

Pohyb je stabilný, otočenie relatívne rýchle.

Pohyb *walkback2*

Hráč vykoná pomocou špičiek krok dozadu.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	2	2	2	2	3	2	2	2	2	2	2

Table 45: Test pohybu walkback2

Pohyb je relatívne pomalý a nie veľmi stabilný.

Pohyb *walk_fine_back*

Hráč sa prikrčí a vykoná krok dozadu.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 46: Test pohybu walk_fine_back

Pohyb je síce stabilný ale extrémne pomalý. Posunutie o jeden krok prakticky nie je zaznamenateľné.

Pohyb *walk_fine_fast1*

Hráč sa prikrčí a vykoná relatívne veľký krok dopredu.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	2	2	2	3	2	2	3	2	2	3	2,3

Table 47: Test pohybu walk_fine_fast1

Pohyb je celkom rýchly ale veľmi nestabilný. Hráč sa pri pohybe neustále kolíše a počas chôdze sa postupne otáča doľava.

Pohyb *walk_fine_fast2*

Hráč sa prikrčí a vykoná relatívne veľký krok dopredu.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

Table 48: Test pohybu walk_fine_fast2

Pohyb je stabilný a celkom rýchly. Hráč sa otáča menej ako pri predošlom pohybe.

Pohyb *walk_fine_slow*

Hráč sa prikrčí a vykoná malý krok dopredu.

Číslo testu	1	2	3	4	5	6	7	8	9	10	Priemer
Stabilita	1	1	1	1	1	1	1	1	1	1	1

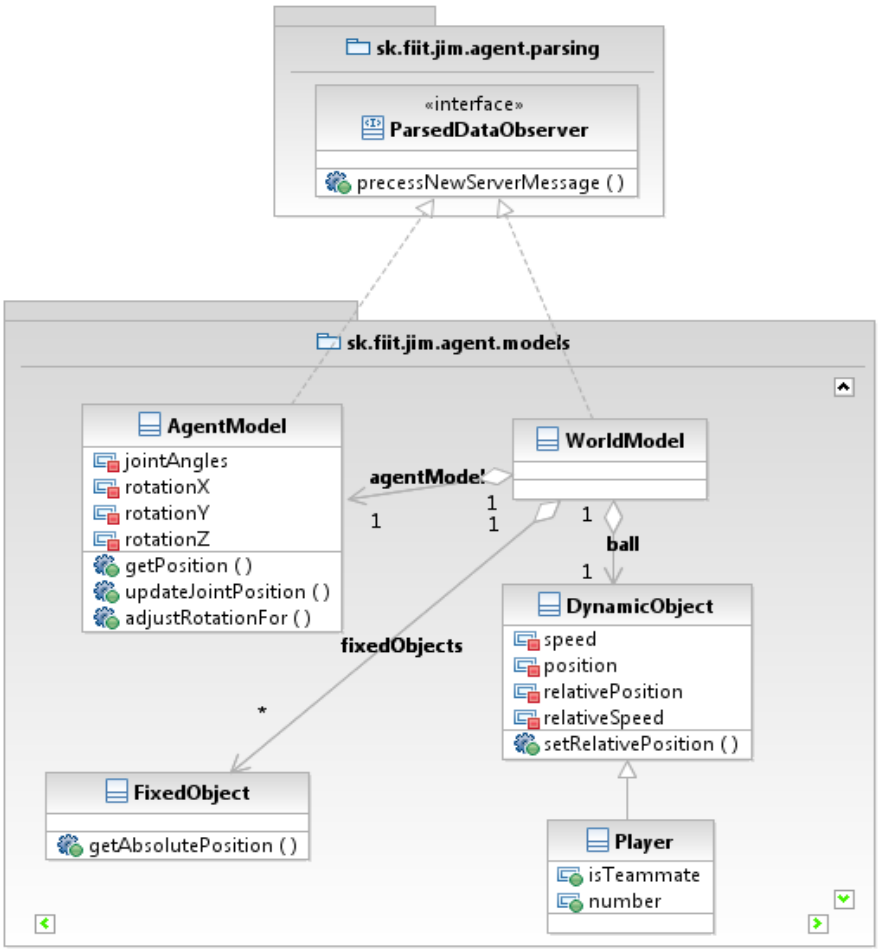
Table 49: Test pohybu walk_fine_slow

Pohyb je stabilný ale chôdza je veľmi pomalá. Môže slúžiť na úpravu polohy hráča pred loptou.

2.3 Model Sveta

2.3.1 Analýza a testovanie modelu sveta agenta Jim

Model sveta predstavuje v implementácii agenta Jim tímu Androids hlavná trieda WorldModel, ktorá udržiava a aktualizuje informácie o svete. Logické previazanie tried súvisiacich s modelom sveta je viditeľné na diagrame tried na obrázku 37.



Obrázok 37: Diagram tried agenta Jim tímu Androids.

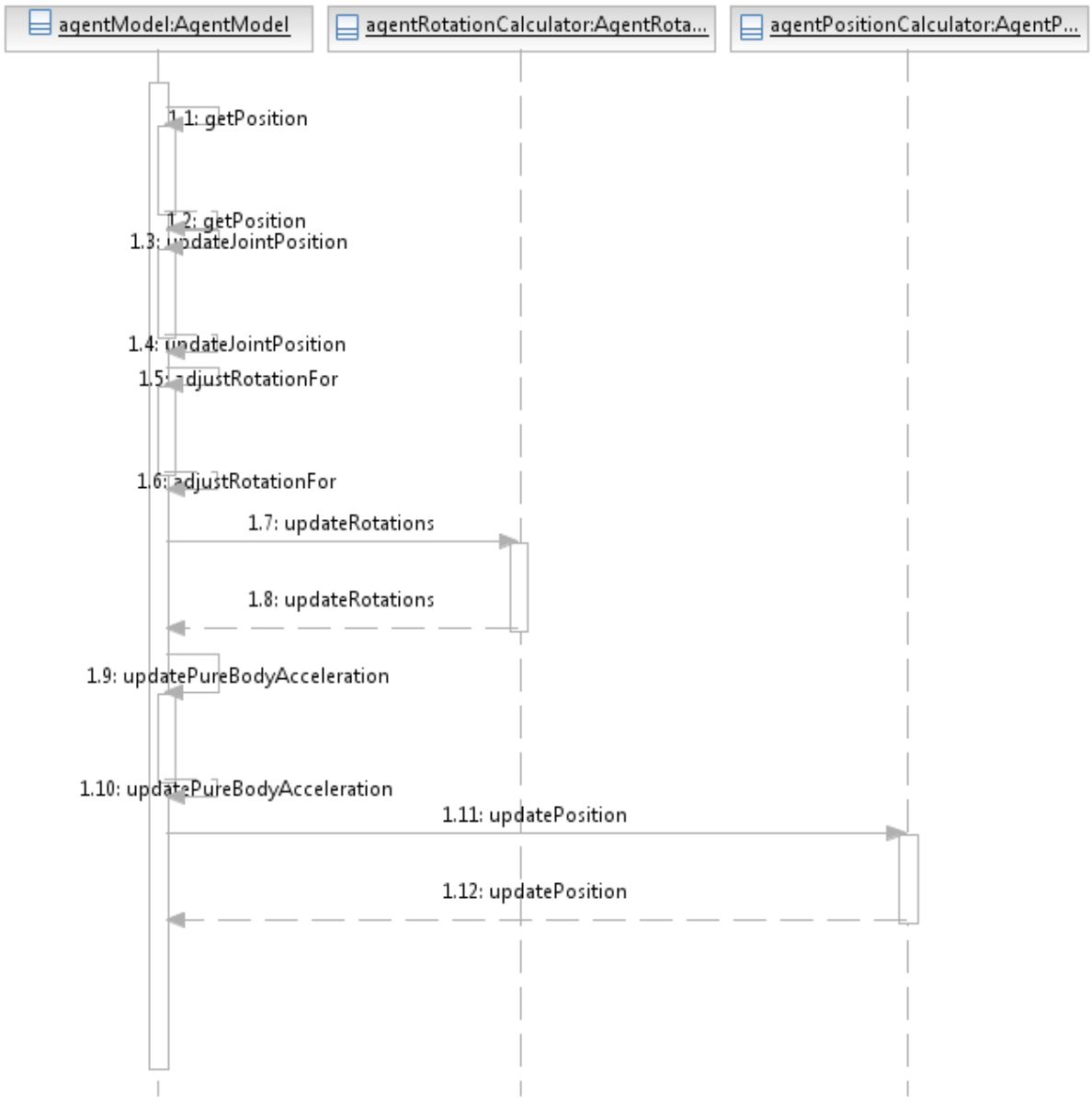
WorldModel aj AgentModel sú observery a dostávajú informácie zo servera v každom cykle prostredníctvom parsera. Správy sú už predspracované, napr. správy zo Seen perceptora sú prepočítané na relatívnu polohu videného predmetu vzhľadom na agenta, konkrétne na jeho stred hlavy, kde je umiestnený Seen perceptor.

Model Sveta

V aktuálnej verzii WordModelu sa aktualizujú dáta lopty a vytvára sa model Agenta.

Model agenta

Spracovanie správy AgentModelom je znázornené na nasledujúcom sekvenčnom diagrame na obrázku 38:



Obrázok 38: Sekvenčný diagram spracovania správy zo servera AgentModelom

V modeli sveta má agent informácie o sebe samom, ako sú aktuálna pozícia, rýchlosť, natočenie kĺbov a natočenie vzhľadom na hlavné osy.

Stav agenta

Metóda getPosition zisťuje na základe akcelerometra aktuálny stav agenta. Agent môže byť v stave Standing alebo On ground. Pričom On ground je ešte rozdelený na On belly(na bruchu) a na On Back(Na chrbte). Meranie týchto stavov je funkčné.

Test: sledovanie výpisov, pričom agent vykonával pohyb pád vpred(prípadne pád vzad)

Natočenie kĺbov

UpdateJointPosition ukladá informácie o aktuálnych otočeniach kĺbov, získaných zo správ zo servera.

Test: Debugovanie zdrojových kódov.

Otočenie agenta

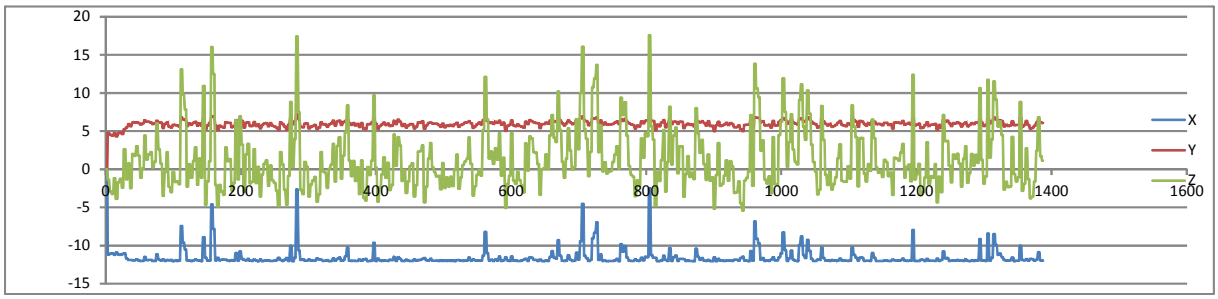
AdjustRotationFor počíta natočenie pomocou informácií z GyroRate perceptora, ktorý je ale umiestnený v trupe robota nao.

AgentRotationCalculator počíta natočenie z minimálne troch fixných bodom, ktoré vidí. Keďže Seen perceptor je v strede hlavy, nastáva tu problém s presným definovaním bodu v nao modeli vzhľadom na ktorý sa bude počítať natočenie.

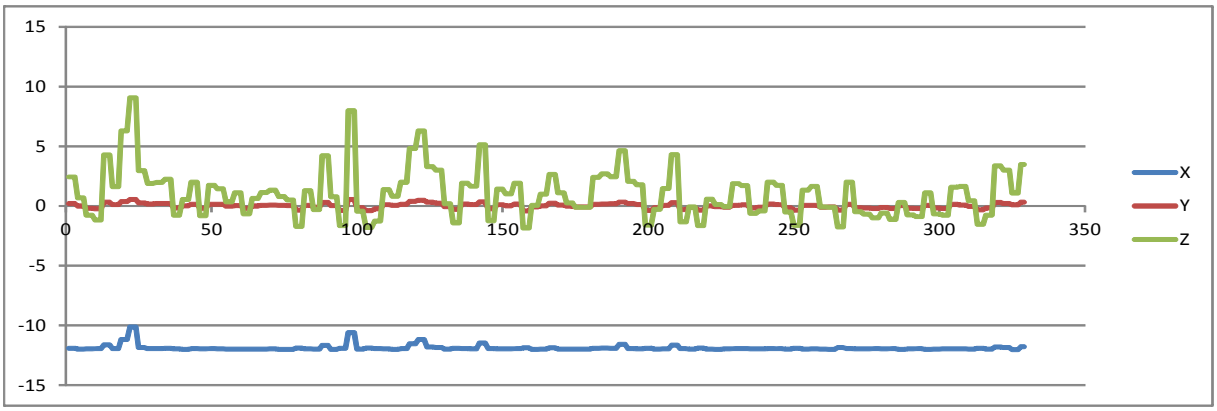
Aktuálna pozícia agenta v ihrisku

Aktuálna pozícia je počítaná z informácií o relatívnej polohe rohových zástavok, a zástavok na bránkach. Potrebná je minimálne jedna zástavka aby bolo možné určiť polohu zástaviek. Do správy o polohe videných objektov je serverom úmyselne vkladaná chyba, ktorá sa prejavuje hlavne, keď je robot viac vzdialený od objektu.

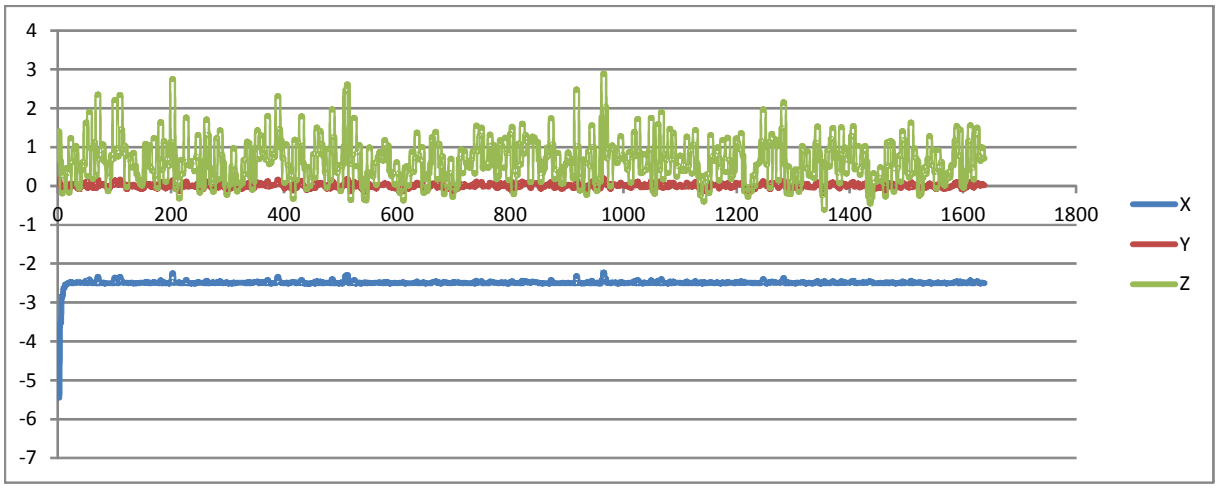
Test: Umiestnenie agenta beam efektorom na rôzne polohy v ihrisku a zaznamenávanie jeho polohy v čase, pričom sa agent fyziky nehýbal. Vybrané testy sú zaznamenané v nasledujúcich grafoch.



Obrázok 39: Testovanie polohy. Umiestnenie robota v bode [-11, 6, 0]



Obrázok 40: Testovanie polohy. Umiestnenie robota v bode [-11, 0, 0]



Orázok41 Testovanie polohy. Umiestnenie robota v bode [-1, 0, 0]

Cieľom testu bolo ukázať vplyv vzdialenosti od videných objektov na určenie polohy agenta. Agent bol vždy umiestnený vždy tak aby videl na rohové zástavky a bránku na opačnej strane ihriska a určoval polohu vždy z tých istých fixných bodov, pričom je od nich rôzne vzdialený. Výsledky testu ukazujú, že bližšie fixne objekty poskytujú presnejšie hodnoty.

2.3.2 Návrh vylepšení modelu sveta

Model spoluhráčov a protihráčov v poli

Seen perceptor poskytuje informácie aj o ostatných hráčoch, ktorých vidí na ihrisku. V správe o zvyšných hráčoch, ktorých agent vidí na ihrisku sa nachádza:

- číslo hráča
- poloha jednotlivých častí tela

Na základe toho navrhujeme vytvorenie modelu hráča, ktorý bude poskytovať dostupné informácie o príslušnom spoluhráčovi-protihráčovi. V tomto modeli by mala byť zaznamenaná poloha všetkých častí tela, ktoré agent na ostatných hráčoch zaregistruje.

Zlepšenie určenie pozície agenta

V súčasnej implementácie je určenie polohy agenta podľa fixných bodov na ihrisku. Tieto informácie nie sú postačujúce a nie sú dostupné vždy(agent môže byť natočený tak, že žiadnu zástavku nevidí). Server taktiež poskytuje informácie o čiarach, ktoré agent vidí na ihrisku. Hustota čiar na ihrisku je väčšia ako fixných bodov a je možné podľa nich vylepšiť určenie polohy agenta v hracom poli.

Zlepšenie určenia natočenia agenta

Navrhujeme presné určenie bodu, v strede trupu, vzhľadom na ktorý sa bude počítať natočenie. Gyroskop poskytuje celkom presné informácie a je umiestnený v strede trupu. Keďže sa ale na gyroskop nemôžeme úplne spoliehať, bude natočenie korigované aj správami zo seen perceptora, pričom toto natočenie sa prepočíta vzhľadom na stred trupu pomocou natočenia kĺbov krku robota.

Pohybový model dynamických objektov

Vytvoriť históriu zmien stavov(napríklad. pozícia) pre jednotlivé dynamické objekty v modeli sveta. Táto história môže byť použitá na následne odhady dynamického správania sa objektov s čoho je možné odvodiť základné akcie, ktoré ma agent vykonať.

2.4 Analýza možnosti paralelizácie výpočtov

2.4.1 Využitie školského superpočítača

Spracovanie 3D robotického futbalu je pomerne náročné na výpočtové prostriedky, preto sme sa rozhodli preskúmať možnosti jeho paralelizácie. Pre projekt môže priniesť možnosť paralelného spúšťania a testovania hráča úžitok vo forme rýchleho získavania väčšieho množstva relevantných údajov a skrátenia času pri opätovnom spúšťaní podobných scenárov. Týmto priamo otvára dvere rýchlej simulácii veľkého množstva scenárov a strojovému učeniu hráča.

Preto sme sa rozhodli preskúmať ako jedno z riešení paralelizácie využitie superpočítača dostupného na STU FIIT.

Školský superpočítač

Na fakulte je dostupný pre náročné paralelné výpočty nižšie popísaný hardware - jedná sa o sieťovo prepojené zariadenia nasledovnej konfigurácie:

- 1 x HP DL160
  - 2 x Intel Xeon L5520 (4x2.26GHz)
  - 24GB DDR RAM
  - 10Gb Ethernet
- 16 x HP BL460c G6
  - 2 x Intel Xeon X5570 (4x2.93GHz)
  - 36GB DDR RAM
  - 1Gb Ethernet
- zdieľané diskové pole HP StorageWorks 2000sa (48TB)

Pre projekt podstatný software:

- Debian GNU/Linux 6.0
- Java 1.6.0_24

Použiteľné technológie

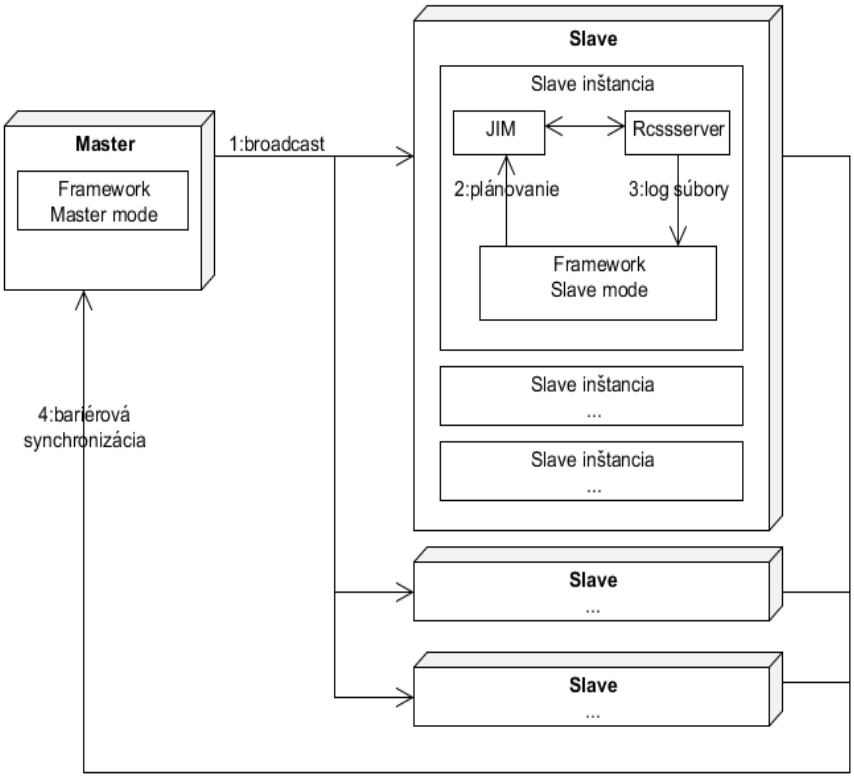
	Výhody	**Nevýhody**
MPI	-Priamočiare riešenie -Dobrá zrozumiteľnosť -Rýchle pochopenie architektúry začiatočníkom	-Žiadne skúsenosti v tíme s daným riešením na viacerých zariadeniach -Nie natívne riešenie pre Javu
Hadoop	-Dobré skúsenosti jedného člena tímu s daným riešením na školskom superpočítači -Napísaný v Jave	-Nie je stavaný na tento druh úloh -Zbytočné režijné náklady navyše -Ťažšie pochopiteľný pre začiatočníka

Tabuľka 50: Porovnanie použiteľných technológií

Návrh na paralelizáciu výpočtov

Po konzultácii s Ing. Petrom Lackom, PhD sme dospeli k záveru, že na pre paralelizáciu behov hráča JIM bude vhodnejšie použitie technológie MPI z nasledovných dôvodov. Architektúra daného riešenia bude vyzerať nasledovne (Obrázok 42):

- Framework bude môcť byť spustený v master a slave móde.
- Jedno zariadenie bude mať spustený master framework, ktorý bude spojený so všetkými slave frameworkami na slave zariadeniach.
- Každý slave framework bude spustený s jemu priradeným rcssserverom a hráčom (hráčmi).
- Vhodnosť počtu slave inštancií (framework+hráči+server) bude overená až záťažou na jednotlivých uzloch, teoreticky sa ale ponúka možnosť spustiť ich až 128, pretože máme k dispozícii 16 zariadení x 2 procesory x 4 jadrá. Tento model ale predstavuje potrebu riešiť spustenie viacerých slave inštancií na jednom zariadení.



Obrázok 42: Popis komunikácie distribuovaného spracovania

Jedna slučka evolučného algoritmu môže vyzerať nasledovne (Obrázok 42):

1. Master framework (pomocou MPI) spôsobom broadcast rozdistribuuje údaje jednotlivým slave frameworkom. Obsahom údajov môžu byť príkazy, skripty, alebo zmeny pohybov.
2. Každý slave framework na základe údajov od master framewoorku naplánuje hráčovi vykonávanie pohybov.

3. Slave framework bude spracovávať údaje získané zo servera. Na základe nich môže vyhodnotiť úspešnosť správania sa hráča (vypočíta fitness funkciu).
4. Master framework prostredníctvom bariérovej synchronizácie zozbiera všetky výsledky jednotlivých slave frameworkov. Na základe nich potom vygeneruje novú generáciu jedincov, ktorých rozdistribuuje (krok 1).

Identifikované problémy s daným riešením:

- potreba riešiť inicializáciu slave inštancií na diaľku,
- potreba vyriešiť komunikáciu s viacerými slave inštanciami na jednom zariadení,
- schopnosť reagovať na neštandardné správanie rcssservera,
- potreba implementácie master a slave módu frameworku,
- vzhľadom na nenatívnosť MPI v Jave môže vzniknúť neprehľadný kód,
- technológia MPI na superpočítači doposiaľ nie je zavedená,
- otázna je reálna potreba prínos riešenia pre projekt.

2.5 Analýza vyšších pohybov a taktické pohyby

2.5.1 Analýza

Pre vytvorenie plnohodnotného hráča a celého tímu je potrebné, aby títo hráči dokázali robiť nielen jednoduché pohyby a bezcieľne pobehovať po ihrisku, ale je potrebné ich naučiť zložitejšie a zaujímavejšie pohyby. Takéto pohyby nazývame vyššie pohyby. Zvyčajne vznikajú správnym spájaním nižších pohybov, prípadne ich prispôsobením danej situácii. Druhou skupinou pohybov sú taktické pohyby, kedy hráč na základe hernej stratégie, rozpoloženia hráčov na ihrisku, či možností danej situácie vyhodnotí stav a rozhodne sa pre najlepšiu možnú stratégiu v danej chvíli.

Už tím Androids navrhoval na vytváranie vyšších pohybov jazyk XABSL. XABSL (Extensible Agent Behavior Specification Language) je jazyk vhodný na opísanie vyššej logiky hráča. Vzhľadom na nadviazanie na agenta implementovaného v jazyku Java je výhodou, že je k dispozícii aj Java XABSL library. XABSL je dialekt jazyka XML. Prvky nižšieho správania sú implementované v jazyku Java, ale XABSL vytvára možnosti definovania vyšších pohybov pomocou stavových automatov.

Na vytvorenie vyšších pohybov je potrebná zmena, prípadne vytvorenie úplne nových pohybov. Na druhú stranu, aby sme mohli implementovať taktické pohyby, je potrebná už spomínaná úprava nižších pohybov, ale navyše je potrebné upraviť model sveta, aby náš hráč vnímal omnoho viac vecí a vedel s nimi aj pracovať.

Požiadavky na vyššie a taktické pohyby:

1. Presnejšie vnímanie sveta
2. Pamäť pre uchovávanie histórie modelu sveta
3. Sledovanie a hľadanie lopty
4. Pravidelná seba lokalizácia
5. Pravidelná lokalizácia spoluhráčov a vytváranie histórie a predikovať ich pohyb
6. Pravidelná lokalizácia protihráčov a vytváranie histórie a predikovať ich pohyb
7. Vyššia komunikácia medzi hráčmi pomocou krátkych správ

Na základe daných požiadaviek bude možné pomocou nižších a vyšších pohybov upravovať správanie agenta podľa modelu sveta. Bez implementácie týchto požiadaviek nebude možné naplno využiť potenciál navrhnutých vyšších pohybov.

