**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE**

**Fakulta informatiky a informačných technológií**

Ilkovičova 2, 842 16 Bratislava 4

**Tímový projekt**

**RoboCup**

**Dokumentácia k inžinierskemu dielu**

**Bc. Filip Blanárik**

**Bc. Michal Blanárik**

**Bc. Štefan Horváth**

**Bc. Štefan Linner**

**Bc. Martin Markech**

**Bc. Roman Moravčík**

**Bc. Tomáš Nemeček**

Tím č. 9: Gitmen

Vedúci projektu: Ing. Ivan Kapustík

Predmet: Tímový projekt I

Ročník: 1

Akademický rok: 2013/2014, zimný semester

Mailový kontakt:gitmen09@gmail.com

**Obsah**

1 Úvod 1

1.1 Ciele projektu 1

2 Prvý šprint 2

2.1 Rozbehanie simulačného prostredia robotického futbalu 2

Windows 2

Popis a riešenie problémov Windows 3

Linux 4

Popis a riešenie problémov Linux 6

Popis a riešenie problem TestFramework 7

2.2 Opis východiskového stavu projektu 7

Projekt Jim 7

Projekt RoboCupLibrary 14

Projekt TestFramework 15

2.3 Vytvorenie unit testu 17

isInHalfPlane() 17

calculateDistance() 17

whereIsTarget() 17

bIsBallNearestToMe() 18

substract() a addiotion() 18

isForward() 18

isOnPosition() 18

2.4 Zmena správania hráča 19

Modifikovaný plan „p*lanTactic*“ 1. 19

Modifikovanie plánu „planTactic“ 2. 20

Vytvorenie nového pohybu a použitie „planZakladny“. 20

Vytvorenie nového pohybu a modifikovanie „planZakladny” 20

Vytvorenie nového highskillu, lowskillu a modifikovanie „planZakladny“ 20

Vytvorenie nového plánu 21

Vytvorenie nového plánu 21

# Úvod

Hlavným cieľom projektu je vylepšenie súčasného hráča 3D robotického futbalu a ostatných častí riešenia ako sú testovací framework, ktorý slúži ako pomôcka pri vývoji hráča, editor pohybov, pomocou ktorého je možné vytvoriť zložené pohybové sekvencie cez grafické používateľské rozhranie. Súčasťou projektu je aj udržiavanie aktuálnej dokumentácie hráča a ostatných súčastí riešenia. Dokumentácia je vo forme wiki stránky. Projekt 3D robotického futbalu je na škole vyvíjaný už niekoľko rokov v rámci predmetu Tímový projekt, Bakalársky a Diplomový projekt. Vďaka tomu je aktuálna verzia hráča pomerne rozsiahlo implementovaná a schopná rôznych pohybov (kopnutie, vstávanie zo zeme, ...) pričom je k dispozícií niekoľko plánovačov, ktoré určujú čo daný agent najbližšom kroku vykoná.

## Ciele projektu

Medzi ciele projektu patrí aj:

* odstránenie duplicitných implementácií rovnakej funkcionality. Z dôvodu oddelenej práce na projekte v rámci rôznych predmetov a počas niekoľkých rokov vznikala architektúra riešenia nekoordinovane a tým pádom sú niektoré časti duplicitne implementované, čo sťažuje spravovanie kódu a spomaľuje vývoj.
* zjednodušiť a následne odstrániť časti riešenia, ktoré sú implementované v jazyku ruby, pretože nie je efektívne spravovať program, ktorý je implementovaný vo viacerých programovacích jazykoch. Ďalším dôvodom je, že máloktorý študent, ktorý pracuje na projekte, má predchádzajúce skúsenosti s týmto programovacím jazykom, pretože jeho výučba nie je realizovaná na fakulte.
* Vytvorenie architektúry, ktorá umožní plánovanie na niekoľkých úrovniach rozhodovania (strategická, taktická, high skill a low skill úroveň) a týmto spôsobom zabezpečiť požadované správanie hráča tak, aby bol schopný posúdiť stav hry a podľa toho sa rozhodovať.
* vylepšenie pohybov hráča.
* integrácia riešení, ktoré boli súčasťou diplomových prác a majú potenciál prispieť k zlepšeniu celkovej kvality hráča.

# Prvý šprint

## Rozbehanie simulačného prostredia robotického futbalu

V rámci úlohy rozbehanie simulačného prostredia bol vytvorený návod, podľa ktorého sa postupuje pri prvej inštalácií simulačného prostredia a hráča. Návod je rozdelený podľa typu operačného systému Windows / Linux a podľa integrovaného vývojového prostredia, ktoré sa použije pri spustení agenta a testovacieho framweorku.

Zmeny v inštalačnej príručke sa premietli aj v dokumentácií na wiki v sekcií Návody a inštalácie.

### Windows

1. **Inštalácia simulačného prostredia**

Na inštaláciu simulačného prostredia sú potrebné nasledujúce programy:

1. MS Visual C++ 2008 Redistributable
2. Simspark
3. Rcsserver3d
4. Ruby

V tabuľke 1 sú uvedené verzie, ktoré boli otestované a zistilo sa, že pre ne projekt funguje správne.

|  |  |
| --- | --- |
| **Program** | **Verzia** |
| Operačný systém | 7 64bit / Vista SP2 64bit / 8 64bit / 8.1 Pro N |
| IDE | NetBeans IDE 7.3.1 / Eclipse Kepler 20130919-0819 |
| Ruby | 1.9.3-p448 / 2.0.0-p247 64bit |
| Simspark | 0.2.4 |
| Rcsserver3d | 0.6.7 |
| Java JDK | 1.7.0\_40 / 1.6.0\_18 |

Tabuľka 1 prehľad použitých verzií programov pre Windows

Postup inštalácie

1. Nainštalujte MS Visual C++ 2008 Redistributable.
2. Nainštalujte som Simspark
3. Nainštalujte som rcsserver3d
4. Nainštalujte Ruby
5. V adresári, kde sa nainštaloval rcsserver3d v adresári adresára *bin* otvorte v editore 3 súbory - *rcssserver3d.cmd*, *rcssmonitor3d.cmd*, *rcssagent3d.cmd*. Do týchto súborov dopíšte premenné, ktorým priraďte adresy simspark-u a rcsserver3d. Napríklad ak ste nainštalovali do Program Files (x86), ta to vyzerá takto:

SET SPARK\_DIR=C:\Program Files (x86)\simspark

SET RCSSSERVER3D\_DIR=C:\Program Files (x86)\rcssserver3d 0.6.7

* + tento krok je možné na niektorých systémoch vynechať

1. Uložte úpravy v súboroch. Otestujte, či všetko správne funguje, spustením príkazov v nasledujúcom poradí: *rcssserver3d.cmd*, *rcssmonitor3d.cmd*, *rcssagent3d.cmd*. Malo by sa na obrazovke objaviť okno futbalového ihriska s jedným hráčom. Týmto krokom ste úspešne rozbehali simulačné prostredie.

