**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

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.

## 1.1 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.

# 2 Prvý šprint

## 2.1 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 2.1 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 2.1** 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
6. 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/

**2. 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.
    a. 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*.
    b. Tu nastavíte adresár, kde sa má projekt naklonovať.
    c. 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 2.2 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 2.2** 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), 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.

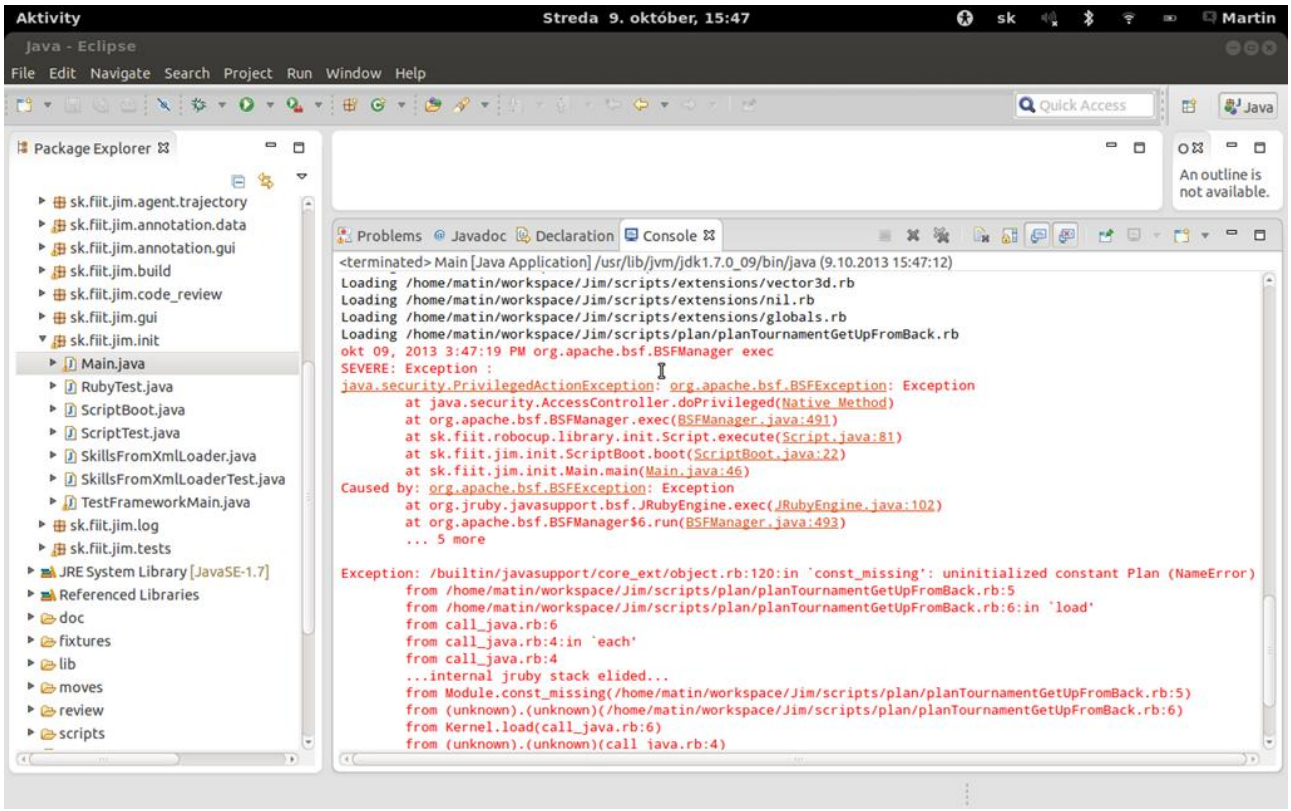
## Popis a riešenie problémov Linux

Problém z obrázku 2.1 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 2.1** 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.

## 2.2 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.

## 2.3 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*“.

## 2.4 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*“. 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“.

# 3 Druhý šprint

## 3.1 Analýza diplomových prác

Cieľom úlohy analýza diplomových prác bolo oboznámenie sa s ich výsledkami a možnosťami zakomponovania týchto výsledkov do spoločného riešenia robotického futbalu.

## Bc. Peter Paššák - Optimalizovanie pohybov robota pomocou evolučných algoritmov v 3D simulovanom robotickom futbale

Hlavným cieľom práce bolo pomocou evolučného algoritmu vylepšiť súčasné pohyby (LowSkills) agenta. Autor sa zameral na optimalizovanie štyroch esenciálnych pohybov:

- kop do lopty
- chôdza vpred
- chôdza vzad
- úkrok do strany

Experimentovaním s rôznymi parametrami genetického algoritmu a prispôsobovaním použitej fitness funkcie sa autorovi podarilo značne zlepšiť každý zo spomínaných pohybov. Na tieto účely bola rozšírená implementácia agenta o nástroj na automatické spúšťanie genetického algoritmu s možnosťou iniciálneho nastavenia požadovaných parametrov. Pri ohodnocovaní nových vyvinutých pohybov bolo nutné spustiť ich testovanie v simulačnom prostredí. Z toho dôvodu muselo byť vytvorené aj automatické testovanie požadovaného pohybu. Testovanie nie je obmedzené iba na testovanie vzniknutých pohybov genetického algoritmu, ale je možné ho použiť aj s manuálnych zvolením ľubovoľného existujúceho pohybu. Samozrejme, nejedná sa o úplne univerzálne testovanie pohybov a prináša isté obmedzenia, kedže každý pohyb je niečím špecifický a je potrebné sledovať rôzne vlastnosti, no ponúka dobrý základ, ktorý je možné v prípade potreby prispôsobiť. Nástroje je možné používať pomocou grafického rozhrania. Obidva sú súčasťou jedného okna.

V nasledujúcej časti priblížime konkrétne výsledky optimalizácie pohybov, vonkajší pohľad na prácu s nástrojom a vnútorný pohľad na implementáciu nástroja a genetického algoritmu.

**Práca s nástrojom**

Pri spustení zdrojového kódu agenta sa zobrazí okno s názvom *WorldMonitor*. Kliknutím na tlačidlo *Automatic testing* sa zobrazí spomínaný nástroj. Ako je vidieť na *obr.3.1*, implementovaný nástroj ponúka široké spektrum možností testovania a evolúcie pohybov. Skladá sa z troch hlavných častí:

- **Typ testovania (resp. evolúcie)** - Táto časť sa skladá dvoch boxov (Test type a Evolution type), ktoré obsahujú prepojený zoznam ovládacích prvkov typu radio button. Je možné zvoliť iba jedne zpomedzi všetkých možností typov testovania a evolúcie.
- **Parametre testovacieho prostredia** - Zvyšné prvky vrchného boxu, obsahujúce polia pre zadanie pozície hráča, dĺžky jedného testu zvoleného pohybu a tlačídlá pre zvolenie

testovaného pohybu a prevrátenie zvoleného pohybu (vysvetlené neskôr). V tejto časti sa ešte nachádzajú polia pre zadanie celkového počtu testovaní a počtu testov pre vypočítanie priemernej fitness hodnoty, ktoré sa sprístupnia na úpravu až po zvolení jedného z typu evolučného testovania.

- **Parametre evolučného algoritmu** - Podstatnú časť tvoria v spodnej časti parametre genetického algoritmu, ktoré je možné upravovať až po zvolení niektorého z ponúknutých typov evolučného testovania (Evolution type).


**Obrázok 3.1** Grafické rozhranie nástroja na testovanie a evolúciu pohybov

## Testovanie

Najskôr sa pozrieme na použitie nástroja ako univerzálneho testovacieho prostredia. Prvým typom testovania je Automatic test. V prípade zvolenia tohto typu testu nie je možné upravovať ani dĺžku jedného testu pohybu, ani zvoliť požadovaný pohyb. Po spustení testovania tlačidlom Start sa nič neudialo. Z pohľadu implementácie tried (opísané neskôr) sa tento typ testovania javí ako hlavné testovanie, ktorého rozšírením vznikajú potrebné špecifické testovania, ako napríklad ďalšie tri typy testovaní. Pri zvolení testovania Kick easy test, Walk easy test alebo Sidestep easy test už je možné pracovať s dĺžkou jedného testu a je možné zvoliť požadovaný pohyb.

Princíp testovania, ktorý je spoločný pre všetky typy si teraz priblížime bližšie. Na dosiahnutie efektu, aby boli každé podmienky testovania v rámci možností zhodné, je potrebné zabezpečiť presnú polohu hráča, a v prípade potreby aj lopty, pred každým

vykonaním testovaného pohybu. Z pohľadu architektúry celého simulačného prostredia robotického futbalu, je toto možné docieliť jedine kombináciou funkcií tzv. monitora simulačného prostredia a samotného agenta.

Pri spustení jedného z typov testovania sa pomocou monitora nastaví mód hry BeforeKickOff. Výhodou tohto módu je pozastavenie simulácie hry (časovač hry nie je spustený) a je v ňom možné umiestňovať agentov na ľubovoľnú pozíciu pomocou príkazu beam. Testovač teda v tomto kroku umiestni agenta a loptu na pozíciu, zadanú v grafickom rozhraní. Pozícia lopty je vo všetkých prípadoch prednastavená vždy na stred ihriska. Ak teda potrebujeme testovať kop do lopty, agenta umiestnime mierne za loptu. Následne testovač pomocou príkazu monitora zmení mód hry na PlayOn. V tomto móde je simulácia hry spustená a teda je spustený aj časovač. Testovač vykoná zadaný pohyb, zmeria požadované metriky testu, ako napríklad dĺžku kopu alebo rýchlosť chôdze a znovu pozastaví hrací mód zmenou na BeforeKickOff. Maximálne trvanie medzi jednotlivými pozastaveniami je dané spomínaným parametrom "dĺžka jedného testu" (Test length in s). Na výstupe sa následne zobrazia výsledky meraných metrík a testovanie pohybu je spustené znovu, umiestnením hráča a lopty na pôvodnú (iniciálnu) polohu a následnou zmenou módu na PlayOn.

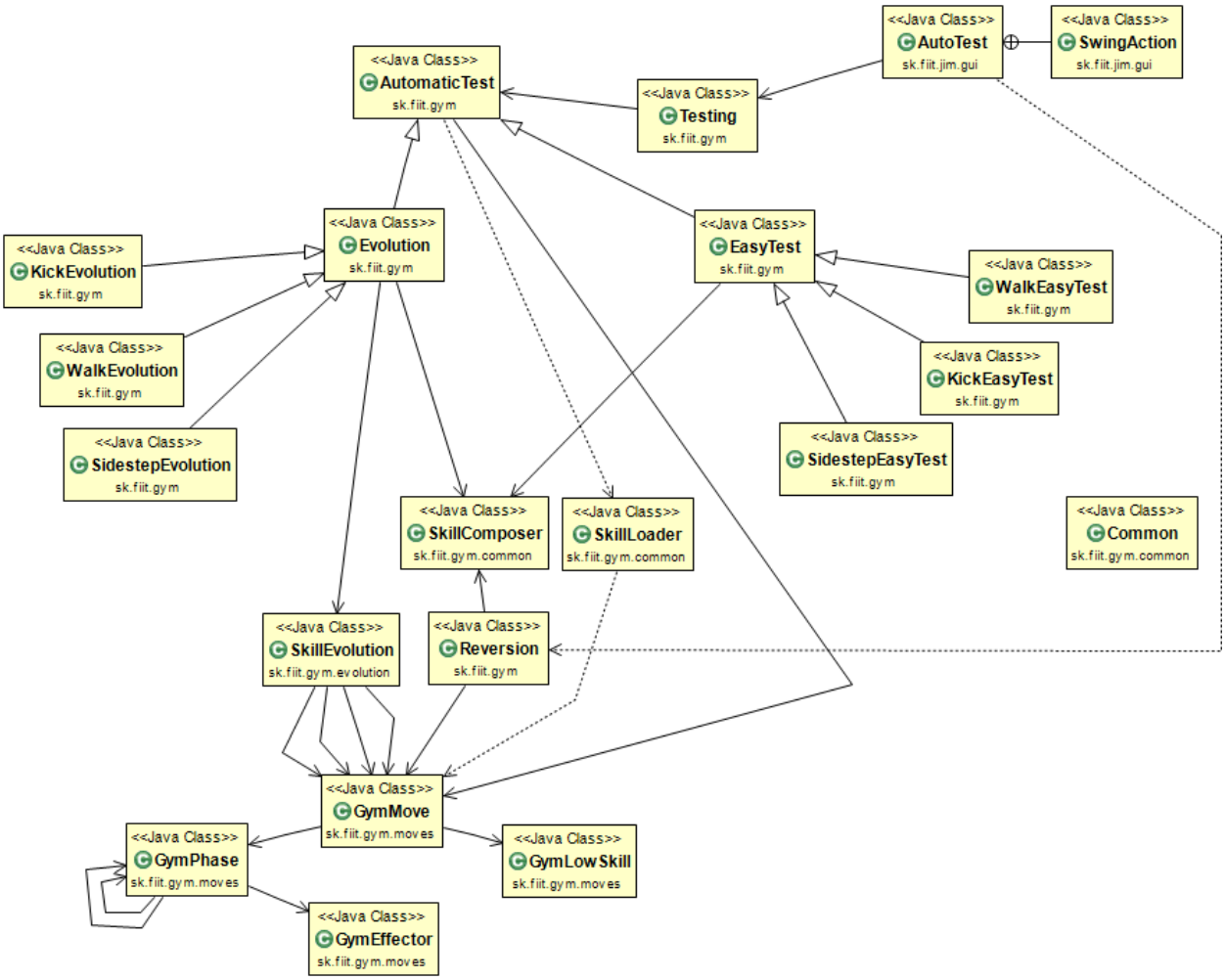
Dĺžka testovania je obmedzená prednastavenou celkovou dĺžkou simulácie hry. Následne je potrebné server manuálne reštartovať a testovanie spustiť znovu. Parametre testovanie je možné upraviť aj počas behu testu, pomocou tlačidla Update.

**Evolúcia**

Samotná evolúcia a spomínané testovanie spolu úzko súvisia. Evolučný mechanizmus priamo využíva testovanie novo-vytvorených pohybov. Priebeh evolúcie je rovnaký ako pri testovaní, s rozdielom v použitom testovanom pohybe. Pred spustením evolúcie je potrebné zadať pohyb, z ktorého bude algoritmus vychádzať. Následne sa vytvorí nová populácia pohybov, ktoré testovač postupne na základe zadaných parametrov otestuje a zobrazí výsledky meraných metrík vrátane fitness hodnoty každého pohybu. Po otestovaní každého pohybu sa vyvinie nová populácia pohybov, ktoré sa tiež všetky otestujú a tento cyklus pokračuje do vypršania času simulácie alebo do zadaného počtu generácií.
Vzniknuté pohyby sú ukladané do priečinka /trener_moves/.

## Implementácia

Implementácia výsledného riešenia je znázornená diagramom tried na obr.3.2. Všetky triedy, okrem triedy AutoTest, sú súčasťou balíka sk.fiit.gym. Túto implementáciu priblížime opísaním každej znázornenej triedy. Predpokladá sa, že všetky opísané triedy, okrem triedy AutomaticTest, vytvoril autor analyzovanej práce. Triedu AutomaticTest ale modifikoval. V grafe pre zjednodušenie nie sú znázornené všetky závislosti (dependencies).




**Obrázok 3.2** Diagram tried výsledného riešenia

**AutoTest** - Trieda reprezentujúca grafické rozhranie spomínaného nástroja na riadenie testovania a evolúcie. Obsahuje rozloženie grafických prvkov a ActionListeners. Po spustení testu, kliknutím na tlačidlo Start, sa vytvorí objekt triedy Testing.

**Testing** - Táto trieda spúšťa požadovaný typ testovania ako samostatné vlákno. Obsahuje metódu na aktualizovanie parametrov testovanie bežiaceho vlákna a metódu na zastavenie vytvoreného vlákna, teda testovania.

**AutomaticTest** - Základná trieda, špecifikujúca priebeh testovania, od ktorej dedia konkrétne typy testovaní. V minulosti pravdepodobne slúžila na automatické testovanie určitého pohybu, avšak nebolo možné toto testovanie kontrolovať počas behu programu. Potrebné parametre sa nastavovali pred spustením programu.

**EasyTest** - Rozširujúca trieda triedy AutomaticTest. Pridáva možnosť zvolenia testovaného pohybu počas behu programu, na čo využíva triedu SkillComposer, a parameter dĺžky vykonania jedného pohybu.

**WalkEasyTest, KickEasyTest, SidestepEasyTest** - Konkrétne triedy typov testovania. Ich hlavný účel je špecifikovanie meraných metrík daného testu.
**Evolution** - Upravuje testovanie pre potreby evolúcie. V podstate sa stále jedná o testovanie, ktoré je obohatené o vývoj testovaných pohybov, spolu s ich automatickým testovaním. Obsahuje hlavný cyklus na riadenie evolučného procesu, základnú metódu na výpočet fitness, na generáciu iniciálnej populácie, informačné výpisy o parametroch evolúcie a ďalšie metódy potrebné ku kooperácií testovania a evolúcie. Hlavný cyklus evolúcie využíva triedu SkillEvolution, ktorá obsahuje populáciu jedincov a konkrétne metódy kríženia a mutácie. Na zostavenie XML pohybu vyžíva triedu SkillComposer.

**WalkEvolution, KickEvolution, SidestepEvolutionEvolution** - Rozširujú triedu Evolution o konkrétne metódy na výpočet fitness a výpis výsledkov testovaného pohybu, ktoré sú pri každom type evolúcie mierne odlišné.

**SkillComposer** - Slúži na zostavenie a uloženie XML pohybu z pohybu typu GymMove (ktorý je vytvorený pre potreby evolúcie). Pohyb v tvare XML je uchovaný v štruktúrovanej podobe (ako ostatné pohyby) a v tomto tvare je možné ho testovať v simulačnom prostredí.
SkillLoader - Táto trieda je využívaná metódami tried AutomaticTest a Reversion na účely načítania XML pohybu a vytvorenie jeho podoby ako GymMove. Využíva teda triedu GymMove.

**SkillEvolution** - Obsahuje parametre a konkrétne implementácie metód evolúcie (kríženie, mutácia, generácia populácie, ruleta, turnaj), ktoré využíva trieda Evolution.

**GymMove** - Trieda reprezentujúca pohyb na účely evolúcie, obsahujúca napríklad hašovaciu tabuľku jednotlivých fáz pohybu a fitness pohybu. Pracuje s triedou GymLowSkill.
GymPhase - Trieda využívaná triedou GymMove na reprezentáciu jednotlivých fáz pohybu.
GymEffector - Trieda využívaná triedou GymPhase na reprezentáciu konkrétnych efektorov danej fázy pohybu.

**GymLowSkill** - Trieda reprezentujúca pohyb z pohľadu LowSkill-u, využívaná triedou GymMove.

**Reversion** - Trieda by mala slúžiť na prevrátenie pohybu. Napríklad kop ľavou nohou by mal byť prevrátený na kop pravou nohou. Volaná je priamo z triedy AutoTest (grafické rozhranie nástroja) tlačidlom Revert move a pracuje s pohybom, ktorý je zvolený pomocou tlačidla Choose move. Výstupom (v prípade úspechu) je XML pohyb, uložený v priečinku /trener_moves/.

**Common** - Trieda obsahujúca bežné metódy využívané viacerými implementovanými triedami. Ponúka napríklad rôzne generátory náhodných čísel alebo generátor názvu súboru.

**Výsledky optimalizácie pohybov**

Deklarované výsledky práce som overoval pomocou spomínaného nástroja na automatické testovanie pohybov. Pôvodný vylepšovaný kop do lopty dosahoval v priemere vzdialenosť 4,4 m a rozptyl 0,41 m. Autor práce pri vylepšovaní pomocou genetického algoritmu prezentoval 5 nových kopov do lopty, s priemernou vzdialenosťou kopu od 5 do takmer 10 metrov. Testovanie najlepších z nich však bohužiaľ neprinieslo očakávané a deklarované výsledky. Najlepšie pohyby dosahoval priemerné výsledky okolo 5-7 metrov. Domnievam sa, že problém môže byť v umiestnení hráča pred loptou, no pri teste sme použili rovnaké súradnice ako autor (hráč [-0.05:-0.25:0], lopta [0:0:0.4]). Hráč si totiž pri úvodnom úkroku k lopte takmer vždy jemne posunie loptu. Pri miernom posunutí hráča o približne 10 cm som však raz dosiahol hodnotu kopu vyše 9 metrov, no tento pokus sa mi nepodarilo zopakovať.

Ostatné optimalizované pohyby chôdze vpred, vzad a do strany dosahovali deklarované výsledky:

- chôdza vpred
    - turbo_walk(20130414_054044_1634)
    - priemerná rýchlosť: 0,68 m/s
    - priemerné vychýlenie: 3,88 °
    - úspešnosť: 100 %
- turbo_walk(20130506_092638_4347)
    - priemerná rýchlosť: 0,97 m/s
    - priemerné vychýlenie: 8,32 °
    - úspešnosť: 80 %

Najrýchlejšiu chôdzu som sa pokúsil použiť v aktuálne používanom pláne “PlanTactic.rb”. Vo viacerých prípadoch však agent spadol už pri začiatku chôdze, a pri zastavení spadol vždy (testovanie a vyvíjanie chôdze bolo totiž vykonávané bez zastavenia agenta do stabilizovanej polohy).

- chôdza vzad
    - walk_back2(20130417_065658_6195)
    - priemerná rýchlosť: 0,54 m/s
    - priemerné vychýlenie: 7,99 °
    - úspešnosť: 92 %
- úkrok do strany
    - stepleft_new_smaller(20130417_184700_2011)
        - priemerná rýchlosť: 0,09 m/s
        - priemerné vychýlenie: 4,92 °
        - úspešnosť: 100 %

- stepleft_new_smaller(20130509_080409_4791)
  - priemerná rýchlosť: 0,16 m/s
  - primerné vychýlenie: 14,11 °
  - úspešnosť: 93 %

**Zhodnotenie**

Autorovi sa úspešne podarilo výrazne vylepšiť každý z uvedených pohybov. Mrzí ma však skutočnosť, že sa mi nepodarilo nasimulovať deklarované vzdialenosti kopov do lopty. Verím však, že problém je v iba v správnom umiestnení hráča k lopte.

Implementovaný testovač nie je dokonalý, no umožňuje automatické testovanie zvoleného pohybu, čo je užitočné pre každého, kto sa na našej univerzite stretne s projektom RoboCup. Preto považujem za užitočné testovač zakomponovať do aktuálne používaného TestFramework-u.