2.5.2 Návrh vyšších pohybov

Návrh všetkých vyšších pohybov nášho tímu pozostáva z úpravy pohybov tímu Androids. Väčšina týchto pohybov je pomerne dobre a stabilne implementovaná, aj keď bude potrebné spraviť určité zmeny. Najzákladnejšou požiadavkou je parametrizácia všetkých týchto pohybov. Je potrebné vytvoriť anotácie daných pohybov a na ich základe a konkrétnej situácie povoliť agentovi upravovať pohyby pre jeho potreby.

Chôdza – schopnosť agenta dostať sa na potrebné miesto. Pomocou parametrizácie chôdze by sme mali mať možnosť upravovať smer chôdze. Táto zmena by nám mala umožniť čo najpresnejšie sa dostať na potrebné miesto, bez nutnosti využívania pomocných pohybov ako je otáčanie. Vďaka tomu by mal byť pohyb plynulejší a hlavne premiestenie sa na dané miesto rýchlejšie. Pri tejto implementácii by mali byť využité nižšie schopnosti agenta ako pohyb dopredu a otočenie sa, ktoré by mali byť v rozumnej miere skombinované pre dosiahnutie nášho cieľa.

Kop do lopty - schopnosť kopnúť do lopty smerom k vopred stanovenému cieľu. Týmto cieľom môže byť bránka, spoluhráč, prípadne iné miesto na ihrisku. V tomto prípade je potrebné parametrizovať dva aspekty daného pohybu. Prvým z nich je sila kopu. Je dôležité, aby hráč v rôznej vzdialenosti do lopty dokázal dostať loptu kopnúť na potrebné miesto. Treba vytvoriť efektívne variácie kopu z rôznej vzdialenosti od lopty, ale s rovnakým silovým efektom. Druhým z nich je kopanie do rôznych smerov, bez potreby otáčania hráča pred loptou, čo je priveľmi pomalé a zdržuje priebeh hry. Na dosiahnutie týchto cieľov by sme mali použiť nižšie schopnosti rôznych typov kopania do lopty a otáčania.

Vedenie lopty - prestavuje schopnosť vedenia lopty jedným agentom. Bude pozostávať zo schopností kopu do lopty a chôdze. Pre implementáciu tohto pohybu je možné využiť už vyššie navrhnuté prístupy k chôdzy a kopu do lopty. Skĺbením týchto dvoch parametrizovaných pohybov, by mal byť hráč schopný viesť loptu hocijakým smerom bez jej straty. Vďaka parametrizovanej chôdzy by mal môcť jednoducho kľučkovať medzi hráčmi a loptu by mal vedieť vždy držať pri sebe vďaka rôznej sile kopania z rôznych pozícií.

2.5.3 Návrh taktických pohybov

Pomocou vyšších pohybou sa pokúsim analyzovať a navrhnúť možné riešenia taktických pohybov vhodných a realizovateľných pre nášho hráča. Takisto sa počíta s daným modelom prostredia, ktorý obsahuje nielen aktuálny stav ale i predpokladaný stav v budúcnosti. Ten sa určuje pomocou aktuálnych (známych) údajov a znalosti dynamiky (napr. pozícia a rýchlosť) lopty a ostatných hráčov.

Veľmi potrebné je pri týchto taktický pohybov komunikácia s ostatnými členmi týmu, bolo by vhodné pred každou realizáciou niektorého z taktických pohybov informovať o svojom zámere svoje okolie, tj. spoluhráčov nachádzajúcich sa v jeho blízkom okolí.

Pri taktických pohyboch som zanalyzoval všetky minuloročné tímy na našej fakulte, ktoré vytvárali agentov pre robotický futbal, ale iba v 2D. Túto možnosť som si zvolil, retože tieto tými mali väčšie možnosti práce na taktických pohyboch ako 3D tými a sú omnoho prepacovanejšie a dá sa teda nimi lepšie inšpirovať aj pre našu situáciu.

1. Získanie lopty - pri tomto chovaní by sa mala využívať predikcia pozície lopty v nasledujúcich dvoch cykloch. Najprv sa vypočíta poloha lopty v nasledujúcom cykle a zistí sa, či hráč dokáže získať loptu posunutím sa do predpovedanej pozície lopty bez otáčania sa. Ak áno, potom hráč získa loptu v nasledujúcom cykle. V opačnom prípade sa predpovedá pozícia lopty o 2 cykly neskôr, v 1. kroku sa hráč posunie, v 2. sa otočí takým smerom, aby mohol získať (spracovať) loptu.
2. Kop na bránu – by mal byť vďaka rozšíreniu modelu sveta o už spomínané prvky pomerne jednoduchý. Hráč by mal zanalyzovať danú situáciu a podľa svojej polohy, polohy brankára a lopty a ich spoločnej polohy voči bráne vybrať čo najlepší parametrizovaný kop. Hráč by mal predpokladať možnosti bránenia pomocou pádu na niektorý bok a teda lopta by mala byť usmernená smerom s najvyššou pravdepodobnosťou úspešnosti.
3. Prihrávka – pri tomto pohybe je možné využívať viacere prístupy. Mnou analyzované a navrhnuté 4 pohybu sú pomerne jednoduché, ale na druhú stranu veľmi účinné.
    - direct – cieľová pozícia lopty je poloha spoluhráča
    - leading – prihrávanie lopty do otvoreného priestoru, kam smeruje aj spoluhráč, ktorý ju dokáže spracovať v jednom cykle (teda nemusí sa otáčať)
    - through – prihrávanie do otvoreného priestoru medzi brankárom a obrancami opačného tímu tak, aby spoluhráč bol schopný získať loptu skôr než protihráči.
    - Give-and-go - hráč nahrá svojmu voľnému spoluhráčovi a potom začne utekať do voľného priestoru a bude čakať prihrávku od tohto spoluhráča, ktorému nahral prvý krát.

4. Bránenie – tento pohyb by mal prebiehať úplne rovnako ako kop na bránu, len v opačnom prístupe. Hráč po analýze všetkých faktorov by mal vybrať najvhodnejšiu polohu, prípadne brániaci manéver.
5. Rozohrávanie – brankár by mal využívať už spomínané prihrávky okrem poslednej z nich. Do tohto konceptu by malo byť zahrnuté rozšírenie, kedy brankár ak má možnosť prihrať viacerím hráčom a jeho schopnosti to dovolujú, prihrá hráčovi najbližšie k súperovej bráne a teda s najvyšším potenciálom streliť gól, prípadne na neho prihrávať.

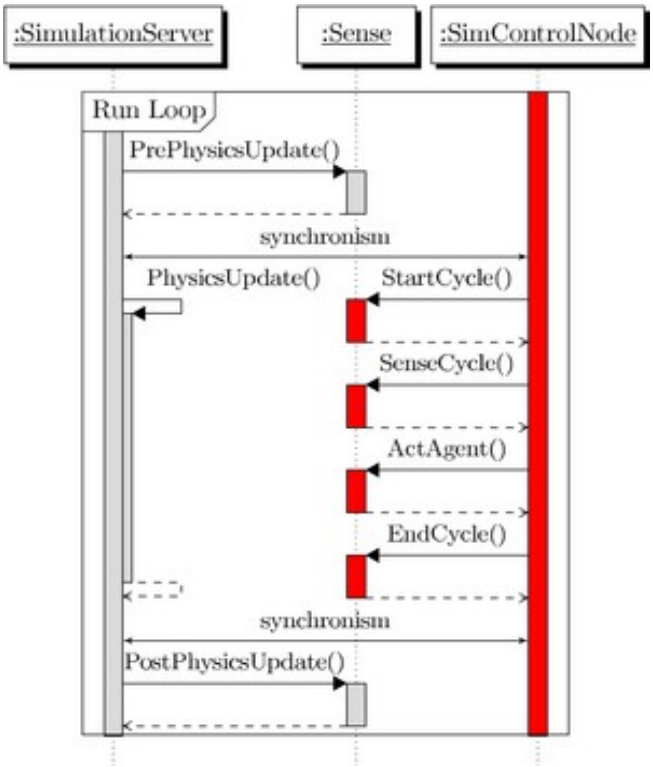
V neposlednom rade je dôležitý výber správneho pohybu. Po analýze som sa rozhodol, že najvhodnejším spôsobom by bolo použiť taktiku rozhodovanie hráča Nexus2D. Hráč Nexus2D pracuje na princípe dvojfázového rozhodovacieho mechanizmu. Pri rozhodovaní sa hráča, akú akciu má uskutočniť najskôr, vyberie sa zvlášť najlepšia možná strela, najlepšia možná prihrávka a najlepšia možná cesta pre driblovanie (pohyb s loptou) a v druhom kroku sa vyberie najlepšia možná akcia z predchádzajúcich troch najlepších akcií. Výber najlepších akcií je ovplyvnený váhami, ktoré sa po každom výbere najlepšej akcie menia.

2.6 Analýza možností paralelizácie robocup servera

2.6.1 Robocup Server

Používaný robocup server umožňuje v najnovšej verzii využitie paralelizmu, teda viacerých CPU počítača. Týmto je možné efektívnejšie a rýchlejšie vykonávať simulácie, čo nám umožňuje lepšie realizovanie testovania hráča a jeho pohybov ako aj efektívnejšie spúšťanie simulácii na slabších počítačoch.

Server interne rozdeľuje simuláciu do viacerých menších oddelených častí nazývaných „SimControlNode“ a jednej časti určenej na fyzikálne výpočty. Všetky tieto časti sú spútané paralelne vo vlastnom vlákne. Na konci simulačnej slučky spolu vytvárajú aktívnu scénu, ktorej tvorenie je už vykonávané len jednom vlákne. Posledný krok je vykonanie posledných fyzikálnych výpočtov na výslednej scéne. Celková náročnosť výpočtu každej iterácie simulácie je teda veľmi znížená. Táto funkcionalita je stále v experimentálnej časti vývoja, no v jej reálnom používaní sme nenarazili na žiadne problémy.



Obrázok 43: UML diagram paralelnej práce robocup servera

2.6.2 Inštalácia Robocup Servera

Pre možnosti využitia rýchlejšieho počítača na vykonávanie simulácii bol robocup server nainštalovaný na školskom tímovom serveri, ktorý bol poskytovaný v rámci predmetu. Inštalovaná bola najnovšia možná verzia pre zabezpečenie čo najlepšieho chodu s experimentálnou funkciou paralelizmu (verzia 0.6.5 revízia 286). Reálne testy servera ukázali beh servera so 4 vláknami, pričom bola viditeľná znížená záťaž na CPU.

Každý člen tímu má možnosť pripojenia sa na tento server pomocou SSH pripojenia a vykonávať simulácie na diaľku. Kvôli striktnému zavedeniu možných prístupných externých portov je pre úspešné ďalšie pripojenia agenta alebo monitora na server nutné presmerovanie týchto portov na lokálny počítač.

Spôsob spustenia servera:

```
rcssserver3d [možnosti]

možnosti:
--agent-port PORTNUM     port pre pripojenie agenta na server
--server-port PORTNUM    port pre pripojenie monitora na server
```

Príklad lokálneho presmerovania portov na server v prostredí UNIX nástrojom OpenSSH client:
ssh [user@]server -L <lokalny_port>:localhost:<port_na_robocup_serveri>

2.7 Nástroj testovací framework

2.7.1 Aktuálny stav testovacieho frameworku

Cieľom testovacieho frameworku je umožniť automatické testovanie a vyhodnocovanie aktivít hráča alebo viacerých hráčov. Medzi výhody ktoré by mal časom prinášať je rýchlejšie automatické testovanie na vzdialených počítačoch prípadne automatizované skúmanie viacerých možností nastavení pohybov pohybov hráča alebo stratégií a ich následné vyhodnocovanie. Základ testovacieho frameworku je prevzatý od tímu Androids, ktorý implementoval viaceré dôležité základné časti pre jeho ďalší správny vývoj. Následné je opísaný aktuálny stav frameworku rozdelení podľa funkčnosti a poskytovanej funkcionality. Tento opis slúži na ujasnenie aktuálnych schopností frameworku a určenie ďalších častí na vývoj alebo úpravu. Do chýbajúci funkcionalít sú zahrnuté aj základné činnosti, ktoré by mal testovací framework vedieť vykonávať a celkovo chýbajú.

Funkčné časti

Identifikované ako správne fungujúce boli následovné hlavné časti testovacieho frameworku. Rozdelené sú do skupín podľa funkcionality ktorú vykonávajú.

- Trénovanie pohybov
  - čítanie XML súborov opisujúcich čo trénovať
  - písanie XML súborov opisujúcich čo trénovať
- Anotácie
  - Vytvorené a funkčné anotácie (Bug, Refactor, Reviewed, TestCovered, UnderConstruction)
- Monitor robocup servera
  - správne príjimanie a spracovávanie správ od servera
  - upravovanie modelu sveta a jeho využitie v iných častiach projektu
- Ovládanie servera
  - monitorovanie stavu hry (čas, stav, skóre)
  - pohyb s loptou
  - pohyb s hráčom
- Ovládanie hráča
  - odosielanie ruby skriptov na vykonanie hráčovi
- Testy
  - umožňuje správne určenie a zistenie pozície lopty, hráča a stav hry
  - správne vykonávanie testu (inicializácie, beh testu, ukončenie s vyhodnotením)
  - jediný funkčný test pohybu kráčania (WalkTestCase)
- JUnit testy
  - testy často nedostatočne kontrolujú správnu implementáciu tried

Nefunkčné alebo chýbajúce časti

Nasledovné časti boli zhodnotené ako nekompletné, chybové alebo celkom chýbajúce v testovacom frameworku.

- Všeobecne
  - nejednotné a málo používané logovanie behu programu
  - žiadna možná konfigurácia nástroja (fixné konštanty priamo v zdrojovom kóde)
  - veľa nepotrebných výpisov priamo do štandardného výstupu (aj výstupy určené na ladenie programu)
  - neexistujúca hlavná spustiteľná metóda, namiesto toho je viacero metód na vykonanie len jednej akcie (spustenie testu, trénera)
- Trénovanie pohybov
  - nefungujúce reálne trénovanie pohybov, teda vykonávanie testov na základe prečítaných XML konfigurácií
- Anotácie
  - nepoužívané na všetkých miestach projektu
- Ovládanie hráča
  - komunikácie s hráčom je len jednosmerná a neexistuje žiadna spätná väzba
  - nemožné spustenie nového hráča (nutné manuálne spustenie pred spustením testovacieho frameworku)
- Ovládanie servera
  - nemožné spúšťanie servera (nutné manuálne spustenie pred spustením testovacieho frameworku)
- Testy
  - implementovaný len jeden test pohybu
- Grafické rozhranie
  - existencia dvoch implementácií, pričom len jedná je automaticky spustiteľná bez zásahu do zdrojového kódu
  - poskytuje celkovo málo možností

2.7.2 Návrh na zlepšenie

Na základe vykonanej analýzy aktuálneho stavu testovacieho frameworku sú navrhnuté následovné časti, ktoré je potrebné vylepšiť v nasledujúcich šprintoch:

- odstrániť ladiace výstupy v normálnom behu programu
- zabezpečiť jednotnú spúšťaciu metódu pre testovací framework (vykonať refactoring modulov pre lepšiu prehľadnosť a modulárnosť)
- implementovať jetnotné a správne logovanie behu programu na kľúčových miestach
- pridať možnosť konfigurácie nástroja na zabezpečenie jeho nasadenia v rôznych prostrediach
- zabezpečiť spätnú komunikačnú väzbu od hráča
- implementovať možnosť spúšťania hráča a robocup servera testovacím frameworkom
- vylepšiť spôsob interakcie s používateľom (napríklad, grafické rozhranie)
- vytvoriť viaceré testy pohybov hráča

3 Šprint 3

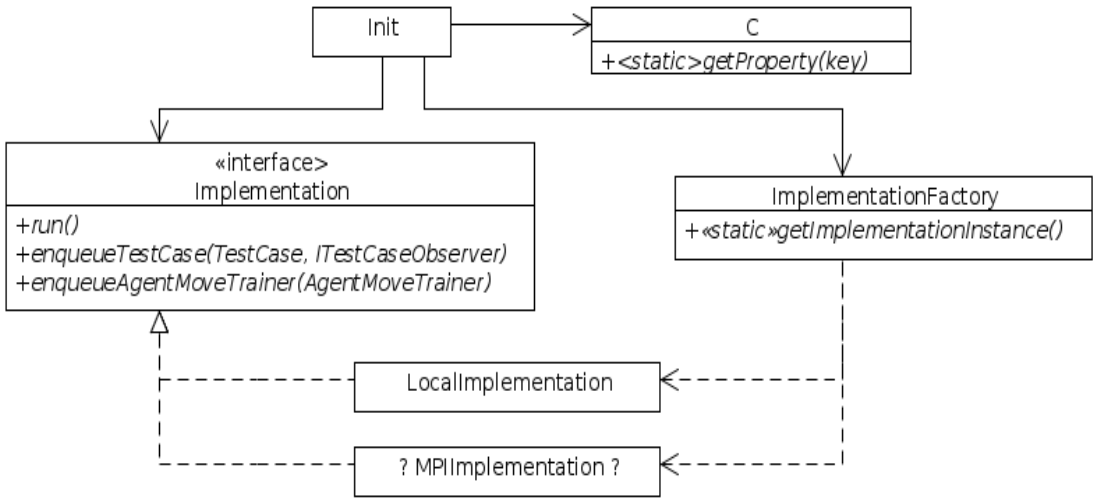
3.1 Refactoring testovacieho frameworku

3.1.1 Návrh štruktúry

Hlavnou myšlienkou zmeny štruktúry testovacieho frameworku je vytvoriť jednotnú metódu na jeho spúšťanie a zjednotiť už predom vytvorené moduly. Súčasťou návrhu je požitie štandardného logovania programovacieho jazyka Java. Jeho konfigurácia je možná vytvorení štandardného súboru „logging.properties“ a následné spustenie programu s príslušnými argumentami. Konfigurácia projektu bude možná priložením konfiguračného súboru, ktorý bude spracovaný pri každom spustení programu. Dodatočná konfigurácia je ďalej možná vhodnými spúšťacími argumentami programu. Výsledné moduly navrhovanej štruktúry sú:

Implementation

Modul predstavuje spôsob správania testovacieho frameworku a obsahuje logiku nutnú na jeho správnu prevádzku. Vytvára potrebné inštancie ostatných modulov a vykonáva hlavnú slučku programu. Pre tento modul bol vybraný návrhový vzor Factory, ktorý zaručí správne zvolenie implementácie na základe nastavenia projektu. V budúcnosti sa počítajú rôzne verzie implementácie napríklad s použitím technológie MPI na urýchlenie testovania.



Obrázok 44: Znázornenie návrhového vzoru Factory pre rôzne implementácie

AgentMonitor

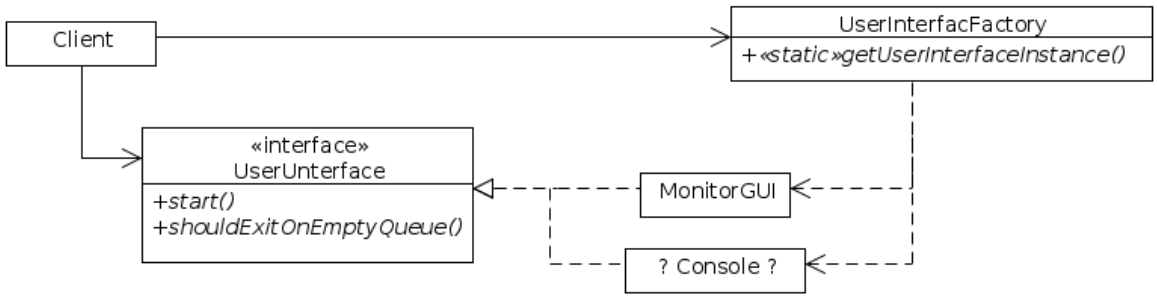
Zabezpečuje spätnú väzbu pre posielanie správn z agenta na zvolenú IP adresu a port testovacieho frameworku. Agent sa musí na túto adresu pripojiť sám. Ostatné moduly testovacieho frameworku sa následne môžu zaregistrovať na prijímanie správ od agenta (vzor observer). Povolená je len jedna inštancia na adresu, pričom trieda obsahuje statické metódy na prístup k už vytvoreným inštanciám tejto triedy

RobocupMonitor

Jeho úlohou je udržiavanie informácií o aktuálnom stave a priebehu hry na robocup serveri a to pripojením na port určený pre monitor servera. Súčasťou tohto modulu je aj parsovací modul, ktorý správy prekladá a vytvára model sveta. Ostatné moduly testovacieho frameworku sa následne môžu zaregistrovať na prijímanie týchto správ od agenta (vzor observer). Povolená je len jedna inštancia na adresu servera, pričom trieda obsahuje statické metódy na prístup k už vytvoreným inštanciám tejto triedy

UserInterface

Opisovaný modul zabezpečuje komunikáciu s používateľom. Do programu je zahrnutý prostredníctvom návrhového vzoru Factory a nieje nutný na samotný automatický beh programu. Jeho činnosť spočíva v jeho zaregistrovaní do tried RobocupMonitor, AgentMonitor, a TestCase na prezentovanie vykonaných častí testovania. Umožnuje taktiež pridávať úlohy danej implementácii pomocou jej statických metód. Konkrétna implementácia je spúšťaná metódou *štart* určenej jej rozhraním. Metóda *shouldExitOnEmptyQueue* je spúšťaná implementáciou vždy po vykonaní všetkých činností a umožňuje používateľovi rozhodnúť o ukončení programu.



Obrázok 45: Znázornenie návrhového vzoru Factory pre používateľské rozhranie

TestCase

Abstraktná trieda TestCase definuje základné operácie pre každý test. Jej vykonanie je zabezpečené pridaním do rady testov na vykonanie v práve zvolenej implementácii metódou *enqueueTestCase*. Každá konkrétna implementácia testu má umožnené reagovať na jeho vytvorenie a zrušenie ako aj vykonávať kontrolu či test neskončil a tým zisťovať a spracovávať výsledok (fitness) pri jej skončení. Výsledky testu sú môžu byť zhromažďované Trénerom na ďalšie spracovanie.

AgentMoveTrainer

Trieda predstavuje samotné trénovanie agenta. Jej úlohou je vytvárať testy s vhodnými parametrami a ich zaradenie na vykonanie pomocou práve zvolenej implementácie testovacieho frameworku (metóda *enqueueTestCase*). O každom výsledku testu je oboznámená a pomocou týchto informácii je umožnené vyhodnotenie najlepšej varianty z testov.

Trieda Init

Základná trieda obsahujúca jediný spustiteľný bod programu. Jej úlohou je spracovanie argumentov z príkazového riadku a následné správne spustenie implementácie a tým spustenie celého testovacieho frameworku.

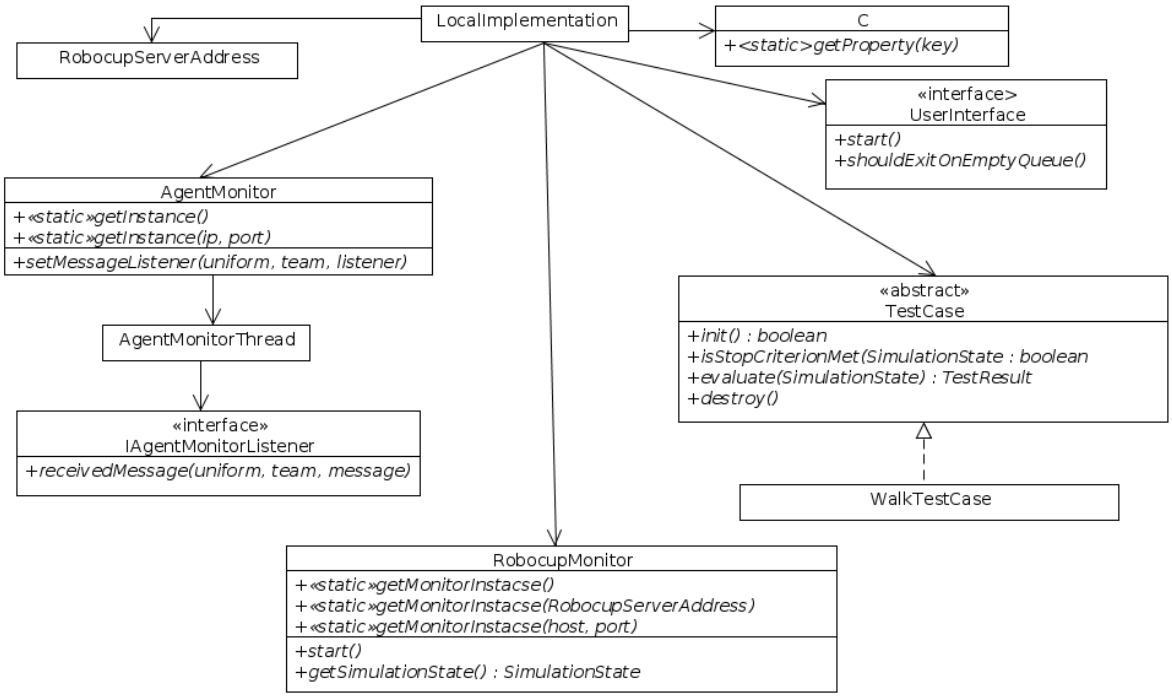
Trieda C

Pomocná trieda C obsahuje všetky konštanty používane v programe. Pri spustení načíta všetky prednastavené hodnoty z vstavaného konfiguračného súboru. Následne sa pokúsi nájsť konfiguračný súbor aj v pracovnom adresári. Umožnené je aj načítavanie iného konfiguračného súboru, túto akciu však musí inicializovať trieda *Init* na základe vstupných argumentov.