Odkazy na stiahnutie programov:

* Simspark 0.2.4 <http://sourceforge.net/projects/simspark/files/>
* rcsserver3d 0.6.7 <http://sourceforge.net/projects/simspark/files/rcssserver3d/>
* Ruby 2.0.0 (x64) <http://rubyinstaller.org/downloads/>

1. **Inštalácia Eclipse IDE a získanie aktuálnej revízie zdrojového kódu hráča**

Na použitie tohto návodu je potrebné mať už dopredu vytvorený GIT repozitár s projektom agenta 3D robotického futbalu.

1. Nainštalujte prostredie Eclipse z <http://www.eclipse.org>
2. Spuste prostredie Eclipse a nastavte pracovný adresár.
3. Importujte existujúci projekt z git repository, ktorý máte rozbehaný napríklad na bitbucket.org. V Eclipse otvoríte cez File -> Import -> Git -> Project from Git -> Clone URI okno, kde vyplníte nasledujúce informácie.
   1. Do sekcie *location* vyplňte adresu repozitára napríklad <https://login@bitbucket.org/robocup_tp09/agent.git>, kde login je váš osobný login. V časti *authentification* vyplňte váš login a heslo a potvrďte *next*.
   2. Tu nastavíte adresár, kde sa má projekt naklonovať.
   3. Teraz sa načítali projekty nachádzajúce sa v repozitári, zvoľte požadované projekty a daljte *finish*.
4. Na otestovanie hráča spustite main.java z package sk.fiit.jim.init v projekte Jim. Kompilácia by mala prebehnúť v poriadku a hráč sa objaví na virtuálnom ihrisku.
5. Na otestovanie zápasu viacerých hráčov spustite Init.java z package sk.fiit.testframework.init v projekte TestFramework. Po kompilácii a spustení sa objaví okno, kde je možnosť pridávať si hráčov pre jeden aj druhý tím. Po spustení simulácie by sa hráči mali pohybovať po virtuálnom ihrisku.

### Popis a riešenie problémov Windows

**Rcssserver3d**

* Pri spustení *rssserver3d.cmd* sa zobrazí chybová hláška o chýbajúcej knižnici (napríklad *libboost\_system-mt.dll is missing*)
  + *Riešenie:* S najväčšou pravdepodobnosťou sú nesprávne nastavené systémové premenné SPARK\_DIR alebo RCSSSERVER3D\_DIR v danom .cmd súbore (spomínaná knižnica sa nachádza v adresári simspark). Skontrolujte, či cesty v tomto súbore sú platné a zodpovedajú hore uvedenému návodu. V prípade potreby je možné tieto premenné definovať aj manuálne v Štart->Ovládací panel->Používateľské kontá->Premenné prostredia.
* Problém so spustením *rssserver3d.cmd*so súčasne nainštalovanými knižnicami *MinGW* (napríklad chyba v knižnici *libgcc\_s\_sjlj-1.dll*).
  + Riešenie: Je potrebné odstrániť z premennej prostredia PATH cestu ku knižniciam MinGW (Štart->Ovládací panel->Používateľské kontá->Premenné prostredia).

**NetBeans IDE**

* Problém build-nut projekty, pretože niektoré súbory boli kódovane ako ANSI, no obsahovali znaky s diakritikou, ktoré NetBeans nevedel prečítať, keďže používa UTF8 ako predvolene kódovanie, ktoré sa nedá zmeniť
  + *Riešenie:* Prepísanie znakov s diakritikou na znaky bez diakritiky, alebo konverzia kódovania z ANSI na UTF8 (bez BOM)

**Eclipse IDE**

* Pri importovani projektu do vývojového prostredia Eclipse sa môže vyskytnúť problém, kedy Eclipse Keppler nevie správne rozoznať aspekt v projekte Jim. Konkrétne *sk.fiit.stuba. ExecutionTimeMonitor.aj.* Riešením tohto problému je pridanie AspectJ nature do projektu.

### Linux

V tabuľke 2 sú uvedené verzie, ktoré boli otestované a zistilo sa, že pre ne projekt funguje správne.

|  |  |
| --- | --- |
| **Program** | **Verzia** |
| Operačný systém | Kubuntu 13.04 64bit / Ubuntu 12.04 LTS 32bit |
| IDE | NetBeans IDE 7.3.1 / Eclipse Kepler 20130919-0819 |
| Ruby | 1.9.3-p327 32bit |
| Simspark | 0.2.4 / 0.2.3-0precise2 |
| Rcsserver3d | 0.6.7 / 0.6.6 |
| Java JDK | 1.7.0\_25 / 1.7.0\_09 |

Tabuľka 2 prehľad použitých verzií programov pre Linux

1. **Inštalácia**

**-** Ubuntu 12.04 LTS 32bit

Pri inštalácii prostredia vychádzajte z návodu na wiki v sekcii „Návody a inštalácie“, Pre verziu Ubuntu 12.04 LTS 32bit, postupujte podľa návodu „LINUX rýchla inštalácia (bez kompilácie)“. V repozitári ppa:gnurubuntu/rubuntu je pre verziu 12.04 iba verzia server 0.6.6, preto nainštalujte túto verziu. Verziu 0.6.5, na ktorej bol vyvíjaný a testovaný projekt sa v binárnej podobe v repozitári nenachádza. Verzia servera 0.6.7 je v binárkach dostupná až pre verziu Ubuntu 13.04. Ruby môžete nainštalovať pomocou RVM ( [https://rvm.io](https://rvm.io/)), vďaka čomu môžete v systéme používať rôzne verzie Ruby pre rôzne projekty a v prípade potreby prepnúť na novšiu verziu Ruby. Verzia Ruby 2.0.0 je spätne kompatibilná s Ruby 1.9.3 vetvou a projekt by mal fungovať na 2.0.0-p247 bez problémov.