Zakomponovanie evolučného algoritmu, ktorý je súčasťou testovača, je rovnako žiaduce. Poskytuje základ pre vylepšovanie existujúcich pohybov, no pred jeho použitím na neznámy pohyb sa pravdepodobne nevyhneme miernej úprave, resp. rozšíreniu, existujúcej implementácie. Toto vyplýva zo samotnej charakteristiky genetického algoritmu, ktorý je pre dosiahnutie čo najlepších výsledkov nutné prispôsobiť každej, čo i len mierne odlišnej doméne problému. Úzkym hrdlom pri vyvíjaní populácie však nie je evolučný algoritmus ako taký, ale výpočet fitness hodnoty každého vyvinutého pohybu, ktorý je závislí od testu pohybu v simulačnom prostredí, ktorý je značne časovo náročný, a stráca sa tak hlavný zmysel algoritmu, ktorý by mal v krátkom čase prehľadať a vyhodnotiť viacero možností doménového priestoru.

Ďalším veľkým problémom testovania (vyhodnocovania) či už zvoleného, alebo vyvinutého pohybu, je skutočnosť, že simulačné prostredie (server) je potrebné po uplynutí hracieho času potrebné manuálne reštartovať (v linuxe je zrejme možné tento problém riešiť skriptom, no neviem či aj na ostatných platformách).

Spomínaná implementácia prevrátenia pohybu pomocou triedy *Reversion*, nám bohužiaľ tiež nefungovala. Pri zvolení kopu ľavou nohou sa síce vytvoril z časti prevrátený pohyb, no nie funkčný. Pri pokuse o prevrátenie pohybu úkrok do strany program vypísal chybu a pohyb sa nevytvoril.

## Bc. Martin Paššák - Interakcia medzi hráčom a loptou v 3D simulovanom robotickom

Autor sa vo svojej práci zameral na vylepšenie práce hráča s loptou a to konkrétne na časti

**Kopy**

### 1. Kopy do lopty

Kop je realizovaný vnútornou bočnou hranou chodidla takže hráč má väčšiu šancu zasiahnuť loptu, avšak je citlivejší na vzdialenosť od lopty.

Testovanie odhalilo nedostatky ako padanie hráča pri vykonávaní krátkych kopov a relatívne nízku úspešnosť kopov.

| Názov kopu | test_right_kick / test_left_kick | test_right_kick_25 / test_left_kick_25 | test_right_kick_45 / test_left_kick_45 |
|---|---|---|---|
| Najdlhší kop v metroch | 7,52 | 6,77 | 6,27 |
| Priemerná dĺžka kopu v metroch | 6,83 | 6,03 | 5,41 |
| Úspešnosť | 60% | 50% | 45% |
| Priemerná uhlová odchýlka v stupňoch | 2,33% | 4.3% | 3,2% |

**Tabuľka 3.1** Vyhodnotenie testovania kopov

### 2. Prihrávky

Neboli vyhodnotené keďže boli veľmi nepresné a ich rozptyl bol veľmi vysoký. Jedná sa o náročnú optimalizačnú úlohu.

### 3. Parametrické kopy

Ide o kopy ktoré je možné na základe parametra nastaviť tak, aby hráč cielene s rovnakým pohybom kopol do vzdialenosti napríklad na 3m ako aj na 6m.

Vylepšené kopy dosahujú väčšiu presnosť ako tie pôvodné, čo je spôsobené optimalizáciou polohy lopty pri kopnutí.

| Názov kopu | test_right_kick / test_left_kick | test_right_kick_25 / test_left_kick_25 | test_right_kick_45 / test_left_kick_45 |
|---|---|---|---|
| Najdlhší kop v metroch | 7,44 | 7,3 | 6,2 |
| Priemerná dĺžka kopu v metroch | 6,44 | 6,1 | 5,16 |
| Úspešnosť | 80% | 90% | 70% |
| Priemerná uhlová odchýlka v stupňoch | 6,2% | 3,4% | 9,3% |

**Tabuľka 3.2** Vyhodnotenie testovania parametrických kopov

### 4. Optimalizácia pozície lopty pri kopnutí

Úspešnosť kopov z optimalizovanej pozície výrazne stúpla. Neúspech bol spôsobený padnutím hráča z nejakých príčin.

### 5. Výber správnej nohy pri kopnutí

Pre správne smerovanie strely je dôležité aby hráč kopal nohou, ktorá je opačná k pozícií cieľa kam sa kope. Hráč sa zakaždým rozhodol správne.

### 6. Vyhodnotenie času kopov

Študent vykonal časové vyhodnotenie kopov, ktoré porovnal s pôvodným stavom, z ktorého vychádzal. Ide o zmeranie času, ktorý hráč potrebuje na príchod k lopte a kopnutie do nej. Vykonalo sa 5 pokusov kde mal hráč loptu 2 metre priamo pred sebou a 5 pokusov kde mal loptu 2 metre pred sebou a 2 metre na ľavo.

| | | | | |
|---|---|---|---|---|
| 3. | 50 s | 12,1 s | 97 s | 28,9 s |
| 4. | 68 s | 12,3 s | 90 s | 27,2 s |
| 5. | 54 s | 13,5 s | 75 s | 21,4 s |
| Priemer | **57,8 s** | **13,86 s** | **82 s** | **25 s** |
| Zlepšenie o | **75%** | | **71.46%** | |

**Tabuľka 3.3** Časové porovnanie kopov

## Chôdza s loptou

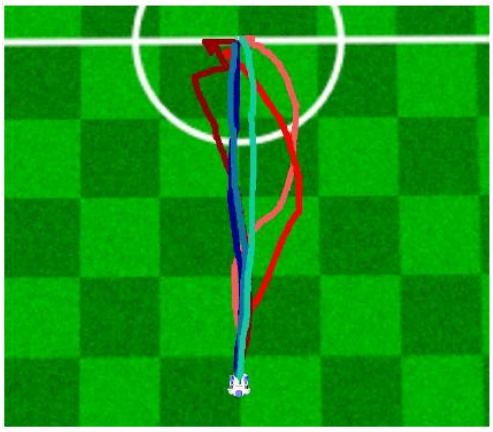
### 1.Úprava nepresností chôdze

Riešenie tohto problému pozostáva z vylepšenia korigovania vychýlenia. Keď hráč následkom zavedeného šumu serveru vykoná kratší, alebo dlhší krok a vychýli sa zo smeru, okamžite vykoná pohyb, aby sa vrátil do pôvodného smeru a opäť smeroval na cieľ.

Korekcia vychýlenia, ako prepínanie medzi kratším a dlhším krokom, bola implementovaná na úrovni pohybu keďže implementácia v podobe skriptov bola pomalá.

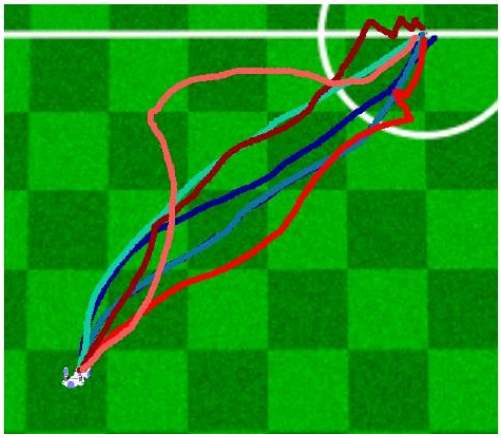
Z výsledkov je viditeľné zlepšenie v čase potrebnom na presun na danú pozíciu o približne 30% v niektorých prípadoch aj o vyše 50%. Pri presunoch, ktoré neboli zamerané priamo na loptu dochádzalo k výraznému zdržaniu pri vykonávaní posledného približovania sa na stanovenú pozíciu, takže tieto časy by sa dali ešte vylepšiť.

Na obrázku 3.3 sú modrou farbou trajektórie nových pohybov a červenou pôvodných pri priamom presune.



**Obrázok 3.3** Porovnanie priameho pohybu

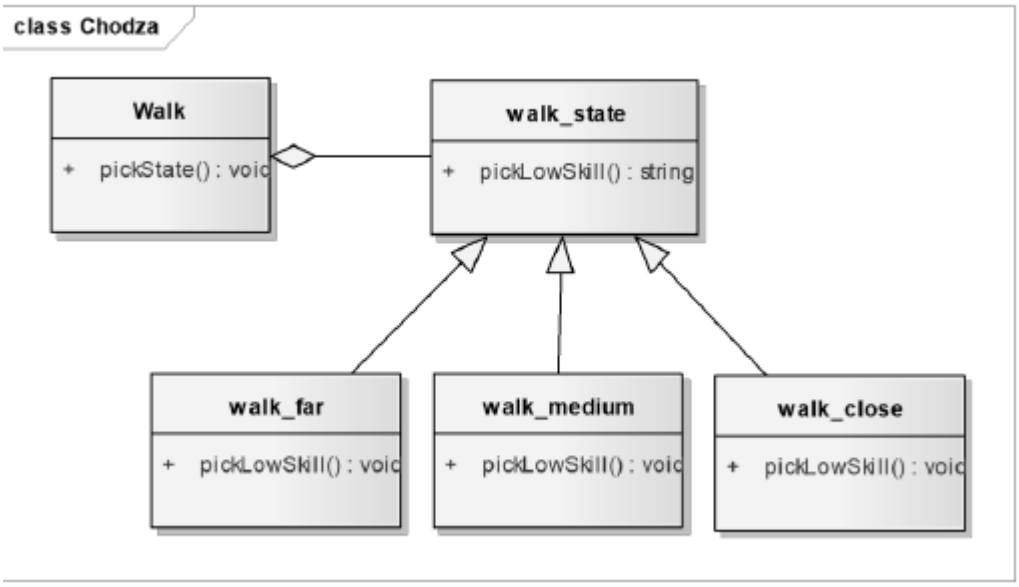
Na obrázku 3.4 sú modrou farbou trajektórie nových pohybov a červenou pôvodných pri diagonálnom presune.



**Obrázok 3.4** Porovnanie pohybu po diagonále

## 2. Úprava vyššej logiky chôdze

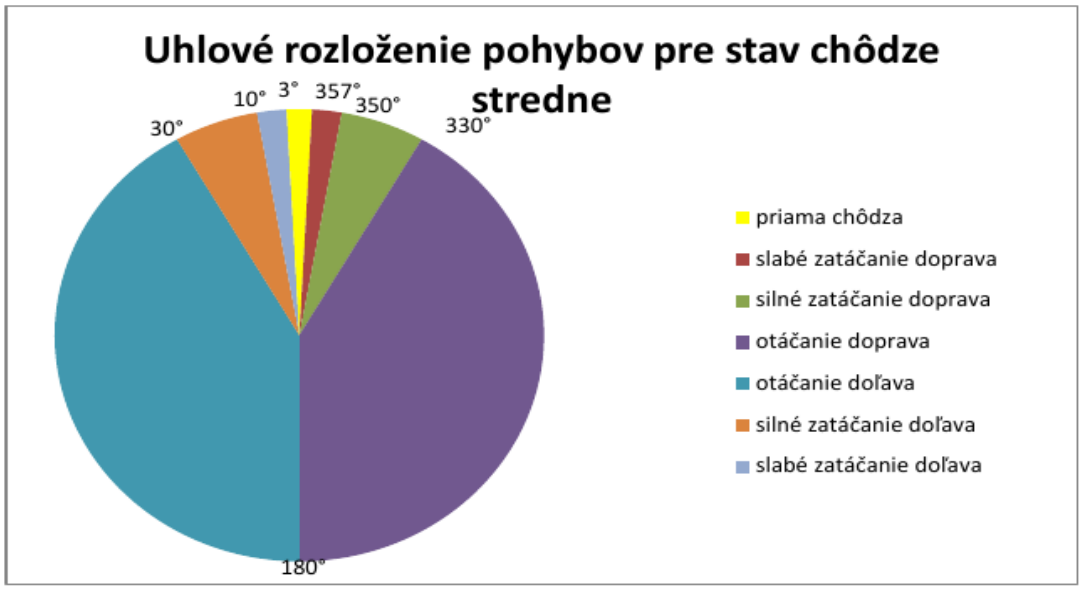
Zaviedol návrhový vzor State keďže pôvodný spôsob bol neprehľadný. Nový rozdelil logiku do viacerých súborov namiesto jedného veľkého skriptu.



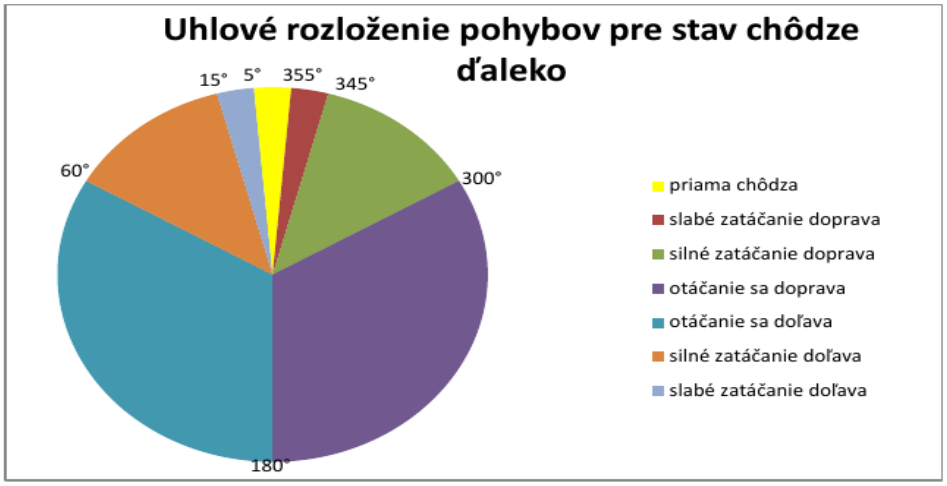

**Obrázok 3.5** State vzor pre chôdzu

### 3. Úprava približovania sa k lopte

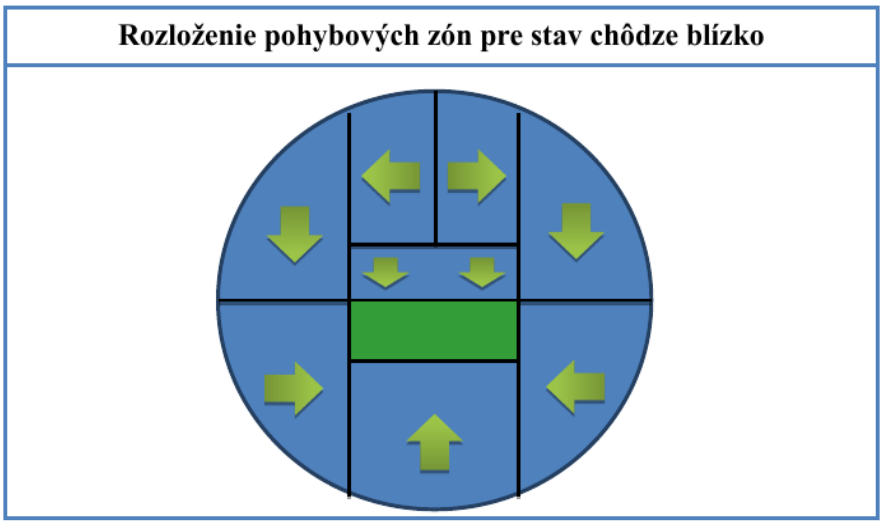
Na obrázkoch 3.6 a 3.7 sú uhlové rozloženia pre chôdzu podľa pôvodnej implementácie chôdze.




**Obrázok 3.6** Uhlové rozloženie pohybov pre strednú chôdzu




**Obrázok 3.7** Uhlové rozloženie pohybov pre vzdialenú chôdzu.

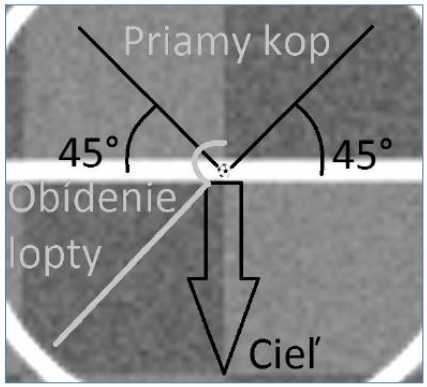



**Obrázok 3.8** Rozloženie pohybových zón pre blízku chôdzu

Hráčovi študent definoval relatívnu vzdialenosť od lopty pomocou súradníc x a y namiesto uhlov a vzdialenosti. Boli vytvorené zóny, obrázok 3.8, a podľa toho, v ktorej zóne sa lopta nachádza, hráč vykonáva príslušné pohyby. V každej zóne je šípkou vyznačený smer pohybu, ktorý sa bude v danej zóne vykonávať. Zeleným štvorcom je vyznačená zóna do ktorej sa má hráč dostať, aby mohol strieľať.

## 4. Obchádzanie lopty

Hráč sa v každej situácii snaží dostať priamo k lopte. Keď sa k nej dostane na požadovanú vzdialenosť, vyhodnotí či je správne otočený na cieľ. Ak nie, vykoná otáčanie okolo bodu s polomerom 30 cm, ktoré zaručí že sa otočí okolo lopty bez toho, aby sa od nej výraznejšie vzdialil alebo aby loptu zasiahol chodidlom. Lopta sa nachádza v strede polomeru otáčania. Tým sa zabezpečuje, že nenastane situácia, kedy hráč zbytočne vykonáva kontraproduktívne pohyby, pretože nesprávne vyhodnocuje pozíciu lopty a bude mať loptu stále pod dohľadom. Tento mechanizmus je znázornený na obrázku 3.9.




**Obrázok 3.9** Obchádzanie lopty

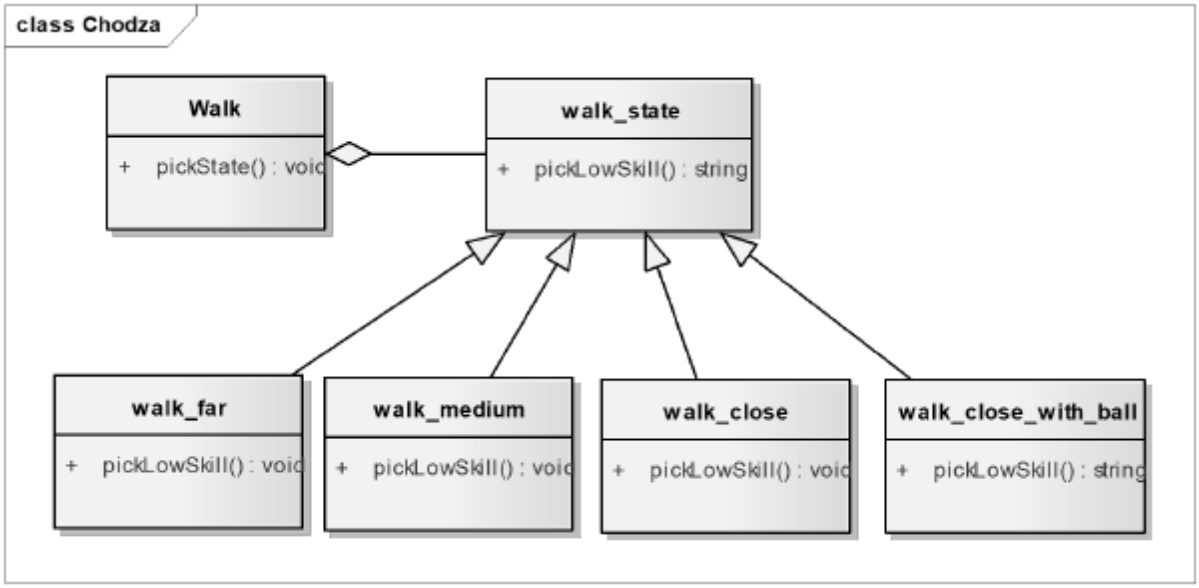
### 5. Odstránenie pádov hráča

Postupné zrýchľovanie a spomaľovanie odstránilo pády hráča.

### 6. Chôdza s loptou

Chôdza s loptou bola implementovaná ako nový vyšší high skill. Diagram logiky je na obrázku 3.10.

Z vyhodnotenia testov priameho vedenia lopty vyplynulo niekoľko záverov. Hráč vie viesť loptu, nedokáže však dostatočne presne udržiavať trajektóriu chôdze a tým pádom zasahovať do lopty správnou časťou chodidla a mať tak loptu pod kontrolou a z tohto dôvodu pomerne často dochádza k strate lopty. Hráč je však schopný znovu si loptu prebrať a pokračovať s ňou požadovaným smerom.



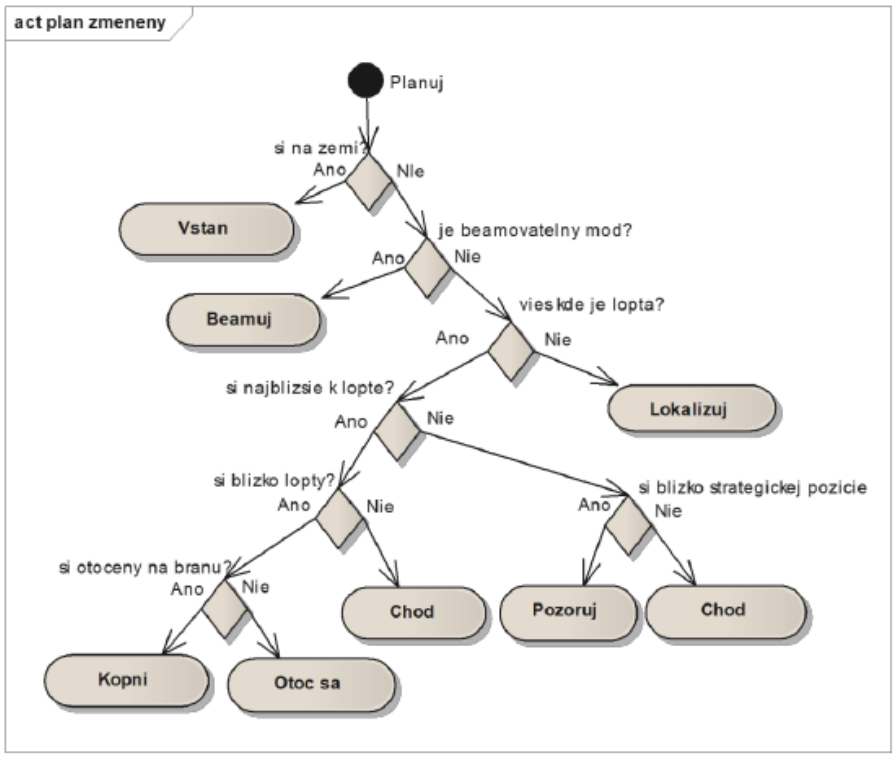

**Obrázok 3.10** Diagram chôdze

## Automatické testovanie

Na vyhodnotenie práce študent vytvoril jednoduchý nástroj na automatizované testovanie, ktorý umožňuje vytvoriť test pre hráča, tj. vytvoriť hernú situáciu, spustiť hráča a po ukončení požadovanej akcie celý proces opakovať. Testovací framework poskytuje aj rôzne možnosti pre manipuláciu hráča ako napríklad podsúvanie vnemov o tom aká prekážka je pri hráčovi alebo posielanie správ.