3.1.2 Vykonané zmeny

Na základe návrhu boli vykonané všetky potrebné zmeny aby návrh reprezentoval skutočný stav projektu. Celkovo boli vykonané následovné zmeny:

- Reorganizácia tried a balíčkov pre lepšiu prehľadnosť projektu.
- Úprava všetkých tried na správne použitie použitých zvolených návrhových vzorov u jednotlivých triedach (triedy TesCase, RobocupMonitor, AgentMonitor, Init, Implementation, MonitorGUI, C)
    - vytvorenie potrebných statických tried
    - vytvorenie Factory tried kde boli potrebné
    - upravené zaobchádzanie s vytvorenými vláknami



Obrázok 46: Znázornenie závistlostí triedy LocalImplementation

- Vytvorenie tried pre prijímanie správ od agenta (AgentMonitor, AgentMonitorThread)
- Vytvorenie triedy Init na inicializáciu testovacieho frameworku.
- Vytvorenie triedy C umožňujúcej jednoduchý prístup ku konštantám projektu. Spolu s triedou bol vytvorený aj konfiguračný súbor obsahujúci všetky predvolené hodnoty.
- Implementované logovanie základných funkcií programu štandardného logovacieho systému jazyka Java *(java.util.Logging).*

- Vytvorenie tried Implementation, LocalImplementation a ImplementationFactory na zabezpečenie správneho behu testovacieho frameworku.
- Úprava triedy WalkTestCase vzhľadom na vykonané zmeny a zabezpečenie jej správneho chodu.
- Drobná úprava iných tried kde bolo potrebné (pridanie komentárov, logovania, úprava správania)

3.1.3 Ďalšie potrebné úpravy

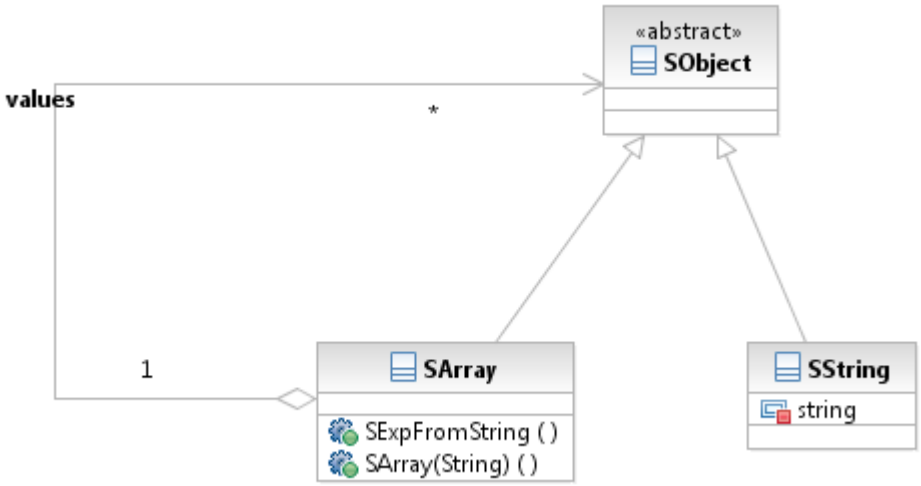
Testovací nástroj je potrebné ďalej rozvíjať v nasledujúcich šprintoch. Medzi konkrétne funkcionality ktoré je potrebné doimplementovať sa radí:

- určenie a spracovanie vstupných argumentov programu
- vytvorenie implementácie pre podporu MPI na lepšie testovanie
- určenie správ a implementácia ich spracovania pre umožnenie obojsmernej komunikácie s agentom JIM (rozšírenie triedy AgentMonitor)
- umožnenie automatického spúšťania RobocupServera a hráča Jim (správa aj viacerých hráčov). Súčasná komunikácia ku hráčovi je možná na len manuálne predom spusteného hráca.
- implementácia trénera schopností hráča (AgentMoveTrainer)
- vytvorenie viacerých testov pohybov hráča
- komentovanie zdrojových kódov projektu a vytvorenie ich komplexnejšej dokumentácie.

3.2 Poloha ostatných hráčov

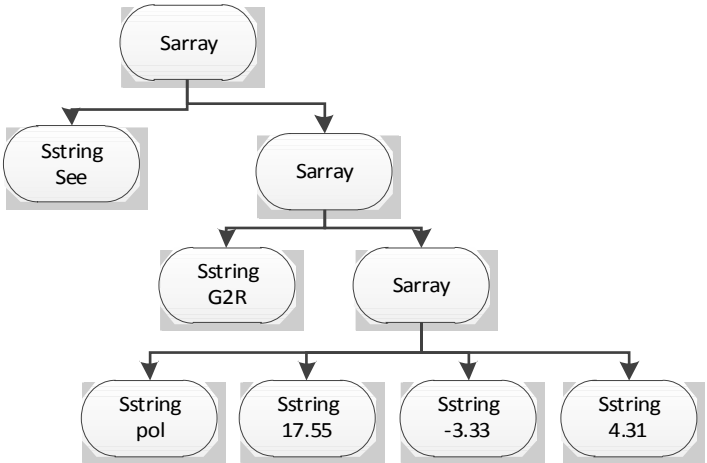
3.2.1 Parser

Jediná relevantná informácia o polohe hráčov prichádza z see perceptora. V pôvodnom riešení síce bolo implementované parsovanie správ zo see perceptora, avšak parsované boli len fixné body a poloha lopty. Preto bolo potrebné upraviť parsovanie see správ. Prvým krokom bolo vytvorenie parsera S-výrazov do stromovej štruktúry. Diagram tried je zobrazený na obrázku 47.



Obrázok 47: Diagram tried stromovej štruktúry s-výrazov

Sarray obsahuje v sebe parser s-výrazov. Sarray je množina, ktorá môže obsahovať reťazce(SString) alebo ďalšie Sarrays. To umožňuje rozloženie výrazu do stromu.



Obrázok 48: Príklad S-výraz (See (G2R (pol 17.55 -3.33 4.31))) uloženého v strome

Ďalším krokom bolo vytvorenie triedy SeenPerceptor, ktorá pomocou vyššie uvedenej stromovej štruktúry, rozparsovala správu seen perceptora. Navyše dáta o hráčoch ukladá do inštancie objektu PlayerData.

Pôvodná štruktúra ParsedData bola upravená tak aby obsahovala informácie o ostatných hráčoch, pričom nerozlišuje do akého tímu patrí. Rozdelenie hráčov do tímov je úohou WorldModelu, úlohou parsera je len poskytovať dáta zo servera v určitých štruktúrach.

V triede Perceptors bolo zmenená metóda retrieveSeenData. Algoritmus, ktorý bol v nej implementovaný, bol nahradený SeenPerceptorom:

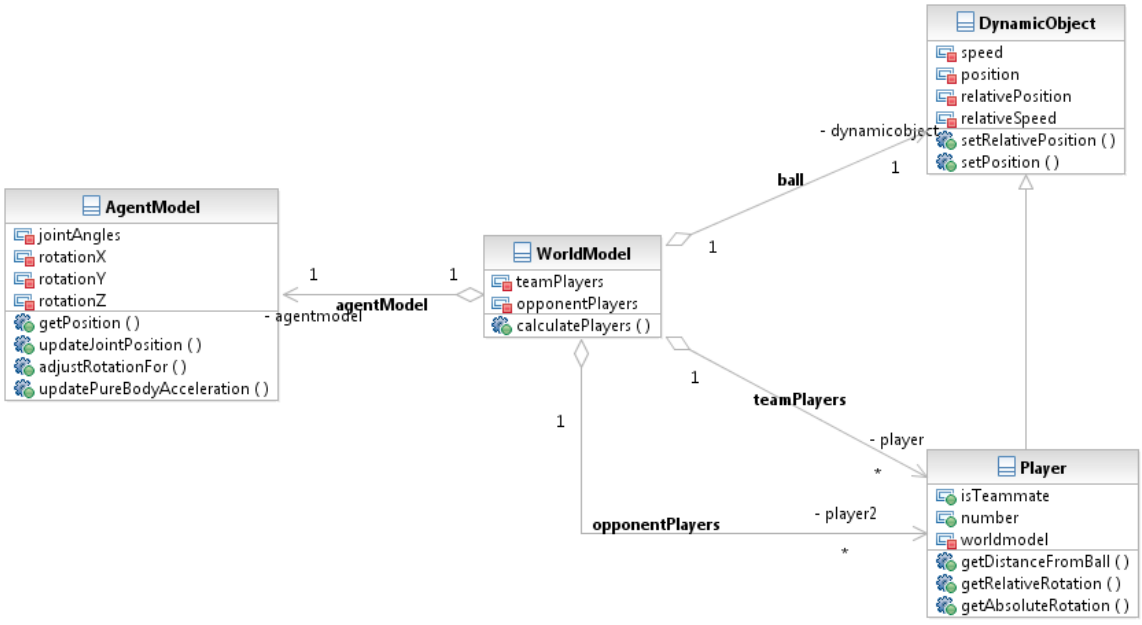
```
private static void retrieveSeenData(String message, ParsedData data){

            SeenPerceptorData seendata =
                  SeenPerceptor.retrieveSeenData(message);

            data.ballRelativePosition = seendata.ball;
            data.fixedObjects = seendata.fixedObjects;
            data.otherplayers = seendata.players;
}
```

3.2.2 Určenie polohy ostatných hráčov

Informácia o jednotlivých hráčoch je uložená v inštanciách tried Player. Pôvodná trieda bola rozšírená tak aby uchovávala všetky informácie o polohách častí hráčov. Absolútna poloha je vypočítaná pomocou AgentModel.globalize(Vector3D) metódy, ktorá vypočíta z relatívnej polohy, pričom zohľadňuje natočenie hráča a jeho aktuálnu pozíciu na ihrisku. Za globálnu polohu hráča je považovaná poloha hlavy. Presnosť určenia polohy hráčov na ihrisku závisí od vzdialenosti hráča od agenta, a v prípade absolútnej polohy na ihrisku aj od presnosti polohy a natočenia agenta. Class diagram WorldModelu je na obrázku 49.



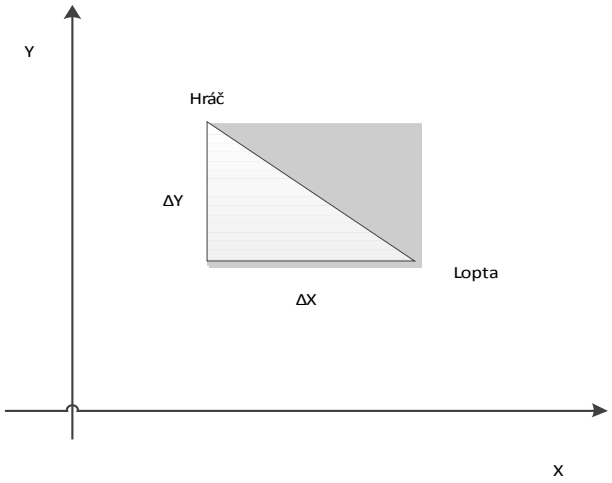
Obrázok49: Časť diagramu tried WorldModelu s dôrazom na hráčov

3.3 Vzdialenosť hráčov od lopty, natočenie hráčov

3.3.1 Vzdialenosť hráčov od lopty

Vzdialenosť od lopty je počítaná na základe absolútnych polôh hráčov aj lopty. Každý Player je umiestnený do určitého contextu, čo je v našom prípade WorldModel. Z tohto kontextu vie potom vyčítať polohu lopty, prípadne v budúcnosti aj iné informácie z modelu sveta.

Výpočet vzdialenosti je vykonávaný pomocou pytagorovej vety 79:



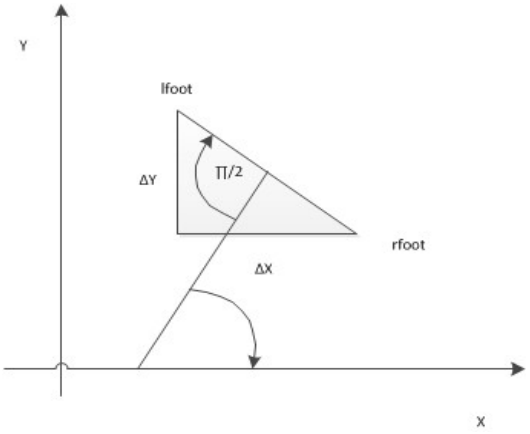
Obrázok 50: Výpočet vzdialenosti lopty

Vzdialenosť lopty od hráča vracia metóda v triede Player:

```
public double getDistanceFromBall()
```

3.3.2 Natočenie hráčov

Natočenie hráča sa považuje uhol, ktorý zviera os úsečky spájajúca nohy. Ideálne je keď hráč stojí v vzpriamenej polohe a nohy ma pri sebe. Problém nastáva, keď nohy nie sú v úplne vzpriamenej polohe(napríklad pri chôdzi), v súčastnosti však máme príliš málo informácii na presnejšie určenie natočenia hráča.



Obrázok 51: Natočenie hráča

Pri výpočte tohto uhla je potrebné brať do úvahu všetky štyri kvadranty. Pre každý kvadrant sú charakteristické znamienka pri delta X a delta Y.

kvadrant	Delta X	DeltaY
0-90	<0	>0
90-180	<0	<0
180-240	>0	<0
240-360	>0	>0

Tabuľka 51: Kvadranty

Výpočet je vykonávaný podľa vzorca v pravouhlom trojuholníku

sin(alfa) = delta x/vzdialenosť nôh.

A potom uhol sa rovná:

alfa = arcsin(dx/c)

Funkcie arcsin vracia len hodnoty v intervale (-PI/2, PI/2), takže je potrebné funkciu rozdeliť na dve časti, a to:

- dy<0: alfa = arcsin(dx/c)
- dy>0: alfa = PI + arcsin(dx/c)

Natočenie hráča vracajú metódy triedy Player:

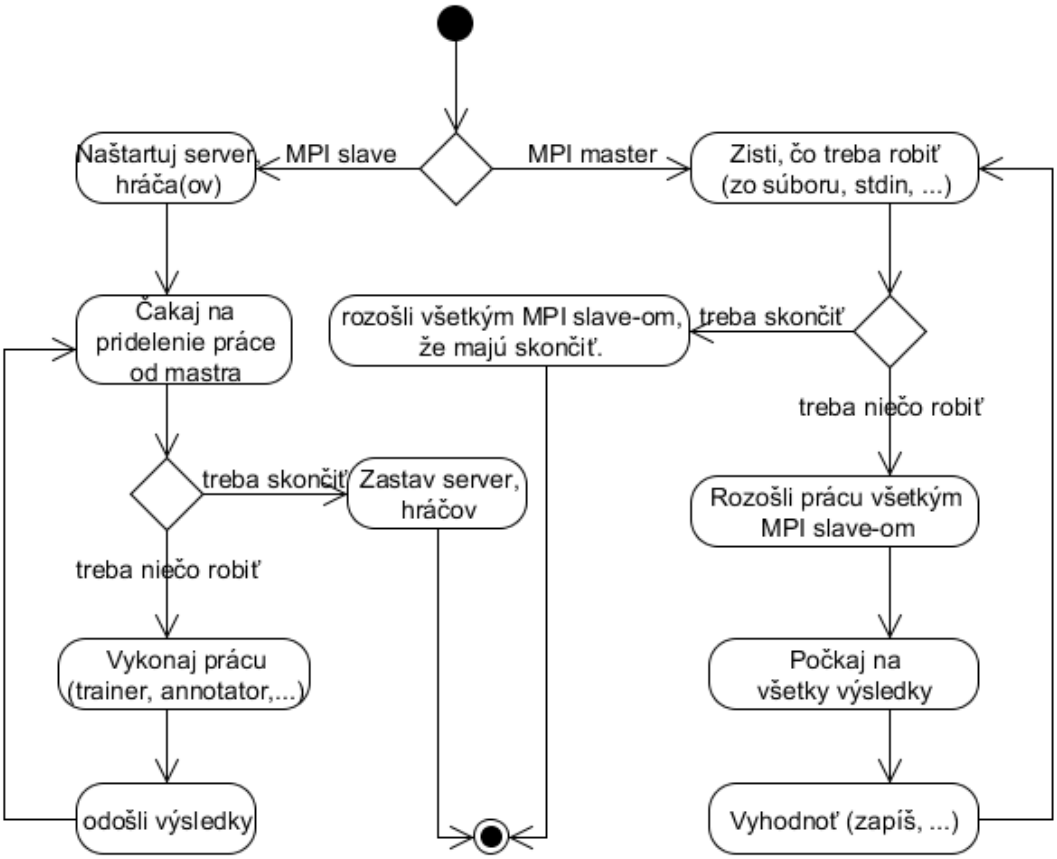
```
public double getRelativeRotation()
public double getAbsoluteRotation()
```

3.4 Prototyp MPI Frameworku

3.4.1 Návrh

Za implementáciu MPI, ktorá bola použitá bol vybraný program MPJ-Express, pretože poskytuje celkom prehľadné API pre Javu, je zdokumentovaný na postačujúcej úrovni.

Opis navrhnutého riešenia z pohľadu frameworku zobrazený je na obrázku 52.



Obrázok 52: Opis návrhnu MPI riešenia z pohľadu frameworku

3.4.2 Implementácia

Pre spustenie MPI implementácie je potrebné nasledovné:

- Zo stránky http://mpj-express.org/ stiahnuťmpj-express a nainštalovať podľa inštrukcií.
- V balíku `sk.fiit.testframework.init` v súbore `default.properties` nastaviť:
  `userInterface = sk.fiit.testframework.ui.MpiInterface`
  `implementation = sk.fiit.testframework.init.MpiImplementation`
- Podľa inštrukcií spustiť framework prostredníctvom MPI ľubovoľným počtom vlákien

Implementácia bola vzhľadom na skutočnosť, že je potrebné najprv implementovať spúšťanie hráča a rcessserver 3D automacitky pomocou frameworku dokončená iba na úrovni existujúcej komunikácie medzi MPI vláknami.

V rámci tejto implementácie bolo vypracované vhodné doplnenie spúšťania Frameworku pod MPI tak, aby v ňom ostal iba jeden spúšťač. To sa podarilo vďaka vhodnému refactoringu a úprav kódu zo strany Ivana Šimka.

Program by mal z master vlákna poslať správu (dvojnásobok čísla procesu, ktorému je správa odosielaná) postupne všetkým slave vláknam, ktoré ich vypíšu na obrazovku). Predpokladá sa neskoršie dopracovanie implementácie MPI podľa návrhu, ak bude doplnená potrebná funkcionalita a podarí sa spustiť Robocup aj na školskom superpočítači.

3.5 Návrh parametrizovaných pohybov a anotácií

Pri návrhu parametrizovaných pohybov som sa zameral na analýzu zahraničných tímov, ktorý už danú problematiku riešili a podarilo sa im pomerne úspešne implementovať dané návrhy. Medzi najpokročilejšie tímy v tejto oblasti patria tímy Cerberus , Nao Team HTWK , Kouretes a hlavne nemecký tím B-Human, z ktorého dokumentácie som informácie čerpal. Všetky tieto tímy majú parametrizácia pohybov riešenú veľmi podobným spôsobom.

3.5.1 Motion engine

Aby bolo možné dynamicky vytvárať parametrizované pohyby je vhodné doimplemenovať niečo ako „motion engine“(ďalej len ME), ktorý sa bude o toto starať. ME by mal byť implementovaný v rámci frameworku a jeho základným princípom by malo byť kombinovať súbory jednoduchých kriviek pohybu k zložitejším.

Na základe tohto spôsobu by mal byť celý pohyb rozdelený do niekoľkých fáz pohybu, ktorých počet bude záležať od zložitosti a typu pohybu. Fázy by mali byť napríklad zdvihnutie nôh, už samotný pohyb pri kope do lopty a tak ďalej. Jednoduché krivky by boli definované pohybom končatiny za nejakú dobu. Tieto krivky môžu byť označované aj ako trajektórie tohto pohybu. Každá fáza pohybu by mala obsahovať 6 rôznych kriviek. Každá ruka a každá noha by mala mať svoju vlasnú krivku, ktorá kontroluje jej pozíciu. Navyše každá noha má jednu krivku, na kontrolu rotačného pohybu.

Vhľadom k tomu, že tieto trajektórie musia byť kombinované do zložitejších trajektórií, ME musí zabezpečiť, že kombinácie každých dvoch fáz je hladká, aby sa zabránilo nežiaducim trhnutiam medzi dvoma prepojenými fázami. Systém by mal zaručovať hladkosť kontrolou, či miesto pripojenia dvoch je spojito diferencovateľné.

ME by mal dostať zoznam dynamických bodov ako vstupy z iných modulov, aby mohol dokončiť dynamické zmeny. Dynamické body by mali obsahovat informácie o zadanej cielovej polohe niektorých kriviek a ich smer pohybu. Podľa týchto dynamických bodov poskytuje systém informácie, ako napríklad poloha lopty, ktorá sa mení počas vykonávania konkrétneho pohybu.

Na obrázku 53 vidíme algoritmus používaný nemeckým tímom B-Human. Daný algoritmus popisuje celkovú štruktúru ME. V prvej časti definície pohybu sú vyžiadané ID požadovaného pohybu. Potom sú na začiatku každej fázy sú dynamické body aplikované na aktuálnu a trajektórie sú inicializované. Okrem dynamicky sa meniacich, je v tomto kroku upravená aj hladkosť kriviek súčasnej fázy. Následne sú získané pozície končatín z aktuálneho okamihu a pozície. Keďže pozícia konečného efektora je známa, uhly kĺbov sú dopočítané pomocou inverznej matematiky. Posledným krokom je úprava rovnováhy.

Algorithm 1. The main procedure of the motion engine

```
1: if wasActive ≠ true then
2:    phase ← 0
3:    currentParameters ← getParameters(id)
4:    addDynValues(phase, currentParameters, dynValues)
5:    initCurrentTrajectories(phase, currentParameters)
6: end if
7: if phase ≤ maxPhase then
8:    time ← getTime(getCurrentTime(), getDeltaT(currentParameters))
9:    if time == 1 then
10:      phase ← phase + 1
11:      addDynValues(phase, currentParameters, dynValues)
12:      initCurrentTrajectories(phase, currentParameters)
13:      time ← 0
14:   end if
15:   positions ← getLimbPos(currentParameters, phase, time)
16:   joints ← calcJoints(positions)
17:   addBalance(joints, currentParameters, gyroData)
18:   wasActive ← true
19:   return joints
20: else
21:   wasActive = false
22:   return stand
23: end if
```

Obrázok 53: Motion engine algoritmus

3.5.2 Parametrizácia

Na základe vyššie uvedenej metódy by bolo vhodné parametrizovať chôdzu a kopy do lopty. Pomocou daných rozdelení pohybov na fázy by sme mali byť schopný vytvoriť plynulú a pomerne rýchlu chôdzu pri ktorej sa nebude agent pohybovať len jedným smerom, ale bude možné aby zabáčal prípadne jednoducho kľučkoval po ihrisku.

Takisto by nám malo byť umožnené parametrizovať konkrétne kopy, či už ich sila prípadne smer strely. Ako už bolo uvedené v analýze vyšších a taktických pohybov, pomocou pojenia týchto dvoch pohybov bude možné implementovať aj pomerne kvalitné vedenie lopty.

Otázkou ostáva kedy je potrebné a vhodné dané parametrizované pohyby použiť. V prípade kopu do lopty je nutné, aby hráč vedel či je potrebné daný pohyb parametrizovať, pretože vďaka vytvorenému anotačnému systému vieme otestovať a zistiť či náhodou daný pohyb už v základnej konfigurácii nebude postačovať na dosiahnutie požadovaného výsledku. Hráč pomocou anotácii bude vedieť rozoznať, či je lopta priďaleko prípadne akou silou vie kopnúť do lopty na aktuálnej pozícii.

V neposlednej rade je potrebné zvážiť aj stabilitu týchto pohybov. Pri veľkom obrate počas chôdze je možné, že bude efektívnejšie vykonať otočenie o požadovaný uhoľ a následne využívať iba jednoduchú chôdzu. Dané anotácie by mali obsahovať aj maximálnu možnú mieru parametrizácie, pri ktorej je ešte pohyb efektívny.

4 Šprint 4

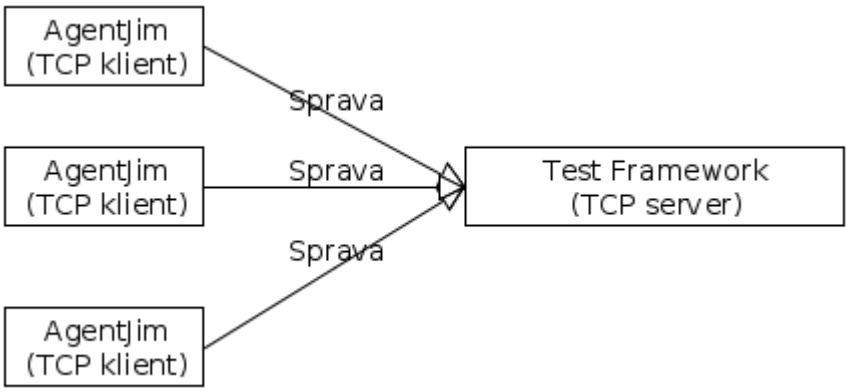
4.1 Spätná väzba od hráča v testovacom frameworku

Na zabezpečenie správneho chodu testovacieho frameworku a jeho testov je potrebné byť informovaný o rôznych činnostiach a rozhodnutiach agenta. Príkladom môžu byť rozhodnutia o naplánovaní pohybov alebo o jeho vnímaní sveta. Terajší stav testovacieho frameworku umožnuje posielanie príkazov len smerom k hráčovi a neumožňuje žiadnu spätnú väzbu.

4.1.1 Návrh

Najjednoduchší a najvariabilnejší spôsob komunikácie je vytvorením TCP spojenia a následného odosielania správ v predom dohodnutej forme. Výhodou tohto riešenia je možnosť behu hráča a testovacieho frameworku na odlišných počítačoch alebo iných platformách ako aj pomerne jednoduchá implementácia.

Testovací framework bude slúžiť ako server na prijímanie nových TCP spojení a následne spracovávať správy. Hráč sa bude pripájať na testovací framework(TCP server), vytvárať správy a následne ich odosielať. Umožnené je aj pripojenie viacerých hráčov na jeden server, pričom testovací framework musí správne odlíšiť správy od jednotlivých hráčov.



Obrázok 54: Znázornenie komukácie hráčov s testovacím frameworkom

Formát správy by mal byť čo najjednoduchší na spracovanie, predom známy a jednoduchý na rozšírenie. Každá správa pritom musí obsahovať identifikátor hráča na jeho jednoznačné identifikovanie. Celkovo boli navrhnuté zatiaľ dva typy správ:

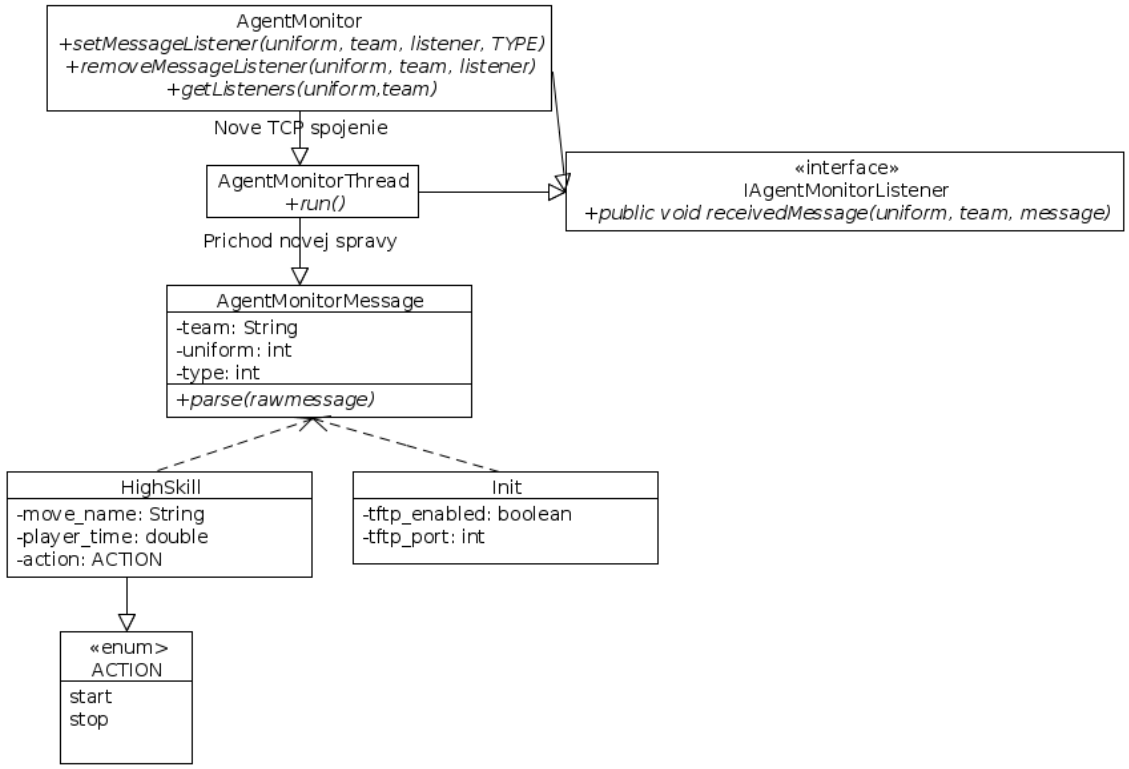
```
(<uniform> <team>) init [<tftp> <tftp_port>]
(<uniform> <team>) highskill <meno_pohybu> <start|stop> <cas_vykonania>
```

Prvý typ správy je posielaný pri prvom spojení so serverom. Testovací framework na základe tejto správy vie určiť presne ktorý hráč sa pripojil a či podporuje vykonávanie ruby skriptov pomocou tftp servera. Druhá správa predstavuje začiatok alebo koniec vykonávania pohybu hráčom. Obsahuje aj informáciu o aktuálnom čase hry. Tento čas je odosielaný pre presné určenie časov testovacím frameworkom nakoľko prijatie správy môže byť oneskorené z rôznych neovplyvniteľných dôvodov spôsobených využitím spojenia TCP.