* Kubuntu 13.04 64bit

Balíky pre simspark sa v repozitári tejto verzie distribúcie už nenachádzajú, je potrebné stiahnuť Simspark a Rcsserver3d, manuálne skompilovať a nainštalovať podľa návodu na wiki. Kompilačné závislosti si vyžadujú stiahnutie balíkov z repozitára a to libode-dev, libsdl1.2-dev, libdevil-dev, libboost-dev, libboost-thread-dev, libboost-regex-dev, zeitgeist, zeitgeist-core, ruby1.9.1-full, ruby1.9.1-dev, libboost-all-dev, cmake, libsdl-dev, latex, latex-make, openjdk-7-jdk openjdk-7-jre, latex2html, doxygen, imagemagick. Kompilácia po splnení závislostí prebiehala už na wiki spomínaným make a ďalšími príkazmi.

**2.1 Inštalácia vývojového NetBeans**

Nainštalujte NetBeans, ktorý je dostupný na oficiálnej stránke projektu. Zdrojové kódy stiahnite do pracovného adresára cez konzolu príkazom git clone git@bitbucket.org:robocup\_tp09/agent.git . V NetBeans následne vytvorte projekt z existujúcich zdrojových kódov projektov. Následne nastavte projekt tj. working directory a main class pre spustenie pridajte požadované knižnice. Tieto informácie sú uvedené aj na wiki.

**2.2 Spustenie projektu Eclipse**

Pri spúšťaní postupujte podľa návodu na wiki, najskôr spustite server z terminálu príkazom rcssserver3d, následne monitor v novom okne terminálu príkazom rcssmonitor3d a zozačiatku vyskúšajte v novom okne spustiť agenta príkazom rcssagent3d. Robot sa na obrazovke zobrazí, nezačne však hrať hru.

Následne som skúste spustiť agenta z Eclipse projektu. Natiahnite som si projekt z git repozitára. V Eclipse kliknite na File -> Import -> Git -> Project from git -> Clone URI a klikol na next. V časti locations zadajte ako adresu napríklad „git@bitbucket.org:robocup\_tp09/agent.git“ , nastavte protokol na ssh a port na 22. Predpoklad pre fungovanie načítania z tejto adresy je nahratie ssh kľúča do Bitbucketu. Na ďalšej obrazovke som ponechajte vetvu master a kliknite na next. Vyberte priečinok, kde sa uloží git a pokračujte kliknutím na next. Následne postupujte podľa návodu v sekcii „Spustenie agenta z IDE“ > „Eclipse“ Pôvodný návod hovoril, že v súbore settings.rb je potrebné zmeniť VERSION\_0\_6\_X konštantu podľa verzie nainštalovaného servera. Vzhľadom na to, že verzie 0.6.6 a 0.6.7 sú relatívne nové a neboli zmeny pre ne implementované v tomto projekte, treba ponechať túto konštantu na hodnote 0\_6\_5, aj pri použití server verzie 0.6.6. Vo verzii 0.6.7 došlo k zmenám rozmeru ihriska a tieto zmeny by sme mali implementovať do projektu.

### Popis a riešenie problémov Linux

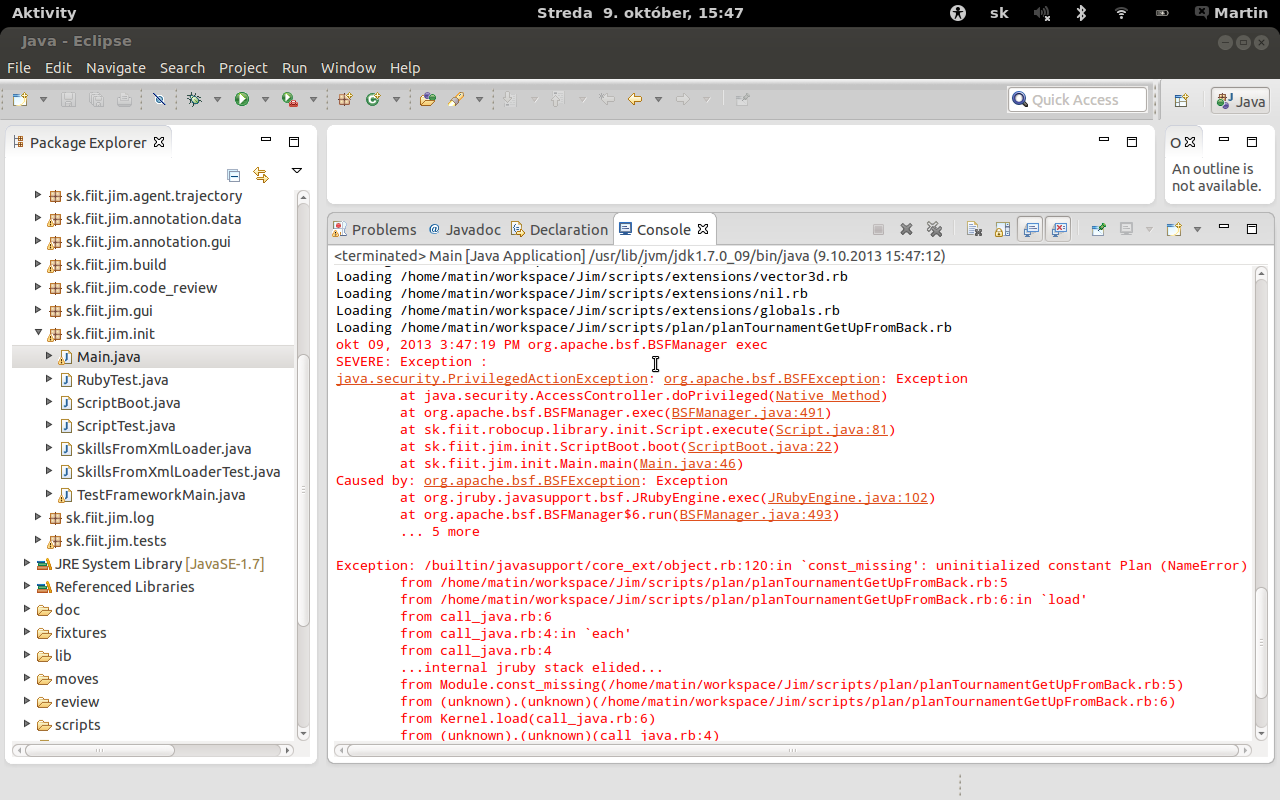
Problém z obrázku 1 môže nastať pri zavádzaní (loading) ruby súborov. Keďže tento problém sa na Windowse nevyskytuje, je pravdepodobne spôsobený tým, v akom poradí sa ruby súbory načítavajú pri zavádzaní. Na súbory *„scripts/high\_skills/walk.rb“* a *"scripts/plan/plan.rb"* sa pravdepodobne odkazuje skôr ako sú tieto súbory načítané, preto na ich skoršie zavedenie je potrebné pridať riadky

*load "scripts/high\_skills/walk.rb"*

*load "scripts/plan/plan.rb"*

v súbore jim/boot.rb za riadok include Java

Ďalší problém so zatuhnutím načítavania na súbore *jim/scripts/high\_skills/GoTo.rb.* To Riešenie je zakomentovať posledný riadok, v ktorom sa vytvárala nová inštancia triedy *GoTo*.