## Úprava plánu hráča

Keďže pôvodný plán bol neprehľadný a rovnaké podmienky sa vyhodnocovali opakovane, takže pri úpravách by sa mohli zabudúť pokryť všetky možnosti, vytvoril študent nový plánovač na princípe rozhodovacieho stromu, kde akcie sú listy a jednoznačne sú určené na základe podmienok na obrázku 3.11.




**Obrázok 3.11** Strom rozhodovania pre plánovač

## Zhodnotenie

Diplomová práca výrazne zlepšila existujúce pohyby na úrovni či už samotných pohybov na tak aj high skillov a prináša nové prístupy k práci hráča s loptou. Testovací framework stojí za pozornosť avšak je lepšie vychádzať z verzie Paššáka P.

## 3.2 Analýza zahraničných tímov

Cieľom úlohy analýza zahraničných tímov je analýza existujúcich riešení vyvíjaných tímami, ktoré sa zúčastnili na tohtoročnom svetovom turnaji v oblasti Robocup3D simulačného futbalu. Výsledky analýzy by mali byť využité pri vlastnom návrhu architektúry strategickej a taktickej vrstvy, a architektúry highskillov.

## Analýza tímu UT Austin Villa

### Úvod

Tím UT Austin Villa pochádza z Texaskej univerzity v Austine. Patrí medzi najlepšie tímy zo svetovej špičky robotického simulačného futbalu. Posledné dva šampionáty v rokoch 2011 a 2012 sa umiestnili na prvom mieste. Vďaka týmto úspechom je vhodné preštudovať ich prístupy a techniky a zanalyzovať ich možný prínos pri vylepšovaní nášho riešenia.

### Implementované vylepšenia

V tejto kapitole sa zaoberáme prístupmi a technikami, ktoré autori uvádzajú ako podstatné zmeny v implementácii, ktoré viedli k víťazstvu v šampionátoch v rokoch 2011 a 2012.

#### Detekcia pádu a optimalizácia High skillu vstavania zo zeme

Pri tejto oblasti sa autori rozhodli detegovať zmenu stavu pri páde robota, kedy akcelometre indikujú, že gravitačná sila „ťahá“ robota kolmo na vektor určujúci smer jeho trupu. Ak robot zaznamená stav, kedy padá, tak v tomto prípade robot rozpaží ruky do priameho uhla. Touto akciou autori docielili, že robot pri páde pristane buď na chrbát alebo trup. Následne sa robot rozhodne, na ktorú stranu dopadol a podľa pozície na ktorej leží vyberá medzi dvoma vyššími činnosťami (High Skilly) stávania zo zeme.

Následne parametre potrebné na zefektívnenie vstávania zo zeme autori optimalizovali použitím algoritmu Covariance Matrix Adaptation Evolution Strategy. Výber tohto algoritmu autori podmienili jeho úspešnosťou pri optimalizácii viacerých druhov vyšších činností, napríklad parametrického kopu.

V nasledujúcej časti autori popísali, akým spôsobom vybraný optimalizačný algoritmus uplatnili pri trénovaní akcie vstávania zo zeme.

Výsledné hodnoty optimalizácie pre vstávanie z polohy v ľahu na chrbte alebo trupe získali z celkového počtu 200 generácii o populácii vo veľkosti 150 jedincov. Po vykonaní optimalizácie zredukovali výsledný čas vstávania na $\frac{1}{3}$ pôvodnej hodnoty a to na hodnotu 0,96 sekundy pre vstávanie z trupu a 0,84 sekundy pre vstávanie z chrbta.

**Kopanie**

V nasledujúcej časti článku autori popisovali 4 druhy kopov, ktoré rozdelili do dvoch skupín podľa pozície pri kope.

Prvú skupinu nazvali **Fixed Pose Keyframe Kicks**, kedy agent pred kopom položí podpornú (stabilnú) nohu blízko lopty a následne prenesie všetku svoju váhu na túto nohu. Následne zdvihne nohu ktorou chce kopnúť a posunie ju dozadu v rovine za loptu. Nakoniec robot švihne nohou dopredu smerom k lopte. Do tejto skupiny autori priradili 3 kopy a to kop do diaľky, kop na strednú vzdialenosť a kop na blízko. Nevýhodou týchto kopov je potreba veľmi presnej pozície pri lopte pre najefektívnejší výsledok.

Nasledujúcu skupinu nazvali **Inverse Kinematics Based Kicks**. DO tejto skupiny zaradili jeden kop , kedy robot

## Zdroje


UT Austin Villa: RoboCup 2012 3D Simulation League Champion.
Patrick MacAlpine, Nick Collins, Adrian Lopez-Mobilia, and Peter Stone.
In RoboCup-2012: Robot Soccer World Cup XVI, Lecture Notes in Artificial Intelligence, Springer Verlag, Berlin, 2013.

Design and Optimization of an Omnidirectional Humanoid Walk: A Winning Approach at the RoboCup 2011 3D Simulation Competition.
Patrick MacAlpine, Samuel Barrett, Daniel Urieli, Victor Vu, Peter Stone
In the Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12) in Toronto, Ontario, Canada, July 2012.

Positioning to Win: A Dynamic Role Assignment and Formation Positioning System.
Patrick MacAlpine, Francisco Barrera, and Peter Stone.
In Proceedings of the RoboCup International Symposium (RoboCup 2012) in Mexico City, Mexico, June 2012.

## Apollo3D Team

### Úvod

Tím Apollo3D pochádza z Nanjing University of Posts and Telecommunications. Tím bol založený v roku 2006. V júli 2013 vyhral tento tím RoboCup robot soccer World Cup v Holandsku v meste Eindhoven. V nasledujúcej časti sa venujeme opisu častí riešení tímu Apollo3D, ktoré sú relevantné pre náš projekt.

### Systém komunikácie

V 3D simulácií robotického futbalu má každý z tímov k dispozícií 11 agentov, ktorý sú simulovaní prostredníctvom 11 paralelných procesov. Tieto procesy nemôžu medzi sebou priamo komunikovať. Iba jeden z agentov môže poslať správu raz za dva cykly pričom ostatní agenti ju príjmu nasledujúci cyklus. Veľkosť správy je limitovaná na 20 bajtov. Každý bajt môže kódovať jeden znak z ASCII pričom nie všetky znaky z ASCII môžu byt použité.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

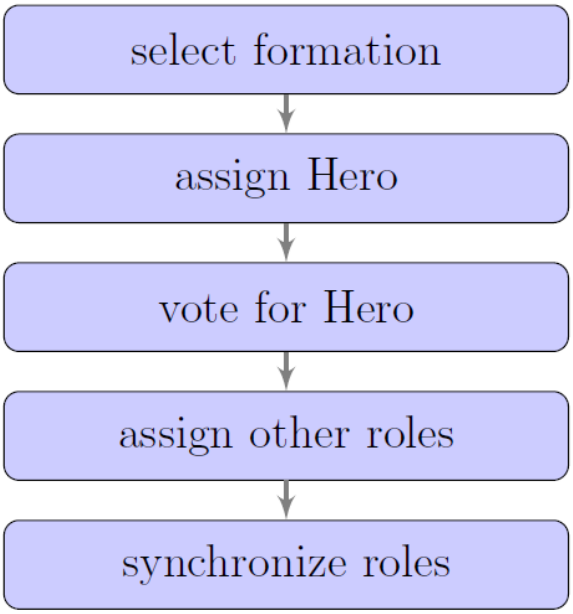
- 1 a 2 bajt reprezentujú identifikáciu tímu
- 3 bajt reprezentuje číslo hráča ktorý správu vytvoril
- 4 a 5 bajt informujú o stave hráča ktorý správu zasielal (hráč padol / hráč vidí loptu ...)
- 6 až 9 bajt informuje o pozícií lopty z pohľadu hráča tvoriaceho správu
- 10 až 13 bajt informuje o pozícií hráča z jeho vlastného pohľadu
- 14 až 18 bajt popisuje úlohy zvyšných členov tímu z pohľadu odosielajúceho hráča
- 19 a 20 bajt tvorí rezervu

Aby bolo možné polohy lopty a hráča zaznamenať do obmedzeného priestoru je celé hracie pole rozdelené na mriežku s rozmermi 5000 x 5000. Na kódovanie sa používajú všetky ASCII znaky ktoré sú pravidlami hry RoboCup povolené. Povolených znakov je 83.

Navrhnutý systém komunikácie sa zdá byť navrhovaný pre starú verziu hry, kde bolo len 6 hráčov v tíme. Pre súčasné pravidlá s 11 hrčami v jednom tíme je veľkosť poľa popisujúceho úlohy ostatných hráčov nedostačujúca.

### Taktika

Zvýšenie počtu hráčov zo 6 v roku 2010 na 9 v roku 2011 a následne na 11 v roku 2012 zvýšilo nároky na taktiku a lepšiu kooperáciu medzi jednotlivými agentmi.
Základom taktiky je hráč ktorý má loptu pod kontrolou pretože v súčasnosti je prihrávanie medzi hráčmi neefektívne pre väčšinu tímov. Pre hráča s držaním lopty je kľúčové, aby efektívne kontroloval túto loptu pri pohybe ihriskom.




**Obrázok 3.12** Diagram prideľovania úloh

Tím Apolo3D má vrstvovú architektúru prideľovania úloh. Na úplnom vrchole, ako je možné vidieť na obrázku 3.12, je výber formácie. Výber formácie prebieha na základe pozície lopty v priestore. V ďalšom kroku sa vyberie hráč, ktorý sa pokúsi získať nad loptou kontrolu.

Keďže pozorovacie schopnosti hráča sú limitované a majú chyby, môže nastať situácia, keď sa viacero agentov pokúsi získať loptu, čo by mohlo spôsobiť kolízie medzi hráčmi rovnakého tímu v dôsledku čoho by sa nemuselo podariť získať loptu, prípadne by mohol tím loptu stratiť z kontroly. Tento problém je riešený pomocou metódy selekcie najvhodnejšieho hráča. Každý agent podľa svojho videnia sveta určí najvhodnejšieho hráča a následne sa tento výber synchronizuje s využitím komunikácie. Keďže komunikačný systém má oneskorenia(obmedzenia komunikačného systému sú popísane v predošlej kapitole) a agent si nie je 100% istý jeho výberom, každý výber má k sebe priradenú pravdepodobnosť s hodnotou (0,1).

Pokiaľ je lopta v kontrole jedného z hráčov, ostatní mu asistujú v útoku. Pridelenie úloh zvyšným agentom sa určuje podľa ich aktuálnej pozície vo formácií. Tento postup sa tiež synchronizuje aby sa predišlo kolíziám.
Kritéria selekcie hráča, ktorý sa pokúsi získať loptu:

- Je hráč na zemi?
- Vidí hráč loptu?
- Aká je vzdialenosť hráča od lopty?
- Je hráč pred loptou alebo za loptou? (ak je pred loptou, potrebuje čas navyše aby sa mohol otočiť)
- Aj v predošlom cykle bol tento hráč určený na získanie lopty?
- Je hráč brankárom?

## Zhodnotenie

Z analýzy vyplynula potreba komunikácie medzi hráčmi ako základná podmienka úspešného postupu tímu vo formácií. Tiež bolo poukázané na limity komunikácie, ktoré sú stanovené pravidlami robotického futbalu.

## Zdroje

http://staff.science.uva.nl/~arnoud/activities/robocup/RoboCup2013/Symposium/TeamDescriptionPapers/SoccerSimulation/Soccer3D/Apollo3D_TDP.pdf
http://www.njupt.edu.cn/s/2/t/2/b3/e8/info46056.htm

## Analýza tímu Bahia3D

### Úvod

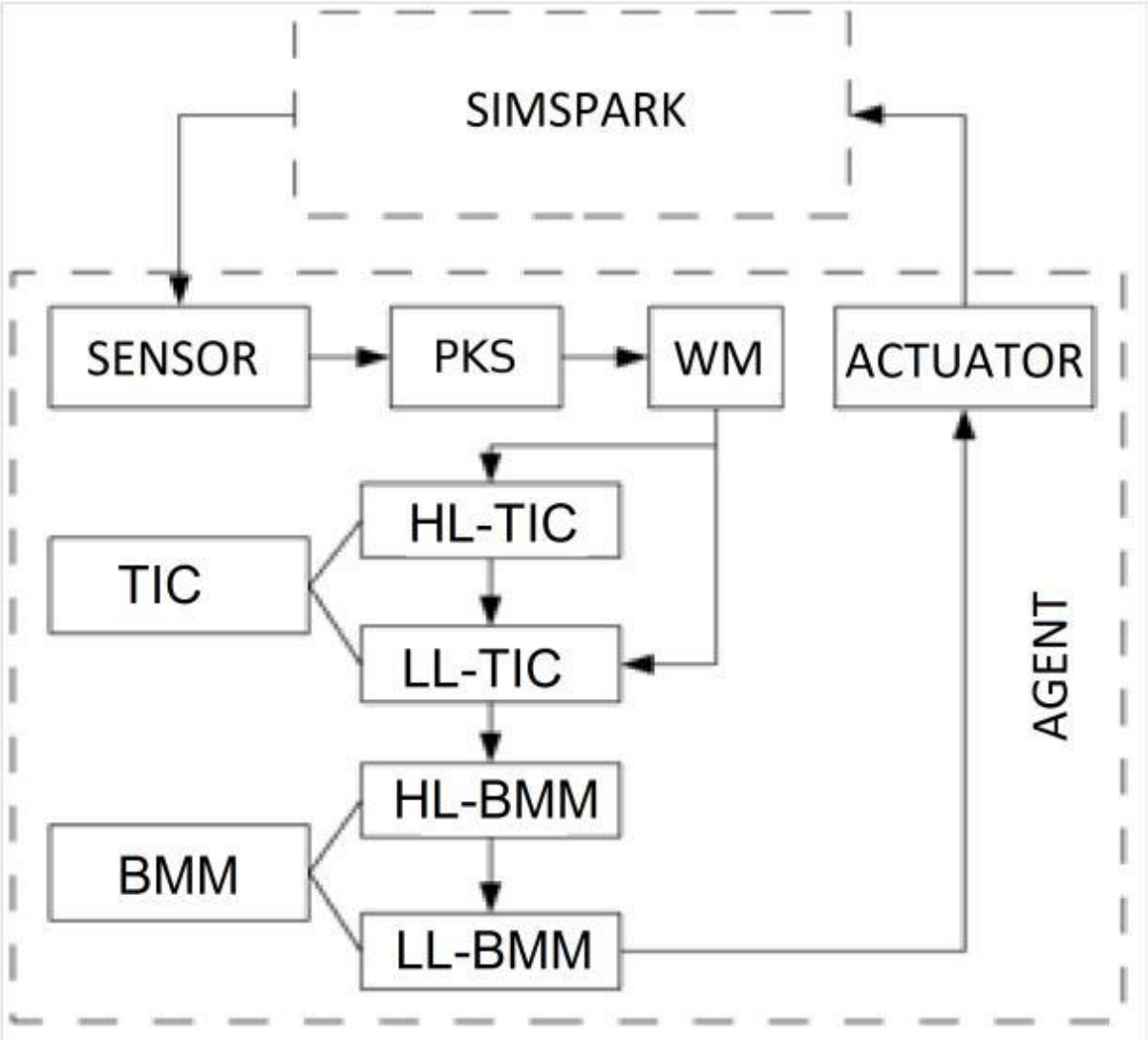
Bahia3D je tím z Brazílie, ktorý začal pracovať na projekte v roku 2008 a zúčastňuje sa na medzinárodných súťažiach v simulačnej lige RoboCup3D od roku 2009. Tím sa umiestnil na 2. mieste v rámci LARC 2010 (American Robotics Competition), kedy podľahol tímu Nexus_3D z Iránu a po komplexnej zmene návrhu architektúry agenta, ktorá prebehla v roku 2011 zvíťazil na LARC 2012. Súčasťou súťaže bola aj CBR 2012 (Brazilian Robotics Competition), ktorú tím vyhral v kategórií 3D robotického futbalu aj v nasledujúcom roku.

### Architektúra

Táto časť sa venuje implementovanej architektúre agenta. Na obrázku 3.13 je zobrazená architektúra agenta.

- Simsprark reprezentuje simulačný server
- Sensor poskytuje informácie o stave agenta zo serveru.
- Actuator posiela informácie o požadovaných zmenách stavu agenta na server.
- PKS (Perception Kinematic State) uchováva informácie o pohybovom stave agenta, lopty a zvyšných agentoch nachádzajúcich sa v hre. Obsahuje dve podmnožiny údajov. Zrakové vnemy a senzorickú inteligenciu, ktorá uchováva zvyšné typy vnemov a doplnkové informácie o samotnom agentovi.
- WM (World model) obsahuje nespracované dáta tak ako boli prijaté zo serveru a ich spracovanú a očistenú reprezentáciu, ktorá poskytuje vrstve rozhodovania širšiu významovú informáciu
- TIC (Tactical Intelligence Center) rozhoduje akú akciu má agent vykonať vzhľadom na jeho rolu v tíme a informácií o stave hry dostupných z WM. Implementované roly sú brankár, útočník a obranca. Pre každú rolu je implementované špecifické uvažovanie, ale obsahuje aj uvažovanie, ktoré je rovnaké pre každého agenta. Každý agent najprv určí svoju pozíciu a stav hry a následne vykoná otočenie hlavou, ktoré je nezávislé od vykonávania ostatných pohybov tak, aby mal loptu vždy v zornom poli.
  - HL-TIC (High Level TIC) na základe dostupných informácií analyzuje, čo urobiť.
  - LL-TIC (Low Level TIC) určuje akú sekvenciu pohybov treba vykonať, aby sa dosiahol požadovaný stav. Parametrami, na základe ktorých je vykonané rozhodovanie, sú presnosť a čas.
- BMM (Body Movement Manager) je zodpovedný za vykonávanie pohybov požadovaných vrstvou TIC.
  - HL-BMM (High Level BMM) predstavuje proces vykonávajúci kombináciu niekoľkých LL-BMM za cieľom dosiahnuť stav definovaný vrstvou LL-TIC. V rámci procesu je možné vykonávaný pohyb zastaviť, či už z dôvodu pádu agenta alebo ak bola zmenená situácia na ihrisku.
  - LL-BMM (Low Level BMM) predstavuje nízkoúrovňové pohyby podobné low skillom v súčasnej verzií projektu robotického futbalu na fakulte.

Architektúra je navrhnutá s dôrazom na modulárnosť a umožňuje pridanie strategického modulu nad taktickú vrstvu rozhodovania, s ktorou sa v budúcnosti počíta.




**Obrázok 3.13** Bahia3D architektúra agenta

Vzhľadom na náš projekt sú zaujímavé vrstvy TIC a BMM prípadne pridanie ďalšej vrstvy strategického rozhodovania nad TIC. Toto riešenie môže slúžiť ako inšpirácia na zmenu architektúry rozhodovania agenta s pridaním taktickej a strategickej vrstvy do nášho projektu.

**Implementované vylepšenia**

**Komunikácia medzi agentmi**

Tím Bahia3D navrhol komunikáciu medzi spoluhráčmi tak, že brankár, ktorý má takmer po celý čas prehľad o kompletnom dianí na ihrisku ohlasuje zvyšným spoluhráčom aktuálnu pozíciu lopty. Toto riešenie bolo zvolené na základe predošlého neúspechu, kedy sa každý hráč pokúšal o komunikáciu no ostatní zachytili len prvú prijatú správu od jedného hráča a ostatné správy sa stratili. Ponúkané riešenie by bolo nutné adaptovať v prípade zmeny rozmerov ihriska natoľko, že brankár už nebude schopný vizuálne pokryť celú situáciu.

**Detekcia pádu spoluhráča**

Pokiaľ chce hráč s loptou prihrať svojmu spoluhráčovi, je vhodné, aby tento spoluhráč vedel prihrávku aj spracovať. Toto ale nie je možné v prípade, že spoluhráč padol a pokúša sa vstať. Riešenie tímu Bahia3D je vizuálne určenie stavu spoluhráča na základe výšky jeho hlavy podľa osy *z*. Tu je potrebné nastaviť parameter, podľa ktorého je možné rozhodnúť že agent stojí alebo je padnutý na zemi.

## Zdroje

SANTOS, Alan, et al. Bahia3D-A Team of 3D Simulation for Robocup; v Eindenhovene, Holandsko;http://staff.science.uva.nl/~arnoud/activities/robocup/RoboCup2013/Symposium/TeamDescriptionPapers/SoccerSimulation/Soccer3D/Bahia3D_TDP.pdf

## RoboCanes

### Úvod

Tím RoboCanes je americký tím z University of Miami. Založený bol v januári 2010, no na jeseň v roku 2010 začali takpovediac “od nuly” kompletne prerábať zdrojové kódy agenta. Vedúcí tímu má však skúsenosti s RoboCup-om už od roku 2000 [1]. Jeden z členov tímu publikoval prácu s názvom "Entropy-based active vision for a humanoid soccer robot" [2], ktorá v roku 2010 vyhrala ocenenie najlepšej publikácie RoboCup-u 2010. Zručnosti toho istého člena následne viedli k vytvoreniu nového 3D monitora simulačného prostredia, ktorý si predstavíme bližšie.

### RoboViz

Tento nástroj nahrádza a vylepšuje aktuálny monitor simulačného prostredia, dodávaný spolu so SimSpark-om. Nástroj neponúka iba zlepšenie vizuálnej stránky, ktorá je viditeľne pokrokovejšia ako v prípade pôvodného riešenia (obrázok 3.14), ale ponúka rôzne ďalšie vylepšenia. Jedným z nich je používateľsky priateľskejšia prácu so základnými funkciami monitora alebo zobrazenie ďalších dostupných informácii o stave hry a hráčov, ktoré v pôvodnom monitore nie sú. V neposlednom rade poskytuje rozhranie pre vizualizáciu vnímania a stavu agenta (obrázok 3.15).

**Obrázok 3.14** Roboviz (v ľavo) a SimSpark Monitor (v pravo) [3]

Spomenieme teraz niektoré z vylepšených funkcií:

- možnosť zapnúť zobrazovanie čísel hráčov na ich hlavami,
- jednoduchšie pohybovanie kamery po ihrisku
  - možnosť približovať kameru kolieskom myši
- možnosť označiť hráčov a loptu a umiestniť ich na ľubovoľné miesto (škoda však že nie je možné zadať presné súradnice miesta, poloha je závislá iba od kliknutia myši)
- možnosť vizuálne prepínať medzí dostupnými módmi hry (v pôvodnom monitore je potrebné si pamätať klávesy jednotlivých pohybov)

Pre využitie vizualizácie vnímania a stavu agenta je potrebné rozšíriť implementáciu agenta. Návod je dostupný na stránke projektu [3].