4.1.2 Implementácia

Testovací framework

Vytvorená bola nová trieda AgentMonitor, ktorý zohráva úlohu TCP servera pre prichádzajúce nové spojenia. Po príchode nového spojenia je vytvorená nová inštancia triedy AgentMonitorThread, ktorá vykonáva čítanie správ z daného spojenia a následne ich posiela na spracovanie triede AgentMonitorMessage. Trieda AgentMonitorThread vytvára ešte nový typ správy TYPE_DESTROY v prípade zrušenia spojenia s daným hráčom. Po úspešnom spracovaní správy sú notifikované všetky registrované triedy implementujúce rozhranie IAgentMonitorListener na počúvanie (návrhový vzor observer). Pri registrovaní na počúvanie je možné určiť presne pre ktoré typy správy chce byť daná trieda notifikovaná a to určením čísla hráča, jeho tímu alebo typu správy.



Obrázok 55: Diagram tried spracovania správ

Príklad registrovania na typ správy triedou implementujúce rozhranie IAgentMonitorListener:

```
AgentMonitor.setMessageListener(this, TYPE_INIT |TYPE_DESTROY);
```

Príklad spracovania správy:

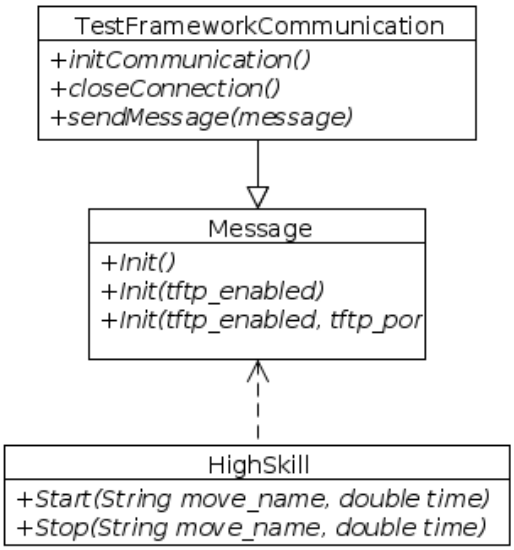
```
@Override
public void receivedMessage(int uniform, String team, AgentMonitorMessage
message) {
      switch (message.type_flags) {
            case TYPE_INIT:
                  Init msgInit = (Init) message;
                  // spracovanie informacie o pripojeni agenta
```

```
                    break;
                case TYPE_DESTROY:
                    // spracovanie informacie o odpojeni agenta
                    break;
        }
}
```

Parser testovacieho frameworku bol vytvorený pomocou Java reflexion API a to tak, že meno triedy je rovnake ako typ správy. Takto parser jednoducho odovzdá ostávajúcu časť spravy v podobe textu danej inštancii triedy na spracovanie. Napríklad pri prijatí správy typu *init* je vytvorená nová trieda *AgentMonitorMessage.Init* pričom argument konštruktora tejto triedy je zvyšok správy.

Agent Jim

Implementované boli dve nové triedy TestFrameworkCommunication a Message. Prvá z týchto tried je spustená hneď pri spustení hráča a jej úlohou je pripojiť sa na TCP server testovacieho frameworku, udržiavať spojenie a posielať správy. Určenie adresy samotného testovacieho frameworku ako aj možnosť zapnutia samotnej spätnej väzby je vykonané prostredníctvom konfiguračného súboru premennými „*TestFramework_monitor_enable*„ a „*TestFramework_monitor_address*“. Trieda Message následne umožňuje vytváranie správ na odoslanie a zabezpečuje správny formát týchto správ. Vytvorenie konkrétnej správy je umožnené rôznymi statickými metódami. Pri spustení triedy s nedostatočnými informáciami sú doplnené údaje z konfiguračného súboru (napríklad tftp port pre vykonávanie príkazov).



Obrázok 56: Znázornenie závistlostí tried zodpovedných za spätnú väzbu hráča

Nové správy je možné jednoducho implementovať vytvorení novej triedy dediacej od Message, ktorá vytvára správu v správnom formáte. Následné je nutné implementovanie parsera tejto triedy v testovacom frameworku a to vytvorením triedy dediacej od AgentMonitorMessage.

4.2 Analýza a návrh algoritmov určenia polohy agenta

4.2.1 Lineárny Kalmanov filter

Kalmanov filter slúži na rekurzívne odhadovanie vnútorného stavu lineárneho dynamického systému z množiny meraní so šumom. Rekurzívny znamená, že nepotrebujeme uchovávať všetky predchádzajúce merania ale nové meranie sa odvodí z predchádzajúceho.

Stav systému v čase k je opísaný vektorom `xk` , potom stav v čase (k+1) je vypočítaný pomocou stavovej rovnice:

```
xk+1 = Fk+1xk + wk+1                                    (1)
```

- `Fk+1` je transformačný model
- `wk+1` je procesný šum

```
yk = Hkxk + rk                                          (2)
```

- `yk` je pozorovanie v čase k
- `Hk` je matica pozorovaní
- `rk` je šum pozorovaní

Kalmanov filter pozostáva z dvoch hlavných krokov:

1. Časová aktualizácia

$$\hat{\boldsymbol{x}}_k = \boldsymbol{F}_k \hat{\boldsymbol{x}}_{k-1}$$

$$\boldsymbol{P}_k = \boldsymbol{F}_k \boldsymbol{P}_{k-1} \boldsymbol{F}_k^T + \boldsymbol{Q}_{k-1}$$

2. Aktualizácia pozorovaním

$$\boldsymbol{K}_k = \boldsymbol{P}_k \boldsymbol{H}_k^T [\boldsymbol{H}_k \boldsymbol{P}_k \boldsymbol{H}_k^T + \boldsymbol{R}_k]^{-1}$$

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k + \boldsymbol{K}_k (\boldsymbol{y}_k - \boldsymbol{H}_k \hat{\boldsymbol{x}}_k)$$

$$\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_k$$

4.2.2 Existujúce riešenie

V agentovy JIM je implementovaný lineárny kalmanov filter, pre jednorozmerný stav. Stavom sa rozumie súradnica x, y, z. Keďže implementovaný kalmanov filter je pre jednorozmerný stav, pri výpočte pozície sa použije zvlášť pre každú súradnicu. Implementovaný filter je používaný na korekciu fixných bodov. Fixné body sú pomocou filtra korigované po parsovaní. Výpočet výslednej polohy je pomocou aritmetického priemeru absolútnych polôh získaných z relatívnych polôh jednotlivých fixných bodov.

4.2.3 Rozšírený Kalmanov filter

Poloha agenta na ihrisku nieje lineárny problém, preto je potrebný použiť takzvaný rozšírený kalmanov filter. Stav agenta bude popisovať v čase t vektor $\boldsymbol{x}_t = [x_t, y_t, v_t, \varphi_t]$ potom funkcia $f(\boldsymbol{x}, \boldsymbol{u})$ bude mat tvar:

$$f_x = x_{t+1} = x_t + s\cos(\varphi_t + 0{,}5\, d\varphi)$$

$$f_y = y_{t+1} = y_t + s\sin(\varphi_t + 0{,}5\,d\,\varphi)$$

$$f_v = v_{t+1} = v_t + adt$$

$$f_\varphi = \varphi_{t+1} = \varphi_t + d\,\varphi$$

Pričom $s = v_{tdt} + 1/2\mathrm{adt}^2$

Potom

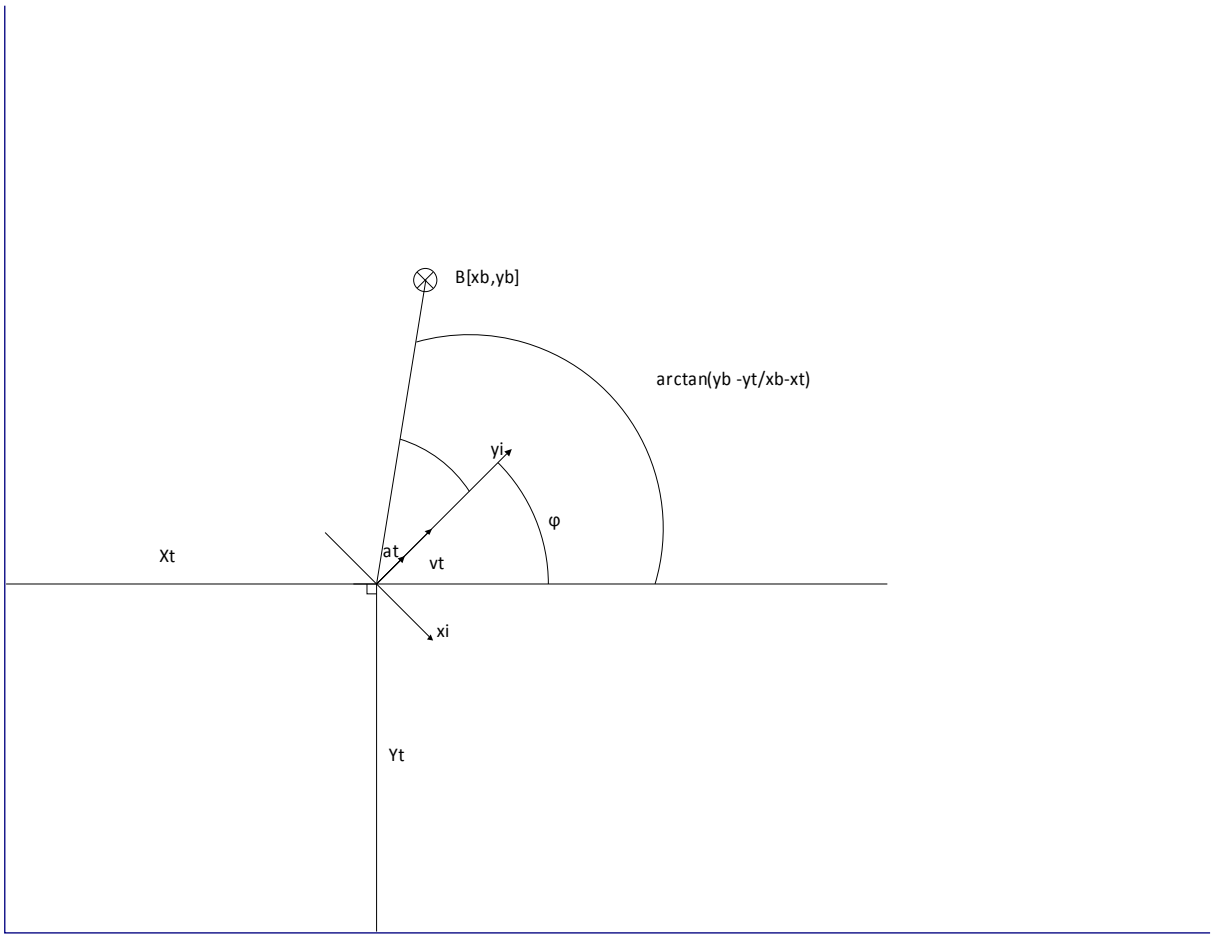
$$\boldsymbol{A}_k = \begin{bmatrix} 1 & 0 & -s\sin(\varphi_t + 0{,}5\,d\,\varphi) \\ 0 & 1 & s\cos(\varphi_t + 0{,}5\,d\,\varphi) \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{B}_k = \begin{bmatrix} 1/2\mathrm{dt}^2\cos(\varphi_t + 0{,}5\,d\,\varphi) & s/2\sin(\varphi_t + 0{,}5\,d\,\varphi) \\ -1/2\mathrm{dt}^2\sin(\varphi_t + 0{,}5\,d\,\varphi) & -s/2\cos(\varphi_t + 0{,}5\,d\,\varphi) \\ dt & 0 \\ 0 & 1 \end{bmatrix}$$

rovnica h bude mať tvar

- $h_x = d * \sin(\arctan((y_b - y_t)/(x_b - x_t)) - \varphi_t)$
- $h_y = d * \cos(\arctan((y_b - y_t)/(x_b - x_t)) - \varphi_t)$
- $d = \sqrt{(x_b - x_t)^2 + (y_b - y_t)^2}$

$$\hat{\boldsymbol{x}}_t = \boldsymbol{f}_t(\hat{\boldsymbol{x}}_{t-1}, u_t)$$
$$\boldsymbol{P}_t = \boldsymbol{F}_t \boldsymbol{P}_{t-1} \boldsymbol{F}_t^T + \boldsymbol{Q}_{t-1}$$
$$\boldsymbol{K}_t = \boldsymbol{P}_t \boldsymbol{H}_t^T [\boldsymbol{H}_t \boldsymbol{P}_k \boldsymbol{H}_t^T + \boldsymbol{R}_t]^{-1}$$
$$\hat{\boldsymbol{x}}_t = \hat{\boldsymbol{x}}_t + \boldsymbol{K}_t(\boldsymbol{y}_t - \boldsymbol{h}_t(\hat{\boldsymbol{x}}_t))$$
$$\boldsymbol{P}_t = (\boldsymbol{I} - \boldsymbol{x}_t \boldsymbol{H}_t)\boldsymbol{P}_t$$



Obrázok 57: Určenie polohy hráča na ihrisku

4.3 Optimalizácia kopnutia

V rámci štvrtého šprintu bola vykonaná optimalizácia pohybov kopnutia. Tím Androids nám zanechal dve základné kopnutia – kopnutie špičkou a kopnutie bokom chodidla.

4.3.1 Kopnutie špičkou

Na základe testovania tohto pohybu sme dospeli k názoru, že nie je nutné jeho zmena. Pohyb bol v poriadku v rýchlosti vykonávania, aj po stránke stability. Jediný problém bol v otáčaní hráča okolo svojej osi pri opakovanom vykonávaní pohybu. Toto sa nakoniec podarilo úspešne vyriešiť. Pre každú nohu boli tiež vytvorené dva ďalšie pohyby, ktoré z tohto kopnutia priamo vychádzajú a líšia sa iba vo veľkosti náprahu a sile kopnutia.

4.3.2 Kopnutie bokom

Tento typ kopnutia spočíva v otočení chodidla o 90 stupňov a následné kopnutie jeho bokom do lopty. Nebol však veľmi dobre implementovaný, pretože pokiaľ sa lopta nenachádzala v bezprostrednej blízkosti pred hráčovou nohou, netrafil sa do nej. Optimalizácia pohybu spočívala vo väčšom zohnutí kolena a členku pri kopaní, čo spôsobilo, že sa hráč naklonil smerom dopredu a loptu bez problémov trafil.

4.4 Optimalizácia blokovania (sadnutia)

Tento pohyb bol od tímu Androids prevzatý ako úplne nefunkčný. Asi v polovici sadania sa hráč vždy prevrátil na chrbát a občas aj na brucho, z ktorého sa potom nevedel ani postaviť. Vyriešili sme to zlepšením stability pohybu pomocou nakláňania trupu a tiež viacerými úpravami záverečných fáz pohybu, ako napríklad zmenšením uhlu medzi nohami robota pri dosadnutí na zem. To zabezpečilo vysokú stabilitu a tiež rýchlosť pohybu. Vznikla tiež experimentálna verzia tohto pohybu, ktorá vykoná posadenie za rekordný čas. Táto však už nie je tak stabilná a občas sa stane, že sa hráč prevalí na chrbát.

4.5 Testovanie pohybov pomocou frameworku

V rámci frameworku je potrebné, aby sme vedeli otestovať rôzne pohyby a vedeli ich ohodnotiť nejakou číselnou hodnotou, ktorá nám bude udávať kvalitu, presnosť prípadne efektívnosť tohto pohybu, samozrejme každý test je jedinečny a tak k nemu je potrebné aj pristupovať. Tím Androids mali v pláne niečo podobné vytvoriť, ale prevdepodobne z časového hladiska túto úlohe nestihli. Podarilo sa im vytvoriť veľmi jednoduchý test pre jeden typ chôdze. Tento test case bol vymazaný a nahradený jedným všebecným, v ktorom sú zisťované počiatočné informácie o stave situácie a následne je tento test rozširovaný podľa toho aký typ pohybu sa ide testovať.

4.5.1 MotionTestCase

Toto je základny, už vyššie spomínaný test case. Daný test case zisťuje inicializačnú polohu hráča, lopty ako aj natočenie hráča pomocou vektora. Daný test case je samotne nespustiteľný slúži iba ako základ pre ďaľšie rozšírené testy.

4.5.2 FallTestCase

Daný test case pracuje iba s hráčom. Jeho hlavnou úlohou je overiť, kde sa hráč nachádza a v akom natočení oproti pôvodnej polohe pred pádom. Tento test je vhodný, aby sme v budúcnosti vedeli po páde a bránení hráča predikovať jeho aktuálnu polohu. Tým pádom že sa hráč postaví naspäť na nohy, bolo by najvhodnejšie, aby sa postavil na pozíciu čo najbližie k pôvodnej a v rovnakom natočení.

Cieľom testu je dosiahnuť čo najnižšiu možnú hodnotu od 0 po N, kedy 0 znamená, že sa hráč postavil do pôvodnej polohy, prípadne ostane sedieť.

Fitnes funkcia je postavená na princípe, kedy podľa rozdielu začiatočnej a finálnej pozície je vyrátaná hodnota, ku ktorej je ešte následne pripočítaný uhoľ otočnia.

Daný test bol vytvorený pre nasledovné pohyby:

Pohyb fall_left - Hráč zámerne spadne na ľavý bok.

Pohyb fall_right - Hráč zámerne spadne na pravý bok.
Pohyb padvpred - Hráč zámerne padne dopredu.
Pohyb padvzad - Hráč zámerne padne dozadu.
Pohyb sit_down - Hráč rozkročí nohy a pokúsi sa sadnúť na zem.

4.5.3 KickTestCase

Daný test case pracuje s hráčom a s loptou. Jeho hlavnou úlohou je overiť, kde sa nachádza lopta a kde sa nachádza hráč a v akom natočení oproti pôvodnej polohe pred kopom. Tento test je vhodný, aby sme vedeli ohodnotiť silu a kvalitu jednotlivých kopov a či hráč po skončení kopu ostáva na rovnakom mieste ako na začiatku

Cieľom testu je dosiahnuť čo najvyššiu možnú hodnotu od 0 po N, kedy 0 znamená, že hráč loptu vôbec netrafil, až po N ktoré predstavuje vzdialenosť prejdenú loptou, pričim hráč ostal na rovnakom mieste a nepootočil sa.

Fitnes funkcia je postavená na princípe, kedy podľa rozdielu začiatočnej a finálnej pozície lopty je vyrátaná hodnota, od ktorej sú následne primerane odpočitavane hodnoty, ak sa pohol hráč, ak sa hráč nejak pootočil, prípadne lopta sa vychýlila zo smeru kopu.

Daný test bol vytvorený pre nasledovné pohyby:

Pohyb kick_left - Hráč sa prikrčí a vykoná jednoduché kopnutie špičkou ľavej nohy pred seba.
Pohyb kick_right - Hráč sa prikrčí a vykoná jednoduché kopnutie špičkou pravej nohy pred seba.
Pohyb kick_straight_edge_l - Hráč sa prikrčí a vykoná kopnutie pred seba vnútornou časťou chodidla ľavej nohy.
Pohyb kick_straight_edge_r - Hráč sa prikrčí a vykoná kopnutie pred seba vnútornou časťou chodidla pravej nohy.
Pohyb sidekick_left - Hráč sa prikrčí a kopne vnútornou časťou ľavej nohy bokom do lopty tak, že ju pošle napravo.
Pohyb sidekick_right - Hráč sa prikrčí a kopne vnútornou časťou pravej nohy bokom do lopty tak, že ju pošle naľavo.

4.5.4 WalkTestCase

Daný test case pracuje iba s hráčom. Jeho hlavnou úlohou je overiť, kde sa hráč nachádza a v akom natočení oproti pôvodnej polohe pred chôdzou. Tento test je vhodný, na zistenie rýchlosti a kvality chôdze jedným smerom.

Cieľom testu je dosiahnuť čo najvyššiu možnú hodnotu od 0 po N, kedy 0 znamená, že sa hráč vôbec nepohol a N, ktoré predstavuje vzdialenosť prejdenej dráhy, v prípade ak sa hráč pohyboval zvoleným smerom.

Fitnes funkcia je postavená na princípe, kedy podľa rozdielu začiatočnej a finálnej pozície hráča je vyrátaná hodnota, od ktorej sú následne primerane strhnuté hodnoty za odklonenie sa od smeru trasy a otočenie sa oproti pôvodnej polohe.

Daný test bol vytvorený pre nasledovné pohyby:

Pohyb stepleft - Hráč vykoná ľavou nohou krok do boku.
Pohyb stepleft1 - Hráč vykoná ľavou nohou krok do boku.
Pohyb stepright - Hráč vykoná pravou nohou krok do boku.
Pohyb stepright1 - Hráč vykoná pravou nohou krok do boku.
Pohyb walkback2 - Hráč vykoná pomocou špičiek krok dozadu.
Pohyb walk_fine_back - Hráč sa prikrčí a vykoná krok dozadu.
Pohyb walk_fine_fast1 - Hráč sa prikrčí a vykoná relatívne veľký krok dopredu.
Pohyb walk_fine_fast2 - Hráč sa prikrčí a vykoná relatívne veľký krok dopredu.
Pohyb walk_fine_slow - Hráč sa prikrčí a vykoná malý krok dopredu.

4.5.5 RotationTestCase

Daný test case pracuje iba s hráčom. Jeho hlavnou úlohou je overiť, kde sa hráč nachádza a v akom natočení oproti pôvodnej polohe pred otočením. Tento test je vhodný, aby sme zistili či sa hráč počas otočenia otočí o požadovanú vzdialenosť a pritom ostane na rovnakom mieste.

Cieľom testu je dosiahnuť čo najpresnešiu možnú hodnotu od -N po N, kedy 0 znamená, že sa hráč vôbec neotočil a N je označenie pre konkrétne otočenie, ktoré sme testovali alebo -N v opačnom smere ak sa otočil, t.j. ak sme testovali otočenie hráča o 20 stupňov do prava a výsledná hodnota je 20, daný pohyb dosiahol najlepší možný výsledok

Fitnes funkcia je postavená na princípe, kedy podľa stupňa otočenia je určená daná hodnota.

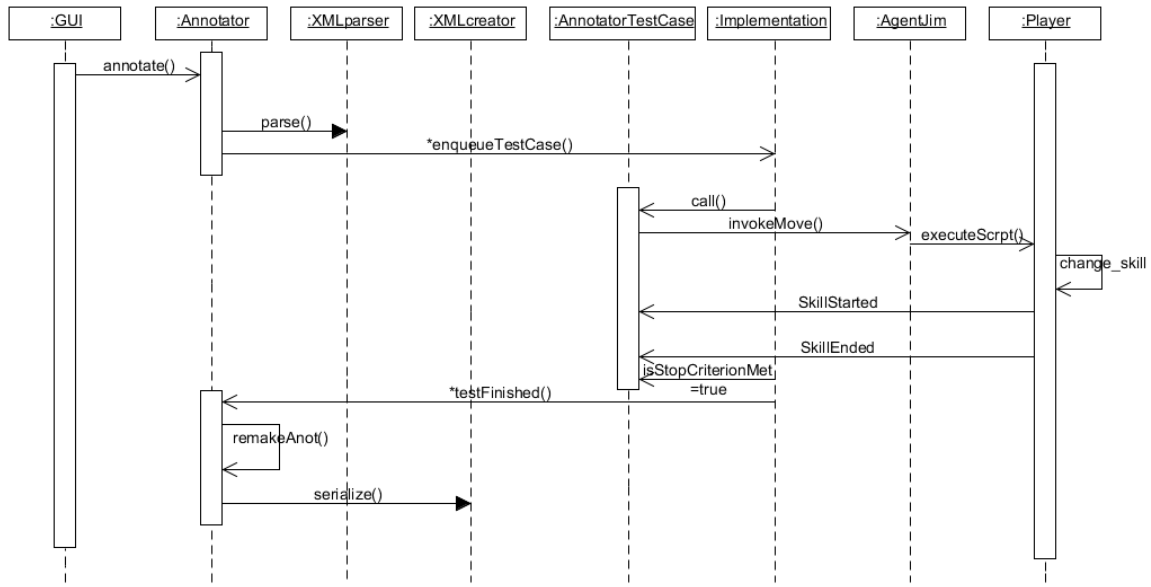
Daný test bol vytvorený pre nasledovné pohyby:

Pohyb turnleft90 - Hráč sa prikrčí a otočí o približne 90 supňov doľava. Najskôr natočí jednu nohu a následne k nej priloží druhú, čím sa otočí.
Pohyb turn_left_cont_20 - Hráč sa otočí o 20 supňov doľava podobným spôsobom ako u predchádzajúceho pohybu.
Pohyb turn_left_cont_5 - Hráč sa otočí o 5 supňov doľava.
Pohyb turnright90 - Hráč sa prikrčí a otočí o približne 90 supňov doprava. Najskôr natočí jednu nohu a následne k nej priloží druhú, čím sa otočí.
Pohyb turn_right_cont_20 - Hráč sa otočí o 20 supňov doprava podobným spôsobom ako u predchádzajúceho pohybu.
Pohyb turn_right_cont_5 - Hráč sa otočí o 5 supňov doprava.

4.6 Automatické anotácie

4.6.1 Implementácia

Popis fungovania automatických anotácií je popísaný v nasledovnom sekvenčnom diagrame (Obrázok 58).