Obrázok 1 chybové hlásenie z dôvodu zlej postupnosti načítavania súborov ruby

### Popis a riešenie problem TestFramework

So spustením testframeworku bol na Linuxoch problém, pretože testovací framework spúšťal Jima s s príkazom java -classpath, kde jednotlivé knižnice boli oddelené bodkočiarkou. To sa však vzťahuje iba na Windows, na Linuxe funguje oddeľovanie dvojbodkou. Tento bol opravený v zdrojovom kóde.

## Opis východiskového stavu projektu

V tejto časti dokumentácie je analýza východiskový stav projektu robotického futbalu vyvíjaného na fakulte. Ako súčasť analýzy projektu bol vytvorený Rdoc, ktorý obsahuje opis tried implementovaných v jazyki Ruby.

### Projekt Jim

V projekte Jim je implementovaný hráč robotického futbalu.

**Balík jim**

*AllTests.java*

- Spustí všetky testy

*Settings.java*

- Nastavuje globálne správanie kódu. Môže zmeniť východiskové nastavenia hry pokiaľ nastavenia nie sú k dispozícií v súbore: *./scripts/config/settings.rb.*

*SettingTest.java*

- Overuje správnosť načítania východiskového nastavenia hry.

**Balík agent**

*AgentInfo.java*

- Obsahuje základné informácie o agentovi. Jedná sa napríklad o meno tímu, strana tímu a ďalšie všeobecne informácie. Špecifické informácie o agentovi sa nachádzajú v triede *AgentModel.java*

*Planner.java*

- Implementuje plánovací modul, ktorý sa skladá z volaní plánovacích ruby skriptov.

*Side.java*

- Trieda obsahujúca jeden *enum* prvok, ktorý značí priradenú stranu tímu (LEFT, RIGHT).

**Balík agent.communication**

*Communication.java*

- Trieda implementuje nízkoúrovňovú komunikáciu so serverom a doručuje správy. Nastavenia pre túto triedu sú uložené v súbore: *./scripts/config/setting.rb*.

* start: Táto metóda sa pokúsi pripojiť na server a v prípade úspechu nastaví vstupný a výstupný kanál
* restart: Táto metóda nie je dokončená. Má slúžiť na reštartovanie komunikácie so serverom.

*CommunicationTest.java*

- Implementácia tejto triedy nie je dokončená. Má slúžiť na otestovanie správnosti metód v triede *Communication.java*.

*CommunicationThread.java*

- Táto trieda sa už v súčasnosti nevyužíva.

**Balík agent.communication.testframework**

Tento balík zabezpečuje prenos informácií medzi agentom a TestFramework.

*Message.java*

- Trieda obsahuje jednoduché metódy, zabezpečujúce odosielanie základných informácií o stave agenta ako napríklad jeho číslo dresu, názov tímu ako aj informácie o stave vykonávaného high-skill-u

*TestFrameworkCommunication.java*

- Realizuje technickú implementáciu komunikácie s TestFramework-om a vyžíva prvú spomenutú triedu *Message.java.*

**Balík agent.models**

**-** popisuje model sveta, ktorý agent pozná

*AgentModel.java*

- trieda na určenie súčasného stavu hráča a jeho polohy na ihrisku. Ďalej určuje polohu jeho kĺbov, rýchlosť, posledné navštívené pozície, poslednú prijatú správu, tím do ktorého je agent zaradený

*AgentPositionCalculator.java*

- trieda slúžiaca na výpočet pozície agenta na ihrisku vzhľadom na fixné objekty- vlajky

*AgentRotationCalculator.java*

- trieda slúžiaca na výpočet otočenia agenta v priestore vzhľadom na fixné objekty- vlajky

*DynamicObject.java*

- trieda slúžiaca na opis polohy a rýchlosti objektov pohybujúcich sa v priestore - lopta

*EnvironmentModel.java*

- trieda slúžiaca na opis simulačného prostredia, v ktorom sa agent pohybuje

*FixedObject.java*

- trieda slúžiaca na opis fixných objektov umiestentých v simulačnom prostredí, v ktorom sa agent pohybuje - vlajky

*KalmanAdjuster.java*

- trieda slúžiaca na výpočet pozície objektov na ihrisku odstránením šumu kalmanovým filtrom

*Player.java*

- trieda opisujúca pozíciu, natočenie kĺbov, prílošnosť k tímu hráča, ktorého agent vníma perceptormi

*TacticalInfo.java*

- trieda slúžiaca na určenie situácie na ihrisku a vyhodnotenia prihrávok a tvorby formácie

*WorldModel.java*

-opisuje stav sveta, teda ostatných hráčov, lopty,( objektov, ktorých agent vidí), odhaduje ich pozície na základe odhadnutej rýchlosti

**Balík agent.models.prediction**

- Balík obsahuje triedy, ktoré umožňujú predpovedať situáciu, ktorá by mohla nastať v budúcnosti s najväčšou pravdepodobnosťou na základe teraz vykonávaných akcií.

*Prophecy*

- Trieda obsahuje údaje o situácií, ktorá by mala podľa očakávania nastať v budúcnosti s najvyššou pravdepodobnosťou.

*Prophet*

- V triede sú implementované metódy na predpovedanie budúcej situácie. Možnosti, ktoré ponúka táto trieda nie sú využívané v plnom rozsahu. Jedným z dôvodov slabého využívania metód triedy je neoverená správnosť implementácie.

**Balík agent.moves**

- V balíku sa nachádzajú rozsahy natočenia jednotlivých kĺbov a spracovanie základných pohybov (low skillov) z XML.

*EffectorData*

- Reprezentácia tagu (značky) effectoru (kĺbu) v XML pohyboch. Využíva sa pri načítaní pohybov z XML

*Joint*

- V triede sú uchované dáta o hraničných natočeniach (rozsahu) jednotlivých kĺbov, a tiež sú namapované označenia kĺbov z XML súborov na označenia používané v programe.

*JointPlacement*

- V triede sú uchované dáta o požadovaných natočeniach jednotlivých kĺbov. Obsahuje metódy pomocou, ktorých sa posielajú natočenia kĺbov na server a počíta sa rýchlosť ako sa daný kĺb má natáčať.