**Obrázok 3.15** Nový 3D monitor simulačného prostredia

## Zdroje

[1] Saminda Abeyruwan, Alexander Härtl, Piyali Nath, Andreas Seekircher, Justin Stoecker and Ubbo Visser: "RoboCanes RoboCup 3D Simulation League Team Description Paper." *Department of Computer Science University of Miami*, 2013.

[2] Seekircher, Andreas, Tim Laue, and Thomas Röfer. "Entropy-based active vision for a humanoid soccer robot." *RoboCup 2010: Robot Soccer World Cup XIV*. Springer Berlin Heidelberg, 2011. 1-12.

[3] Stoecker J, Visser U: RoboViz: Programmable Visualization for Simulated Soccer. In *RoboCup 2011: Robot Soccer World Cup XV*. edited by Röfer T, Mayer NM, Savage J, Saranli U Springer Berlin / Heidelberg; 2012:282–293

## B-Human

### Úvod

B-Human je v súčasnosti jeden z najlepších tímov v RoboCup Standard Platform League. Sú 4-násoný víťazi svetového šampionátu a 5-násobný majstri RoboCup German Open. B-Human vnikol ako univerzitký projekt na Bremenskej Univerzite. Tvoria ho študenti 2-heho a vyššieho stupňa. Tím sa hlacvne venuje RoboCupu s reálnymi robotmi, ale nakoľko simulovaný RoboCup obsahuje model robota Nao ktorý sa používa aj pri klasickom RoboCupe, môžu byť niektoré ich objavy a techiky prínosom pre náš tím. Tím síce neuverejňuje svoje zdrojové kódy, ale uverejňuje svoje objavy a publikácie.

### Analýza zápasu

Podľa analýzy zápasu, roboti neustále točia hlavou a získavaju tak komplexnejšie informácie o scéne. Tieto informácie využívajú na tvorbu stratégie v reálnom čase. Počas zápasu sa viac krát stalo, že robot bežiaci k lopte spadol. Najbližší robot to videl a rozbehol sa za loptou on. Keď prvý robot vstal, skontroloval situáciu a ak bol stále bližšie k lopte, pokračoval v presune.

Roboti sa strategicky rozmiestňujú po ihrisku. Keď vidia hráča blízko lopty, vzdialia sa tak, aby mohli prijať prihrávku. Teda analyzujú priestor a zvolia najlepšie umiestnenie v danej situácii, aby zároveň priamo videli na prihrávajúceho hráča a na bránu alebo iného hráča bližšie k bráne.

Tím využíva bočný kop do lopty. Ak sú hráč a protihráč pri lopte blízko oproti sebe, hráč posunie nohu v pred a do strany od seba a následne odkopne loptu do druhej strany.

Takmer pri každom pohybe roboti využívajú stabilizáciu rukami a telom. Napr. pri ďalekom kope ľavou nohou keď sa pripravuje robot na výkop, posúva pravú ruku vzad, ľavú ruku vpred a telom sa nahne do pravej strany. Pri výkope trhne pravou rukou vpred. Po výkope nohu vráti na miesto a čaká kým sa vráti stabilne na obe nohy. Až potom vráti na miesto aj ľavú ruku.

### Riešenia

Tím najnovšie pracoval na parametrizovanom kope do lopty a následnom vyvážení robota pomocou inverznej kinetiky. Počítajú zero-moment-point (ZMP), ku ktorému posunuli telo robota metódami Linear Quadratic Regulator, local linearization a Cart-Table Preview Controller aby vyvážili robota počas kopu.

Trajektóriu nohy k lopte počítajú cez 6 kontrolných bodov, ktoré si vygenerujú na základe polohy a polomeru lopty, dĺžky trvania pohybu nohy a vzdialenosti. Nad týmito bodmi vygenerujú B-Spline krivku, ktorá popisuje dráhu nohy.

## Zdroje

[1] http://www.b-human.de/wp-content/uploads/2013/07/F.Wenk-Paper.pdf
[2] http://www.youtube.com/watch?v=ClQhOrDHJIo
[3] http://www.b-human.de/

## 3.3 Automaticky build system – CI

**Zadanie úlohy**

Zadaním tejto úlohy bola analýza existujúcich nástrojov použiteľných pre potreby kontinuálnej integrácie a následná integrácia projektu s týmto nástrojom.

Bližšou špecifikáciou bolo vytvorenie denného kompilovania projektového zostavenia, vykonanie jednotkových testov(unit test) a následne oznámenie výsledkov.

**Postup riešenia**

Pri riešení sme analyzovali dva nástroje a to konkrétne Jenkins a Atlassian Bamboo.

Prvou alternatívou, ktorú sme skúšali bol nástroj Jenkins vďaka skúsenostiam riešiteľa s týmto nástrojom. Výsledok integrácie bol neúspešný kvôli problémom spôsobeným pri práci s projektovým repozitárom a to konkrétne pri checkoute repozitára.

Ďalšou skúmanou alternatívou bol nástroj Atlassian Bamboo. Tento nástroj sme si vybrali kvôli už vybraným riešeniam projektového repozitára (Atlassian Bitbucket) a nástroja na podporu riadenia projektov (Atlassian Jira) a to kvôli jednoduchšej integrácii medzi nástrojmi a lepšej prehľadnosti.

Celý proces testovania je tvorený dvomi etapami. Prvá etapa je *Build project*, kedy je vykonaný checkout projektového repozitára a následne je skompilovaný hlavný projekt Jim pomocou Apache Ant, čo je nástroj na automatické zostavovanie softwarových aplikácií.
Druhou fázou je *Unit test*, kedy je využitý zostavený projekt z predchádzajúcej etapy na vykonanie základných jednotkových testov spúšťaných taktiež pomocou Apache Ant.

Následne sú zadefinované vetvy, pre ktoré sa tieto skupiny testov budú vykonávať. Primárne bude vždy zadefinovaná jedna hlavná vetva (master), ktorej periodicita spúšťania je každých 12 hodín. Následne je zadefinovaná aspoň jedna vedľajšia vývojová vetva, ktorej periodicita spúšťania je každé 4 hodiny. Tieto časové intervaly boli zadané vo formáte CRON.
Pre každú pridanú vetvu je potrebné pri prvom spustení nastaviť čas spúšťania pravidelného zostavovania.

**Zdroje**

Atlassian Bamboo:
https://confluence.atlassian.com/display/BAMBOO/Bamboo+Documentation+Home

## 3.4 Transformácia high skillov do Javy

Cieľom úlohy transformácia high skillov z ruby do Javy je umožniť spustenie agenta bez potreby načítavania ruby scriptov, v ktorých sú pôvodné high skilly naprogramované.

V rámci šprintu bola realizovaná reimplementácia high skillov z jazyka ruby do jazyka Java. Vytvorené triedy high skillov sa nachádzajú v balíku „*highSkills*“ pričom názvy sú rovnaké ako pred reimplementáciou s tým rozdielom, že názov triedy sa začína veľkým písmenom.

Pri reimplementácií sa vyskytol problém s anonymnými funkciami (výrazy lambda), ktoré sa nachádzali v niekoľkých high skilloch, ale jazyk java tieto konštrukcie zatiaľ nepodporuje. V skriptoch sa nachádzali mnohé premenné, ktoré sa nepoužívali, ako aj vstupné argumenty high skillu ako napríklad ciele pohybu, sa v skripte samotnom nebrali do úvahy.

Ďalej nebol vykonaný prepis troch high skillov (*walk2BallAccuracyTournament*, *kickTournament* a *walkBehindBallRelative*), ktoré sa v programe nevyužívali a predstavovali len ďalšiu verziu už prepísaného high skillu s drobnými odlišnosťami. Niekoľko high skillov bolo nefunkčných alebo len s čiastočne realizovanou funkcionalitou.

Spomenuté nedostatky nepredstavujú pre ďalšie napredovanie projektu významný problém, pretože v nasledujúcich šprintoch sa mení celá architektúra rozhodovania, s čím súvisí aj komplexné prispôsobenie alebo nová implementácia všetkých súčasných high skillov.

Nasledujúce high skills nie sú okrem jedného s názvom *low_skill* využívanie v žiadnom so súčasných plánov a pravdepodobne slúžili na testovanie rôznych konkrétnych low skillov.

## Niektoré reimplementovné high skilly

**beam**

Pošle správu na server s pozíciou na ktorú má hráča nastaviť. Používa sa pred zápasom na umiestnenie hráčov na pozície napríklad vo formácii.

**get_up**

Postaví hráča keď je na zemi. Ak hráč leží na bruchu, vykoná sa pohyb pre vstávanie z brucha. To isté platí keď hráč leží na chrbte.

**kick**

Tento high skill má na starosť kopnutie do lopty. Podľa vzdialenosti a pozície od lopty si hráča najprv nastaví na vhodnú pozíciu pre kopnutie krokmi a krôčikmi vpred alebo do strany a následne vykoná kop.

**localize**

Pomáha hráčovi nájsť loptu. Hráč najprv otáča hlavou vľavo a potom vpravo. Ak stále nevidí loptu, je pravdepodobne pod ním takže sa otočí celý hráč až kým nenájde loptu.

**turn**

Otočí hráča k zadanému cieľu. Podľa uhla odchýlky vykoná hráč malé otočenie alebo otočenie o 45° až 60°.

**walk2Ball**

Má na starosť príchod k lopte. Na základe pozície lopty sa najprv môže otočiť a potom ísť vpred. Počas pohybu k lopte si kontroluje natočenie a vzdialenosť a podľa potreby koriguje smer otočeniami kôli odchýlke ktora vzniká pri pohybe.

**cyclic_high_skill**

Tento high skill nie je využívaný v žiadnom pláne a slúži na opakované volanie toho istého low skillu, ktorý je jediným parametrom jeho inicializácie.

**linked_high_skill**

Inicializačná metóda tohto high skillu neobsahuje žiadne parametre. Jeho funkciou je postupné vykonávanie pevne zadefinovaných low skillov. Low skilly sú pridané do premennej *skills* (jednorozmerné pole) priamo v inicializačnej metóde. Metóda *pickLowSkill* postupne pri každom zavolaní vráti zo začiatku poľa *skills* jeden vybratý low skill. Postupne sa takto z poľa vyberú všetky pohyby a v taktom prípade metóda zavolaní vráti *null*, čím sa high skill ukončí.

**low_skill_test**

Tento high skill bude po reimplementácii odstránený. Inicializačná metóda prijíma ako parameter názov low skillu, no metóda *pickLowSkill* ani len tento prijatý low skill nevracia.

**low_skill**

Tento high skill je využívaný viacerými plánmi ako prostredník na zavolanie jedného pohybu (low skillu) v plánovači, ktorý je zadaný ako parameter inicializačnej metóde. Metóda pickLowSkill je ošetrená pre prípad opakovaného zavolania tejto metódy kontrolou, či už daný low skill bol jeden krát vrátený, a teda je už vykonávaný.

**goToBall**

High skill sa snaží dostať robota k lopte. Ak loptu nevidí, zisti si jej pozíciu a podľa toho či je ďaleko alebo blízko zvoli príslušnú sériu low skills.

**walkNew**

Tento high skill sa snaží dostať robota k lopte, nie je ale moc prepracovaný nepoužíva sa v žiadnom pláne. Pravdepodobne ho nahradil high skill Walk2Ball.

**walkOld**

Tento high skill bol prepísaný do javy, no pri testovaní sa zdalo, že nefungoval, ale to mohlo byť spôsobené aj zlým plánom. Nepoužíva ho žiaden plán, tj predpoklada sa, že bol nahradený iným.

# 4 Tretí šprint

## 4.1 Oprava unit testov

Po spustení všetkých testov v nástroji kontinuálnej integrácie bol vytvorený zoznam testov, ktoré obsahovali chybu. Zoznam obsahoval 10 testov, ktoré boli riešené.

### LogTest

Chyba v tomto teste spočívala v zlom importe metód. Táto chyba bola odstránená správnym importom metód.

### TacticalInfoTest.testbIsBallNearestToMe

Chyba v tomto teste odhalila zásadnú chyb pri implementácii testovanej metódy, kedy testovaná metóda mala vracať hodnotu true iba v tom prípade ak agent bol najbližším hráčom pri lopte.

Chyba spočívala v hľadaní najnižšej hodnoty vzdialenosti hocakého hráča od lopty a porovnávaním tejto hodnoty samej zo sebou. Metóda vracala hodnotu true v každom z testovaných prípadov.

Na základe tohto faktu bola testovaná metóda preimplementovaná spolu so všetkými metódami určujúcimi najbližšieho hráča k lopte. Taktiež vznikla statická trieda DistanceHelper na výpočet vzdialeností hráčov. Upravenými metódami sú bIsBallNearestToMeInMyTeam, bIsBallOurs, bIsBallTheir.

### LowSkillTest.shouldSkipPhaseIfFlagIsSet

Chyba spočívala v nekorektnom xml súbore, ktorý vznikol najskôr iba na testovacie účely. Načítanie tohto súboru bolo nahradené načítaním súboru z adresára moves, kde sú uložené všetky low skilly, ktoré musia byť korektné. Následne bol celý test upravený podľa hodnôt načítavaného súboru.

### ParserTest.seePerceptor

Chyba bola spôsobená chýbajúcou medzerou v správe odosielanej zo servera. Oprava spočívala v pridaní medzery do tejto správy.

### RubyTest.initializationError

Chyba bola spôsobená chýbajúcou podporou Unit testov v tejto triede. Test slúžil na otestovanie načítania ruby skriptov. Keďže ruby skripty už netvoria súčasť projektu, tak tento test nie je opodstatnený.

### SkillsFromXmlLoaderTest.shouldPopulateSkipFlags

Chyba v tomto teste bola podobná s chybou v teste LowSkillTest. Kde sa načítaval nekorektný xml súbor. Riešením bolo upravenie načítania testu, tak aby nebol závislí na inom teste a následnej úprave testovacích metód podľa nového načítaného xml súboru.

### GoalieTestCaseTest.initializationError

Riešenie tohto testu nie je jednoznačné, keďže tento test neobsahoval žiadnu funkčnú testovaciu metódu (všetky boli zakomentované). Na ďalšom stretnutí je nutné prediskutovať adekvátnosť tohto testu.

### Pozičné testy

Nasledujúce testy slúžia na overenie určenie správnych parametrov pozície agenta(súradnice x,y,z, r, phi, theta). Tieto testy porovnávali hodnoty vypočítané agentom s hodnotami, ktoré boli určené vývojárom. Pozícia agenta sa počítala pomocou pozície fixných bodov na ihrisku. Týmito bodmi sú vlajky a tyčky bránok. Po zmene konštantných hodnôt hracieho ihriska v úlohe ROBOCUPTP-30 9.1 Verzia servera simulacneho prostredia sa automatikcy zmenili aj očakávané hodnoty týchto testov. Riešením týchto chýb je prepočítanie nových očakávaných hodnôt. Do testov bola pridaná poznámka informujúca o verzii servera na ktorom boli testy nadstavované.

Pozičnými testami sú:

- **AgentModelTest.rotationInfer**
- **ParserToAgentModelIntegrationTest.testRotationAndPositioncalculation**
- **AgentModelTest.positionCalculation**

## 4.2 Návrh architektúry agenta

### Selector

Trieda ktorá je mozog systému, zberá údaje od situácií a tried ktoré následne vyhodnocuje a volí najvhodnejšiu taktiku metódou *selectTactic* a taktiež najvhodnejšiu stratégiu selectStrategy.

## ParsedDataObserver

Trieda Selector bude implementovať rozhranie ParsedDataObserver, ktoré je už v kóde projektu prítomné a v kóde je už naprogramované aj správanie pre vzor Observer. Bude stačiť, ak trieda Selector bude mať implementovanú metódu processNewServerMessage(ParsedData data), v ktorej bude volať svoje metódy controlStrategy() a controlTactic()-

Metóda *controlTactic* preveruje či aktuálna taktika stále spĺňa kritéria pre vykonávanie. (Neexistuje taktika s oveľa väčšou fitness, taktika nie je ukončená)

Metóda *controlStrategy* preveruje či aktuálna stratégia stále spĺňa kritéria pre vykonávanie. (Neexistuje stratégie s väčšou fitness)

## Situácie

Trieda situácie reprezentuje jednotlivé situácie, ktoré môžu počas hry nastať. Sú to situácie napríklad:

- loptu má protihráč a som v prvom kvadrante a lopta je v treťom kvadrante
- lopta je v druhom kvadrante, som v prvom kvadrante a nikto nemá loptu

Pre situácie bude vytvorená jedna statická trieda, ktorá bude zodpovedná sa tvorbu ďalších inštancií situácii pri načítaní projektu. Nebudeme tvoriť samostatné triedy pre každú rozdielnu situáciu, vždy to bude objekt situácia avšak s rôznymi atribútmi a odlišným názvom situácie. Štruktúra triedy bude vyzerať nasledovne:

```
public enum Quadrant {
    1,2,3,4
}
class Situation {
    private Integer scoreLeft = 0;
    private Integer scoreRight = 0;
    private boolean iHaveBall = true;
    private boolean rivalHasBall = false;
    private boolean iAmNearBall = false;
    private boolean nobodyHasBall = false;
    private Quadrant iAmInQuadrant = 2;
    private Quadrant  ballIsInQuadrant = 1;
    private integer howManyPlayersAreNearBall = 2;
}
```

Hodnoty, ktoré sa majú do inštancií nastaviť budú spísané v XML súboroch a budú sa z nich pri spustení programu načítavať. Zápis pomocou XML umožní i prípadnú zmenu hodnôt počas behu programu.

## Stratégie

Situácie budú vstupovať ako argument do jednotlivých stratégií a vyberača stratégií, keďže budú mať vplyv i na výber stratégií. Vyberač (stratégií i taktík) bude vyberať vhodnú stratégiu na základe fitness. Výpočet fitness bude spočívať v porovnávaní aktuálnej situácie s predpripravenými situáciami – tj šablónami. Keďže sa nemusí podariť mať ošetrené všetky možné kombinácie situácií, v prípade nezhody budeme brať najbližšie vyhovujúcu situáciu. Vybraná stratégia bude uložená v modeli – agenta.

**Kedy sa mení stratégia?**

- Nenaplnili sa ciele stratégie
- zmena času
- zmena skóre hry

**Implementácia učenia?**

- Ak vybraná stratégia dala gól, zvýšiť jej fitness, popr. prioritizovať danú stratégiu
- Fitness sa bude logovať, čím bude možné vyhodnocovať dosiahnuté výsledky pri zvolenej fitness i po skončení programu

## Taktiky

Taktiky sa vyberajú v triede Selector. Každá stratégia bude mať zoznam vhodných taktík pre dosiahnutie danej stratégie. Selector zavolá metódu selectTactic, v ktorej sa vyberie vhodná taktika vyhovujúca danej stratégie a situácii.

**Kedy sa (ne)mení taktika?**

- Pokiaľ nemám oveľa lepšiu taktiku, tak ju nemením
- Pokiaľ prejdem do iného kvadrantu

**Ako sa zmení aktuálna taktika?**

Každá taktika bude obsahovať svoju metódu *checkProgress* ktorá zistí či je daná taktika ukončená, alebo je potrebné, aby sa ešte vykonávala. Túto informáciu poskytne metóde *controlTactic*.

Do taktiky vstupujú anotácie, čim sa bude vedieť vybrať konkrétny highskill na základe anotácií.

## HighSkilly

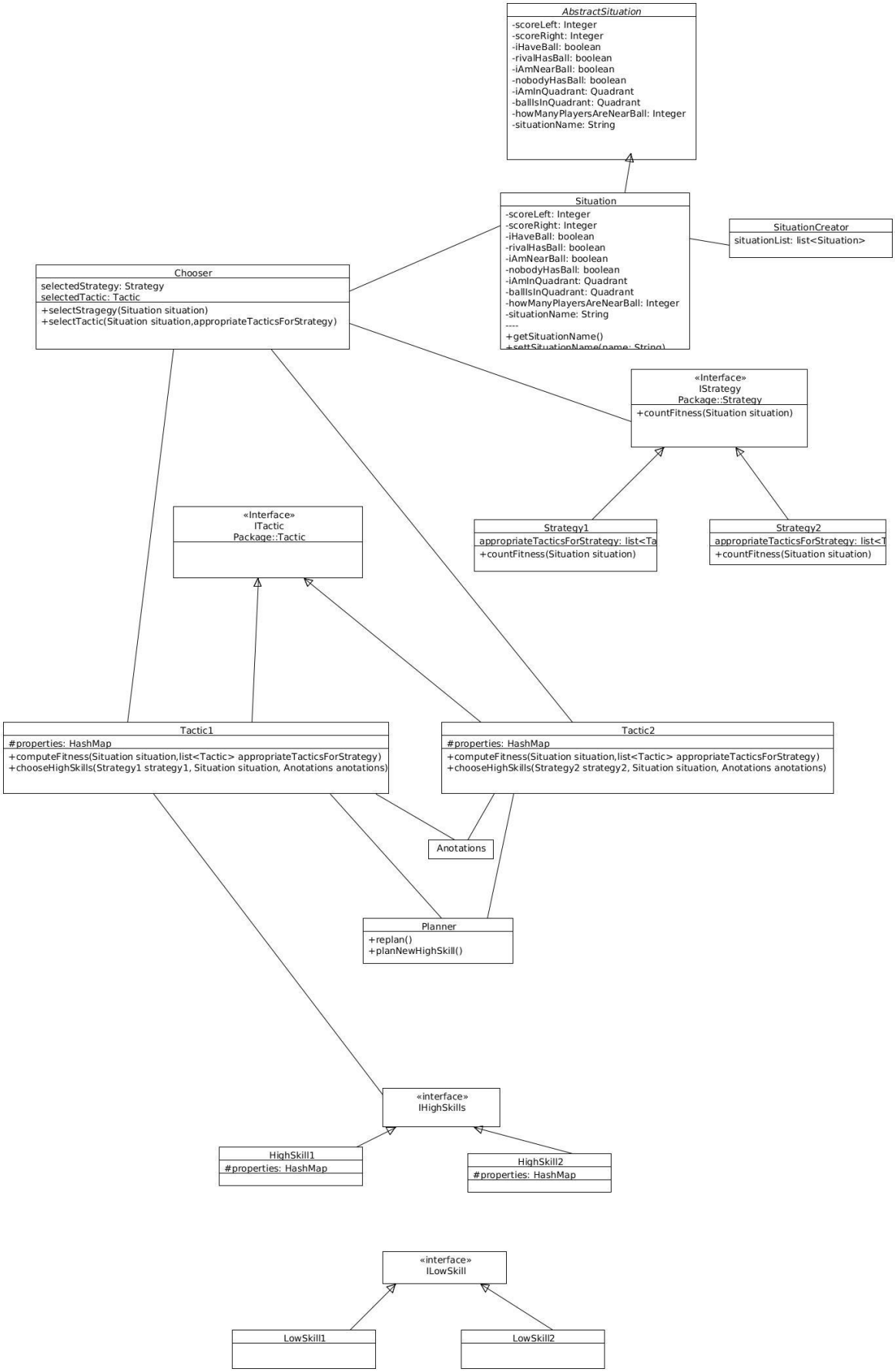
Highskilly sa vyberajú na základe porovnávania vlastností pre vybranú taktiku. Vyberú sa highskilly, ktoré majú najvhodnejšie vlastnosti. Vybrané highskilly sa zaradia do plánovača, z ktorého budú postupne vykonávané.