Obrázok 58: Sekvenčný diagram tvorby automatických anotácií

- Anotátor je vytvorený s paramterami
  - loops – počet opakovaní každého testu,
  - initBallPositions – všetky počiatočné pozície lopty, pre ktoré sa bude testovať,
  - moveToTest – pohyb, ktorý sa bude testovať.
- Automatické anotovanie sa spúšťa pomocou annotate funkcie, ktorá obsahuje parametre:
  - annotSource – vstupný xml súbor s anotáciou (ideálne ručne predpripravený súbor). Pokiaľ daný súbor neexistuje, bude vytvorená prázdna anotácia (ktorú je vhodné následne doplniť ručne).
  - AnnotDest – výstupný xml súbor s anotáciou,
  - xsdFile – XSD súbor na overenie schémy anotácií.
- Anotátor testuje nad testCase: pre každú iniciálnu pozíciu lopty vytvorí toľko testcase-ov, koľko je nastavených opakovaní pre každý test. Každý takýto TestCase zaradí ho do fronty konkrétnej implementácie, takže jednotlivé testcase-y sa začnú postupne vykonávať.
- Samotný testcase pred svojim vykonaním povie hráčovi, aký pohyb má vykonávať
- Počas testovania testcase počúva hráča, vie kedy začal s vykonávaním pohybu a kedy skončil
- Po skončení vykonávania pohybu uloží zozbierané údaje (do testcase result)
- Anotátor sleduje počet ukončených testCase-ov, po dosiahnutí počtu vytvorených testCase-ov začne s vytváraním novej anotácie na základe ich výsledkov.

- Novovytvorenú anotáciu zapíše do XML súboru.

Počas riešenia boli identifikované nasledovné problémy, ktoré je potrebné v budúcnosti vyriešiť:

- Potreba jednotnej implementácie parsovania a serializácie XML aj v hráčovi aj vo Frameworku. Keďže na oboch sa stále pracuje, vhodné je riešenie prostredníctvom importu z nového projektu so spoločnými knižnicami.
- Je otázne, nakoľko navrhnuté anotácie budú hráčovi nápomocné. Osobne by som informácie ohľadom pozície lopty po ukončení pohybu v nejakej miere priraďoval počiatočnej pozícii lopty (či už pre konkrétne hodnoty, alebo určité rozloženie ihriska z pohľadu hráča na logické časti).
- Framework by mal pri ukončovaní testCase vedieť, či sa ešte hýbe lopta – až potom ukončiť anotáciu.
- Pre korektné fungovanie automatických anotácii, hráč nesmie mať naplánované spúšťanie nejakého pohybu dookola. Treba sa zamyslieť nad vylepšovaním plan.rb v hráčovi.
- Pre fungovanie automatických anotácii je kritická obojsmerná komunikácia. Treba vyriešiť kontrolu jej fungovania počas testovania.
- Treba začať vyvíjať GUI Frameworku, anotácie by potrebovali vlastný tab.

4.7 Spúšťanie hráča a servera pomocou testovacieho frameworku

4.7.1 Implementácia

Za účelom vykonávania testov na vzdialených počítačoch je nutné aby si testovací framework vedel sám riadiť chod agenta a robocup servera. Umožní to skutočné automatické testovanie na viacerých počítačoch bez nutnosti manuálneho spúšťania alebo vypínania iných programov/procesov.

Spúštanie agenta Jim

Pre správne spútanie hráča je potrebné aby hráč vedel spracovávať rôzne argumenty na jeho konfiguráciu. Implementované teda bolo parsovanie vstupných argument v hlavnej spúštacej metóde. Konkrétne sú povolené následné argumenty pri spúštaní agenta:

```
jim [options]
      [-testframework host port]
      [-uniform number]
      [-team team_name]
      [-tftp [port]]

Vysvetlenie:
-testframework host port      - zapnutie odosielania spatnej vazby na adresu
-test [port]                  - zapnutie lokalneho tftp servera - mozne
                                dodatocne urcenie portu (default 3070)
-uniform number               - nastavenie atributov agenta na pripojenie k
                                serveru
-team team_name               - nastavenie atributov agenta na pripojenie k
                                serveru
```

Hráč je následne spúšťaný pomocou triedy *java.util.Process.* Štandardný výstup agenta je čítaný vo zvlášť vytvorenom vlákne avšak nieje nijak ďalej spracovávaný. Dôvod tohto čítania je obmedzenosť veľkosti vyrovnávacej pamäti prislúchajúcej na výstup procesu. Pri zaplnení tejto pamäte by bol proces pozastavený.

Celkovú réžiu spúšťania a znovu použitia už pripojených agentov má na starosti vytvorená trieda AgentManager. Pomocou jej metód je možné jednoduché vyžiadanie agenta so zvolenými atribútmi. Trieda sama spustí novú inštanciu agenta alebo použije už pripojeného staršieho agenta. Nasleduje príklad vyžiadania agenta s číslom 1 a tímom „ANDROIDS“. Posledná premenná funkcie robí volanie blokujúce, teda dané vlákno čaká na pridelenie agenta, keďže prípadné spustenie novej inštancie agenta nieje okamžité.

```
agent = AgentManager.getManager().getAgent(5, "ANDROIDS", true);
```

Spúštanie RobocupServer (simspark)

Spúšťanie procesu robocup servera je implementované podobne ako spúštanie novej inštancie agenta, teda triedou *java.util.Process* a je vykonávané triedou LocalImplementation. Táto trieda si pred spustením procesu overí či server už nieje spustený vytvorením TCP spojenia na jeden z portov servera. Pri neúspešnosti je daný proces spustený a následný skúšaním vytvorenia TCP spojenia na jeden z portov servera je overovaná úspešnosť tejto operácie. Operácia spustenia novej inštancie servera sa považuje za neúspešnú po nemožnom vytvorení TCP spojenia na jeden z portov servera ani po piatich sekundách od spustenia procesu.

4.7.2 Konfigurácia

Pri spúštaní nového agenta alebo robocup servera testovací framework vykonáva príkaz určený konfiguračnou premennou *robocup.player.command* a *robocup.server.command.* Túto premennú je možné určiť v konfiguračnom súbore ako aj vstupných argumentoch hlavnej metódy spustenia testovacieho frameworku.

Pri vypínaní procesu umožňuje platforma Java a trieda *java.util.Process* len posielanie signálu SIGTERM, pričom signál je doručený len procesu priamo spusteného triedou *Process* (podprocesy signál nedostanú). Je teda potrebné aby spúšťaný proces správne reagoval na tento signál a prípadne aj ukončil svoje podprocesy.

V prípade robocup servera je spúšťanie problematické, keďže nereaguje na signál SIGTERM. Riešenie predstavuje vytvorenie pomocného programu na spustenie servera, pričom tento pomocný program správne spracováva daný signál a ukončí beh spusteného robocup servera. Implementácia takéhoto pomocného programu je závislá od použitého operačného systému. Príklad takéhoto programu pre operačný systém Linux v skriptovacom jazyku Bash je následovný.

```
#!/bin/bash
rcssserver3d $@ &
trap "kill -sigint $!;exit" TERM SIGTERM
wait
```

4.8 MPI v distribuovanom prostredí

Cieľom tejto úlohy bolo spustiť testovací framework v distribuovanom prostredí prostredníctvom technológie MPI. Východiskovým stavom je schopnosť úspešne spustiť testovací framework v tzv. „cluster“ konfigurácii, hoci iba lokálne na jednom zariadení. Prenos takéhoto riešenia na viaceré zariadenia však predpokladá pomerne priamočiare riešenie. Tento cieľ sa však nepodarilo splniť pre viaceré dôvody:

- V čase vypracovávania úlohy ešte nie je k dispozícii spúšťanie hráča a servera prostedníctvom testovacieho frameworku.
- Pre spustenie simulácie je potrebné nainštalovať na cluser robocup server 3D, čo zasa vyžaduje určitú množinu knižníc, ktoré nie sú na týchto zariadeniach dostupné. Toto by samo o sebe nebolo problémom, pre inštaláciu daných knižníc a robocup servera je potrebné mať prístupové práva, prípadne situáciu riešiť cez ďalšiu osobu, čo poukazuje na dlhodobý charakter tejto úlohy. Alternatívou je štúdium možnosti inštalácie robocup servera iba prostredníctvom kopírovania súborov.
- Pre samotné spustenie MPI je potrebné vedieť pomocou SSH pristupovať na jednotlivé zariadenia, čo momentálne tiež nie je splnené.

Táto úloha ale ostáva do budúcnosti otvorená, keď všetky vymenované problémy skôr poukazujú na jej charakter potreby dlhodobého vypracovania, než neriešiteľnosť.

Zhrnutie ZS

Úspešne vykonaná práca

Model sveta hráča

- Hráč už pozná polohu ostatných hráčov na ihrisku, čo je jeden z kľúčových poznatkov potrebných pre podporu jeho rozhodovania počas hry (kapitola 3.2).
- Okrem polohy ostatných hráčov už pozná aj ich natočenie, ktoré prerátava prostredníctvom pozície ich nôh (kapitola 3.3.2).
- V neposlednom rade je dôležité, že hráč už pozná aj vzdialenosť jednotlivých hráčov od lopty (kapitola 3.3.1).

Testovací framework

- Testovací framework bol prebraný v pomerne zlom stave, preto musel byť signifikantne refaktorovaný. Hlavým prínosom refaktoringu bolo rozdelenie testovacieho frameworku na moduly vykonávajúce činnosti a na implementačné moduly, spúšťa sa iba pomocou jednej inicializačnej metódy, má konzistentné logovanie, pribudol konfiguračný súbor (kapitola 3.1).
- Vďaka obojsmernej komunikácii s hráčom je možné informovať framework o jeho plánovaní veľmi presne (kapitola 4.1).
- Automatické spúšťanie hráča a servera uľahčí život každému, kto bude s týmto nástrojom pracovať. V neposlednom rade podporí automatizáciu testovania a umožní napríklad paralelizáciu prostredníctvom MPI (kapitola 4.7).
- Vytvorené boli testy pohybov, ktoré dokážu ohodnotiť ich úspešnosť, čo je kľučové pre ich budúce zlepšovanie (kapitola).
- Automatické tvorenie anotácií implementované v testovacom frameworku umožňuje pohodlné a presné vytváranie anotácií pohybov (kapitola 4.6).

Pohyby

- Pohyb sadnutia bol optimalizovaný na tú úroveň, že je opäť použiteľný, okrem iného bola vytvorená experimentálna, extrémne rýchla verzia (kapitola 4.4).
- Optimalizácia kopu priniesla viacero úrovní sily kopu špičkou, hráč sa už naďalej neotáča, kopanie bokom sa stalo oveľa použiteľnejšie (kapitola 4.3).
- Vhodne navrhnuté anotácie pohybov budú zákadným kameňom v budúcom rozhodovaní hráča (kapitola 2.1).

Návrhy na prácu do druhého semestra

Hráč

- Hráč by sa mal naučiť plánovať vykonávanie svojich pohybov v závislosti od cieľa, ktorý chce dosiahnuť.

Testovací framework

- Grafické používateľské rozhranie bude potrebovať úpravu a rozšírenie pre možnosti nastavenia anotácií či trénovania

- Do úvahy stále prichádza využitie technológie MPI pre paralelizáciu testovania pohybov (kapitola 4.8)
- Potrebné bude automatizovať vylepšovanie pohybov prostredníctvom implementácie trénera

Pohyby

- Niektoré staršie pohyby by stále potrebovali vylepšenie
- Bude potrebné navrhnúť a implementovať vyššie pohyby
- Do úvahy v budúcnosti prichádza možnosť parametrizovať pohyby

5 Šprint 5

5.1 Optimalizácia pohybov

5.1.1 walk_fine_fast2_optimized2

Chôda walk_fine_fast2_optimized, z ktorej tento pohyb vychádza bola nestabilná. Hráč v mnohých prípadoch nedokázal prejsť väščí úsek naraz a spadol na chrbát. Taktiež skoro vôbec nevyužíval kĺby hráča a ťažil z rýchleho mihania nohami. Nový pohyb bol prerobený do ľudskejšej formy, kedy hráč do istej miery imituje chôdzu človeka. To mu dalo potrebnú stabilitu a bez problémov dokáže prejsť aj celé ihrisko porovnateľnou rýchlosťou ako predošlý pohyb.

5.1.2 walkback_slow

Pohyb dozadu. Vytvorený bol z pohybu walkback3, ktorý je síce výborný, ale hráč vykonáva veľmi veľké kroky dozadu. Bolo nutné vytvoriť pohyb, ktorý bude vykonávať menšie kroky, najmä kvôli pohybom hráča okolo lopty. Výborná stabilita pohybu zostala zachovaná.

5.2 Rozšírenie anotácií

5.2.1 Analýza

V priebehu používania automatických anotácií (najmä pre účely kopania) sa ukázala potreba

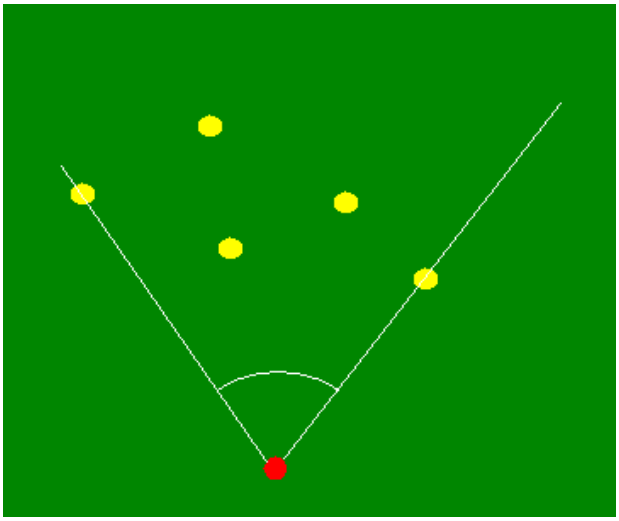
- lepšie reprezentovať vstupné a výstupné podmienky (v rámci polohy lopty),
- vytvárať viacero anotácií pre jeden pohyb,
- pridať rozptyl (v akom rozptyle uhlov sa lopty po vykonaní pohybu nachádzajú),
- určiť bod najvyššej efektivity kopu do lopty.

5.2.2 Návrh

Doteraz bola poloha lopty v rámci vstupných a výstupných podmienok reprezentovaná prostredníctvom maximálnej, minimálnej a priemernej polohy. Toto sa ukazuje ako nedostatočné, keď takáto obdĺžniková reprezentácia môže zavádzať v rámci toho, kde sa naozaj nachádzala lopta. Kruhová reprezentácia má nasledovné výhody:

- stred kruhu dobre reprezentuje, kam zhruba hráč loptu kopne (resp odkiaľ ju kopal),
- polomer kruhu dobre reprezentuje presnosť, teda čím vyššia hodnota polomeru, tým menej presné meranie to bolo,
- je dobre zrozumiteľný,
- je dobre implementovateľný.

V rámci anotácií je vhodné pre lepšie pochopenie, ako sa lopta počas vykonávania pohybu chovala reprezentovať aj rozptyl uhlov, v akom skončila po vykonaní pohybu. Táto hodnota sa dá reprezentovať uhlom, aký je znázornený na nasledovnom obrázku (Obrázok 59).

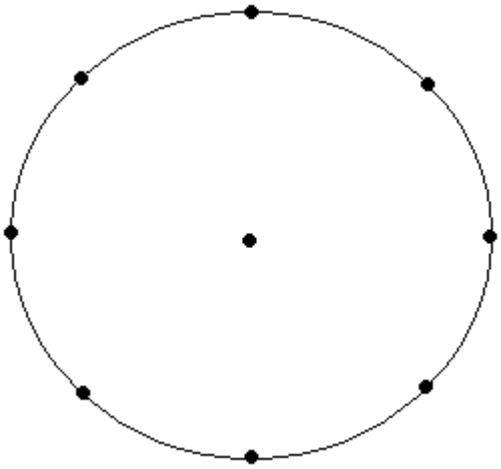


Obrázok 59: uhol vyjadrujúci rozptyl koncových stavov lopty

5.2.3 Implementácia

Kruhová reprezentácia pozície lopty

V rámci vstupného a výstupného stavu je pozícia lopty reprezentovaná kruhom. Vo pre vstupný stav je tento kruh udávaný používateľom pred spustením automatického anotovania zadaním stredovej súradnice a polomeru. V danom kruhu sa tak inicializuje lopta na rôznych miestach, tie sú určené metódou `initPositionsFromCircle(Circle initcircle)`. Túto je možné meniť ľubovoľne podľa predstáv. Inicializácia bodov v súčasnej implementácii rozmietňuje body iba do stredu kruhu a na 8 bodov na jeho okraji – znázornenie je na nasledovnom obrázku (Obrázok 60).



Obrázok 60: Vyznačené body určujú miesta inicializácie lopty pre používateľom daný kruh

Na výstupe sa kruhová reprezentácia počíta ako minimálny spoločný kruh bodov, na ktorých zastavila lopta (počítaný prostredníctvom triedy `MEC`).

Viacero anotácií pre jeden pohyb

Pri vytváraní XML súboru s anotáciou sa program pozerá do používateľom zadaného výstupnéhho priečinka a postupuje nasledovne:

1. nájde všetky anotácie patriace tomu pohybu, ktorý bol práve testovaný,
2. nájde súbor s najvyšším poradovým číslom,
3. vytvorí súbor s poradovým číslo o jeden vyšším.

Všetky tieto operácie prebiehajú na úrovni mien súborov, preto si na ne treba dávať pozor a udržiavať ich v nasledovnom formáte: `meno_pohybu-poradové_číslo_anotácie.xml`.

Popis XML súboru s anotáciami celkovo

Niektoré atribúty, ktoré boli dostupné v anotáciách v minulých verziách už dostupné nie sú (preto, lebo je plánované ich presunutie do XML súboru so samotným pohybom, kde lepšie reprezentujú svoju podstatu – napríklad typ pohybu), chýbajú tiež tie, ktoré boli plánované (napríklad vyššie spomínaný rozptyl uhlov), pretože do dokončenia tohto šprintu neboli presunuté nástroje na tvorbu a čítanie XML súborov do jednej spoločnej knižnice pre framework a hráča. Aby táto práca nemusela byť robená na dva krát, bude toto dokončené neskôr – správne hodnoty sú už ale vo frameworku

dopočítané a reprezentované, chýba iba ich zápis do XML. Nakoniec mnohé atribúty je potrebné dopísať ručne.

Z tohoto dôvodu vzniká určitá nekonzistencia medzi tým, čo anotácie majú obsahovať a čo momentálne obsahujú po ukončení automatického anotovania. V nasledovnom zozname je tak popísaná štruktúra XML súboru s tými atribútmi, ktoré sa do neho reálne zapisujú počas automatického anotovania:

- `annotation` (koreňový atribút)
  - `name` (meno anotovaného pohybu)
  - `description` (detailnejší popis)
    - `duration` (trvanie pohybu – aj s pohybom lopty, pokiaľ sa hýbala dlhšie ako hráč)
    - `rotation` (otočenie hráča v stupňoch okolo jednotlivých osí)
    - `move` (posun hráča v rámci jednotlivých osí)
    - `fall` (pravdepodobnost pádu hráča)
    - `ball_distance` (vzdialenosť, ktorú prešla lopta počas pohybu)
    - `max_ball_distance_position` (iniciálny bod, z ktorého lopta prešla maximálnu vzdialenosť)
    - `ball_move` (vzdialenosť, akú lopta prešla po jednotlivých osiach)
  - `preconditions` (vstupný stav)
    - `ball_positions_circle` (vstupný kruh inicializácií lopty zadaný používateľom)
  - `end_state` (výstupný stav)
    - `ball_positions_circle` (minimálny kruh obsahujúci všetky body, ktoré)

6 Šprint 6

6.1 Rozšírenie pohybov

Táto úloha má nadväznosť na úlohu vytvárania automatických anotácií. Ako sa totiž ukázalo pri takomto vytváraní anotácií, reprezentácia niektorých atribútov, ktoré sú vzhľadom na pohyb zaujímavé, je v nich pomerne zložitá.

Príkladom za všetky je atribút *typ pohybu*, teda či ide o kop, otočenie, chôdzu a podobne. Kým v rámci anotácie by bolo potrebné pre každý automaticky vygenerovaný súbor manuálne dopĺňať túto hodnotu, reprezentovaním tohto atribútu v rámci samotného pohybu by znamenalo jednorazovú záležitosť (pričom vytváranie pohybu je doposiaľ čisto ľudskou záležitosťou). Takáto informácia o type pohybu je navyše spojená čisto s daným pohybom a vôbec nie s meraním daného pohybu, čo je skôr záležitosť samotnej anotácie. Preto bolo vhodné rozšíriť štruktúru XML súborov s pohybmi.

Pridané boli teda v rámci atribútu low_skill nasledovné atribúty:

- `type` (typ pohybu – môže nadobúdať hodnoty: fall, look, kick, stand_up, rotation, walk, other)
- `author` (autor daného pohybu)
- `extendedFromMove` (z akého pohybu sa pri danom pohybe vychádzalo)
- `description` (voľný popis pohybu – či je stabilný, rýchly, pomalý, dobrý, zlý, čo vlastne robí a pod...)

Tieto atribúty boli následne pre všetky už implementované pohyby dodatočne vyplnené.

6.2 Testy na súťaže

Pre jednoduchšie meranie výsledkov a porovnávanie úpra alebo viacerých hráčov boli v testovacom frameworku implementované testy pre jednotlivé disciplíny v rámci súťaže „Robocup at FIIT'11“.

6.2.1 Pravidlá súťaže

Vstávanie z brucha

- hráč leží na bruchu, všetky kĺby má nastavené v polohe, v akej ich mal pri prihlásení na server s jedinou výnimkou, robot musí mať ruky pripažené (tzn. pri tele)
- spolu je umožnených 5 pokusov
- výsledok je súčet časov
- za neúspešný pokus je pripočítaných 20 sekúnd nadčas,
- meranie času sa zastaví po 20 sekún alebo 1 sekundu po tom, ako sa robot dostane do vzpriamenej polohy a prestane sa viditeľne hýbať

Vstávanie z chrbta

- hráč leží na chrbte, všetky kĺby má nastavené v polohe, v akej ich mal pri prihlásení na server s jedinou výnimkou, robot musí mať ruky pripažené (tzn. pri tele)
- umožnených 5 pokusov
- výsledok je súčet časov
- za neúspešný pokus je pripočítaných 20 sekúnd nadčas
- meranie času sa zastaví po 20 sekúnd alebo 1 sekundu po tom, ako sa robot dostane do vzpriamenej polohy a prestane sa viditeľne hýbať

Chôdza (stabilita)

- hráč stojí na nohách, všetky kĺby má nastavené v polohe, v akej ich mal pri prihlásení na server
- meria sa čas, za ktorý sa robot dostane na druhú polovicu ihriska
- 2 pokusy, pričom sa započítava lepší čas
- meranie času sa zastaví vo chvíli, keď sa ťažisko robota dostane na vzdialenejšiu polovicu ihriska oproti pozícii, z ktorej chôdza začala.
- za každý dotyk hracej plochy inou časťou robota ako je chodidlo sa k výslednému času pripočíta 10 trestných sekúnd
- ak robot nepríde do cieľa za 180 sekúnd, vypočíta sa výsledný čas lineárnou aproximáciou podľa prejdenej vzdialenosti (so zarátaním trestných sekúnd).

Chôdza (rýchlosť)

- počiatočná poloha: hráč stojí na nohách, všetky kĺby má nastavené v polohe, v akej ich mal pri prihlásení na server
- meria sa čas, za ktorý sa robot dostane na druhú polovicu ihriska
- 2 pokusy, počíta sa súčet časov.
- meranie času sa zastaví vo chvíli, keď sa ťažisko robota dostane na vzdialenejšiu polovicu ihriska oproti pozícii, z ktorej chôdza začala.

- za dotyky hracej plochy inou časťou robota ako je chodidlo sa trestné sekundy nepripočítavajú.
- ak robot nepríde do cieľa za 180 sekúnd, vypočíta sa výsledný čas lineárnou aproximáciou podľa prejdenej vzdialenosti.

Kopanie do lopty (vzdialenosť)

- počiatočná pozícia je ľubovoľná
- meria sa vzdialenosť, do akej sa lopta po kopnutí dostane
- 5 pokusov, započítava sa pokus s najväčšou vzdialenosťou
- ak hráč počas kopnutia spadne (tzn. dotkne sa ihriska inou časťou tela ako chodidlami), vydelí sa výsledná vzdialenosť dvomi.

Kopanie do lopty (presnosť)

- počiatočná pozícia: (?; 0)
- počiatočná poloha: hráč stojí na nohách, hráč je otočený čelom k stredu súperovej brány
- meria sa odchýľka smeru kopnutia od smeru k niektorému z rohov ihriska
- 5 pokusov, započítava sa pokus s najmenšou odchýľkou smeru lopty od smeru k niektorému z rohov ihriska.
- ak hráč počas kopnutia spadne (tzn. dotkne sa ihriska inou časťou tela ako chodidlami), vynásobí sa výsledný uhol dvomi.
- ak lopta po kopnutí neprejde minimálne 1,5 metra, pokus sa považuje za neplatný.

Otočenie hráča

- počiatočná poloha: hráč stojí na nohách, hráč je otočený čelom k stredu brány
- meria sa čas, za ktorý sa hráč otočí okolo svojej osi o 180 stupňov s toleranciou 5 stupňov.
- 2 pokusy, započítava sa rýchlejší
- meranie času sa zastaví vo chvíli, keď hráč dosiahne predpísané otočenie a je pri tom vo vzpriamenej polohe.

Voľná jazda

- počiatočná poloha/pozícia je ľubovoľná
- vyhodnocuje sa umelecký dojem, estetika pohybov, zábavnosť predvedeného výkonu, náročnosť technického riešenia
- robot má 2 minúty na to, aby na ihrisku „niečo" predviedol, organizátori súťaže môžu tento čas v odôvodnených prípadoch predĺžiť
- je možné spustiť naraz aj viacero robotov (treba ale počítať so zníženým výkonom servera)
- hodnotenie sa uskutoční na základe hlasovania súťažiacich a divákov

6.2.2 Implementácia testov

Vstávanie z brucha a chrbta

Na vstávanie brucha a chrbta je implementovaný jeden rovnaký test. TestFramework v súčastnosti nevykonáva kontrolu či hráč leží na bruchu alebo na chrbte. Pri spustení testu sa samotné meranie nespustí pokým sa hra neprepne do módu „PlayOn". Na tento signál by mal reagovať aj hráč a podľa toho začat následne vykonávať pohyb.

Test sa ukončí po uplynutí 20 sekúnd alebo po jednej sekunde bez pohybu kedy overí či hráč stojí podla danej definíce vzpriamenej polohy (ťažisko 38 cm nad zemou).

Test je implementovaný v triede „StandUpTest“ a sériu viacerých testov a ich následné vyhodnotenie je implementované v triede „StandUp“. Táto trieda je obyčajný Runnable objekt ktorý sa stará o viacnásobné spúštanie testov a zbieranie ich výsledkov.

Chôdza (stabilita a rýchlosť)

Samotné meranie testu sa začne po prechode hry na mód „PlayOn“. V tomto momente by hráč mal začat vyknonávať svoj pohyb. Na začiatku testovania je hráč presunutý na miesto ku bráne a lopta je presunutá mierne za stredovú čiaru. Úlohou hráča je prejsť za stredovú čiaru a nespadnúť pri tom.

Test sa ukončí po oplynutí 180 sekúnd alebo po úspešnom prejdení polky ihriska. Ďalej testovanie vykonáva kontrolu či hráč nebol resetnutý serverom (za veľmi krátky čas prešie veľmi veľku vzdialenosť za čiaru ihriska).