*LowSkill*

- Reprezentuje základný pohyb, ktorý môže hráč vykonať - low skill. Medzi takéto pohyby patrí kop, chôdza, otočenie, sadnutie si,...

*LowSkillTest*

- Testy metód triedy *LowSkill.* Niektoré z testov hlásia chybu.

*LowSkills*

- Namapuje inštancie *LowSkill*-ov, každý low skill má len jednu inštanciu a v prípade opätovného použitia daného skillu sa opäť použije tá istá inštancia.

*Phase*

- Trieda *Phase* reprezentuje jednu fázu low skillu, počas ktorej sa vykonajú presne definované otočenia jednotlivých kĺbov. V triede sa nachádzajú dáta o pohyboch, ktoré sa majú počas danej fázy vykonať. Fázy sú definované v XML v rámci low skillu.

*Phases*

- V triede sa nachádza zoznam (mapa), obsahujúca všetky fázy low skillu.

*SkipFlag*

- Flag umožňuje preskočenie fázy low skillu.

*SkipFlags*

- V triede sa nachádzajú metódy, pomocou ktorých je možné pracovať s flagmi (mastavit hodnotu, reset , zistenie hodnoty).

**Balík agent.parsing**

- V balíku sú implementované triedy, ktoré zabezpečujú uloženie, spracovanie a interpretáciu údajov prijatých z jednotlivých perceptorov a receptorov.

*ForceReceptor*

- V triede sa uchovávajú hodnoty sily pôsobiacej na nohy. Každá noha má samostatné centrum pôsobenia sily. Používa sa v triede *Perceptors*.

*HearReceptor*

- V triede sa uchováva správa, ktorú agent prijal pomocou sluchového receptoru. Okrem samotného textu správy sú k dispozícií aj informácie o tom, komu bola správa určená, z akej vzdialenosti, približnej pozície a kedy prišla. Používa sa v triede *Perceptors*.

*ParsedData*

- Trieda obsahuje dáta, ktoré boli získané z prijatej správy od servera. Sú to dáta z receptorov (zrak, sluch, sila pôsobiaca na nohy), a tiež dáta opisujúce aktuálnu situáciu na ihrisku.

**Balík ParsedDataObserver**

*- ParsedDataObserver* je rozhranie a slúži triedam, ktoré ho implementujú, na oznámenie, že bola prijatá správa od servera. Implementujúce triedy sa tiež zapíšu do observera pomocou metódy *subscribe* triedy *Parser*, aby mohli tieto správy prijímať.

*Parser*

- Trieda obsahuje metódy na zabezpečenie fungovania observera. Tiež poskytuje funkcionality na prijatie, pretransformovanie a rozloženie správy od servera.

*ParserTest*

- Trieda obsahuje unit testy, ktoré overujú správnu funkcionalitu rozparsovania prijatej správy od servera. Dva testy sú implementované korektne a jeden je zakomentovaný pretože neprebehne úspešne.

*Perceptors*

- V triede sa nachádzajú metódy, pomocou ktorých sa transformujú prijaté údaje zo všetkých receptorov do dátovej reprezentácie v programe.

*PlayerData*

- V triede sú zapuzdrené dáta z videnia hráča.

*SExpression*

- Výraz prijatý zo servera, ktorý sa následne syntaktickou analýzou rozloží na jednotlivé informácie.

*SeePerceptor*

- Metódy triedy aktualizujú dáta uložené v triede *ParsedData.* Aktualizujú sa len tie dáta, ktoré je možné získať zo zrakového perceptora.

*SeenPerceptor*

- Metódy triedy parsujú informácie prijaté zo zrakového perceptora.

*SeenPerceptorData*

- V triede sú zapuzdrené dáta zo zrakového perceptora.

**Balík agent.parsing.sexp**

- V balíku sú implementované triedy, ktoré slúžia na spracovanie výrazu pomocou syntaktickej analýzy pre potreby získania informácií z daného výrazu.

*SArray*

- Na základe syntaktickej analýzy výrazu vytvorí strom, ktorého vrcholy sú buď opäť typu *SArray* alebo *SString*. Prijatý výraz je typu *SExpression*.

*SException*

- Výnimka vzťahujúca sa na proces parsovania výrazu.

*SObject*

- Abstraktná trieda, od ktorej dedia triedy SString a SArray.

*SString*

- Zapuzdrenie jednej premennej typu *String* do triedy.

**Balík agent.server**

*TFTPServer.java*

- Riadi pripojenie TFTP servera, prípadne jeho ukončenie.

**Balík agent.skills**

Obsahuje hlavnú implmentáciu vyšších schopností agenta - HighSkill.

**Balík annotation.data**

*Annotation.java:*

- Definuje položku v zozname anotácií.

*AnnotationManager.java:*

- Zo súboru: *./move/annotations* sa načítajú všetky anotácie.

*Axis.java:*

- Definuje rotáciu okolo jednotlivých osí.

*Joint.java:*

- Definuje kľb.

*Main.java:*

- Spusti grafické rozhranie pre anotácie implementované v balíku *sk.fiit.jim.annotation.gui.*

*MEC.java:*

- Minimal enclosing circle.

*MoveValidator.java:*

- V triede sú implementované metódy slúžiace na overenie správnosti špecifikácie pohybu.

*State.java:*

- Definuje stav v akom sa robot nachádza, využíva triedu *Joint.java*.

*Values.java:*

- Definuje minimálne, maximálne a priemerné hodnoty rotácií okolo jednotlivých osí použitých v anotácií.

*XMLCreator.java:*

- Metódy v triede slúžia na serializáciu anotácie do Xml súboru.

*XMLParser.java:*

- Parsuje Xml a vytvára Annotation.

**Balík annotation.gui**

V balíku je definované grafické užívateľské rozhranie pre načítanie Xml súboru a jeho kontrolu.

*MyFilter.java*

*Window.java*

**Balík Gui**

Implementácia tejto triedy nie je dokončená. Momentálne umožňuje opätovné načítanie Xml “*low-skillov*“ a plánov.

*ReplanWindow.java*

**Balík Init**

*Main.java*

Trieda obsahuje jedinú metódu *main* umožňujúcu spustiť agenta Jim

- V metóde sa vytvorí inštancia triedy SkillFromXmlLoader. Pohyby ktoré dokáže agent vykonať sa získajú z adresára *./moves* použitím metódy *load().* Jedná sa o “low-skill”

- V metóde sa vytvorí inštancia triedy *AnnotationManager* ktorá sa nachádza v balíku: *sk.fiit.jim.annotation.data*. Popisy pohybou ktoré dokáže agent vykonať sa získajú použitím metódy *loadAnnotation* z adresára *./moves/annotations.*

- Načítajú sa všetky ruby skripty agent Jim vykonaním skriptu: *./scripts/boot.rb.*

*- Posledným krokom je zavolanie metódy start()* z triedy *Communication.java,* ktorá je implementovaná v balíku: *sk.fiit.jim.communication.*

*RubyTest.java*

- skúša či je možné načítať všetky skripty.