## Rozhodovanie

Rozhodovanie na základe hash tabuľky. Na začiatku veľká réžia v podobe vypočítania fitness pre rôzne situácie. Fitness sa pridá do hash tabuľky a na základe výberu kľúča bude rýchlejší výber.

## Diagram architektúry agenta

Na obrázku 4.1 je zobrazený návrh architektúry agenta.




**Obrázok 4.1** diagram architektúry agenta

## 4.3 Úprava projektu pre verziu servera 0.6.7

Od predchádzajúcej nasadenej verzie servera (0.6.5) a aktuálnej (0.6.7) vznikli zmeny ktoré ovplyvňujú priamo kód hráča, alebo aj spôsob hrania prípadne rozvíjajú ďaľšie možnosti vývoja.

**Pravidlá**

- Automatický rozhodca kontroluje dodržiavanie pravidiel kedykoľvek hráči môžu hrať, nielen v PlayOn stave.
- Nie je možné streliť priamo gól z výkopu. Lopta sa musí minimálne dotknúť ďalšieho hráča.

**Rozmery**

- Veľkosť ihriska sa zväčšila na veľkosť 30m x20m z predchádzajúcich 21m x 14m.
- Vzdialenosť pre voľný kop sú 2m.
- Pozícia pre rohový kop je stred medzi bránkou a rohom ihriska aby boli rohové kopy rýchlejšie.
- Výška chodidla robota Nao je 0.02m namiesto pôvodných 0.03m. S tým sa zmenila aj pozícia členka.

**Fauly**

- Bolo pridané zaznamenávanie faulov. Fauly nie sú penalizované ale ich zaznamenanie sa posiela na monitor, do ktorého komunikačného protokolu boli pridané.

**Iné**

- Pre testovacie účely môže byť v vnímanie pozície zapnuté nastavením parametra setSetSenseMyOrien na enable v .rsg súbore agenta.
- Hráč môže kopnúť do lopty vo výkope ak sú iba dvaja hráči na ihrisku na podporu aktuálneho pokutového módu.
- Zlepšená podpora automatického výkopu. Čas parametra WaitBeforeKickOff bol zvýšený na 30s z 5s aby sa všetky tímy stihli pripraviť a naštartovať. Čas sa začína počítať od pripojenia prvého hráča namiesto od začiatku spustenia simulácie.
- Hra môže byť spustená výkopom ľavého tímu alebo s použitím hodu mincou, podľa ktorého sa rozhodne ktorý tím začne. Túto vlastnosť je možné nastavením premennej CoinTossForKickOff v naosoccersim.rb na true.
- Je možné nastaviť automatické ukončenie simulácie po skončení hry nastavením premennej AutomaticQuit v naosoccersim.rb na true.
- Bola pridaná podpora heterogénnych hráčov. Správa pre server má tvar "(scene rsg/agent/nao/nao_hetero.rsg TYPE_NO)" kde TYPE_NO je typ hráča. Hodnota 0 je pre štandardný typ a heterogénne typy začínajú od 1. Parametre robotov sú definované v súbore *naorobottypes.rb*.

Táto zmena ovplyvnila aj pravidlá, keď každý tím môže použiť limitovaný počet heterogénnych hráčov. Limity je možné nastaviť zmenou premenných MaxTotalHeteroCount aMaxHeteroTypeCount v naosoccersim.rb. Prvá premenná je celkový počet hetero hráčov a druhá maximálny počet daného typu.

## Úpravy v kóde

Konštanty v zdrojovom kóde boli upravené na najnovšie rozmery ihriska so zachovaním spätnej kompatibility s predchádzajúcimi verziami. Jedná sa o pozície fixných objektov a rozmery ihriska ako také.

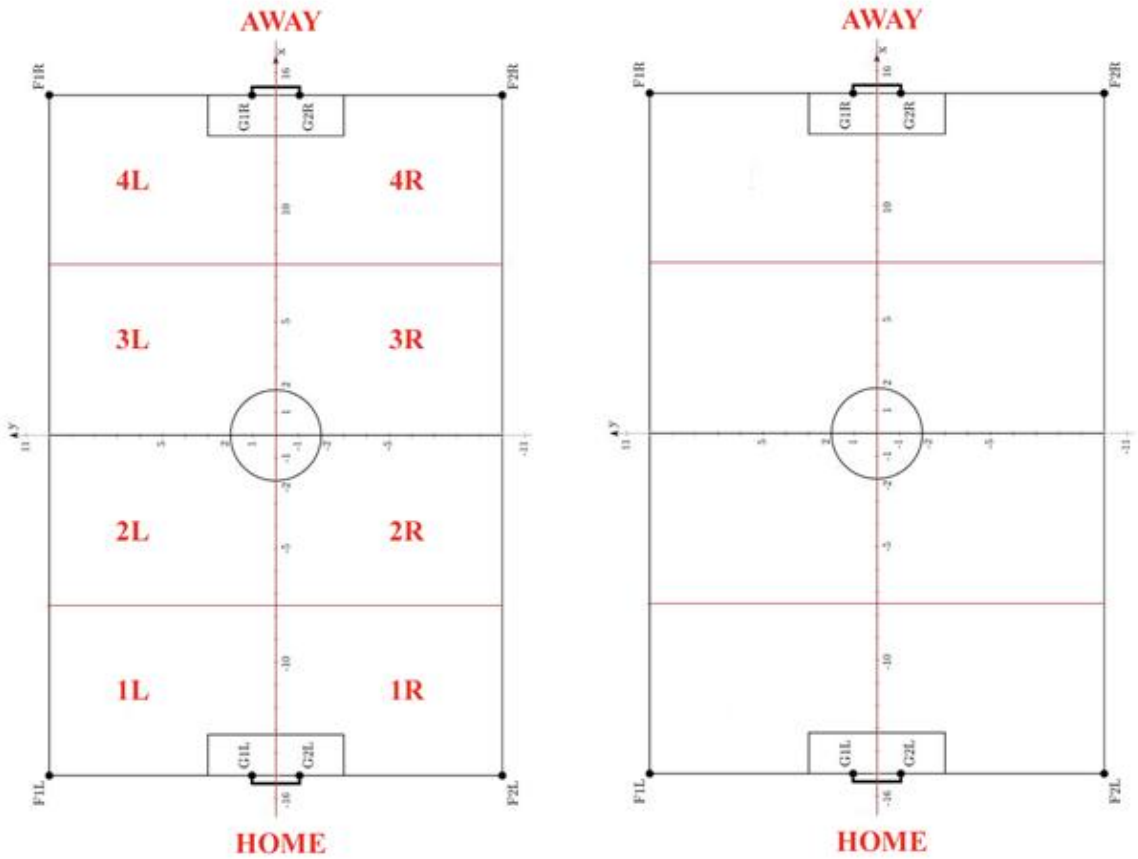
## Možnosti pre testovanie

Súbor *naosoccersim.rb* obsahuje nastavenia servera ako sú rozmery, vzdialenosti, časy pre postavenie a rôzne limity. Toto je možné využiť pri testovaní, napríklad predĺžením hracieho času.

# 5 Štvrtý šprint

## 5.1 Papierový prototyp

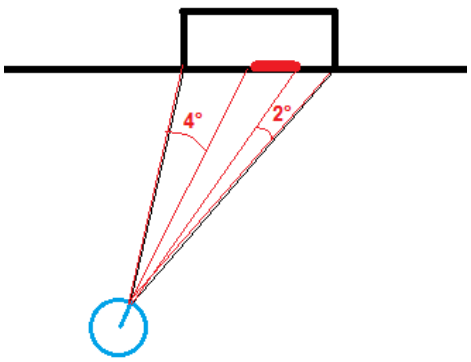
Na obrázku 5.1 je zobrazený model ihriska s jeho logickým rozdelením na osem častí – oktanty. V kapitole sú opísané príklady situácií, v ktorých sa môže hráč ocitnúť. Všetky situácie sú robené pre ľavú polovicu ihriska (oktanty 1L, 2L, 3L a 4L) pretože pre pravú časť ihriska platia zrkadlovo obrátené situácie.




**Obrázok 5.1** rozdelenie ihriska na oktanty a ich pomenovania

### Identifikované parametre

1. boolean **mamLoptu**
2. boolean **superMaLoptu**
3. boolean **spoluhracMaLoptu**
4. double **volnyPriestorBrany** = 6° (v stupňoch)

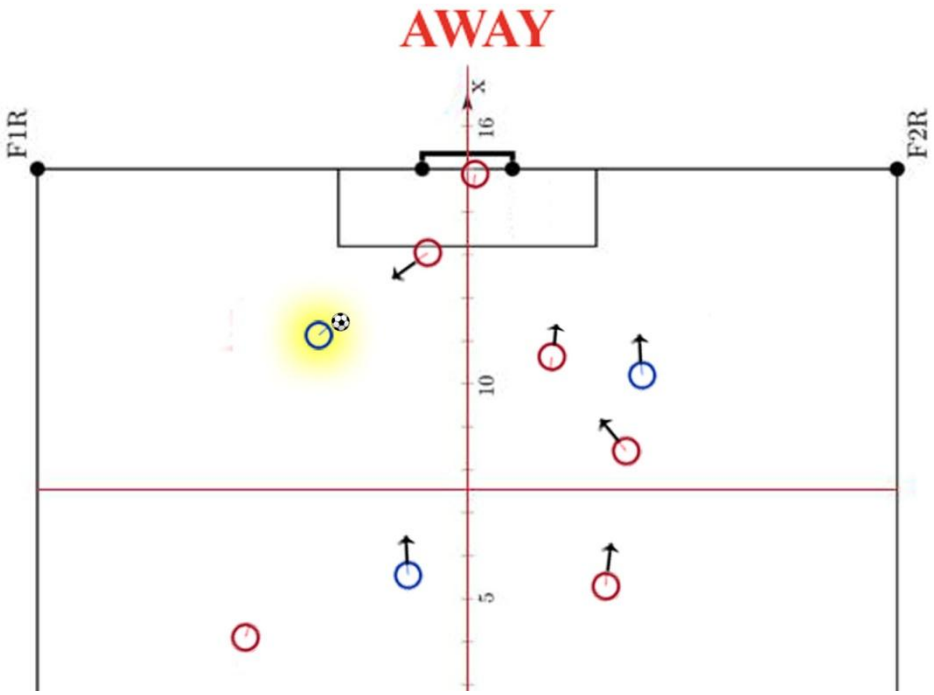



**Obrázok 5.2** ukážka parametru volnyPriestorBrany

5. double **mojaVzdialenostOdBrany**
6. double **superPrideKuMneZa** (v sekundách)
7. boolean **vOctanteXYvidimVolnehoSpoluhraca**
8. boolean **vOctanteXYjeVolnySpoluhrac**
9. int **pocetProtihracovVMojomOkoli**
10. int **pocetSpoluhracovVMojomOkoli**
11. int **pocetProtihracovVOctante**
12. int **pocetSpoluhracovVOctante**
13. int **octatLopty**
14. int **octatAgenta**
15. String **najblizsiTimKLopte**
16. boolean **somVBrankovisku**
17. Player **najblizsiHracKLopte**
18. boolean **somNajblizsieKLopte**

## Útočník v 4L

**Situácia 1**




**Obrázok 5.3** vzorová situácia 2 útočníka v oktante 4L

Situácia:

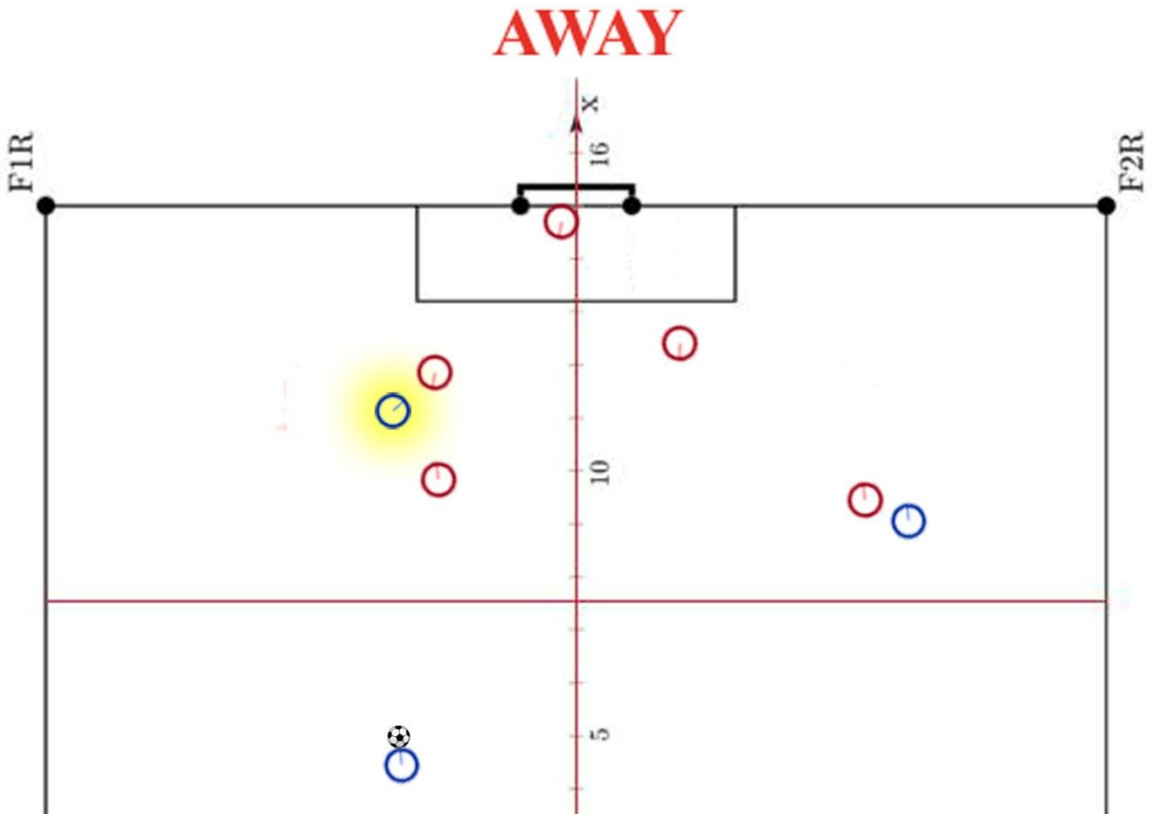
- mám loptu
- vzdialenosť od brány je viac ako 3m
- voľný priestor brány je menej ako 3° (viď identifikované parametre)
- medzi mnou a bránou je súperov obranca, vzdialený viac ako 2m, ktorý sa blíži ku mne rýchlosťou menej ako 0,5 m/s

- OR medzi mnou a bránou je súperov obranca, menej ako 1m, ktorý sa hýbe menej ako 0,2 m/s
  *(tu by si zišiel nejaký komplexný parameter, napríklad hráč príde ku mne za menej ako 2 sekundy)*
- v 4R nie je voľný spoluhráč, t.j. je medzi nimi prekážka, alebo k voľnému hráčovi príde niekto do 2s
- v 3L je voľný spoluhráč, a ľubovoľný protihráč v 3L je vzdialený od neho minimálne 3m alebo 2m ale otočený aspoň o 90°

Cieľ:
- otočiť sa a prihrať spoluhráčovi v 3L

Podmienky:
- spoluhráč v 3L je voľný, resp. ľubovoľný protihráč je vzdialený viac ako 0,5m
- protihráč medzi bránou a mnou sa pohol (t.j. hráč môže kopať na bránu)

**Situácia 2**




**Obrázok 5.4** vzorová situácia 2 útočníka v oktante 4L

Situácia:
- spoluhráč má loptu v 3L (momentalne sa da zistit pomocou dvoch parametrov **octantLopty** a **spoluhracMaLoptu**)
- v mojom okolí sú dvaja protihráči (vo vzdialenosti do 2m)

- v 4R nie je náš voľný hráč, t.j. je pri každom našom hráčovi protihráč, otočený k nemu (s toleranciou 60°)

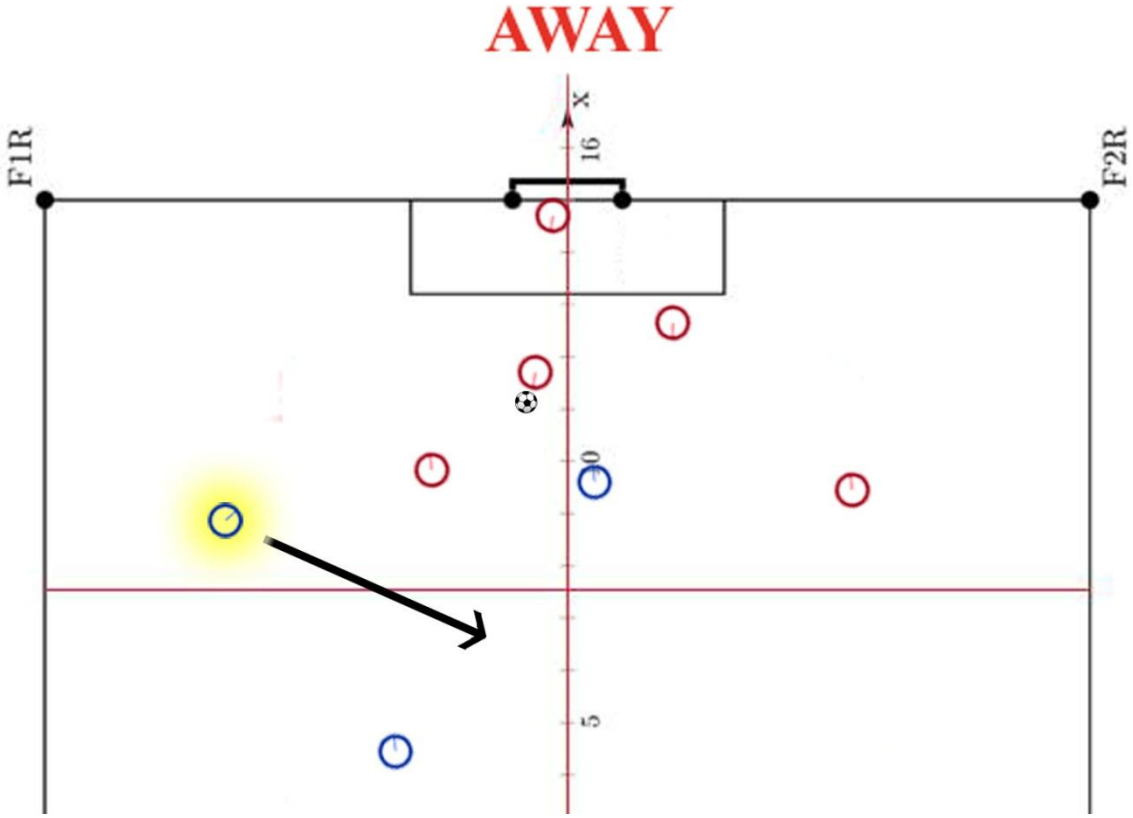
Cieľ:

- prejsť do 4R a otočiť sa smerom k ľavej tyčke súperovej brány (tým pádom sa možno so mnou presunie aj niektorý zo súperových hráčov, alebo sa uvoľním na prihrávku)

Podmienky:

- Náš hráč v 3L má stále loptu

**Situácia 3**




**Obrázok 5.5** vzorová situácia 3 útočníka v oktante 4L

Situácia:

- súper má loptu (ľubovoľný octant)
- je k nemu bližšie nejaký náš hráč a je otočený k nemu so 60° toleranciou a nie je bližšie k súperovej bránkovej čiare ako tento hráč

Cieľ:

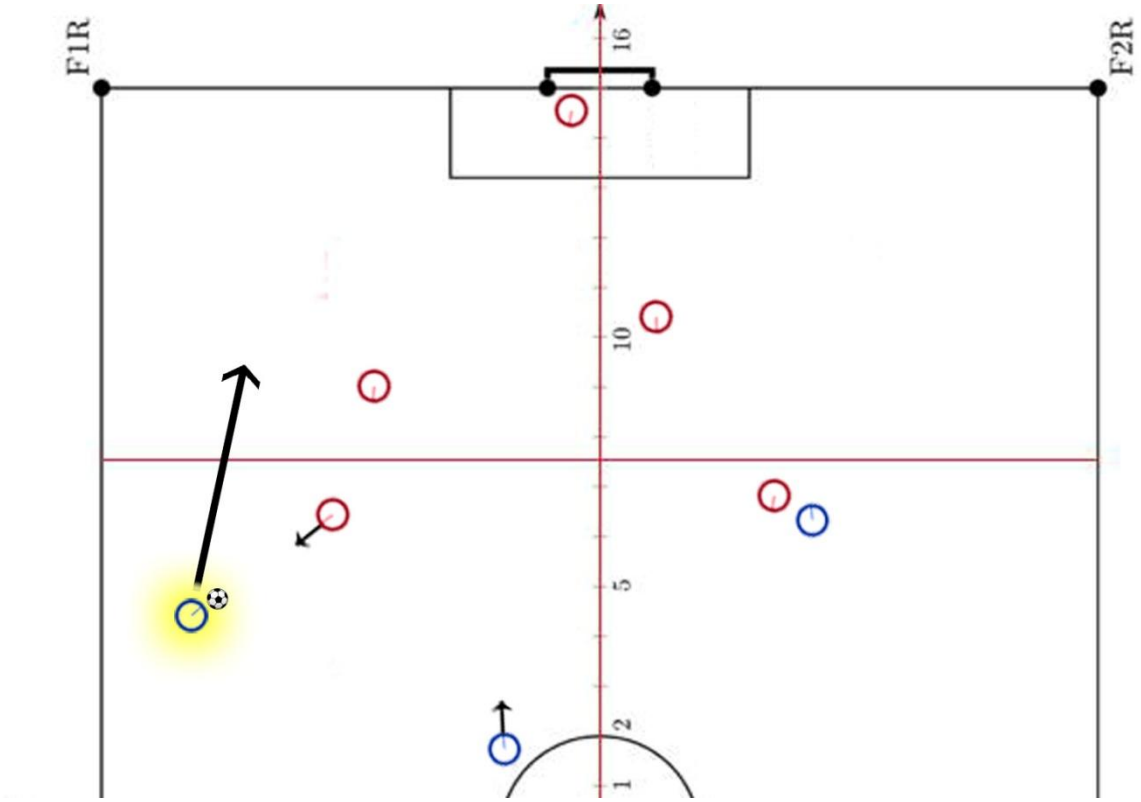
- zdvojovať a teda snažiť sa byť medzi súperom s loptou a našou bránou s odstupom minimálne 4m