Pri testovaní stability chôdze je za každý pád pripočítavaný čas ku finálnemu času ako penalkta. Pád sa testuje kontrolou výšky ťažiska hráča.

Opisované testy boli implementované v triedach „WalkStableTest“ a „WalkFastTest“. Ich spúštanie je možné vykonať pomocou tried „WalkStable“ a „WalkFast“. Tieto dve triedy sú Runnable objekty ktoré sa starajú o viacnásobné spúštanie testov a zbieranie ich výsledkov.

Kopanie do lopty (vzdialenosť)

Samotné meranie testu sa začne po prechode hry na mód „PlayOn“. V tomto momente by hráč mal začat vyknonávať svoj pohyb. Na začiatku testovania je hráč presunutý do stredu ihriska pričom lopta je posunutá pred hráča Úlohou je kopnúť loptu čo najďalej.

Vyhodnotenie testu spočíva v meraní vzdialenosti, ktorú prešla lopta. Ak je po kopnutí hráč na zemi, celková výsledok je vydelený dvomi.

Otočenie hráča

meranie testu sa začne po prechode hry na mód „PlayOn“. V tomto momente by hráč mal začat vyknonávať svoj pohyb. Na začiatku testovania je hráč presunutý do stredu ihriska.

Ukončenie testu na základe rotácie hráča ešte nebolo implementované. Vyskytli sa problémi pri počítani jeho otočenia. Hráča sa pri otáčaní stálen nakláňa na rôzne strany čo výpočet rotácie robí obťiažnym a v pravidlách turnaja nieje stanovené že hráč sa musí po otočení zastaviť a ostať v vzpriamenej polohe.

Voľná jazda

Test na túto disciplínu nebol implementovaný, nakoľko sa hodnotí hlasovaním a je subjektívne.

6.3 Vylepšenie GUI testovacieho frameworku

Daná úloha vznikla na základe dvoch požiadaviek – kvôli neprehľadnosti kódu a zhoršenej funkcionalite pôvodného GUI.

6.3.1 Analýza

Počas používania a snaženia sa o rozšírenie pôvodného GUI vzniklo niekoľko základných požiadaviek na nové GUI

- refaktoring, prípadne vytvorenie úplne novej verzie
- možnosť pridávania nových agentov pomocou GUI
- možnosť sledovania atribútov nových agentov
- možnosť prepínania medzi sledovanými agentmi
- vypisovanie log informácií priamo do GUI
- možnosť výberu aké log informácie chceme vypisovať

Na základe našej predchádzajúcej práce vznikla úplne nová požiadavka, a to vytvorenie vlastného tabu pre možnosti anotovania.

6.3.2 Návrh

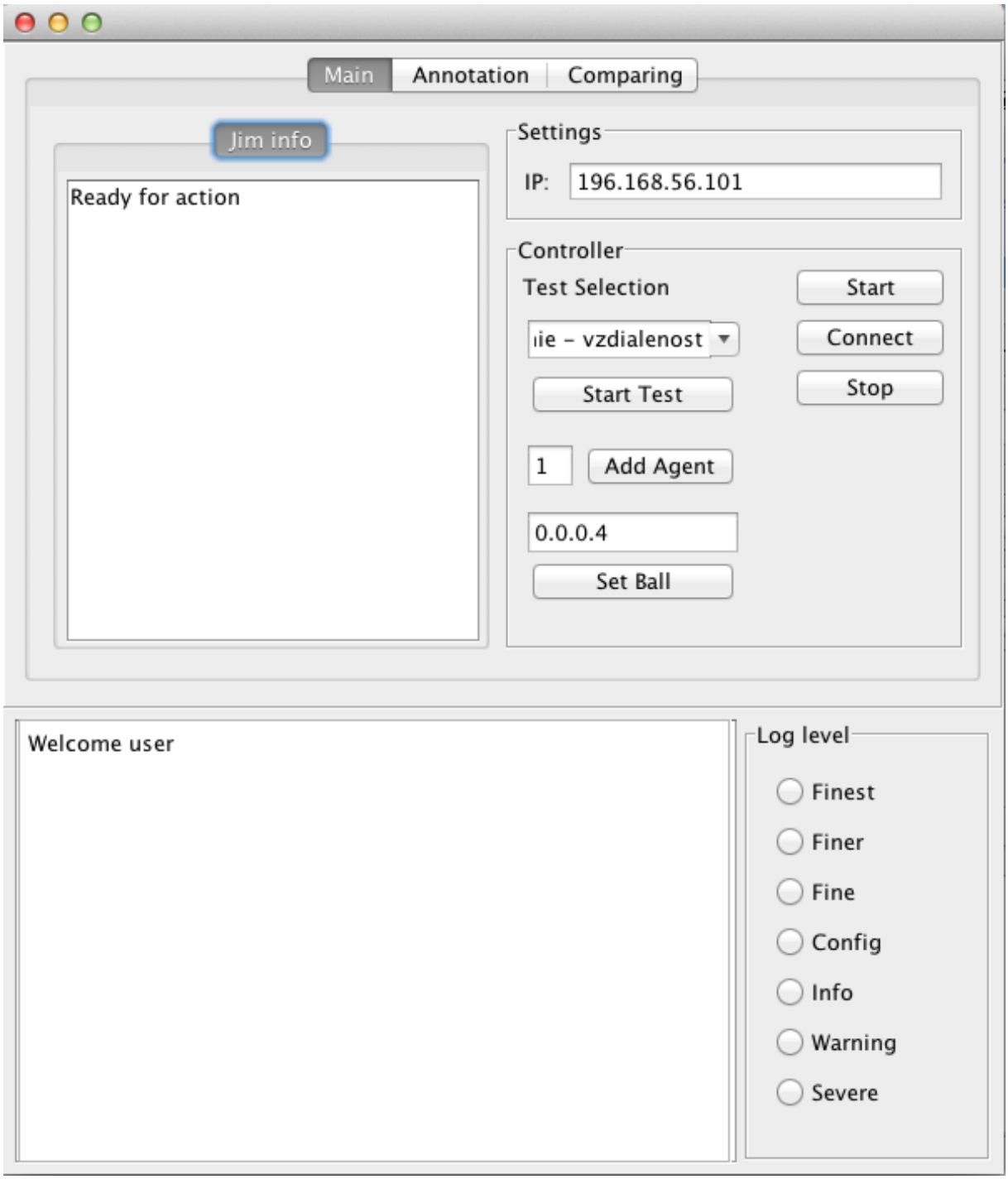
Návrh GUI testovacieho frameworku bol vytvorený na základe vyššie definovaných požiadaviek a prekonzultovaný s celým tímom a hlavne Mirom Bimbom, ktorý mal na starosti vytváranie anotácií. Dané GUI počas návrh prešlo niekoľkmi zmenami, či už grafickými ako aj funkčnými podľa potrieb.

6.3.3 Implementácia

Do konkrétnej implementácie sa dostala väčšina z návrhov. Na základe analýzy som sa rozhodol zdrojový kód starého GUI vôbec nepoužiť a iba sa jemne inšpirovať grafickou časťou. Tým pádom nebol refaktoring potrebný a bolo spustená funkčnosť pridávania nových hráčov a najdôležitejšou časťou bola možnosť sledovania logov a možnosť ich konkrétneho výberu zo 7 kategórií.

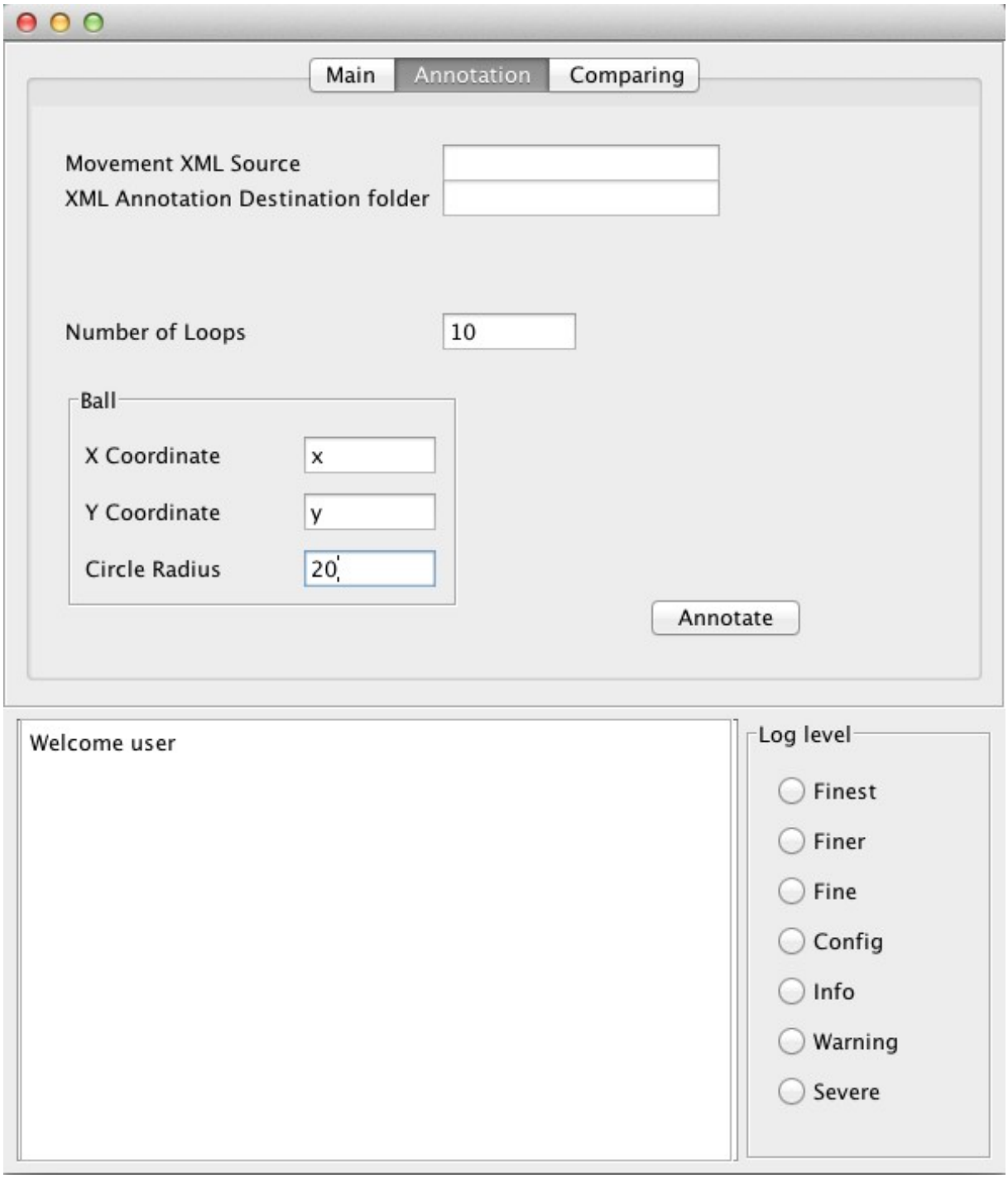
- Finest (najmenej detailov)
- Finer
- Fine
- Config
- Info
- Warning
- Severe (najviac detailov)

Podľa výberu levelu logovania sa nám následne zobrazovali všetky detaily od danej úrovne vyššie (viď Obrázok 61). Tým pádom ak sme si vybrali level logovania Info, zobrazovali sa nám informácie od levelu Info až po level Finest.



Obrázok 61: Nové Framework GUI - úvodná obrazovka

Druhou významnou časťou bola implementácia anotovacieho tabu (Obrázok 61). Pri vytváraní anotácií je potrebné zadať druh pohybu, počet opakovaní pri anotovaní, následne inicializačnú polohu lopty a rádius okolo danej polohy, kde bude daná lopta umiestňovaná. Poslednou položkou je priečinok, kde bude daná anotácia vytvorená.



Obrázok 62: Nové Framework GUI - anotovací tab

6.4 Refaktoring a vytvorenie spoločných knižníc

Daná úloha vznikla na základe dvoch požiadaviek – kvôli neprehľadnosti kódu a z dôvodu viacnásobnej duplicity kódu, či už v rôznych triedach Jima alebo Testovacieho frameworku, ako aj veľmi podobné, dokonca mnohokrát tiež duplicitné triedy v oboch projektoch.

6.4.1 Analýza

Analýza daného problému vznikala počas celej doterajšej práce, až daný problém vyústil do vytvorenia konkrétnej úlohy. Refaktoring by mal prebehnúť v hlavných baličkoch, kde podobnosť, či prípadná duplicita tried bude na veľmi vysokej úrovni. Analýza odhalila podobnosť kódu v 7 balíčkoch.

6.4.2 Návrh

Navrhnutý bol refaktoring kódu na základoch správneho písania kódu v objektovo-orientovaných jazykoch a následne odstránenie duplicitných častí. Vytvorenie úplne nového projektu, v ktorom sa budú nachádzať všetky spoločné časti, využívané Jimom ako aj Testovacím frameworkom, ktorý bude slúžiť ako knižnica pre obydva projekty.

6.4.3 Implementácia

Na základe daného návrhu bol vytvorený tretí projekt s názvom RoboCupLibrary, ktorý bol následne importovaný do Jima ako aj do testovacieho frameworku. V danom projekte vzniklo 5 balíčkov

- sk.fiit.robocup.library.annotations
- sk.fiit.robocup.library.geometry
- sk.fiit.robocup.library.init
- sk.fiit.robocup.library.math
- sk.fiit.robocup.library.review

Najpodstatnejšími časťami sú knižnice geometry a math, ktoré zahŕňajú všetky matematické a geometrické výpočty doteraz implementované v oboch projektoch.

7 Šprint 7

7.1 Analýza pohybu na základe údajov z akcelerometra a gyroskopu

Hráči si každý takt servera spracovávajú informácie zo svojich receptorov. Medzi ne patrí aj akcelerometer a gyroskop. Akcelerometer slúži na zobrazenie momentálneho preťaženia v určitom smere, čiže zaznamenáva zmenu iba počas vykonávania pohybu. Gyroskop ukazuje presné natočenie hráča v rámci súradnicovej sústavy a teda poskytuje dáta aj v prípade, že sa hráč práve nepohybuje.

Obidva receptory fungujú na báze klasickej súradnicovej sústavy, takže používajú osi X, Y a Z. Pre akcelerometer ukazujú zrýchlenie tela hráča v smere jeho jednotlivých osí (X – doprava a doľava, Y – dopredu a dozadu, Z – gravitácia a preťaženie smerom nadol). Avšak, dáta z gyroskopu (po patričnej úprave, zobrazenej nižšie) znázorňujú rotáciu polohy hráča okolo jednotlivých osí (X – dopredu a dozadu, Y – doprava a doľava, Z – smer chôdze vzhľadom na pomyselnú zvislú priamku). Z toho vyplýva, že dáta z osi X akcelerometra súvisia s dátami z osi Y gyroskopu a naopak.

Príprava na analýzu

Pre vykonanie analýzy je najskôr potrebné upraviť surové dáta do požadovaného tvaru. Dáta z akcelerometra nám hovoria aktuálnu hodnotu preťaženia v jednom takte, čo je pre nás výhodné. Upraviť potrebujeme dáta z gyroskopu, ktoré zo servera prichádzajú v tvare uhlovej rýchlosti trupu. Najskôr je nutné získať celkovú polohu hráča a potom túto prekonvertovať z radiánov na stupne.

Akcelerometer

```
System.out.println(data.accelerometer.getX() + ";" +
data.accelerometer.getY() + ";" + data.accelerometer.getZ());
```

Uprava dat z gyroskopu

```
rotationX += Math.toRadians(gyroscope.getX() * TIME_STEP);
rotationX = Angles.normalize(rotationX);
rotationY += Math.toRadians(gyroscope.getY() * TIME_STEP);
rotationY = Angles.normalize(rotationY);
rotationZ += Math.toRadians(gyroscope.getZ() * TIME_STEP);
rotationZ = Angles.normalize(rotationZ);
```

Gyroskop

```
System.out.println(Math.toDegrees(rotationX) + ";" +
Math.toDegrees(rotationY) + ";" + Math.toDegrees(rotationZ));
```

Po takýchto úpravách môžme spustiť pohyb, nazbierané dáta vložiť do tabuľkového procesora a nechať si vykresliť graf.

Čítanie z grafu

Pre účely tohto testu sme použili pohyb walk_fine_fast2_optimized, z tvorivej dielne tímu Číslo 17 žije. Je to chôdza, ktorá sa ale vďaka snahe o veľkú rýchlosť stal nestabilnou a je takmer isté že po niekoľkých desiatkách krokov hráč spadne na chrbát. Zozbierané dáta sme vložili do grafu.

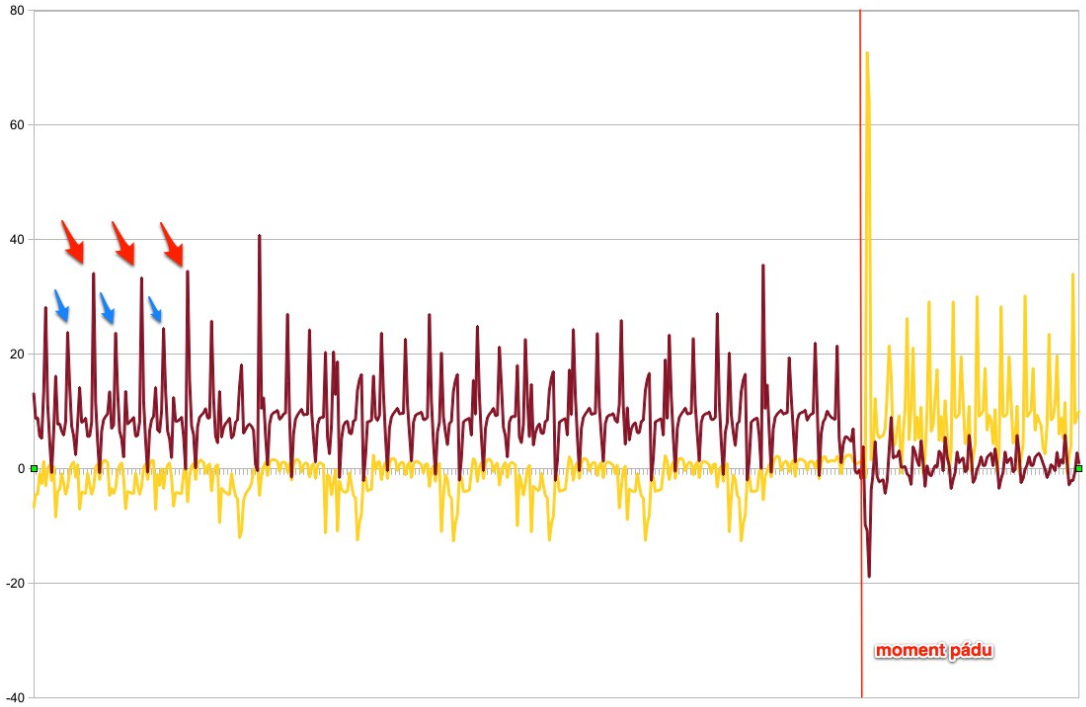
Keďže sa jedná o chôdzu, robot sa nakláňa iba v smere dopredu – dozadu. Preto môžeme zanedbať dáta znázorňujúce nakláňanie sa bokom doprava a doľava (Os X akcelerometra a os Y gyroskopu).

Akcelerometer

Keďže sme zanedbali os, ktorá znázorňuje zrýchlenie doprava a doľava, zostali nám osi znázorňujúce zrýchlenie dopredu a dozadu (žltá krivka) a gravitačné zrýchlenie (fialová krivka). Zvislá červená priamka znázorňuje moment pádu hráča na chrbát.

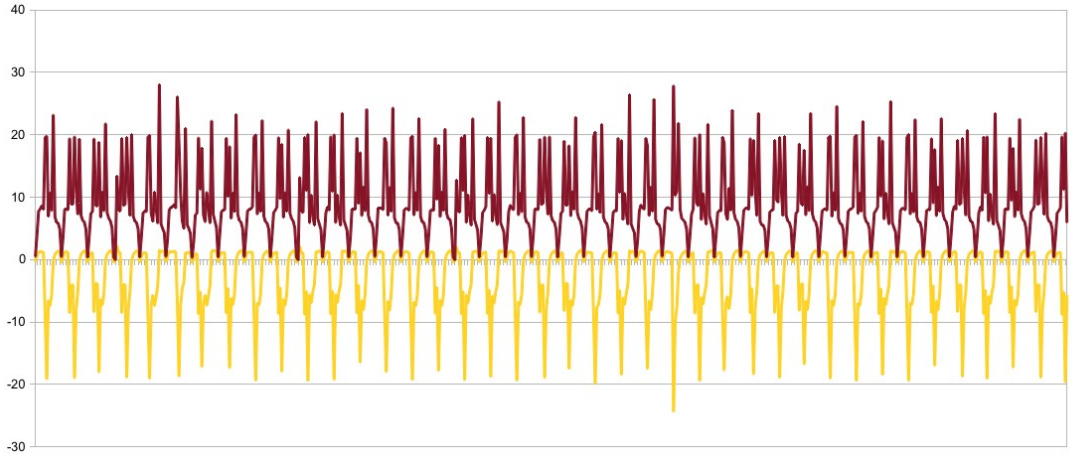
Z grafu (Obrázok 63) sa dajú vyčítať viaceré zaujímavé informácie o pohybe:

- Hráč sa pri pohybe periodicky kolíše dopredu, čo možno vidieť na mínusovej časti žltej krivky (os Y akcelerometra)
- Pri páde sa hráč zvalí smerom dozadu na chrbát a potom sa dalej hýbe na zemi – krásne vidieť zýšenie momentálneho preťaženia smerom dozadu
- Fialová krivka znázorňujúca zrýchlenie smerom k zemi (aj gravitačné) ukazuje, že počas krokov pravou a ľavou nohou na hráča nepôsobí rovnaké preťaženie – čo sa napokon aj potvrdilo pri skontrolovaní definície dotyčného pohybu, kde fázy pohybu oboch nôh neboli rovnako rýchle (modré a červené šípky). Jednou z nôh hráč výkoná krok rýchlejšie a teda preťaženie na najnižšej polohe nohy je vyššie. Toto môže mať vplyv napríklad na trajektóriu chôdze.



Obrázok 63: Graf pádu pri pohybe walk_fine_fast2_optimized (akcelerometer)

Pre porovnanie uvedieme aj graf (Obrázok 64), ktorý vznikol pri stabilnej chôdzi. Je možné vidieť, že hodnoty nie sú také rozkývané.



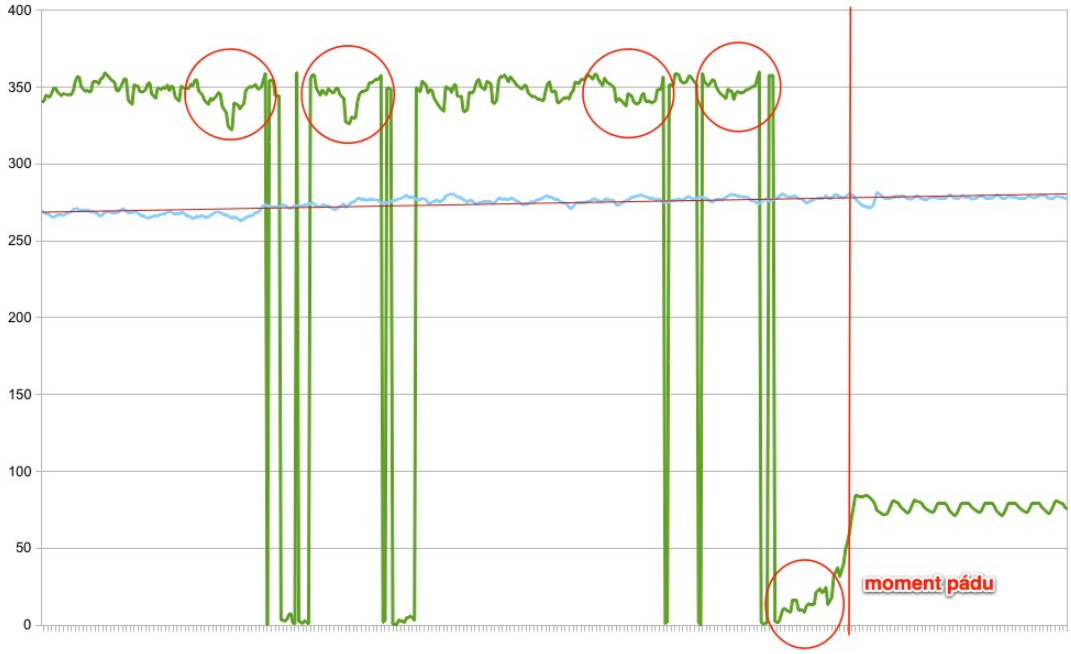
Obrázok 64: Graf stabilnej chôdze (akcelerometer)

Gyroskop

Aj pri tomto grafe (Obrázok 65) bola vynechaná krivka znázorňujúca náklon doprava a doľava (os Y gyroskopu). Zelená krivka znázorňuje os X gyroskopu (poloha hráča smerom dopredu a dozadu) a svetlomodrá krivka ukazuje otočenie trupu okolo pomyselnej zvislej čiary – osi Z (a síce aj smer chôdze). Zelená krivka sa môže javiť ako chybná, ale zvislé čiary sú spôsobené osciláciou hodnôt medzi prvým a štvrtým kvartálom jednotkovej kružnice (stredná hodnota je 0 stupňov = 360 stupňov).

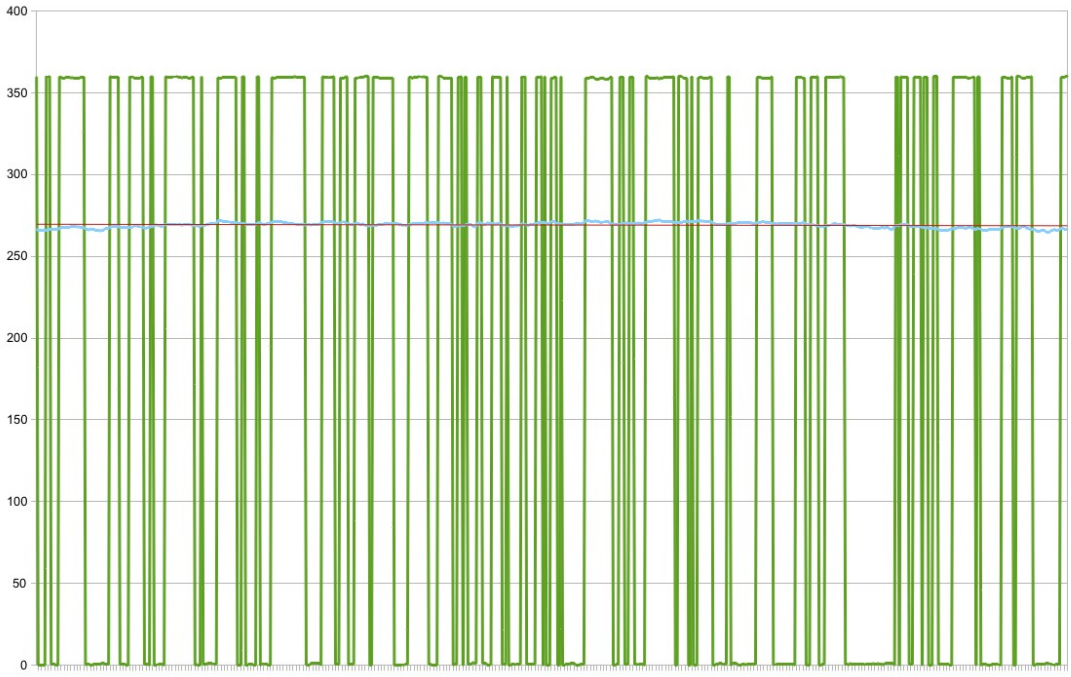
Pri analýze kriviek vieme vysloviť nasledujúce zistenia:

- Hráč sa počas vykonávania pohybu niekoľko krát výrazne naklonil dopredu (horné štyri znázornené vychýlenia).
- Pred momentom pádu prichádza výrazné vychýlenie dozadu, ktoré už hráč nakoniec nevyvážil (spodné znázornené vychýlenie).
- Jasne vidieť, kedy už hráč leží na chrbte.
- Lineárnou aproximáciou hodnôt otočenia vzhľadom na os Z (svetlomodrá krivka) môžeme vidieť, že hráč sa nepohyboval po rovnej priamke, ale postupne sa vychyľoval do jedne zo strán. Toto sa dá v budúcnosti aplikovať do adaptívnej chôdze, kde hráč sám detekuje zmenu smeru a zmenou niektorých atribútov pohybu ho v reálnom čase upraví.