*ScriptBoot.java*

- vykoná *./scripts/boot.rb* skrip, ktorý načíta všetky ostatné ruby skripty. Vykonaním metódy *boot* sa reštartujú všetky nastavenia na hodnoty určené v skripte *scripts/congif/setting.rb* a spôsobí reset plánovača.

*ScriptTest.java*

- obsahuje metódy na otestovanie scriptov

* + *Public void javaBridging()*
  + *public void performacne()*
  + *public void testIngeritance()*

*SkillsFromXmlLoader.java*

- Obsahuje metódy, ktoré načítajú pohyby ktoré dokáže agent vykonať zo súboru ./moves. Tieto pohyby môžu byť uložené v cache.

*SkillFromXmlLoaderTest.java*

- Obsahuje metódy slúžiace na overenie funkčnosti načítavania “low skillov“ z Xml súborov.

*TestFrameworkMain.java*

- Navyše od triedy *Main.java* je schopný vkladať agenta s parametrami. Táto trieda je potrebná pre strojové učenie.

* + Možné argumenty:
    - testframework host port - zapnutie odosielania spätnej väzby na adresu host:port
    - test [port] - zapnutie lokálneho tftp servera - možné dodatočné určenie portu (default 3070)
    - uniform number - nastavenie atribútov agenta na pripojenie k serveru
    - team team\_name - nastavenie atribútov agenta na pripojenie k serveru

**Balík Tests**

- V balíku sú implementované triedy, ktoré obsahujú unit testy. Pravdepodobne z dôvodu neúplnej realizácie implementácie brankára sú všetky testy v danom balíku zakomentované a ich funkčnosť je nekorektná vzhľadom na bežného hráča. Sú to triedy *GoalieTestCase*, *GoalieTestCaseTest*, *TestJim*.

### Projekt RoboCupLibrary

- knižnica slúžiaca na prácu so základnými operáciami použitými pri riešení RoboCupu

**Balík annotations**

- slúži na prezeranie kódu

*Bug.java*

-anotácia na určenie chyby v kóde

*Refactor.java*

- anotácia na určenie potreby refaktorizácie

*Reviewed.java*

- anotácia na označenie kódu za prezretý

*TestCovered.java*

- anotácia informujúca o pokrytí funkcionality testom

*UnderConstruction.java*

- anotácia určujúca kód za nedokončený

**Balík geometry**

**-** opisuje základné geometrické objekty využívane v robocupe

*Angles.java*

- popisuje určenie rozdielu medzi uhlami

*Circle.java*

- popisuje objekt určujúci kruh v 2D priestore

*Line2D.java*

- popisuje objekt a geometrické výpočty s úsečkou v 2D priestore

*MEC.java*

- popisuje výpočet najmenšieho kruhu obsahujúceho n bodov

*Point3D.java*

- popisuje objekt reprezentujúci bod v 3D priestore

*Vector2.java*

- popisuje objekt reprezentujúci vektor v 2D priestore

*Vector3.java*

- popisuje objekt reprezentujúci vektor v 3D priestore - depricated

*Vector3D.java*

- popisuje objekt reprezentujúci vektor v 3D priestore - komplexnejšia ako predchádzajúca trieda

**Balík init**

*Script.java*

- popisuje triedu vykonávajúcu ruby skripty

**Balík math**

*KalmanForVariable.java*

- popisuje triedu určujúcu výpočet linearno-kvadratického Gaussovho regulátora pre premennú

*KalmanForVector.java*

- popisuje triedu určujúcu výpočet linearno-kvadratického Gaussovho regulátora pre vektor

*MathExpressionEvaluation.java*

- popisuje triedu určujúcu výpočet matematického vzorca z textu za pomoci Ruby

*TransformationMatrix.java*

- popisuje triedu určujúcu transformačnú maticu - vektor v priestore

**Balík review**

- slúži na prezeranie kódu

*ReviewOk.java*

- anotácia informujúca o tom, že kód splnil všetky podmienky:

1.pokrytie unit testom

2.prezretý iným programátorom

3.testovaný voči serveru

### Projekt TestFramework

**Balík agenttrainer**

*AgentMoveConfigReader.java*

- Načíta konfiguráciu pohybov zo súboru do dokumentu.

*AgentMoveReader.java*

- Načíta pohyby zo súboru do dokumentu.

*AgentMoveWriter.java* – Zapíše pohyby do dokumentu.

**Balík agenttrainer.models**

Štruktúrne triedy so setermi a getermi.

**Balík annotator**

*Annotator.java*

- výpočet hodnôt pre anotáciu a vyhodnotenie testu.

*AnnotatorTestCase.java*

- Spustenie test case na zaklade ktorého sa vytvára anotácia.

**Balík annotator.serialization**

*MoveValidator.java*

- Skontroluje či pohyb má správy zápis.

*XMLcreator.java*

- serializuje anotáciu a vytvorí XML.

*XMLparser.java*

- z xml anotácie vytvorí objekt anotácie.

**Balík communication.agent**

*AgentJim.java*

- trieda ktora reprezentuje agenta. Umožňuje pripojenie na samostatne spusteného agenta a reload skriptov agenta.

*AgentManager.java*

- Pridávanie a mazanie hráčov.

**Balík communication.robocupserver**

*RobocupServer.java*

- Pripojenie na server a spustenie príkazu.

**Balík monitor**

*AgentMonitor.java*

- Pripojenie sa na hráča a aktualizácia údajov o jeho stave, pozícii.

**Balík parsing**

Parsovanie správ zo servera a ich prevod do štruktúr.

**Balík parsing.model**

Triedy štruktúr.

**Balík trainer.testsuit**

Triedy s testovacími prípadmi pohybov.

## Vytvorenie unit testu

Cieľom úlohy vytvoriť unit testy bolo oboznámenie sa so spôsobom tvorby takéhoto typu testov. V rámci tejto úlohy boli vytvorené a zdokumentované nasledovné testy metód:

### isInHalfPlane()

Bol vytvorený unit test pre metódu „*isInHalfPlane(double a, double c, Vector3D position)*”triedy „*AgentInfo*“, ktorá je súčasťou balíka „*sk.fiit.jim.agent*“. Metóda vypočíta či bod spadá do určenej polroviny definovanej súradnicami a, c. Metóda vracia hodnotu „*true*“ ak bod leží v určenej polrovine a „*false*“ ak neleží. Unit test sa nachádza v triede „*AgentInfoTest*“ v tom istom balíku.