Podmienky:

- Je splnená druhá podmienka, t.j. buď som k nemu najbližšie ja, alebo naši hráči sú otočený mimo
- Súper má loptu

## Útočník v 3L

**Situácia 1**




**Obrázok 5.6** vzorová situácia 1 útočníka v oktante 3L

Situácia:

- mám loptu
- v mojom okolí je najviac jeden protihráč, ktorý smeruje ku mne OR v mojom okolí sú dvaja protihráči, ale nesmerujú ku mne
- v 4L alebo v 4R nie je voľný spoluhráč

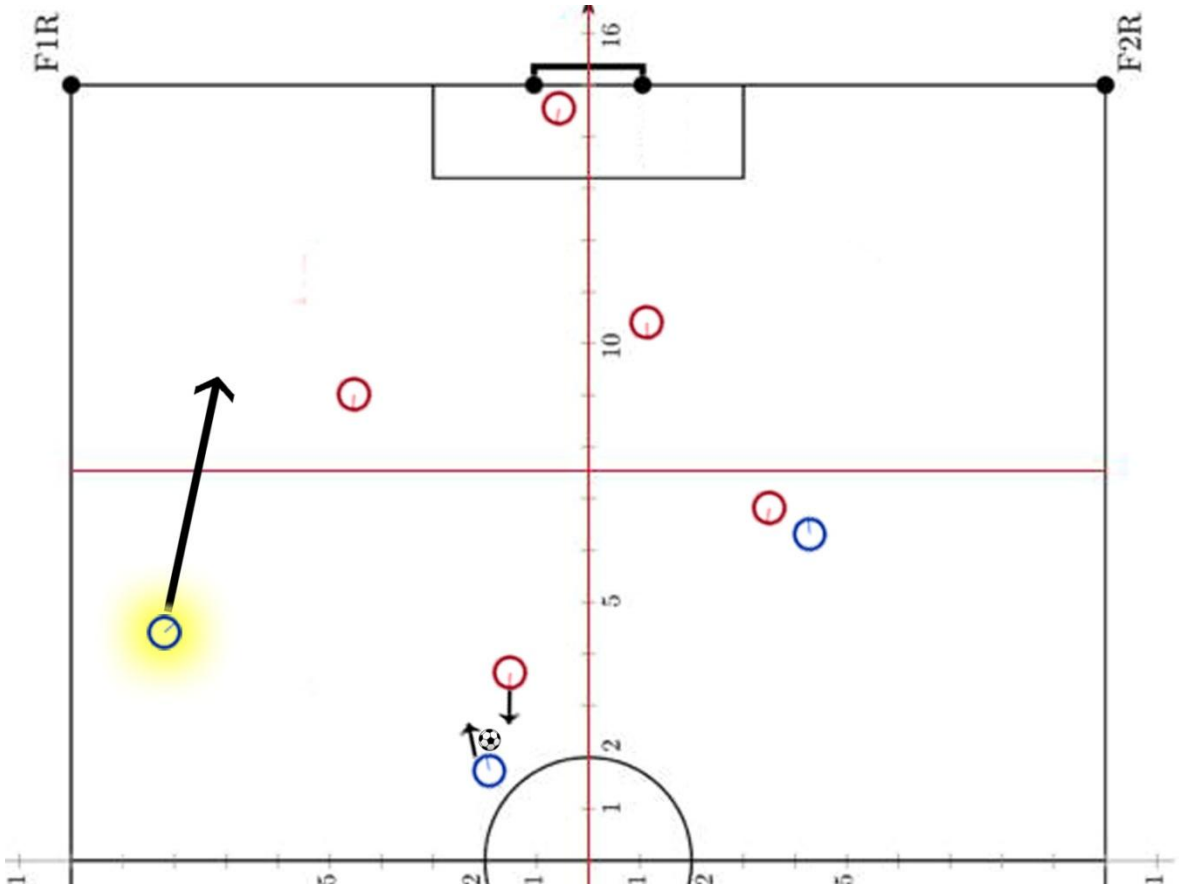
Cieľ:

- prejsť s loptou do 4L

Podmienky: (rovnaké ako situácia)

- mám stále loptu
- počet protihráčov v mojom okolí nie je viac ako 2
- v 4L alebo v 4R nie je voľný spoluhráč

**Situácia 2**




**Obrázok 5.7** vzorová situácia 2 útočníka v oktante 3L

Situácia:

- nemám loptu
- spoluhráč tiež v 3L má loptu
- pri spoluhráčovi s loptou nie je viac ako 1 protihráč
- v 4L nie je náš spoluhráč
- v 3R alebo v 4R je aspoň jeden spoluhráč

Cieľ:

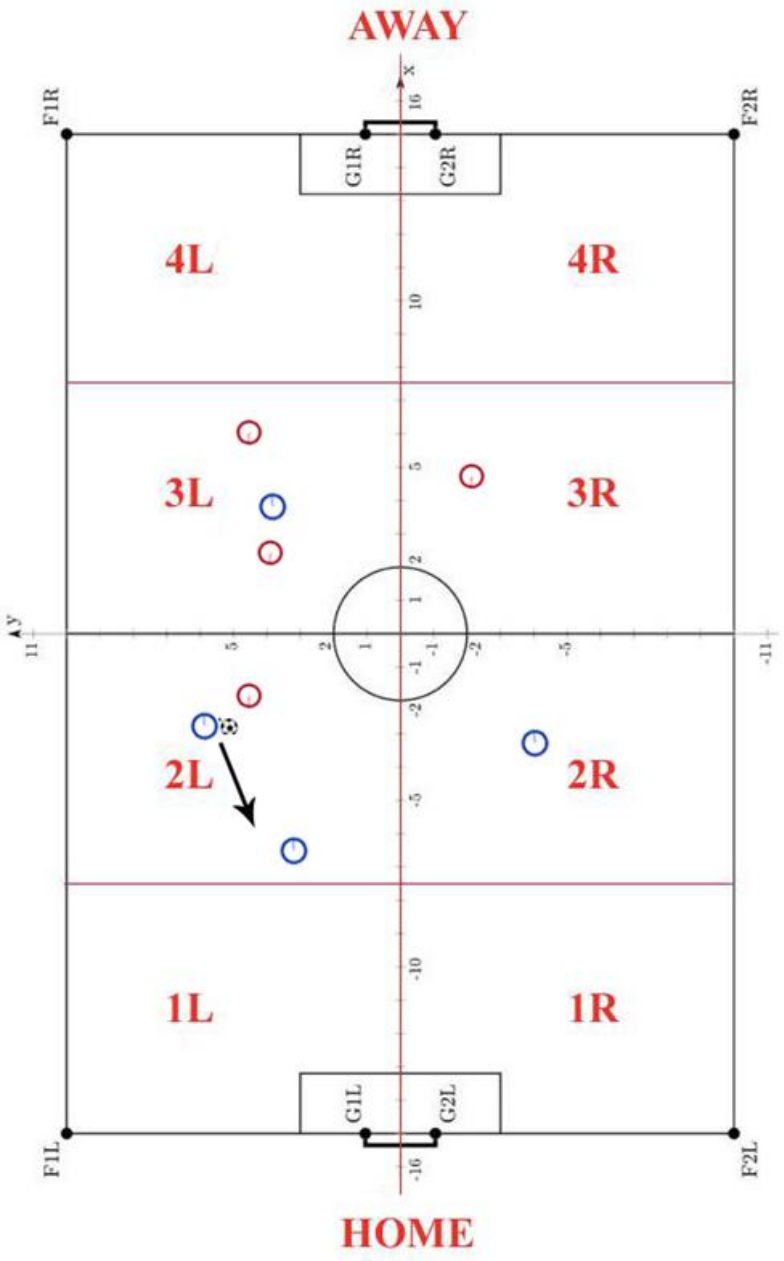
- ísť do 4L a uvoľniť sa, no nie do ofsajdu

Podmienky:

- náš spoluhráč má loptu
- v 4L nie je náš spoluhráč
- pri spoluhráčovi s loptou nie je viac ako 1 protihráč
- OR pri spoluhráčovi s loptou sú dvaja protihráči, ale aj jeden spoluhráč
- Nie som v ofsajde

## Útočník v 2L

**Situácia 1**




**Obrázok 5.8** vzorová situácia 1 útočníka v oktante 2L

Situácia:

- Mam loptu
- Vo vzdialenosti 3m odo mňa v rovnakom kvadrante je protihráč, ktorý ma ide obsadiť
- V oktante 3L nemám voľného hráča z môjho tímu
- V rovnakom oktante 2L mám spoluhráča, ktorý je za mnou a je voľný

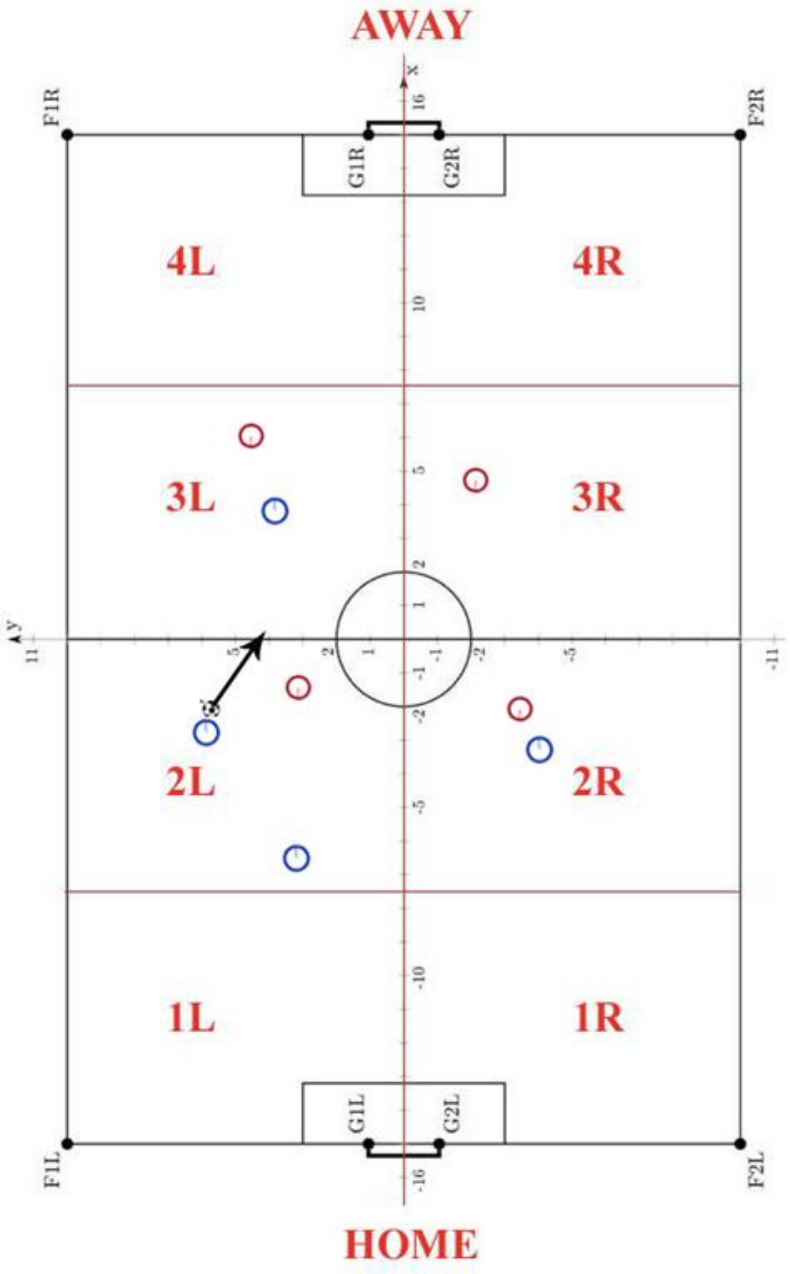
Cieľ:

- Otočiť sa a prihrať spoluhráčovi za mnou

Podmienky:

- Spoluhráč v rovnakom oktante je voľný
- Spoluhráč v 3L oktante nie je voľný (ak vôbec nejaký je)

**Situácia 2**




**Obrázok 5.9** vzorová situácia 2 útočníka v oktante 2L

Situácia:

- Mam loptu
- Protihráč sa približuje ku mne, avšak stále som voľný a mám dosť času na prihrávku s loptou spoluhráčovi

- V rovnakom oktante mám voľného spoluhráča
- V oktante 2R nemám voľného spoluhráča
- V oktante 3L mám voľného spoluhráča, ku ktorému sa približujú súperi

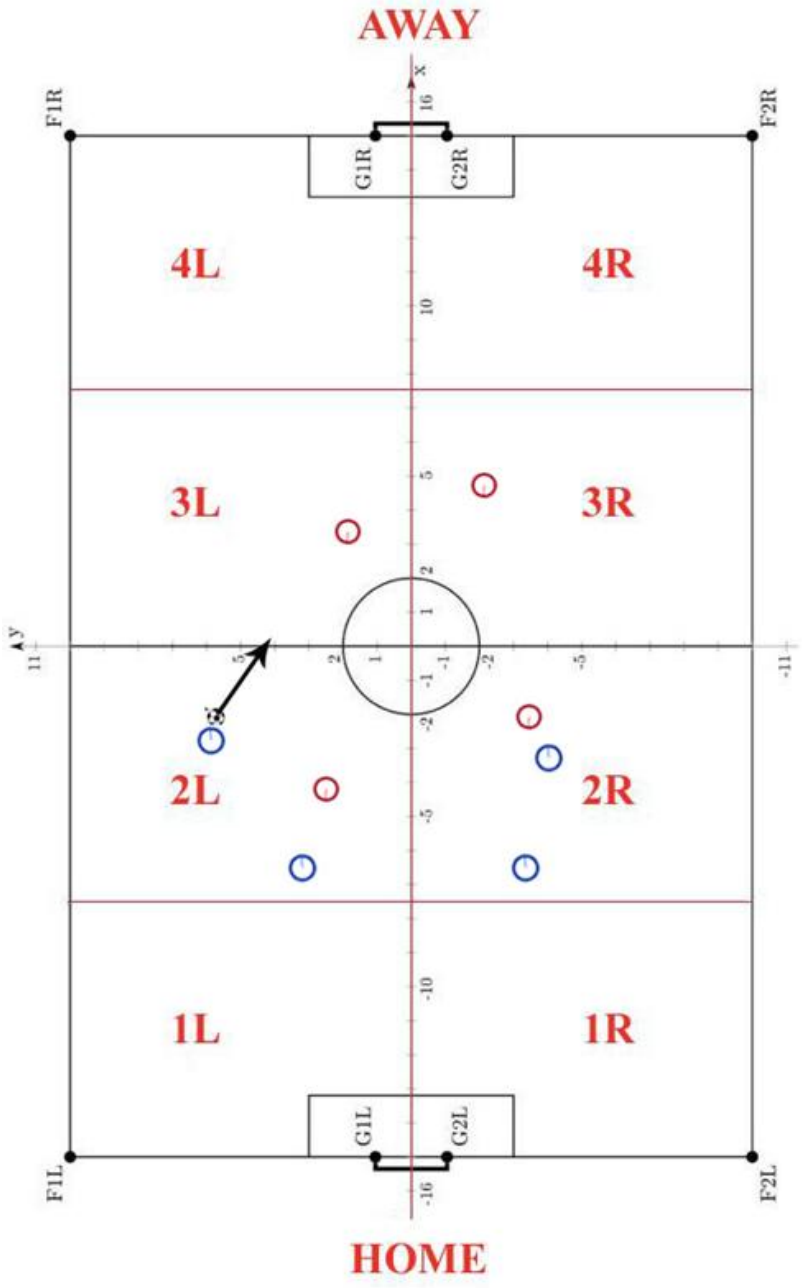
Cieľ:

- Prihrať loptu spoluhráčovi do oktantu 3L

Podmienky:

- Nemám voľného spoluhráča v oktante 2R
- Mám voľného spoluhráča v oktante 3L, ktorý nie je v postavení mimo hry

**Situácia 3**




**Obrázok 5.10** vzorová situácia 3 útočníka v oktante 2L

Situácia:
- Mam loptu
- Protihráč zle prihral a prihral mi rovno na kopačku
- Som neobsadený
- V oktánoch 3L a 3R nemám voľného spoluhráča, nemám komu prihrať vpredu – iba spoluhráčom za mnou

Cieľ:
- Prechod s loptou do 3L oktánu

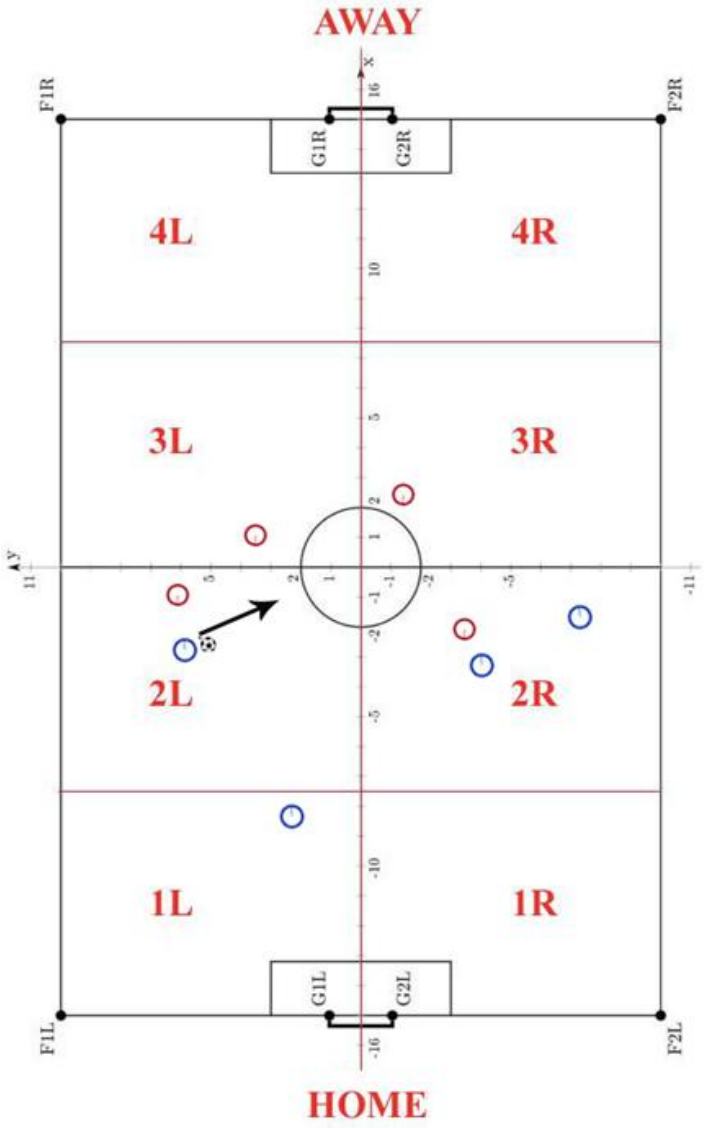
Podmienky:
- Som voľný
- V oktánoch 3L a 3R nemám voľného spoluhráča
- V rovnakom kvadrante nemám voľného spoluhráča, ktorý by bol viac vpredu ako ja

**Situácia 4**

Situácia:
- Mám loptu
- V rovnakom oktante ako ja nemám voľného spoluhráča
- V oktantoch 3L a 3R nemám spoluhráčov
- V oktante 2R mám spoluhráčov, pričom aspoň jedného voľného a rovnako ďaleko alebo bližšie k bráne ako ja.

Cieľ:
- Krížna prihrávka cez ihrisko do oktantu 3R tak, aby bola nahraná voľnému spoluhráčovi do pohybu

Podmienky:
- Vo vzdialenosti 3-6 metrov odo mňa je protihráč
- V oktantoch 3L a 3R nemám spoluhráčov
- V oktante 2R mám voľného 1 spoluhráča




**Obrázok 5.11** vzorová situácia 4 útočníka v oktante 2L

**Situácia 5**

Situácia:

- Mám loptu
- V rovnakom oktante ako ja mám voľného spoluhráča
- V oktantoch 3L a 3R nemám spoluhráčov
- Približuje sa ku mne hráč, ktorý ma chce obsadiť

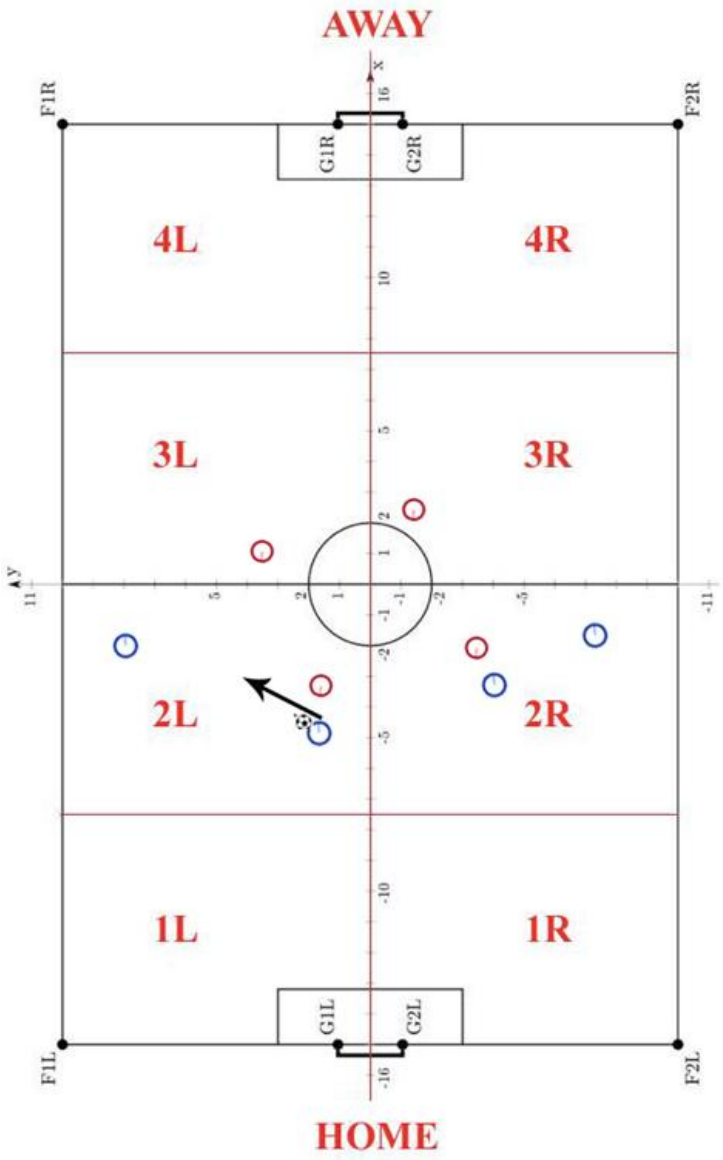
Cieľ:

- Prihrávka – vysunutie spoluhráča po ľavom krídle

Podmienky:

- Mám loptu

- V rovnakom oktante ako ja mám voľného spoluhráča
- V oktantoch 3L a 3R nemám spoluhráčov




**Obrázok 5.12** vzorová situácia 5 útočníka v oktante 2L

## Brankár

**Situácia LoptaBrankara:**

- Lopta v 1L alebo 1R
- Moja pozícia – 1L alebo 1R
- Som v bránke
- Najbližší hráč k lopte – Ja

Ciel:

- Rozohrať loptu od bránkoviska
    - ak je ku mne bližšie spoluhráč - prihrať najbližšiemu spoluhráčovi
    - ak je ku mne bližšie oponent - vykopnúť loptu dopredu

**Situácia LoptaSpoluhraca**

- Lopta v 1L alebo 1R
- Moja pozícia – 1L alebo 1R
- Som v bránke
- Najbližší hráč k lopte – spoluhráč

Ciel:

- ostaň v bránkovisku a pozoruj hru

**Situácia LoptaOponenta**

- Lopta v 1L alebo 1R
- Moja pozícia – 1L alebo 1R (bránkovisko)
- Som v bránke
- Najbližší hráč k lopte – oponent

Ciel:

- ostaň v bránkovisku a chytajLoptu

**Situácia MimoBrankoviskaOponent**

- Lopta (nie je v 1L alebo 1R) a (nie je v 4L alebo 4R)
- Moja pozícia – 1L alebo 1R
- Nie som v bránke
- Najbližší hráč k lopte – oponent

Ciel:

- Návrat do bránkoviska a chytajLoptu

**Situácia MimoBrankoviskaSpoluhrac**

- Lopta (nie je v 1L alebo 1R) a (nie je v 4L alebo 4R)
- Moja pozícia – 1L alebo 1R
- Nie som v bránke
- Najbližší hráč k lopte – spoluhráč

Ciel:

- Pozoruj hru

## Obranca v 1L

**Situácia 1**

Popis situácie:

- som v blízkosti vlastnej bránky(1L) bez lopty a súper útočí – je potrebné zabrániť gólu. Som v blízkosti hráča s loptou číže JA by som mal zasiahnuť.