Obrázok 65: Graf pádu pri pohybe walk_fine_fast2_optimized (gyroskop)

Aj tu pre porovnanie uvedieme graf, ktorý vznikol pri stabilnej chôdzi. Je možné vidieť, že lineárna aproximácia ukazovateľa smeru chôdze je vodorovná, takže hráč ide rovno dopredu. Zelená krivka ukazujúca pohyb dopredu a dozadu neprestajne osciluje okolo nuly, takže pohyb je stabilný.



Obrázok 66: Graf stabilnej chôdze (gyroskop)

Zhrnutie

Týmito analýzami jednotlivých hodnôt z akcelerometra a gyroskopu sme dokázali rozoznať, ktoré hodnoty je nutné sledovať aby sme vedeli predpovedať možný pád robota. Do budúcna sa to dá využiť ako pomôcka pri tvorení pohybov ale aj ako navigácia pri ataptabilnej chôdzi, kde napríklad informácia o zmene smeru chôdze môže pomôcť pri jeho napravení.

7.2 Koexistencia viacerých plánovačov

7.2.1 Analýza

Potreba mať implementoovaných viacero plánovačov vyšla z nasledovných dôvodov:

- rôznorodosť tímov, ľudí pracujúcich na plánovaní,
- možnosť mať viacero plánovacích stratégií a ich následné porovnávanie,
- potreba napr aby hráč neplánoval sám žiadne pohyby, ale aby čakal na príkaz od testovacieho frameworku (napr. Annotator)

7.2.2 Implementácia

Každý plánovač sa nachádza v `./scripts/plan/`. Hlavný plánovač je `plan.rb` s implementovanými metódami, ktoré môžu byť znovu použité aj v iných plánovačoch. Sám ale neplánuje nič.

Ostatné plánovače dedia od `plan.rb`, implementujú metódu `replan`, ktorá určuje aký pohyb sa bude vykonávať na základe stavu hry, hráča, atď.

Spustenie konkrétneho plánovača je možné prostredíctvom `./scripts/config/settings.rb` pomocou príkazu `Settings.setValue("Planner", "Meno triedy plánovača")`

PlanOld

Tento plánovač je vytvorený prakticky ešte tímom Androids. Je predurčený na priame vykonávanie konkrétneho zadaného HighSkillu (alebo na vykonávanie ničoho). Keď hráč spadne, znovu sa postaví, keď je hra v beamable móde, hráč vykoná HighSkill `Beam.rb`.

Plan5ko

Plánovač vytvorený v tíme 17, nebol zatiaľ zdokumentovaný.

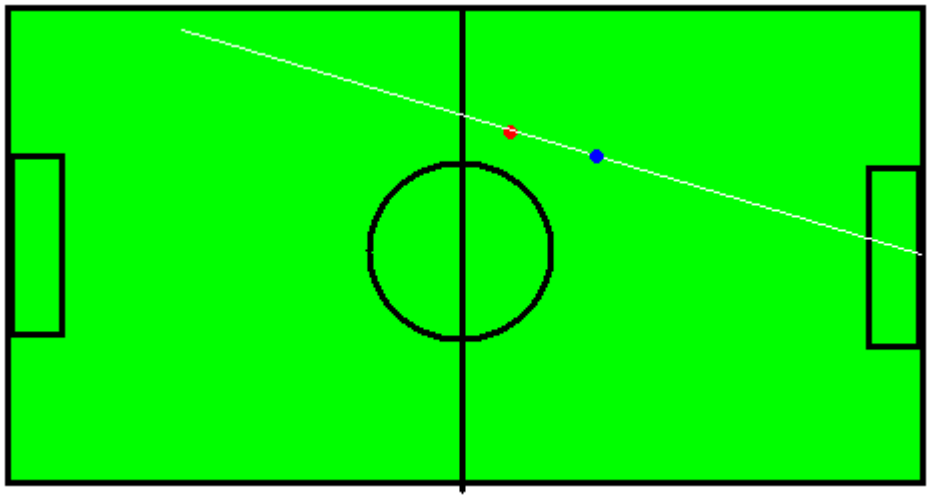
7.3 Pohyb hráča za loptu

7.3.1 Analýza

V tejto fáze implementácie už hráč vie robiť základné pohyby (LowSkills), tj. chodiť, postaviť sa, kopať a pod. Podstatné je ale naučiť ho aj spájať takéto pohyby do postupností tak, aby z toho pramenila hra (HighSkills). Jedným z takýchto vyšších pohybov by mohol byť taký, ktorý postaví hráča vždy za loptu tak, aby sa lopta nachádzala medzi ním a cieľom, na ktorý chce kopať. Inak povedané, *postaviť hráča za loptu*.

7.3.2 Návrh

Dosiahnuť, aby sa hráč dostal za loptu je možné viacerými spôsobmi. Pri tomto riešení som sa pokúsil čo najväčšmi použiť matematiku priamok v rovine. Vhodne toto riešenie opisuje nasledovná séria obrázkov. Na obrázku 67 je možné vidieť, ako by sme chceli, aby hráč (červená bodka) stál. Napríklad ak chce kopať loptu (modrá bodka) smerom na pravú bránku, mal by byť na priamke spájajúcej loptu a stred brány. Takúto priamku volajme *smerová priamka*.

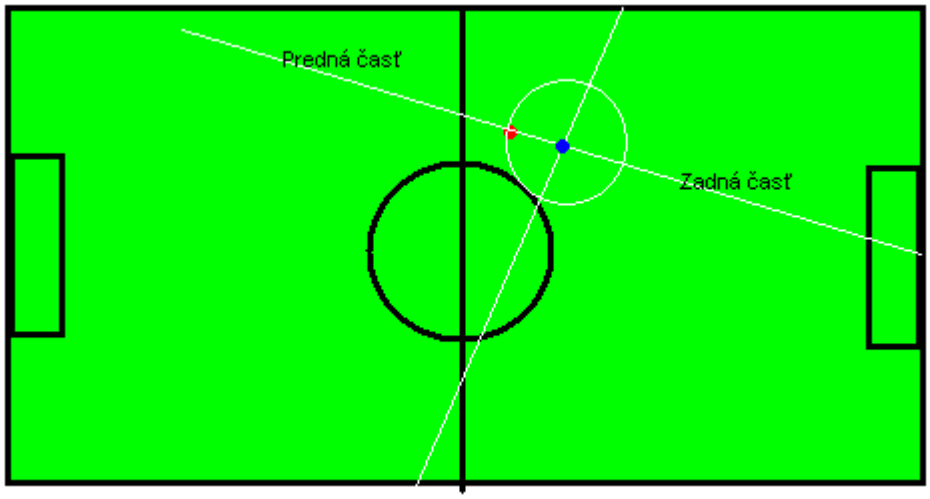


Obrázok 67: Želané postavenie hráča (červený bod) pred snahou o kop lopty (modrá bodka) na bránku

Aby sa však hráč postavil za loptu a nie pred ňu, môžeme ihrisko rozdeliť na dve časti pomocou priamky kolmej na smerovú priamku, ktorú pretína v bode, kde sa nachádza lopta (volajme ju *kolmá priamka*). Táto priamka teda delí ihrisko na časť, kde chce hráč stáť (volajme ju *predná časť*) a kam chce kopať (volajme ju *zadná časť*). Toto môžeme vidieť na obrázku 68.

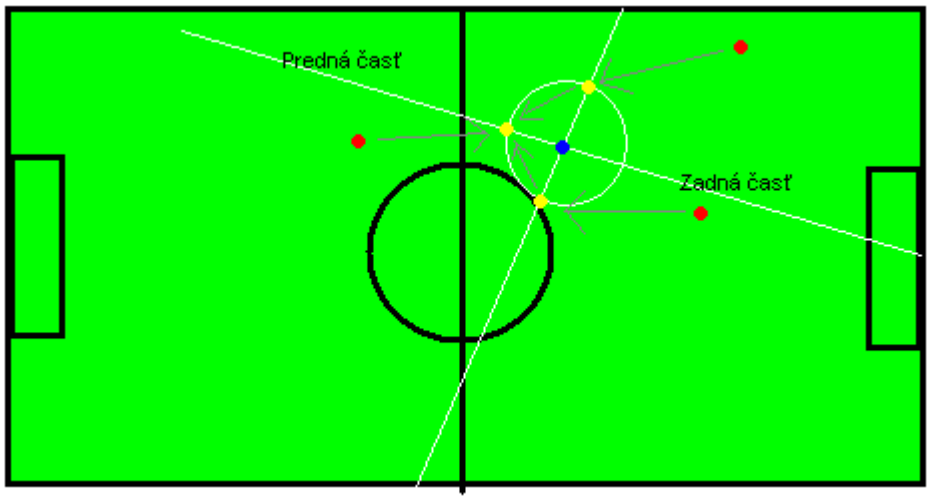
Takto bude hráč určovať, či je za loptou alebo pred ňou. Toto má oproti použitiu pripočítavania a odpočítavania súradníc tú výhodu, že ak je hráčov cieľ napríklad nahrávka spoluhráčovi kolmo smerom na smer hry (tj. zo strany na stranu), hráč nie je závislý od porovnávania súradníc ihriska. Jednoducho povedané, nemusí byť vždy dobré, aby sa hráč staval bližšie ku svojej bráne – dobré je, ak lopta stojí medzi ním a cieľom, kam chce kopať (nahrávať).

Aby sa hráč postavil v určitej vzdialenosti od lopty, určíme si aj kružnicu okolo lopty s danou vzdialenosťou. Hráč sa potom postaví na prienik kružnice a smerovej priamky vo vhodnej polrovine.



Obrázok 68: Rozdelenie ihriska na dve časti podľa priamok

Navyše, pokiaľ sa hráč nachádza v *zadnej časti*, mohol by pri návrate do prednej časti nechtiac kopnúť do lopty. Aby sa tomuto vyhol, môže využiť body, ktoré vznikajú na prieniku *kolmej priamky* a kružnice okolo lopty. Zo *zadnej časti* teda namiesto priamej chôdze k cieľovému bodu pôjde najprv k takýmto bočným bodom a až potom cieľovému bodu. Z prednej časti môže ísť rovno k cieľovému bodu. Príklad takýchto situácií je možné vidieť na obrázku 69.

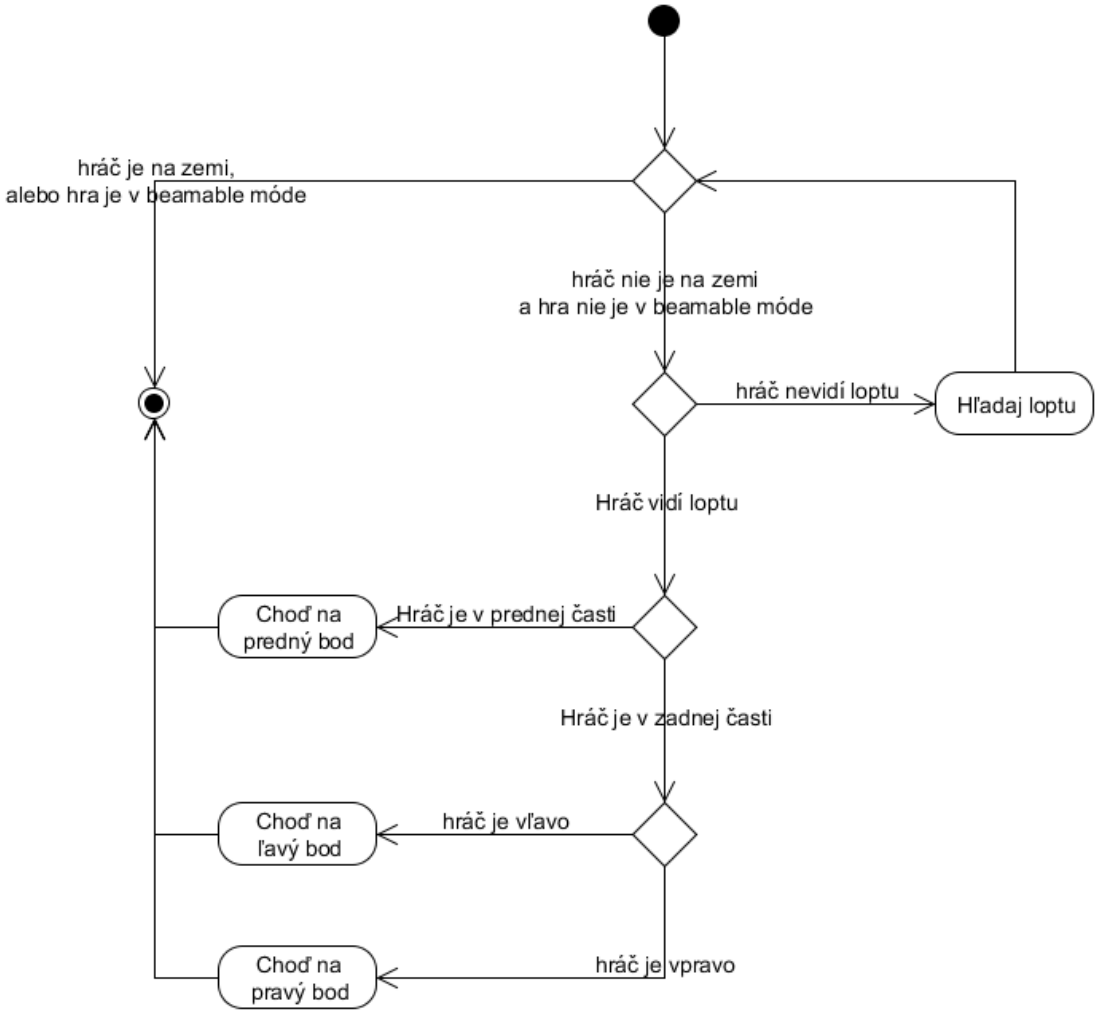


Obrázok 69: Tri rôzne situácie hráča (červené body) a ich riešenia (šedé šípky).

7.3.3 Implementácia

Tento pohyb je implementovaný ako vyšší pohyb v súbore `WalkBehindBall.rb`, diagram jeho rozhodovania je možné vidieť na nasledovnom obrázku (Obrázok 70). Ako chôdza je použitý vyšší pohyb `WalkOld.rb` (mierne upravená pôvodná chôdza od Androidov), ktorý vráti `nil`, ak sa dostane

dosť blízko požadovanému cieľu. Na hľadanie lopty je použitý vyšší pohyb `Localize.rb`, napísaný tímom 17.



Obrázok 70: Diagram rozhodovania hráča idúceho poza loptu

7.3.4 Testovanie

Počas testovania sa ukazuje, že tento návrh možno nebude až taký vhodný, ako sa očakávalo. Problémom je najmä to, že hráč počíta polohu seba a teda aj lopty z nepresných informácií (rozdiely oproti realite sú niekedy aj 2 metre!), a kým sa dostane na požadované miesto, myslí si, že sa lopta nachádza niekde úplne inde ako predtým (aj keď stála na jednom mieste). Rozpory samozrejme tvorí aj vždy rôzna informácia o polohe samotného hráča. Končí sa to často poskakovaním z miesta na miesto, alebo chôdzou na úplne nevhodné miesto.

Tento pohyb je zatiaľ prakticky pre hru nepoužiteľný, treba ho teda v ďalšom priebehu zlepšovať.

7.4 Prenos modelu sveta z Jim do TestFramework

Testovací framework umožnuje prijímať správy od hráča, čo mu umožnuje lepšie vyhodnocovať jeho správanie. Na posielanie správ je vytvorené štandardné TCP spojenie. Typická správa nepresahuje dĺžku jedného riadka a obsahuje číslo hráča, meno tímu, typ správy a posielané hodnoty.

Pridaná bola podpora na nový typ správy, ktorý umožnuje posielať binárne dáta. Začiatok správy je identický s ostanými správami, avšak na konci obsahuje textový reťazec „beginData“. Následne je na ďalšom jednom riadku očakávaný „sha1“ hash dát, na kontrolu správnosti prenesených údajov. Správa ďalej pokračuje dátami zakódovaným polom bajtov dát v Base64 formáte pre bezpečné posielanie cez Java bjekty StreamReader a StreamWriter. Posledný riadok správny musí končit textovým reťazcom „endData“.

Príklad správy je následovný:

```
(1 ANDROIDS LEFT) worldmodel beginData
6488e0907bc62ad6429963910a7c6c975d44275b
rO0ABXNyACNzay5maWl0LmppbS5hZ2VudC5tb2RlbHMuV29ybGRNb2RlbM8irrx9aZAgAETAAK
8FbJQDWQwJTKvbpzcQB+AAg/QAAAAAA...ADHcIAAAAEAAAAAB4c3EAfgAIP0AAAAAAAAx3CAAA
ABAAAAAAeA==
endData
```

Do parsera správ od hráča v Testovacom Frameworku bola implementovaná logika na spracovanie tohto typu dát. Posielaný objekt musí byt v classpath danej aplikácie a je následne deserializovaný a daný na spracovanie metóde v AgentMonitorMessage. Tento objekt sa postará o vytvorenie správnej inštancie AgentMonitorMessage objektu a notifikovanie registrovaných listenerov. Pre typ správy „WorldModel“ bola vytvorená nová konštanta typu správy „WORLD_MODEL“ v objekte AgentMonitorMessage.

Pridaná metóda do triedy AgentMonitorMessage je nasledovná:

```
parse(String firstLine, Object deserialized) {
      // spracovanie prveho riadka spravy
      ...
      // vytvorenie modelu sveta z poslanych dat
      WorldModel message = new WorldModel(unusedtokens, deserialized);
      return message;
}
```

Testovací framework zaťial umožnuje prijímať v takto serializovanej forme len object „WorldModel“ a teda implementácia parsovania správy (prvého riadka poslanej správy) je zatiaľ triviálna a správne spracuje len tento typ správy. Celkový návrh je však veľmi jednoducho rozšíriteľný.

7.5 GUI pre prenos modelu sveta z Jim do Testframework

Na základe nepresností, ktoré vznikali pri pohľade na model sveta v Jimovi, bolo potrebné vytvoriť porovnanie týchto dvoch modelov pre možné budúce vylepšenia modelu sveta v Jimovi.

7.5.1 Analýza

Pridanie daľšieho tabu pre porovnanie modelov sveta v Jimovi a v Testframeworku. Možnosť zobrazenia daného modelu sveta pre každého hráča na ihrisku a takisto možnosť sledovať rôzne objekty nachádzajúce sa na ihrisku.

7.5.2 Návrh

Na základe analýzy bol navrhnutý najnovší tretí tab do GUI. Daný tab by mal zahŕňať 2 podtaby, v ktorých bude možné sledovať loptu, prípadne iného hráča z pohľadu agenta, ktorého si užívateľ bude mocť zvoliť.

7.5.3 Implementácia

Podľa návrhu boli implementované dva podtaby Ball a Players do jedného hlavného tabu Comparing. V oboch podtaboch sú informácie rozdelené do troch skupín

- From server
- From player
- Differences

Ako už názvy napovedajú dané informácie hovoria o konkrétnych modeloch sveta a ich porovnanie.

Prvý podtab Ball (Obrázok 71) obsahuje jeden hlavný ComboBox, v ktorom je možné vybrať si z pohľadu ktorého agenta budeme sledovať dané atribúty lopty

- Axis X
- Axis Y
- Axis Z
- Speed
- Angle

Na základe daných atribútov môžeme pohodlne sledovať porovnanie modelov sveta pre danú loptu a teda jej konkrétnu polohu, uhol smerovania ako aj rýchlosť.

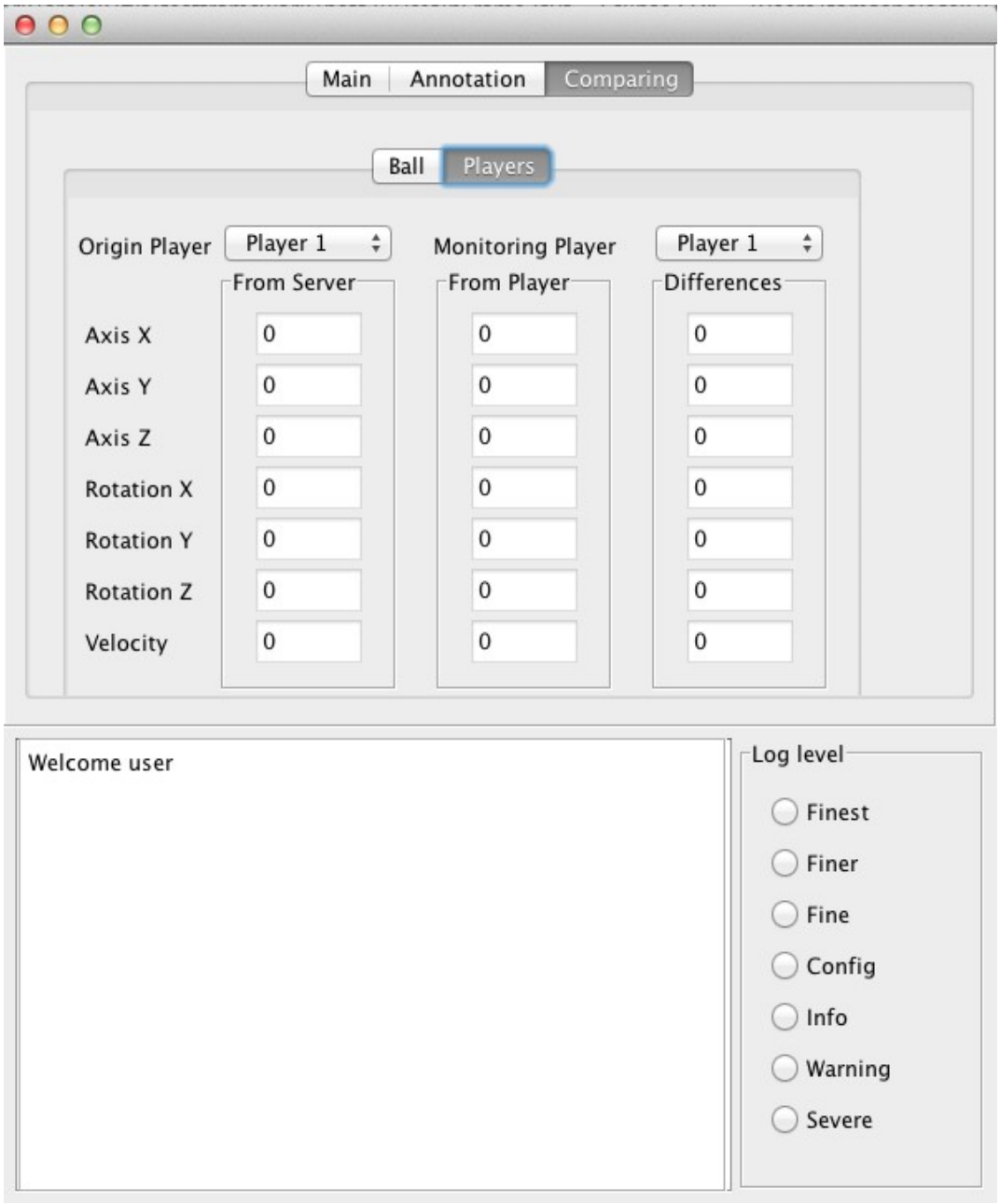
Obrázok 71: Framework GUI - Porovnávací tab pre loptu

Druhý podtab Players (Obrázok 72) obsahuje dva ComboBoxy, kde si v jednom vyberáme pozorovateľa a v druhom pozorovaný objekt. Atribúty, ktoré je možné pomocou dané tabu porovnávať sú nasledovné

- Axis X
- Axis Y
- Axis Z
- Rotation X

- Rotation Y
- Rotation Z
- Velocity

Pomocou týchto údajov vieme pomerne presne určit (predikovať) nadchádzajúce správanie sledované objektu a preto je veľmi dôležité, aby dané hodnoty čo najpresnejšie odpovedali realite.