V rámci unit testu boli určené dvojice bod a polrovina, o ktorých bola dopredu zistená vzájomná poloha. Následne bola pre každú dvojicu spustená metóda „*isInHalfPlane*“ a overená správnosť výstupov.

Unit test bol zapísaný do triedy„*AllTests*“ balíka „*sk.fiit.jim*“, v ktorej sa nachádzajú všetky testy projektu.

### calculateDistance()

Bol vytvorený unit test pre metódu „*calculateDistance(Vector3D startPosition, Vector3D stopPosition*)”triedy „*AgentInfo.java*“, ktorá je súčasťou balíka „*sk.fiit.jim.agent*“. Metóda slúži na výpočet vzdialenosti medzi dvoma pozíciami a vracia hodnotu „*doubledistance*“.

Vytvorený unit test je umiestnený v triede „*AgentInfoTest.java*“ v rovnakom balíku ako trieda implementujúca testovanú metódu.

Unit test pozostáva z testovacích prípadov, v ktorých je napevno zadané aký výsledok ma metóda vrátiť pre konkrétne vstupné parametre.

Unit test bol zapísaný do triedy „*AllTest.java*“ balíka „*sk.fiit.jim*“, v ktorej sa nachádzajú všetky testy projektu.

### whereIsTarget()

Bol vytvorený unit test pre metódu „*whereIsTarget(Vector3D target)*”triedy „*AgentInfo*“, ktorá je súčasťou balíka „*sk.fiit.jim.agent*“. Metóda vypočíta, na aký smer je otočený cieľ pre hráča definovaný vstupným vektorom. Metóda vracia hodnoty „*front*“, „*back*”, „*right*”, „*left*“ v závislosti od vstupných hodnôt vektora. Unit test sa nachádza v triede „*AgentInfoTest*“ v tom istom balíku.

V rámci unit testu boli zistené možné kombinácie vstupných dát vo vektore a test bol navrhnutý tak, aby pokryl všetky možné výstupné hodnoty funkcie.

Unit test bol zapísaný do triedy„*AllTests*“ balíka „*sk.fiit.jim*“, v ktorej sa nachádzajú všetky testy projektu.

### bIsBallNearestToMe()

Bol vytvorený unit test pre metódu „*bIsBallNearestToMe()*“triedy „*TacticalInfo*“, ktorá je súčasťou balíka „*sk.fiit.jim.agent.models*“. Metóda určuje, či je zvolený agent najbližšie k lopte. Metóda vracia hodnoty „*true*“ alebo „*false*“ v závislosti od údajov získaných z „*WorldModel*“. Unit test sa nachádza v triede „*TacticalInfoTest*“ v rovnakom balíku ako je umiestená testovaná trieda.

V rámci unit testu boli pevne zadefinované pozície štyroch hráčov relatívne blízko pri sebe. Následne bola zvolená pozícia lopty na nulový bod. V teste určujeme vzdialenosť agentov k lopte.

Unit test bol zapísaný do triedy„*AllTests*“ balíka „*sk.fiit.jim*“, v ktorej sa nachádzajú všetky testy projektu.

### substract() a addiotion()

Bol vytvorený unit test pre triedu „*Vector3*“ v balíku „*sk.fiit.robocup.library.geometry*“ pre metódy „*subtract*“ a „*addition*“. Metódy vracajú vektor, ktorý je súčtom alebo rozdielom dvoch vektorov, ktoré prichádzajú ako parametre. Unit test bol vložený do rovnakého balíka ako je trieda „*Vector3*“.

Na začiatku unit testu sa zadefinujú dva vektory. Následne sa volá metóda „*addition*“, ktorá dostane tieto dva vektory ako parametre. Vrátený vektor sa porovnáva, či majú jeho zložky správne hodnoty. Podobne je to aj pre metódu „*substract*“.

### isForward()

Bol vytvorený unit test pre metódu “*isForward(Vector3D startPosition, Vector3D stopPosition, Vector3D opponentPlayerPosition)*” triedy “*AgentInfo*“, ktorá je súčasťou balíka “*sk.fiit.jim.agent*“. Metóda slúži na určenie či je súper pred hráčom. Metóda vracia hodnotu typu „*bool*“*.*

Vytvorený unit test je umiestnený v triede “*AgentInfoTest*“ v rovnakom balíku ako trieda implementujúca testovanú metódu.

Unit test pozostáva z testovacích prípadov, v ktorých je napevno zadané aký výsledok ma metóda vrátiť pre konkrétne vstupné parametre.

Testovacie vstupné parametre boli zvolené tak aby dali

1. Jednoznačný pozitívny výsledok.
2. Pozitívny výsledok ktorý nie je určiteľný zo vstupných premenných.
3. Jednoznačný negatívny výsledok.

Unit test bol zapísaný do triedy „*AllTest*“ balíka „*sk.fiit.jim*“, v ktorej sa nachádzajú všetky testy projektu.

### isOnPosition()

Na testovanie bola zvolená metóda „*isOnPosition()*“ triedy „*TacticalInfo*“. Jednou z úloh tejto triedy je vytvorenie formácie tímu v závislosti od aktuálnej polohy lopty a zapísanie vypočítanej polohy do premennej „*pos*“typu „*Vector3D*“. Testovaná metóda slúži na overenie, či je aktuálna poloha hráča zhodná s vypočítanou polohou hráča v premennej„*pos*“. Metóda vracia premennú typu „*bool*“.

Keďže formácia je vypočítavaná na základe identifikačného čísla hráča, pred spustením samotného testu metódy je potrebné zadať toto číslo fiktívnemu hráčovi a zavolať metódu „*TacticalInfo.setMyFormPosition()*“, ktorá nastaví spomínanú premennú „*pos*“. Následne je potrebné zadať pozíciu fiktívneho hráča. V tomto bode je možné spustiť testovanie, ktoré bolo vykonané pre dve situácie, vracajúce „*true*“a dve situácie, vracajúce „*false*“.

Teste je súčasťou triedy „*TacticalInfoTest*“, ktorá už je zapísaná v hlavnej testovacej triede „*AllTest*“.

## Zmena správania hráča

Cieľom úlohy zmena správania hráča bolo oboznámenie sa s možnosťami pohybov robota, spôsobu vytvárania pohybov a tvorba jednoduchého plánu. Výsledkom úlohy je niekoľko vytvorených pohybov a plánov, ktoré ale nemajú praktické využitie.