Situácia:

- mamLoptu = false
- superMaLoptu = true
- spoluhracMaLoptu = false
- octatLopty = 1L
- octatagenta = 1L
- najblizsiTimKLopte = „Meno súperovho tímu“
- somVBrankovisku = false
- najblizsiHracKLopte = nejaký súperov player
- somNajblizsieKLopte = true

Cieľ:

- obrat protihráča s loptou alebo zablokovať strelu (zmenšiť strelecký uhol) – v oboch prípadoch sa jedná o pristúpenie pred protihráča s loptou s tým, že sa odhadne jeho predpokladaná pozícia v budúcom čase (na základe smeru a rýchlosti protihráča).

**Situácia 2**

Popis situácie:

- som v blízkosti vlastnej bránky(1L) podarilo sa mi dostať k lopte – je potrebné dostať loptu z Octantov 1.

Situácia:

- mamLoptu = **true**
- superMaLoptu = **false**
- spoluhracMaLoptu = false
- octatLopty = 1L
- octatagenta = 1L
- najblizsiTimKLopte = „Meno môjho tímu“
- somVBrankovisku = false
- najblizsiHracKLopte = JA
- somNajblizsieKLopte = true

Cieľ:

- Ak je splnená podmienka 1
    - Prihrať najbližšiemu spoluhráčovi ak je na dostrel a voľný (v 2L alebo 2P mám spoluhráča)
    - Odkopnúť dopredu

- Ak je splnená podmienka 2
    - Vyviesť loptu z polovice

Podmienky:
- Nemám voľnu cestu – protihráč v zornom poli je bližšie ako 2m
- Mám voľnu cestu – najbližší protihráč je za mnou (nevidím ho) alebo nie je bližšie ako 2m

**Situácia 3**
Popis situácie:
- som v blízkosti vlastnej bránky(1L) bez lopty a súper útočí – je potrebné zabrániť gólu. NIE som v blízkosti hráča s loptou číže sa pokúsim dostať sa k bránkovisku tak aby som blokoval prípadnú strelu..

Situácia:
- mamLoptu = false
- superMaLoptu = true
- spoluhracMaLoptu = false
- octatLopty = 1L
- octatagenta = 1L
- najblizsiTimKLopte = „Meno súperovho tímu“
- somVBrankovisku = false
- najblizsiHracKLopte = nejaký súperov player
- somNajblizsieKLopte = **false**

Cieľ:
- postaviť sa na okraj bránkoviska na priamku medzi hráčom lopty a stredom bránky.

**Situácia 4**
Popis situácie:
- som v blízkosti vlastnej bránky(1L) bez lopty spoluhráč už získal loptu a ideme dopredu.

Situácia:
- mamLoptu = false
- superMaLoptu = false
- spoluhracMaLoptu = **true**
- octatLopty = nezáleží
- octatagenta = 1L
- najblizsiTimKLopte = „Meno môjho tímu“
- somVBrankovisku = nezáleží
- najblizsiHracKLopte = nejaký hráč z môjho tímu
- somNajblizsieKLopte = false

Cieľ:

- postupovať dopredu do octantu 2L.
-

**Situácia 5**

Popis situácie:

- v 1L a spoluhráč mi prihral loptu alebo protihráč stratil.

Situácia:

- mamLoptu = false
- superMaLoptu = false
- spoluhracMaLoptu = false
- octatLopty = 1L
- octatagenta = 1L
- najblizsiTimKLopte = „?“
- somVBrankovisku = nezáleží
- najblizsiHracKLopte = JA
- somNajblizsieKLopte = **true**

Cieľ:

- čo najrýchlejšie získať loptu.

## Obranca v 2L

**Situácia 1**

Popis situácie

Obranca je opustený v svojom octante a má v držaní loptu. Snaží sa vytrvať v držaní lopty pokým nebude ohrozený protihráčom a postupuje smerom od *HOME* bránkoviska.

Situácia:

- mám loptu = true
- počet protihráčov v octante 2L = 0

Cieľ:

- Dostať loptu za vzdialenejší okraj od *HOME* brány 2L sektora
- Obranca postupuje od *HOME* brány, lopta v držaní obrancu
- Vytiahnuť súperových hráčov z ich obrannej polovice

Podmienky:

- Žiaden protihráč v mojom okolí

**Situácia 2**

Popis situácie:

Obranca má v držaní loptu ale v jeho okolí sa nachádza súper, ktorý je ale dosť ďaleko na to aby obranca stihol vykonať prihrávku spoluhráčovi ak sa spoluhráč nachádza v jeho okolí.

Situácia:

- mám loptu = true
- Počet protihráčov v mojom okolí > 0

Cieľ:

- Dostať loptu ďalej od *HOME* brány
- Obranca kopne loptu smerom od brány
    - Medzi spoluhráčom a *HOME* bránou je vačšia vzdialenosť ako medzi obrancom a *Home* bránou a súčasne spoluhráč je voľný: prihrať spoluhráčovy
    - Inak kop do voľného priestoru smerom od brány
- Preniesť loptu (kopom )za niekoľkých súperových hráčov ak je to možné.

Podmienky:

- Najbližší super príde ku mne za čas >> prihrávka spoluhráčovi

**Situácia 3**

Popis situácie:

V blízkosti obrancu sa nachádza jeden alebo viac protihráčov. Obranca nemá dostatok času na vykonanie prihrávky. Vykoná okamžitý kop smerom od brány do prázdneho priestoru.

Situácia:

- mám loptu = true
- Počet protihráčov v mojom okolí > 0

Cieľ:

- Dostať loptu za vzdialenejší okraj od *HOME* brány 2L sektora
- Okamžitý kop do voľného priestoru vzdialenejšieho od *HOME* brány
- Zabrániť prieniku súpera do našej obrannej polovice.

Podmienky:

- Najbližší super príde ku mne za čas <= prihrávka spoluhráčovi

**Situácia 4**

Popis situácie**:**

Nemám loptu ale v mojom okolí sa nachádza spoluhráč ktorý loptu má.

Situácia:

- mám loptu = false
- spoluhráč má loptu

- lopta v 2L

Cieľ:
- Asistovať hráčovi s loptou: obranca sa pohybuje tak aby bol medzi hráčom a *HOME* bránou
- Zabrániť prieniku protihráčov v prípade, že spoluhráč stratí loptu

Podmienky:
- Spoluhráč s loptou v oktante 2L

**Situácia 5**

Popis situácie:

Nemám loptu ale v mojom okolí sa nachádza spoluhráč ktorý loptu má.

Situácia:
- mám loptu = false
- spoluhráč má loptu
- lopta nie je v 2L

Cieľ:
- Postupujem spolu so spoluhráčom ale neasistujem mu priamo

Podmienky:
- Spoluhráč s loptou v inom oktante ako 2L

**Situácia 6**

Popis situácie:

Nemám loptu ale, súper má loptu a je bližšie k *HOME* bráne ako ja.

Situácia:
- mám loptu = false
- spoluhráč má loptu = false
- súper má loptu = true
- lopta v 2L
- moja vzdialenosť od brány > vzdialenosť lopty od brány

Cieľ:
- Rýchly pohyb smerom kde predpokladám, že bude lopta
- Dostať sa medzi *HOME* bránu a loptu, ak je to možné obrať hráča o loptu

Podmienky:
- Som najbližší hráč z môjho tímu k lopte

**Situácia 7**

Popis situácie:

Nemám loptu ale, súper má loptu a je ďalej od *HOME* brány ako ja.

Situácia:

- mám loptu = false
- spoluhráč má loptu = false
- súper má loptu = true
- Som najbližší hráč z môjho tímu k lopte

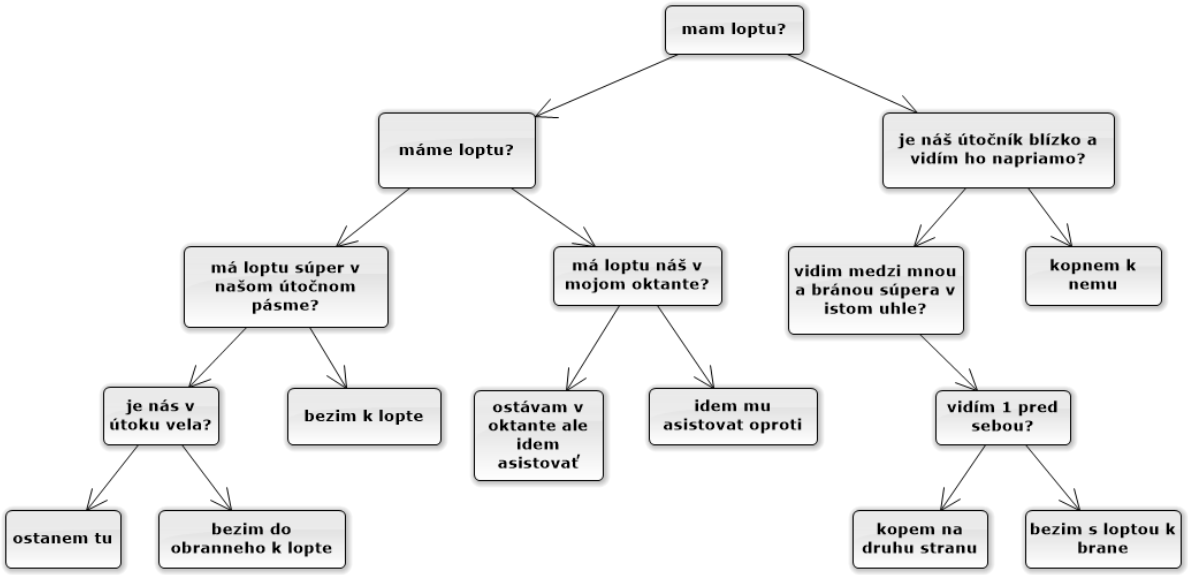
Cieľ:

- Udržať sa v pozícií medzi súperom a bránou
- Pokus o obratie hráča

Podmienky:

- Som najbližší hráč z môjho tímu k lopte

## Obranca v 3L




**Obrázok 5.13** rozhodovací strom obrancu v oktante 1L (napravo – true, naľavo - false)

**Situácia 1**

Popis:

- Súper útočí, predpoklad že niekto z našich vykopne loptu sem

Definícia:

- superMaLoptu = true
- vlastnikLoptyVPasmeKdeSomJa = false
- nasichHracovVUtocnomPasme < 2

Cieľ:

- Zostať v útočnom pásme, prípadne sa presunúť do stredného a čakať na loptu

**Situácia 2**

Popis:

- Súper útočí, ale v útoku nás je veľa a málo nás je v obrane

Definícia:

- superMaLoptu = true
- vlastnikLoptyVRovnakomPasme = false
- nasichHracovVUtocnomPasme >= 2

Cieľ:

- Bežať do obranného pásma, v ňom rozhodnúť ďalej

**Situácia 3**

Popis:

- Súper má loptu v našom útočnom pásme, idem ho obrať

Definícia:

- superMaLoptu = true
- vlastnikLoptyVRovnakomPasme = true

Cieľ:

- Bežať k lopte a zablokovať súpera, prípadne mu prevziať loptu

**Situácia 4**

Popis:

- Spoluhráč má loptu a útočí v mojom oktante

Definícia:

- spoluhracMaLoptu = true
- vlastnikLoptyVRovnakomOktante = true

Cieľ:

- Bežať od spoluhráča a k bráne za účelom asistencie na prihrávku

**Situácia 5**

Popis:

- Spoluhráč má loptu a útočí v oktante oproti

Definícia:

- spoluhracMaLoptu = true
- vlastnikLoptyVRovnakomOktante = true

Cieľ:

- Asistovať na prihrávku v oktante kde je hráč

**Situácia 6**

Popis:

- Hráč vlastní loptu a má priamy výhľad na bránu

Definícia:

- mamLoptu = true
- spoluhracNaPriamoABlizko = false
- branaNaPriamo = true

Cieľ:

- Bežať s loptou k bráne

**Situácia 7**

Popis:

- Hráč vlastní loptu ale nemá priamy výhľad na bránu, spredu sa blíži jeden súper

Definícia:

- mamLoptu = true
- spoluhracNaPriamoABlizko = false
- branaNaPriamo = false
- superovPredBranou = 1

Cieľ:

- Bežať s loptou k bráne, obehnúť súpera

**Situácia 8**

Popis:

- Hráč vlastní loptu ale nemá priamy výhľad na bránu, oproti je niekoľko blokujúcich súperov

Definícia:

- mamLoptu = true
- spoluhracNaPriamoABlizko = false
- branaNaPriamo = false
- superovPredBranou > 1

Cieľ:

- Kopnúť loptu oproti, je pravdepodobné že tam bude spoluhráč

**Situácia 9**

Popis:

- Hráč vlastní loptu, vidí útočníka napriamo a blízko

Definícia:

- mamLoptu = true
- spoluhracNaPriamoABlizko = true

Cieľ:

- Nahrať loptu útočníkovi

## 5.2 Implementácia taktiky

V rámci riešenia tohto zadania sme doimplementovali telá metód do už predpripravenej kostry projektu, ktorú pripravoval tím 4. Doimplementovanie spočívalo v duplikovaní metód z implementácii rozhrania IStrategy slúžiacich na overovanie podmienky začatia a pokračovania vykonávania taktík a následným pridaním množín situácií pre tieto podmienky. Počas riešenia tejto úlohy sme sa stretli s viacerými chybami v implementácii, ktoré sú uvedené v nasledujúcich bodoch:

- **SelectorObserver** – chýbajúca podmienka pre currentTactic rovné null
- **SituactionManager** – zoznam všetkých situácii sa iba dopĺňal a teda sa nedala jednoznačne vyhodnotiť metóda getSuitability
- **SituationManager** – do zoznamu všetkých situácii sa vkladali hodnoty vytvorené z označenia objektu (situation.toString() ) namiesto získania názvu triedy tohto objektu(situation.getClass().getName()). Následne dochádzalo k nevyhodnocovaniu funkcie getSuitability.

# 6 Piaty šprint

## 6.1 Implementácia taktiky

Pri implementácii taktiky sme použili na otestovanie funkčnosti taktík existujúci plán PlanTactic, ktorý sme vložili do metódy run všetkých existujúcich taktík pre stratégiu OffensiveStrategy. Týmto krokom sme si uľahčili prácu s vyššími schopnosťami, ktoré nie sú doposiaľ prispôsobené k aktuálnej architektúre.

Pri testovaní sme narazili na viacero problémov, ktoré zhrnieme do nasledujúcich bodov:

- Pri spustení simulácie v čase, kedy je agent v stave beam sa navrhnuté vyššie rozhodovanie zasekne a agent nevykonáva žiadnu akciu. V tomto prípade bolo potrebné agenta vypnúť a spustiť nanovo.
- V prípade, keď agent spadol, nastal rovnaký stav, teda vyššie rozhodovanie sa zaseklo a bolo potrebné reštartovať agenta.

Pri riešení tejto úlohy sme sa rozhodli využiť dynamické načítanie taktík využívaných v konkrétnej stratégii. Pre načítanie taktík sme použili reflexiu podporovanú v jazyku java. Počas testovania sme sa stretli s problémom, kedy agent vykonával pohyby nesprávnymi kĺbmi v nesprávnom smere, čo spôsobilo nesprávne správanie pohybov.

## 6.2 Bug – hráč útočí po polčase na vlastnú bránku

Identifikovaná chyba ROBOCUPTP-122 Hrac utocí po polcase na vlastnu branku bola spôsobená tým, že správa, na základe ktorej sa nastavuje hráčovi strana na ktorej hrá, prišla hráčovi iba jeden krát a to po jeho pripojení na server. Po polčase, keď sa zmenia strany, hráčovi informácia o zmene strany nepríde, takže všetky jeho výpočty ohľadom smeru a jeho strany sú nesprávne.

Chyba bola opravená pridaním premenných, na základe ktorých sa nastavuje či sa v hre menia strany počas polčasu a čas polčasu. Ak sa strany po polčase menia a hrací čas prekročí nastavený čas polčasu, hráčovi sa zmení strana.

Aktuálne chyba nebola kritická, keďže zmena strany je už z inštalácie vypnutá, takže hráč útočí stále na pravú stranu. Keďže sa hráč spúšťal a testoval väčšinou samostatne, a nie v reálnom zápase tím proti tímu, chyba bola nájdená náhodou. Pri hre v turnaji, kde sa strany počas polčasu menia, by ale bola kritická keďže po zmene strany by hráč útočil na vlastnú bránku. Možnosť zapnúť/vypnúť zmenu strán po polčase je v nastaveniach servera v súbore naosoccersim.rb.

## 6.3 Unit testy

Test na taktiku Goalie, ktorá je základom taktiky pre bránenie bránky, kontroluje správnosť vyhodnocovania inicializačnej podmienky. Najprv sa skontroluje pozitívny výsledok keď sa do inicializačnej podmienky nastaví zoznam situácii, ktoré môžu spustiť danú taktiku. Druhá časť testu je kontrola na negatívny výsledok keď podmienka má vrátiť false pre zoznam situácii ktoré nesúvisia s taktikou.

# 7 Šiesty šprint

## 7.1 Reflection - dynamické načítanie objektov (situácií, stratégií a taktik) podľa balíkov

Cieľom tejto úlohy bolo umožnenie dynamického načítania objektov situácii, stratégií a taktík. Týmto procesom by sa uľahčila práca s jednotlivými objektmi pri riešení vyššieho rozhodovania.

Pri implementácii sa vychádzalo z už existujúcej podúlohy, ktorá sa nepodarila úspešne implementovať v jednom z predchádzajúcich šprintov pri implementácii taktiky. Test ktorý bol vykonaný na otestovanie možností reflexie v spomínanej úlohe bol použitý pri implementácii tohto načítania objektov.

Rozšírenie spočívalo v zásadnej zmene rozhraní ITactic, IStrategy a ISituation na abstraktné triedy, čo pomohlo na jednoduchšie načítanie objektov. Ďalej boli upravené metódy v jednotlivých triedach TacticList, StrategyList a SituationList tak aby umožňovali dynamické načítanie objektov. V poslednom kroku úlohy bola v triede Settings doplnená inicializačná metóda, ktorá je zodpovedná za načítanie objektov pri nastavovaní premenných samotného agenta.

## 7.2 Implementácia anotácií

Cieľom úlohy bolo upraviť existujúcu implementáciu anotácií tak, aby bolo možné anotácie využiť pri rozhodovaní aký low skill sa využije pri realizácií high skillu.

V rámci high skilu by sa low skill mal vyberať zo zoznamu pohybov podľa parametrov, ktoré sú určené z vyšších vrstiev rozhodovania, ale aj samotným high skillom. Tieto parametre by sa mali porovnávať s údajmi v anotácií. Aby bolo možné realizovať toto riešenie, je potrebné, aby všetky relevantné pohyby mali k dispozícií korektnú anotáciu so všetkými potrebnými údajmi.

V rámci analýzy implementácie anotácií sa určili dva parametre, ktoré je nutné pridať. Je to parameter *walkSpeed* – ktorý obsahuje rýchlosť chôdze v m/s a parameter *minDistance* – ktorý obsahuje informáciu o najkratšom stabilnom kroku v metroch, ktorý je pomocou danej chôdze možné vykonať.

Implementácia anotácií po doplnení nových parametrov. Nové parametre sú vyznačené žltým podfarbením. Okrem novo pridaných parametrov je veľmi dôležitý aj parameter *fall*, ktorý poskytuje informáciu o pravdepodobnosti pádu agenta pri použití danej chôdze.