Obrázok 72: Framework GUI - Porovnávací tab pre hráčov

8 Šprint 8

8.1 Vylepšenia pohybov hráča

Popri vývoji hráča a testovacieho frameworku dochádzalo samozrejme aj ku kontinuálnemu vylepšovaniu jeho pohybov. Jedná sa o úpravu XML súborov, v ktorých sú definované. Počas práce na projekte boli vytvorené tieto nové pohyby:

kick_left_normal

Jedná sa o kopnutie dopredu špičkou ľavej nohy. Tento pohyb sa od jeho predchodcu Kick_left líši iba minimálne, boli tam vykonané určité zmeny v rýchlosti pohybu.

kick_left_fast

Kopnutie dopredu špičkou ľavej nohy. Kopajúca noha však vykoná väščí náprah a rýchlejší švih nohou. To vyústi do dlhšieho kopu.

kick_left_slow

Kopnutie dopredu špičkou ľavej nohy. Kopajúca noha však vykoná menší náprah a pomalší švih nohou. To vyústi do kratšieho kopu.

kick_right_normal

Jedná sa o kopnutie dopredu špičkou pravej nohy. Tento pohyb sa od jeho predchodcu Kick_right líši iba minimálne, boli tam vykonané určité zmeny v rýchlosti pohybu.

kick_right_fast

Kopnutie dopredu špičkou pravej nohy. Kopajúca noha však vykoná väščí náprah a rýchlejší švih nohou. To vyústi do dlhšieho kopu.

kick_right_slow

Kopnutie dopredu špičkou pravej nohy. Kopajúca noha však vykoná menší náprah a pomalší švih nohou. To vyústi do kratšieho kopu.

kick_straight_edge_l2

Tento pohyb vychádza z pohybu kick_straight_edge_l, ktorý vykoná kopnutie dopredu vnútornou časťou ľavého chodidla hráča. Pohyb musel byť od základov prerobený, pretože predošlý pohyb potreboval mať loptu tesne pred hráčovou nohou a aj v takom prípade ju ledva trafil. Nový pohyb bol tiež spravený aby sa viac podobal na ľudský pohyb, čiže hráč pracuje s ťažiskom pomocou krčenia kolien a členkov. To vyústi v oveľa väščí náklon robota a tým pádom nie je problém trafiť sa do lopty.

kick_straight_edge_r2

Tento pohyb vychádza z pohybu kick_straight_edge_r, ktorý vykoná kopnutie dopredu vnútornou časťou pravého chodidla hráča. Pohyb musel byť od základov prerobený, pretože predošlý pohyb potreboval mať loptu tesne pred hráčovou nohou a aj v takom prípade ju ledva trafil. Nový pohyb bol tiež spravený aby sa viac podobal na ľudský pohyb, čiže hráč pracuje s ťažiskom pomocou krčenia kolien a členkov. To vyústi v oveľa väščí náklon robota a tým pádom nie je problém trafiť sa do lopty.

sit_down2

Jedná sa o posadenie sa hráča, využívané najmä pri bránení. Predošlý pohyb sit_down bol veľmi nedokonalý a hráč si prakticky nevedel sadnúť. Preto prišlo k vylepšeniu a výraznému zrýchleniu tohto pohybu. Po jeho vykonaní hráč ukážkovo sedí na zemi.

sit_down2-1

Jedná sa o posadenie sa hráča, využívané najmä pri bránení. Tento pohyb vznikol z nového pohybu sit_down2, ešte väčším zrýchlením jeho vykonávania. Teraz sa hráč dokáže posadiť veľmi rýchlo. Pohyb však kvôli rýchlosti občas skončí pádom.

walk_fine_fast2_optimized2

Chôdza walk_fine_fast2_optimized, z ktorej tento pohyb vychádza bola nestabilná. Hráč v mnohých prípadoch nedokázal prejsť väščí úsek naraz a spadol na chrbát. Taktiež skoro vôbec nevyužíval kĺby hráča a ťažil z rýchleho mihania nohami. Nový pohyb bol prerobený do ľudskejšej formy, kedy hráč do istej miery imituje chôdzu človeka. To mu dalo potrebnú stabilitu a bez problémov dokáže prejsť aj celé ihrisko porovnateľnou rýchlosťou ako predošlý pohyb.

walkback_slow

Pohyb dozadu. Vytvorený bol z pohybu walkback3, ktorý je síce výborný, ale hráč vykonáva veľmi veľké kroky dozadu. Bolo nutné vytvoriť pohyb, ktorý bude vykonávať menšie kroky, najmä kvôli pohybom hráča okolo lopty. Výborná stabilita pohybu zostala zachovaná.

8.2 Komplexný vyšší pohyb

8.2.1 Analýza

Napriek tomu, že táto úloha nebola zadaná v rámci product backlogu, bola vykonaná vzhľadom na to, že sa hodila pri riešení úlohy. Keďže začali vznikať vyššie pohyby, ktoré kombinujú nižšie pohyby, bolo vhodné vytvoriť aj také vyššie pohyby, ktoré kombinujú viacero jednoduchších vyšších pohybov – napr. lokalizuj loptu, choď za loptou a pod – je tu totiž predpoklad, že by mohlo byť vhodné, keby mali všetky takéto pohyby podobné (rovnaké) správanie. Prvý komplexný vyšší pohyb je `WalkBehindBall.rb` .

8.2.2 Návrh

Takýto komplexný pohyb by mal mať nasledovné vlastnosti:

- nezaťažuje programátora pri programovaní komplexného pohybu rozhodovaním, čo má hráč robiť, ak spadne na zem, alebo ak je hra v beamable móde (toto samozrejme nemusí byť vždy tak, predpokladá sa ale, že väčšinou sa očakáva štandardné správanie)
- odpovedá na požiadavku pickLowSkill() (pretože dedí od obyčajného vyššieho pohybu)
- nezaťažuje programátora prevádzaním odpovede vyšších pohybov na požiadavku pickLowSkill().
- poskytuje rozumné rozhranie na rozhodovanie o výbere vnorených vyšších pohybov.

Otázkou ale ostáva, či by mal takýto komplexný vyšší pohyb povoliť vykonanie vždy len jedného vyššieho pohybu a potom vrátiť rozhodovanie na plánovač (medzičasom sa mohla zmeniť taktika pre daného hráča – ak nie,tak sa aj tak zavolá ten istý komplexný pohyb – ako by sa nič nestalo), alebo má ostať vyberať vyššie pohyby, až kým sa neuspokojí s výsledkom (zatiaľ je to navrhnuté týmto spôsobom)

8.2.3 Implementácia

Trieda je implementovaná tak, že ak z nej dedí iná trieda, musí implementovať jedine metódu pickHighSkill() - v tej treba vrátiť HighSkill, ktorý sa bude vykonávať. Ukončenie komplexného vyššieho pohybu nastane, ak:

- metóda pickHighSkill vráti null,
- vybraný HighSkill vráti hneď prvý LowSkill null,
- hráč spadne,
- nastane beamable mód.

9 Inštalačná príručka

9.1 Inštalácia a konfigurácia vývojového prostredia

Všetok vývoj hráča Jim ako aj testovacieho frameworku bol uskutočnovaný na platforme Java, Ruby v prostredí Eclipse. Zdrojové kódy sú uložené v centrálnom repozitári spravovaného nástrojom Subversion, s ktorým vie Eclipse komunikovať pomocou pluginu Subclipse. Pre správne spustenie hráča Jim je ďalej nutné inštalovať pluginu AspectJ pre správne pracovanie aspektov v nástroji Eclipse. Celkovo boli teda použité nasledovné nástroje a pluginy v daných verziách:

- Eclipse (verzia Indigo 3.7.1, Java verzia 1.6)
- Subclipse (verzia 1.6)
- AspectJ verzia 1.6.12)
- Eclipse AspectJ Development Tools (verzia 2.1.3)

9.1.1 Eclipse

Pre správnu inštaláciu nástroja Eclipse je potrebné vykonať nasledovné kroky (vývoj sa uskutočnoval pomocou verzie Indigo 3.7.1, Java verzia 1.6):

1. Stiahnuť najnovšiu verzi nástroja Eclipse zo stránky http://www.eclipse.org/downloads/
2. Stiahnutý balík rozbaliť do na zvolené miesto obľúbeným nástrojom na komprimáciu súborov.
3. Eclipse je platformovo nezávislé prostredie a nepotrebuje žiadnu dodatočnú inštaláciu. jeho spustenie predstavuje už len spustenie súboru „eclipse“.

9.1.2 Subclipse

Pre správnu prácu s repozitárom subversion (SVN) bol používaný Eclipse plugin Subclipse vo verzii 1.6. Pre jeho inštaláciu je potrebné vykonať následovné kroky:

1. Spustiť nástroj Eclipse
2. Voľbami Help->Install new software otvoriť obrazovku inštalácie pluginov
3. Do adresy zadať URL http://subclipse.tigris.org/update_1.6.x a stlačit klávesu enter
4. V možnostiach balíkov stačí vybrať „subclipse“ a potvrdiť voľbu tlačítkom „next“
5. Potvrdiť licenčné podmienky a pokračovať s inštaláciou.
6. Reštartovať nástroj Eclipse.

9.1.3 AspectJ a Eclipse AspectJ Development Tools

Hráč Jim potrebuje na správne fungovanie niektorých jeho častí knižnicu AspectJ. Pre jej správnu inštaláciu je potrebné vykonať následovné:

1. Spustiť nástroj Eclipse
2. Voľbami Help->Install new software otvoriť obrazovku inštalácie pluginov
3. Do adresy zadať URL http://download.eclipse.org/tools/ajdt/36/update a stlačit klávesu enter
4. V možnostiach je potrebné zvoliť možnosti „AspectJ Develipment Tools (Requires)“ a „AJDT Development Tools“.
5. Potvrdiť voľbu tlačítkom „next“,
6. Potvrdiť licenčné podmienky a pokračovať s inštaláciou.

7. Reštartovať nástroj Eclipse.

9.1.4 Nastavenie repozitáru Subversion (SVN)

Pri vývoji bol používaný repozitár subversion na školskom virtuálnom serveri poskytovaným v rámci predmetu tímový projekt. Obsahuje najnovšiu verziu projektov. Pre jeho konfiguráciu v nástroji Eclipse je potrebné vykonať následovné kroky:

1. Inštalácia pluginu Subclipse v predošlej časti
2. Prepnúť sa do perspektívy „SVN Repository Exploring“ prostredníctvom Window->Open perspective->Other a zvoliť „SVN Repository Exploring“
3. Pridať repozitár pravým klikom na zoznam repozitárov v záložke „SVN Repositories“ (teraz prázdny) a zvolením New->Repository Location
4. Zdanie adresy repozitára, v našom prípade http://vm01.ucebne.fiit.stuba.sk/code/robocup a potvrdiť adresu stlačením „Finish“
5. Zadanie hesla pre prístup do repozitára ak k tomu Eclipse vyzve.
6. Repozitár je nakonfigurovaný.

9.1.5 Jim

Pre vybratie projektu Jim do prostredia Eclipse pomocou nástroja Subclipse je potrebné vykonať nasledovné operácie:

1. Mať nainštalovaný nástroj Subclipse a nakonfigurovaný repozitár.
2. Prepnúť sa do perspektívy „SVN Repository Exploring“ prostredníctvom Window->Open perspective->Other a zvoliť „SVN Repository Exploring“
3. Rozbalenie daného repozitára a vybratie časti, ktorá sa má vybrať ako projekt. V tomto prípade je to zložka „Jim/trunk“.
4. Kliknutie na danú zložku a vybrať voľbu „Checkout“
5. Nastaviť lokálne meno projektu v prostredí Eclipse a potvrdiť voľbu stlačením „Finish“
6. Prebehne kopírovanie najnovšej verzie hráča Jim na lokálny počítač. Táto akcia môže chvílu trvať
7. Projekt v nástroji je vytvorený a pripravený na použitie.

9.1.6 TestFramework

Pre vybratie projektu TestFramework do prostredia Eclipse pomocou nástroja Subclipse je potrebné vykonať nasledovné operácie:

1. Mať nainštalovaný nástroj Subclipse a nakonfigurovaný repozitár.
2. Prepnúť sa do perspektívy „SVN Repository Exploring“ prostredníctvom Window->Open perspective->Other a zvoliť „SVN Repository Exploring“
3. Rozbalenie daného repozitára a vybratie časti, ktorá sa má vybrať ako projekt. V tomto prípade je to zložka „TestFramwork/trunk“.
4. Kliknutie na danú zložku a vybrať voľbu „Checkout“
5. Nastaviť lokálne meno projektu v prostredí Eclipse a potvrdiť voľbu stlačením „Finish“
6. Prebehne kopírovanie najnovšej verzie hráča TestFramework na lokálny počítač. Táto akcia môže chvílu trvať
7. Projekt v nástroji je vytvorený a pripravený na použitie.

9.2 Inštalácia Robocup Servera

Pre vývoj a testovanie hráča Jim ako aj testovacieho frameworku bol používaný najnovší dostupný robocup server (verzia 0.6.5, subversion revízia 286). Server je dostupný na všetkých platformách. Jeho inštalácia je veľmi odlišná na každej platforme. Opis krokov pre inštaláciu na každej platforme je zrhnutý na oficiálnej stránke simspark servera:

(http://simspark.sourceforge.net/wiki/index.php/Main_Page).

9.2.1 Inštalácia na operačnom systéme Microsoft Windows

Inštalácia je umožnená vykonaním následovných krokov:

1. Stiahnutie a inštalovanie najnovšej verzie „MS Visual C++2008 Redistributable Package (x86)“ (http://www.microsoft.com/downloads/details.aspx?FamilyID=9b2da534-3e03-4391-8a4d-074b9f2bc1bf&displaylang=en)
2. Stiahnutie a inštalovanie servera simspark (http://sourceforge.net/projects/simspark/files/)
3. Stiahnutie a inštalácia robocup servera vo verzii 0.6.5 (http://sourceforge.net/projects/simspark/files/)
4. Pri štandardnej inštalácii je možné robocup server ako aj monitor spustiť pomocou príkazov:
   - C:\Program Files\rcssserver3d 0.6.3\bin\simspark.cmd
   - C:\Program Files\rcssserver3d 0.6.3\bin\rcssmonitor3d.cmd

9.2.2 Inštalácia na operačnom systéme Ubuntu

Pre úspešnú inštaláciu robocup servera je potrebné mať povolené balíky repozitárov „Universe“ a „Multiverse“. Následne je možné pridať repozitár obsahujúci server a vykonať jeho inštaláciu príkazmi:

1. sudo apt-add-repository ppa:gnurubuntu/rubuntu
2. sudo apt-get update
3. sudo apt-get install rcssserver3d

10 Vývojárska príručka pre testovací framework

10.1 Nastavenia a beh testovacieho frameworku

Všetky nastavenia testovacie frameworku sú vykonávané štandardným spôsobom s využitím „java.properties“. Aplikácia je dodávaná s isými prednastavenými hodnotami definovanými v súbore „default.properties“ umiestneného v balíku „sk.fiit.testframework.init“. Vlastné nastavenia pre tieto hodnoty je možné určit v súbore „configuration.properties“ umiestnenom v terajšej pracovnej zložke (zložke odkial je aplikácia spúštaná).

Medzi možné nastaviteľné hodnoty patria:

- robocup.server.command – príkaz na spustenie robocup servera
- robocup.server.killcommand – príkaz na vypnutie robocup servera
- robocup.server.ip - IP adresa robocup servera
- robocup.server.port.monitor – monitor port robocup servera pr
- robocup.server.port.player – port na pripojenie hráča k robocup servera
- robocup.player.dir – zložka, v ktorej sa nachádza hráč
- robocup.player.command – príkaz na spustenie hráča
- testframework.monitorAgent.ip – IP adresa, na ktorej počúva testframework na spatnú väzbu
- testframework.monitorAgent.port - port na ktorom počúva testframework na spatnú väzbu
- userInterface – classpath triedy, ktorá je zodpovedná za userintefrace (musí implementovať rozhranie „UserInterface“)
- implementation – classpath triedy, ktorá riadi celú aplikáciu (musí iplementovať rozhranie „Implementation“)

Konfiguračný súbor je prečítaný pri spustení aplikácie. Využitie takto definovaných premenných je možné pomocoi triedy „C“ (sk.fiit.testframework.init.C) použítím metódy „getProperty(key)“.

Posledné dve premenné (userInterface a Implementation) sú požité na vytvorenie inštancií daných tried a riadia beh celej aplikácie. Zapínanie, vypínanie a vykonávanie testov má na starosti trieda „Implementation“, ktorú je možné na základe konfigurácie vymeni't.

10.2 Spatná väzba od hráča

Testovací framework umožnuje prijímať správy od hráča, čo mu umožnuje lepšie vyhodnocovať jeho správanie. Na posielanie správ je vytvorené štandardné TCP spojenie. Typická správa nepresahuje dĺžku jedného riadka a obsahuje číslo hráča, meno tímu, typ správy a posielané hodnoty.

Príklad takejto správy je informácia o začatom pohybe, ktorá má nasledovnú formu:

```
(1 ANDROIDS LEFT) highskill start rollback 0.0
```

Výnimkou pri posielani správ je odosielanie aktúalneho stavu sveta ako ho vidí hráč (prenos celého objektu WorldModel). Stav sveta je na strane hráča deserializovaný do pola bajtov a následne zakódovaný do textového reťazca pomocou Base64. Okrem štandardných informácií ako pri predošlých správach je odosielaný checksum dát v novom riadku a samotné dáta v ďalších riadkov. Celá jedna správa je ukončená textovým reťazcom „endData“.

Príklad správy je následovný:

```
(1 ANDROIDS LEFT) worldmodel beginData
6488e0907bc62ad6429963910a7c6c975d44275b
```

```
rO0ABXNyACNzay5maWl0LmppbS5hZ2VudC5tb2RlbHMuV29ybGRNb2RlbM8irrx9aZAgAETAAK
8FbJQDWQwJTKvbpzcQB+AAg/QAAAAAA...ADHcIAAAAEAAAAAB4c3EAfgAIP0AAAAAAAAx3CAAA
ABAAAAAAeA==
endData
```

Správy sú následne dekódované na strane testovacieho frameworku. Správy sú dekodóvané dvoma rôznymi metódami, a to podľa toho či obsahuje dáta alebo nie. V obidvoch prípadoch však funkcie musia parsovať prvý riadok správy, a zistiť čo sa prijíma. Dve funkcie sú následovné:

```
parse(String input)
parse(String firstLine, Object deserialized)
```

V druhom prípade, keďže je ojedinelí sa okamžite vytvorí správa typu WorldModel. V prvom prípade sa postupne prechádza slovami prijatej správy, pokiaľ sa text slova nerovná názvu niektorej z tried definovaných v triede „AgentMonitorMessage“. Zvyšky prijatej správy sú poslané do konštruktora tejto triedy v podobe pola textových reťazcov a trieda má spracovať (odobrať) z pola prvky ktoré jej patria. Nespracované časti správy (elementy pola prijatých textových reťazcov) sú neskôr spracované hlavnou triedou „AgentMonitorMessage“, ktorá v tomto poly bude očakávať už len číslo hráča a meno tímu.

Príkladom je spracovanie správy o začatom pohybe:

```
(1 ANDROIDS LEFT) highskill start rollback 0.0“
```

Na spracovanie tejto triedy je potrebné definovať následnú novú triedu v triede „AgentMonitorMessage“:

```
public static class HighSkill extends AgentMonitorMessage {
            // rozne pomocne premenne

            public static class Start extends HighSkill {
                  public Start(Stack<String> tokens) {
                        player_time = Double.parseDouble(tokens.pop());
                        move_name = tokens.pop();
                        type_flags |= TYPE_HIGHSKILL;
                        super.init(tokens);
                  }
            }
```

Definovaná bola trieda „AgentMonitorMessage.HighSkill.Start“. Pri parsovaní sa úspešne identifikovala časť správy „highskill start“ na túto triedy. Do konštruktora tejto triedy bolo teda vložené následové pole textových reťazcov: {0.0, rollback, Left, Androids, 1}. Prvé dva argumenty musí táto trieda spracovať a o ostatné sa už postará trieda od ktorej dedí (pomocou spustenia metódy „super.init(tokens)“.

Po úspešnom zpracovaní správy je rozoslaná všetkým zaregistrovaným listenerom pre daný typ správy (typ správy je určenú pomocou premennej „type_flags“). Typy správ sú definované ako konštanty v triede „AgentMonitorMessage“. **Vykonávajú sa nad nimi logické operácie AND a OR na zisťovanie či daná správa vyhovuje registrovanému listeneru. Možné je teda definovať listener na viacero typov správ naraz.** Pre zaregistrovanie a odregistrovanie listenera je umožnené vykonať zavolaním následovných metód:

```
AgentMonitor.setMessageListener(uniform,team,listener,message_type_flags)
AgentMonitor.removeMessageListener(IAgentMonitorListener listener)
```

10.3 Nastavenie automatického spúštania hráča a servera

Pri spúštaní nového agenta alebo robocup servera testovací framework vykonáva príkaz určený konfiguračnou premennou robocup.player.command a robocup.server.command. Túto premennú je

možné určiť v konfiguračnom súbore ako aj vstupných argumentoch hlavnej metódy spustenia testovacieho frameworku.
Pri vypínaní procesu umožňuje platforma Java a trieda *java.util.Process* len posielanie signálu SIGTERM, pričom signál je doručený len procesu priamo spusteného triedou *Process* (podprocesy signál nedostanú). Je teda potrebné aby spúšťaný proces správne reagoval na tento signál a prípadne aj ukončil svoje podprocesy.

V prípade robocup servera je spúšťanie problematické, keďže nereaguje na signál SIGTERM. Riešenie predstavuje vytvorenie pomocného programu na spustenie servera, pričom tento pomocný program správne spracováva daný signál a ukončí beh spusteného robocup servera. Implementácia takéhoto pomocného programu je závislá od použitého operačného systému. Príklad takéhoto programu pre operačný systém Linux v skriptovacom jazyku Bash je následovný.

```
#!/bin/bash
rcssserver3d $@ &
trap "kill -sigint $!;exit" TERM SIGTERM
wait
```

RobocupServer je automaticky spúštaný pri štarte aplikácie (ak je nastavený spúštací príkaz). Jednotlivých hráčov je možné spúštať dynamicky, napríklad pri inicializácii testov následovn (posledná premenná určuje či je volanie blokujúce – teda je prakticky na kotrolu či je hráč pripojený)

```
agent = AgentManager.getManager().getAgent(5, "ANDROIDS", true);
```

10.4 Vytvorenie testu

Pre vytvorenie nového testu na skúšanie hráča je vytvoriť triedu ktorá dedí od „sk.fiit.testframerotk.trainer.testsuite.TestCase“. Poskytuje následovné metódy ktoré je možné „overridovať“:

- `init()`
- `isStopCriterionMet(SimulationState)`
- `evaluate(SimulationState)`
- `destroy()`

Metóda init je spúštaná tesne pred začať vykonávania testu. Vhodné je teda vykonať akúkoľvek inicializáciu (pripojiť hráčov, nastaviť ich pozíciu, pozíciu lopty a iné). Metóda má návratovú hodnotu *boolean.* Pri vrátenej hodnote „false“ je inicializácia a teda aj celý test požadovaný za neúspešný a test končí.

Metóda „isSopCriterionMet“ je volaná v častým intervaloch. Jej návratovou hodnotou je *boolean* jej hlavnou úlohou je zisťit či má test skončiť. Pri hodnote „true“ test končí.

Po ukončení behu testu je spustená metóda „evaluate“. Jej úlohou ve určit úspešnosť testu. Návratová hodnota testu je objekt „TextCaseResult“ do ktorého je možné uložit výsledok testovania. Tento výsledok dostanú všetky listenery na dianie testov.

Posledná metóda je „destroy“ v ktorej je možné dealokovať všetky nepotrebné objekty (aj hráčov).

Okrem týchto metód trieda „TestCase“ poskytuje aj rôzne premenné. Poskytujú prístup k bežným objektom a premenným testovacieho frameworku.

- RobocupServerAddress address
- RobocupServer server
- RobocupMonitor monitor
- SimulationState simulationState
- AgentMonitor agentServer

Spusťit testCase je následovne možné pridať do rady na vykonanie pomocou inštancie aktúalnej „Implementácie“ aplikácie. Tent krok je ale väčšinou vykonávaný inou častou testovacieho frameworku (GUI, sama implementácia pomocou argumentov na spustenie, …). Pri spúštaní testu je možné zaregistrovať objekt ako listener na prijatie výsledku daného testu.

```
Implementation impl = ImplementationFactory.getImplementationInstance();
impl.enqueueTestCase(testcase, listener);
```

10.5 Spustenie testu

Každý test dedí od triedy „TestCase“. Ich spúštanie je umožnené pomocou konkrétnej Implementácie rozhrania „Implementation“, ktorá riadi celý beh aplikácie. Pridanie testu na vykonanie v danej implementácii je umožnené pomocou metódy:

```
enqueueTestCase(TestCase testCase, ITestCaseObserver observer)
```

Druhý argument tejto metódy je listener (môže byť null), ktorý je následne upozornení na vúsledok testu (TestCaseResult objekt ktorý daná konkrétna TestCase implementácia vráti). Spúštanie viacerých testov naraz a následne porovnávanie výsledok je umožnené napríklad nádledovným Runnable objektom:

```
public class StandUp implements Runnable, ITestCaseObserver {
	// … inicializacia objektu
	public void run() {
		Implementation impl =
			ImplementationFactory.getImplementationInstance();

		impl.enqueueTestCase(new StandUpTest(), this);
		impl.enqueueTestCase(new StandUpTest(), this);
		impl.enqueueTestCase(new StandUpTest(), this);
		impl.enqueueTestCase(new StandUpTest(), this);
		impl.enqueueTestCase(new StandUpTest(), this);
	}

	// notifikacia o ukonceny testu – teda ulozenie vysledku
	public void testFinished(TestCaseResult result) {
		testResults.add(result);
		if (testResults.size() == 5) {
			evaluateFastTestResults(testResults);
			testResults.clear();
		}
	}

	// vyhodnotenie celkoveho vysledku
	public void evaluateFastTestResults(List<TestCaseResult> results) {
		double result = 0;
		for (TestCaseResult testCaseResult : results) {
			result += testCaseResult.getFitness();
		}
	}

}
```

V opisovanom príklade je spustení 5-krát test „StandUpTest“. Pri ukončení jedného z testov je vždy spustená metóda „testFinished“, ktorá výsledok testu uloží. Pri ukončení všetkých testov (výsledkov je 5) sa celkové časy spočítaju a je dostupný finálny výsledok testovania.

11 Používateľská príručka pre testovací framework

Vysvetlenie ako správne nastaviť, aby nám fungoval framework je vysvetlené vo vývojárskej príručke a v danej časti sa budeme už len zaoberať konkrétnym používaním jednotlivých častí frameworku. Rozdelíme to do troch hlavných častí podľa funkcionality

11.1 Main Tab

Hlavný tab určený na spustenie a ovládanie základných funkcionalít testovacieho frameworku. Daný tab je potrebné rozdeliť do troch hlavných častí

- Settings
- Controller
- Jim info

V settings sa nám zobrazuje informácia o IP adrese, na ktorú sme pripojený a ktorá ma iba informatívny charakter.

Hlavné ovládacie prvky sa nachádzajú v controlleri. Pomocou tlačidla connect pripojíme daný testovací framework k serveru a po stlačení tlačidla štart je nám umožnené sledovať model sveta pozorovaný testovacím frameworkom v časti Jim info. Pomocou tlačidla stop samozrejme daný beh zastavíme.

Ďalšími ovládacími prvkami je možné pridávanie hráča s konkrétnym číslom pomocou Add agent, prípadne nastavenie pozície lopty pomocou Set ball.

Jednou z najužitočnejších častí hlavne v závere tímoveho projektu je možnosť spúšťania konkrétnych úťažných testov priamo z testovacieho frameworku vďaka ComboBoxu Test Selection.

11.2 Annotation tab

Annotation tab slúži na výtváranie anotácií pre konkrétne pohyby. Ovládanie danej časti ej veľmi jednoduché príčom je potrebné do GUI zadať iba pohyb, o ktorého anotovanie máme záujem, počet opakovaní daného pohybu a následne iba inicializačnú polohu lopty a rádius v ktorom bude počas testov rozostavovaná. Po stlačení tlačidla Annotate, je vo vopred vybranom priečinku vytvorený anotačný súbor, ktoré atribúty sú bližšie popísané v časti o anotáciach.

11.3 Comparing tab

Comparing tab slúži na porovnávanie modelov sveta z Jima a z Testframeworku.

Daný tab je rozdelený na dva podtaby podľa druhu sledovaného objektu.

V prvom tabe Ball je možné sledovať modely sveta a ich rozdiely z pohľadu hociktorého hráča na ihrisku.

V druhom tabe je umožnené používateľom sledovať hráčov z pohľadu iného hráča, pričom je potrebné vybrať iba daných dvoch hráčov pomocou ComboBoxov a o všetko ostatné sa už postará testovací framework.

11.4 Log Level

Počas celej práce s testovacím frameworkom v každom tabe je možné sledovať priebeh logovania a je nám umožnené aj výber množstva výpisov od Finest po Sever, podľa požiadaviek používateľa.