### Modifikovaný plan „p*lanTactic*“ 1.

Na začiatku si Jim(hráč) zvolí plán „*PlanTactic*“. Tento plán má nastavený v „*Settings.rb*”, ktoré si načíta pri vytváraní. Po začatí zápasu sa začne vykonávať plán. Tento plán je v podobe ruby skriptu, ktorý si vyžiada nejaké informácie napr. pozícia lopty, vlastník lopty, informácie o stave na ihrisku atď. Následne sa rozhoduje na základe situácie (ako je lopta ďaleko, či ju má vlastný tím alebo súper, ...), ktorý highskill sa použije. Na začiatku hráč stojí ďaleko od lopty a keďže ho testujem samého na scéne, plán zvoli prípad keď je lopta voľná a nikto loptu nevlastní. V tomto prípade sa zvoli highskill „*Walk to ball*“, ktorý prevezme riadenie hráča.

Highskill „*Walk to ball*“ sa snaží dostať hráča najbližšie k lopte a natočiť ho tak, aby mohol do nej kopnúť. Správanie hráča bolo modifikované tak, aby sa na miesto kopnutia do lopty hodil dozadu.

Toto bolo možné docieliť zmenou skriptu „*Walk to ball*“. V skripte je časť, kde hráč kontroluje svoju pozíciu s pozíciou lopty. V prípade, ak vzdialenosť od lopty je väčšia ako definované limity, skript vyberá lowskill, ktorý posunie hráča bližšie v danom smere odchýlky. Ak sa hráč nachádza v stanovených intervaloch, vyberie sa lowskill kopnutia. Na tomto mieste bol nahradený lowskill „*kick\_right\_faster*“ za „*fall\_back*“. To spôsobilo, že hráč stojaci pred loptou sa hodí dozadu. Tento proces sa opakuje až do skončenia polčasu.

### Modifikovanie plánu „planTactic“ 2.

Zmena správania hráča bola docielená úpravou existujúceho plánu s názvom „*planTactic*”. Plán obsahuje zložitejšiu logiku plánovania highskill-ov a lowskill-ov. Táto logika bola nahradená jednoduchšou, ktorá zahŕňa iba úvodné umiestnenie hráča na svoju pozíciu (beam) a test, či sa hráč nachádza na zemi. V prípade, že áno, postaví sa, a vykoná zadaný lowskill. Tento test prebieha po každom vykonaní lowskillu.

Daný plán bol teda upravený tak, aby umožňoval testovanie zadaného lowskill-u, bez rizika, že by daný hráč spadol a nemohol tak ďalej testovať zadaný lowskill. Pri testovaní lowskillu boli využité jednoduché pohyby rôznych kĺbov, skladajúce sa z viacerých fáz.

### Vytvorenie nového pohybu a použitie „planZakladny“.

Zmena správania hráča bola dosiahnutá pomocou vytvorenia plánu na základe „*planZakladny*“, ktorý sa nachádza na stránke „*/wiki/index.php/Planovac*“ a vytvorením nového pohybu, ktorý pozostáva z pomalého čupnutia si hráča (4s), a následného predpaženia jednej z rúk. Počas vykonania pohybu hráč zakýva hlavou. Plán bol nastavený ako primárny plán hráča a po zavedení hráča na server bol otestovaný novovytvorený pohyb.

Pohyb bol vytvorený pomocou editoru pohybov a následne prepísaný do XML štruktúry.

### Vytvorenie nového pohybu a modifikovanie „planZakladny”

Bol vytvorený plán: planfilip.rb, ktorý je modifikovaný “p*lanZakladny*“ prebratý z „[*http://team09-13.ucebne.fiit.stuba.sk/wiki/index.php/Planovac*](http://team09-13.ucebne.fiit.stuba.sk/wiki/index.php/Planovac)“. Do pôvodného plánu bola pridaná podmienka využívajúca high-skill *“get\_up“*, vďaka ktorej sa hráč dokáže postaviť v prípade pádu. Ďalej bol plán upravený tak aby využíval nový lowskill *“filipskuska.xml”*. Tento pohyb pozostáva z kombinácie vertikálneho pohybu hlavy [he2] a oboch rúk [lae1 , rae1]. Aby sa plán načítal pri umiestnení nového hráča na server, bol upravený súbor *“Settings“.* Výsledný plán a pohyb bol spustený a otestovaný.

### Vytvorenie nového highskillu, lowskillu a modifikovanie „planZakladny“

Vytvorený bol nový plán s názvom „*planZakladny*“ na základe návodu na wiki. V ňom sa volá novovytvorený highskill „*DoExercising*“ pokiaľ nie je hra začatá - aby sa pred hrou hráč rozcvičil. Pribudol nový lowskill „*rukami\_hore\_dole*“, v ktorom hráč cvičí oboma rukami naraz striedavo hore-dole. Po čase sa pohyb rúk zastaví a hráč s predpaženými rukami urobí drep. Po ňom otočí hlavou postupne do oboch smerov. Lowskill pohyb bol vytvorený priamo prepisovaním v xml súbore podľa dokumentácie spísanej na wiki.

### Vytvorenie nového plánu

Pri riešení tohto zadania bol vytvorený nový plánovač „nemecekPlan“, ktorý je zddedený od „*Plan5ko*“. V rámci tohto plánovača bola upravená podmienka kopu lopty, kde bol vymenený pôvodný highskill slúžiaci na kop lopty za highskill slúžiaci na demonštráciu robocupu s názvom „*freeRide*“. Tento high skill pozostáva z rôznych pohybov od pohybu hlavy až po stojku. Následne v highskille „*Turn*“ slúžiaceho na posun hráča pri lopte bola upravená podmienka určujúca smer posunutia. Pri tejto zmene si agent namiesto posunutia doľava vybral náhradný lowskill zo zoznamu (točenie hlavou, priamy pád na tvar a stojka na hlavu).

### Vytvorenie nového plánu

Zmena správania hráča bola docielená vytvorením nového testovacieho plánu s názvom „*planRoman*“. Plán obsahuje jednoduchú logiku plánovania highskill-ov a lowskill-ov na základe podmienok.

Logika pozostáva z pohybu hráča k lopte s použitím highskillu „*GotoBall*“. V prípade pádu hráča na zem, sa aktivuje „*GetUp*“ na postavenie hráča. Ak sa hráč nachádza pri lopte, začne sa otáčať na jednu stranu s použitím lowskillu „turn\_right“.