- String name; //názov pohybu, ku ktorému anotácia patri

- Values duration = **new** Values(); //trvanie pohybu v milisekundách
- **boolean** rot = **false**; //otočí sa behom vykonávania pohybu o nenulový uhol ?
- Axis rotation = **new** Axis(); //rotácia okolo jednotlivých osi
- **boolean** mov = **false**; //pohne sa behom vykonávania pohybu o nenulovú dĺžku ?
- Axis move = **new** Axis(); //dĺžka pohybu v smere jednotlivých osi robota v metroch
- **double** fall; //pravdepodobnosť pádu v percentách
- **boolean** b_mov = **false**; //pohne sa následkom vykonania pohybu lopta ?
- Axis b_move = **new** Axis(); //dĺžka pohybu lopty v smere jednotlivých osi robota v metroch
- **boolean** prec = **false**; //je vykonanie pohybu obmedzene predpodmienkami ?
- State preconditions = **new** State(); //prepodmienky na vykonanie pohybu
- State end = **new** State(); //konečný stav robota po vykonaní pohybu
- String note; //poznámka k pohybu
- String checksum; //chcecksum anotovaného pohybu pomocou SHA-1
- String id; //jedinečný názov anotácie
- **private** Values ballMoveDistance = **new** Values();
- **private boolean** maxPos = **false**;
- **private** Vector3 maxBallDistancePosition = **new** Vector3();
- **double** walkSpeed; – //rýchlosť chôdze v [m/s]
- **double** minDistance; – //minimálna vzdialenosť od cieľa potrebná na vykonanie pohybu v [m]

V rámci analýzy anotácií sa ďalej zistili parametre potrebné pre anotácie kopov
- je potrebné brat do úvahy pozíciu voči lopte, ako sa ideálne postaviť, aby bol agent schopný trafiť loptu, respektíve kam sa postaviť
- dĺžka kopu – ako ďaleko agent loptu dokopne pri použití daného kopu
- rozptyl – aký je rozptyl kopu od požadovaného smeru
- uhol kopu – pod akým uhlom agent kope pri použití daného pohybu
- úspešnosť – aká je úspešnosť trafenia lopty

Pridaním parametrov do anotácií bolo potrebné zmeniť definičný súbor XML, ktorý zabezpečuje dodržanie štruktúry všetkých XML súborov anotácií pomocou verifikácie každej anotácie k tomuto definičnému súboru. Tiež bolo potrebné rozšíriť načítavanie a zrkadlovo aj vytváranie XML súborov anotácií o nové parametre.

Ďalším cieľom bolo dosiahnuť stav, kedy agent využíva len aktuálne anotácie pre low skilly, pretože v riešení sa nachádzalo už zhruba 30 anotácií, ale mnohé z nich boli riešené

len ako ukážkové či testovacie anotácie a neboli aktualizované spolu s low skillmi, ktoré rokmi prechádzali drobnými alebo veľkými úpravami. Často bol problém priradiť anotáciu k pohybu z dôvodu zmeny názvu tohto pohybu, kedy nebolo možné jednoznačne určiť príslušnosť anotácie. Tento stav bol napravený vytvorením nových anotácií pre všetky chôdze a v nasledujúcom šprinte budú vytvorené anotácie aj pre všetky kopy.

Implementácia načítavania anotácií bola zmenená tak, že sa načítajú všetky mená lowSkillov zo súboru /moves a potom sa načítavajú anotácie do mapy podľa toho, či sa vzťahujú na nejaký z lowSkillov podľa mena. Ak nie, tak sa daná anotácia nenačíta, ak áno, tak sa pridajú do mapy. Názvy jednotlivých low skillov a anotácií sú pomerne jasne definované a jednoducho sa pomocou nich dá identifikovať, čo daný low skill realizuje. Prvé slovo názvu vždy popisuje čo daný pohyb robí (kick, turn, step, walk,), potom je podtržník, za ktorým nasleduje bližší popis pohybu, za ktorým je opäť podtržník a ešte bližšia špecifikácia pohybu. Týmto spôsobom sa dajú low skilly rozdeliť do tried a podtried (kopy, úkroky, otočenia, chôdze a chôdzu dozadu, postavenie sa)

## Namerané hodnoty pohybov:

Agent pri vykonávaní testovaných pohybov prakticky nikdy nepadá a samotné pohyby sú spoľahlivé aj pri dlhších vzdialenostiach ako boli použité pri testovaní.

Chovanie agenta pri vykonávaní pohybu **turbo_walk** zodpovedá deklarovaným hodnotám (priemerná rýchlosť: 0,68 m/s, priemerné vychýlenie: 3,88°, úspešnosť: 100%), pričom rýchlosť dosahovaná týmto pohybom je viac ako dvakrát tak veľká ako pri najrýchlejšom testovanom pohybe **walk_forward.**

V nasledujúcich tabuľkách 7.1 až 7.4 sú zaznamenané údaje namerané pri viacnásobnom testovaní jednotlivých low skillov chôdze. Automaticky anotátor z test framworku by značne uľahčil vytváranie anotácií, ale vzhľadom na rozsiahle úpravy architektúry agenta je potrebné vykonať množstvo úprav aj na testovacom frameworku. Tieto úpravy nie je možné realizovať pokým sa neustáli finálna verzia architektúry hráča.

**walk_forward**:

| Meranie | Rýchlosť | Čas | Vzdialenosť | Úspešnosť | Odchýlka (na 15m) |
|---|---|---|---|---|---|
| **1.** | 0,3125 m/s | 48 sekúnd | 15 m | nespadol | 11,1 m |
| **2.** | 0,3127 m/s | 47,97 sekúnd | 15 m | nespadol | 1,0 m |
| **3.** | 0,3067 m/s | 48,90 sekúnd | 15 m | nespadol | 1,2 m |
| **4.** | 0,3093 m/s | 48,50 sekúnd | 15 m | nespadol | 1,2 m |
| **5.** | 0,3138 m/s | 47,80 sekúnd | 15 m | nespadol | 0,9 m |

| | | | | | |
|---|---|---|---|---|---|
| **6.** | 0,3124 m/s | 48,01 sekúnd | 15 m | nespadol | 1,1 m |
| **Priemer:** | 0,3112 m/s | | | 100 % | 1,083m – 6,18° |

**Tabuľka 7.1** Hodnoty testovania chôdze walk_forward

**walk_back**

| Meranie | Rýchlosť | Čas | Vzdialenosť | Úspešnosť | Odchýlka (na 10m) |
|---|---|---|---|---|---|
| **1.** | 0,1261 m/s | 1:59 | 15 | nespadol | // |
| **2.** | 0,1190 m/s | 1:24 | 10 | nespadol | 0,5 m |
| **3.** | 0,1250 m/s | 2:00 | 15 | nespadol | // |
| **4.** | 0,1205m/s | 1:23 | 10 | nespadol | 0,5 m |
| **5.** | 0,1176m/s | 1:25 | 10 | nespadol | 0,3 m |
| **6.** | 0,1176 m/s | 1:25 | 10 | nespadol | 0,3 m |
| **Priemer:** | 0,1234 m/s | | | 100 % | 0,4 m - 2,29° |

**Tabuľka 7.2** Hodnoty testovania chôdze walk_back

**walk_slow**

| Meranie | Rýchlosť | Čas | Vzdialenosť | Úspešnosť | Odchýlka |
|---|---|---|---|---|---|
| **1.** | 0,05 m/s | 40 sekúnd | 2 | nespadol | // |
| **2.** | 0,05 m/s | 40 sekúnd | 2 | nespadol | // |
| **3.** | 0,05 m/s | 40 sekúnd | 2 | nespadol | // |
| **4.** | 0,476m/s | 3:30 | 10 | nespadol | 0,5 m |
| **5.** | 0,476m/s | 3:30 | 10 | nespadol | 0,3 m |
| **6.** | 0,481m/s | 3:28 | 10 | nespadol | 0,3 m |
| **Priemer:** | 0,05 m/s | | | 100 % | 0,366 m - 2,096° |

**Tabuľka 7.3** Hodnoty testovania chôdze walk_slow

**walk_slow2**

| Meranie | Rýchlosť | Čas | Vzdialenosť | Úspešnosť | Odchýlka |
|---|---|---|---|---|---|
| **1.** | 0,1 m/s | 20 sekúnd | 2 | nespadol | // |
| **2.** | 0,1 m/s | 20 sekúnd | 2 | nespadol | // |
| **3.** | 0,1 m/s | 20 sekúnd | 2 | nespadol | // |
| **4.** | 0,1 m/s | 100 sekúnd | 10 | nespadol | 0,4 m |
| **5.** | 0,1 m/s | 100 sekúnd | 10 | nespadol | 0,45 m |
| **Priemer:** | 0,1 m/s | | | 100 % | 0,425 m - 2,43° |

**Tabuľka 7.4** Hodnoty testovania chôdze walk_slow2

**walk_turn_left**

Rovnaká rýchlosť ako pri walk_forward , približne 72 krokov na urobenie otočky o 360 stupňov (jeden krok je cca 5° stupňov), stabilita asi 90 %, jedno kolo trvá spraviť približne 55 sekúnd.

**walk_turn_right**

Rovnaká rýchlosť ako pri walk_forward , približne 72 krokov na urobenie otočky o 360 stupňov (jeden krok je cca 5° stupňov), stabilita asi 90 %, jedno kolo trvá spraviť približne 55 sekúnd.

**walk_turbo**

V rámci úlohy bola prebraná chôdza z diplomovej práce Bc. Peter Paššáka. Daná chôdza sa vyznačuje dvojnásobnou rýchlosťou ako doteraz najrýchlejšia chôdza v implementácií. Problémom je zastavovanie, kedy pre chýbajúcu ukončovaciu fázu pohybu agent vždy spadne. Pre mimoriadne vysokú rýchlosť chôdze sa chôdza bude aj napriek istoty pádu oplatí využiť na dlhé vzdialenosti. Parametre chôdze sú turbo_walk(20130414_054044_1634):

- turbo_walk(20130414_054044_1634)
- priemerná rýchlosť: 0,68 m/s
- priemerné vychýlenie: 3,88 °
- úspešnosť: 100 %

## 7.3 Refaktor highskillov

## Lokalizácia lopty

**Problém:**

Hráč sa pri pokuse o lokalizáciu lopty zasekáva. Pri volaní highskill-u Localize sa zároveň vytvára aj lambda funkcia, ktorá sa vráti výnimku. Highskill Localize sa teda tiež nezavolá.

Ďalším problémom je, že v samotnom highskill-e Localize sa zle kontroluje, ci agent náhodou počas vykonávania nespadol. Namiesto „kontroly“, či spadol, sa „nastavuje“, že spadol (namiesto funkcie *agentModel.isFalled()* sa vola funkcia *agentModel.falled() )*.Počas riešenia tohto problému sa ešte zistilo, že spomínaná funkcia *agentModel.falled()* je nesprávne použitá aj v iných highskill-och a plánovačoch.

**Riešenie:**
Daná lambda funkcia bola odstránená. Namiesto toho sa priamo v highskill-e Localize kontroluje, či agent nespadol pomocou funkcie *agentModel.isOnGround()*. Rovnako aj v ostatných plánovačoch a highskill-och bola funkcia *agentModel.falled()* nahradená funkciou *agentModel.isOnGround()*.

## Beam a GetUp

**Problém:**
Umiestnenie hráča pred začatím polčasu (Beam) nie je po refaktoring-u highskill-ov funkčné. Rovnaký problém ako pri lokalizovaní lopty, chybná funkcia lambda, ktorá bola vytváraná pri volaní highskill-u Beam spôsobila, že sa tento highskill nevykonal.

**Riešenie:**
Daná lambda funkcia bola odstránená. Taktika *AttackLeft* bola doplnená o korektné volanie highskill-u Beam. V rámci tohto problému bol aj vyriešený problém s nesprávnym vkladaním highskill-u GetUp do fronty. Aby bolo možné vkladať highskill-y aj na začiatok fronty, jej typ bol zmenený z *Queue* na *BlockingDequeue*. Pre potreby správneho umiestnenia hráča pred začatím bol upravený aj hlavný plánovač highskill-ov *HighSkillPlanner*.

# 8 Siedmy šprint

## 8.1 Vytvorenie anotácií pre rôzne druhy kopov

Úloha nadväzuje na úlohu *Implementácia anotácií*, ktorá bola riešená v rámci 6. šprintu. Cieľom bolo modifikovať anotácie tak, aby do nich bolo možné zaznamenať relevantné údaje o kopoch, a tiež tieto anotácie vytvoriť pre jednotlivé kopy.
Do anotácií boli pridané nasledujúce parametre:

- *variance* – rozptyl – určuje, aký je uhlový rozptyl smeru kopnutej lopty v stupňoch
- *deviation* – odchýlka – určuje, aký je uhol medzi natočením agenta a smerom kopnutej lopty v stupňoch
    - *min* – minimálna nameraná odchýlka
    - *max* – maximálna nameraná odchýlka
    - avg – priemerná odchýlka
- *kickTime* – čas potrebný na vykonanie pohybu kopnutia do lopty (samotné vykonávanie pohybu bez započítania prípravnej fázy, kedy sa agent nastavuje k lopte) v sekundách
- *kickSuccessfulness* – úspešnosť trafenia lopty agentom v percentách
- *kickDistance* – priemerná vzdialenosť, ktorú prejde lopta po odkopnutí
- *agentPosition* – pozícia agenta voči lopte, z ktorej je možné vykonať úspešne kop a očakávať výsledok podobný ako je uvedený v ostatných parametroch anotácie.
    - *x-min* – minimálna vzdialenosť od lopty
    - *x-max* – maximálna vzdialenosť od lopty
    - *y-min* – minimálne posunutie do strany
    - *y-max* – maximálne posunutie do strany

Okrem uvedených parametrov je pre kopy dôležitý aj parameter *fall* opísaný už v predošlom šprinte.
Následkom pridania nových parametrov do anotácií bolo potrebné upraviť triedy *Annotation, XMLParser, XMLCreator,* pridať triedu *AgentPosition* všetko v rámci balíku *annotation.data*. Ďalej bolo nutné modifikovať definičný súbor pre XML anotácie.
Staré a neaktuálne anotácie boli presunuté do súboru annotationOld, pre prípad možného využitia v budúcnosti.

### Namerané údaje pre jednotlivé kopy

Nasledujú namerané údaje pre jednotlivé kopy uvedené v tabuľkách 8.1-8.4. Na meranie údajov bol využitý testframework z diplomovej práce Ing. Petra Paššáka.

**kick_left_normal**

*agentPosition*

- x-min -0,15
- x-max -0,20
- y-min -0,02
- y-max -0,08

*kickSuccessfulness* 100%
*fall* 0%
*variance* 0,07264

| | deviation | kickTime [s] | kickDistance [m] |
|---|---|---|---|
| **1** | -4,91 | 1,6 | 2,00 |
| **2** | -4,35 | 1,6 | 2,03 |
| **3** | -4,30 | 1,6 | 2,04 |
| **4** | -4,91 | 1,6 | 2,04 |
| **5** | -4,46 | 1,6 | 2,04 |
| **Priemer:** | -4,586 | 1,6 | 2,03 |

**Tabuľka 8.1** Hodnoty testovania kopu kick_left_normal

*deviation*
- min -4,30
- max -4,91

**kick_right_normal**
*agentPosition*
- x-min -0,15
- x-max -0,20
- y-min 0,02
- y-max 0,08

*kickSuccessfulness* 100%
*fall* 0%
*variance* 0,067856

| | deviation | kickTime [s] | kickDistance [m] |
|---|---|---|---|
| **1** | 4,30 | 1,6 | 2,03 |
| **2** | 4,89 | 1,6 | 2,02 |
| **3** | 5,03 | 1,6 | 2,03 |
| **4** | 4,91 | 1,6 | 2,00 |
| **5** | 4,93 | 1,6 | 2,04 |
| **Priemer:** | 4,812 | 1,6 | 2,024 |

**Tabuľka 8.2** Hodnoty testovania kopu kick_right_normal

*deviation*
- min 4,30
- max 5,03

**kick_left_faster**

*agentPosition*

- x-min -0,15
- x-max -0,25
- y-min -0,02
- y-max -0,08

*kickSuccessfulness* 100%

*fall* 0%

*variance* 1,490865

| | deviation | kickTime [s] | kickDistance [m] |
|---|---|---|---|
| **1** | -4,97 | 1,5 | 2,98 |
| **2** | -5,06 | 1,5 | 3,77 |
| **3** | -4,36 | 1,5 | 2,95 |
| **4** | -4,87 | 1,5 | 3,68 |
| **5** | -4,59 | 1,5 | 3,06 |
| **6** | -5,96 | 1,5 | 3,72 |
| **7** | -4,18 | 1,5 | 2,95 |
| **8** | -8,19 | 1,5 | 3,72 |
| **9** | -6,21 | 1,5 | 3,69 |
| **10** | -3,76 | 1,5 | 2,92 |
| **Priemer:** | -5,215 | 1,5 | 3,344 |

**Tabuľka 8.3** Hodnoty testovania kopu kick_left_faster

*deviation*

- min -3,76
- max -8,19

**kick_right_faster**

*agentPosition*

- x-min -0,15
- x-max -0,25
- y-min 0,02
- y-max 0,08

*kickSuccessfulness* 100%

*fall* 0%

*variance* 1,490865

*kickTime [s]* 1,5

*kickDistance [m]* 3,334

*deviation*

- min 3,76
- max 8,19
- avg 5,215

**kick_step_strong_left**

*agentPosition*

- x-min -0,25
- x-max -0,30
- y-min -0,05
- y-max -0,05

*kickSuccessfulness* 100%

*fall* 13%

*variance* 2,879706

| | deviation | kickTime [s] | kickDistance [m] |
|---|---|---|---|
| **1** | -7,93 | 2,8 | 5,01 |
| **2** | -7,82 | 2,7 | 4,95 |
| **3** | -7,06 | 2,7 | 4,89 |
| **4** | -8,04 | 3,0 | 4,62 |
| **5** | -6,44 | 2,7 | 5,0 |
| **6** | -10,48 | 3,0 | 4,57 |
| **7** | -10,72 | 2,9 | 5,15 |
| **8** | -11,31 | 2,9 | 5,87 |
| **9** | -10,32 | 2,7 | 5,71 |
| **Priemer:** | -8,9022 | 2,8222 | 5,0855 |

**Tabuľka 8.4** Hodnoty testovania kopu kick_step_strong_left

*deviation*

- min -6,44
- max -11,31

**kick_step_strong_right**

*agentPosition*

- x-min -0,25
- x-max -0,30
- y-min 0,05
- y-max 0,05

*kickSuccessfulness* 100%

*fall* 13%

*variance* 2,879706

*kickTime [s]* 2,8222

*kickDistance [m]* 5,0855

*deviation*

- min 6,44
- max 11,31
- avg 8,9022

## 8.2 Mocup taktika

Pri optimalizácií highskillov nám prišlo veľmi vhodné implementovanie funkcionality zachovania vykonávania rovnakej taktiky, aby sa nám taktika neustále nemenila a vedeli by sme jednotlivé highskilly lepšie optimalizovať.

Implementácia bola vykonaná tak, že sa v triede SelectorController doplnil if, ktorý kontroluje, či je v nastaveniach - trieda Settings, hodnota debugTactic nastavena na true. Ak áno, vykonáva sa metóda selectTacticFromSettings, ktorá načíta hodnotu debuggingTacticName z nastavení a ako taktiku pri každom volaní metódy controlTactics vyberie taktiku definovanú v nastaveniach.

## 8.3 Zmena plánovača

Počas siedmeho šprintu bola vykonávaná výraznejšia reimplementácia. Triedy z balíka sk.fiit.jim.plan - jednotlivé plány sa už nebudú v projekte používať, vzhľadom na to, že sme implementovali v spolupráci s tímom tp04 taktickú i strategickú vrstvu. Vzhľadom k tomu bol i kvôli lepšej prehľadnosti prepísaný pôvodný plánovač v triede Plan do dvoch tried - HighskillPlanner a HighSkillRunner. Implementácia plánovača je podobná ako v predchádzajúcom riešení. Oproti pôvodnému sme použili inú štruktúru na rad highskillov pre vykonávania - ArrayList sme vymenili za obojsmerný rad - LinkedBlockingDeque. Umožnilo nám to pridávať požiadavky ako na začiatok radu, tak i na koniec radu. Obrovskou výhodou je, že highskilly ako GetUp vieme zaradiť na začiatok radu - aby sa vykonávali prioritne. V plánovači pribudli metódy na ukončenie aktuálne vykonávaného highskillu a logovanie informácii o aktuálne vykonávaných highskilloch v plánovači. Plánovač bol presunutý do balíka sk.fiit.jim.agent.highskill.runner.

## 8.4 Pridávanie a vykonávanie highskillov

Požiadavky na vykonávané highskilly posiela do plánovača taktická vrstva pomocou metódy addHighskillToQueue. Pokiaľ ide o kopy alebo pohyby, posiela tieto požiadavky cez metódy MovementHighSkill.move(Vector3D position, MovementSpeedEnum speed) , resp KickHighSkill.kick(KickTypeEnum kickType) . Na tieto metódy môžme nazerať ako na metódy sprostredkujúce rozhranie medzi taktickou a highskill vrstvou. Taktická vrstva by nemala vedieť príliž interné detaily o highskill a lowskill vrstve a i preto ako argument nám posiela rýchlosť, ktorá predstavuje enum MovementSpeedEnum - pomalá, stredne rýchla, rýchla. Metóda move obsahuje v súčasnosti iba 2 argumenty - vektor cieľa - kam má agent prísť a enum - rýchlosť. V budúcnosti vďaka tomuto prístupu bude relatívne jednoduché rozšíriť metódu o ďalšie atribúty, popr hashmapu s atribútmi. Metóda na základe dodaných atribútov hľadá na základe anotácii najvhodnejší highskill - počíta tzv. fitness. Fitness sa počíta v metóde getWalkSuitability. Fitness sa počíta z troch vlastností - minimálnej vzdialenosti, na ktorú je highskill možné použiť, rýchlosti a stability. Tieto tri vlastnosti sú vo vzorci navzájom násobené. Vychádza sa z predpokladu, že rýchla chôdza nemôže byť použitá

na veľmi krátku vzdialenosť a podobne pomalá chôdza na veľmi dlhú. Implementované boli highskilly WalkFast, WalkMediu a WalkSlow, ktoré dedia od highskillu Walk, čím je ich funkcionalita rovnaká, až na výber lowskillu, ktorý vyberajú na základe rýchlosti, ktorou majú kráčať.

Ak bude treba pridať nový highskill a bude typu walk, bude musieť byť pridaný do enumu MovementEnum a musí mať v enume implementovanú metódu computeSuitability. Výhoda computeSuitability v enumoch spočíva v tom, že každý highskill, ktorý bude zaradený do enumu a bude môcť byť volaný taktickou vrstvou bude musieť mať implementovanú metódu na výpočet vhodnosti, ale navyše každý highskill v enume ju môže mať modifikovaný - tu sa počítalo s možnosťou zakomponovania strojového učenia, kedy by sa mohla fitness počas vykonávania hry dynamicky meniť podľa úspešnosti daného pohybu.

## 8.5 Reimplementácia Beam-u

Po prepísaní plánovača Planner na HighSkillPlanner bolo úlohou prerobiť beamovanie hráča na ihrisko, ktoré nefungovalo korektne ani v predchádzajúcej verzii (verzii dodanej na začiatku semestra). Na začiatku semestra hráč pred beamovaním vykonával trhavý pohyb - bol neustále v cykle beamovaný na pozíciu -5, -1, 0. Pravdepodobne toto správanie zabezpečilo, že po postavení na ihrisko hráč nepadol. Nakoľko po prerobení plánovača nám hráč neustále padal na ihrisko po jeho postavení na ihrisko, bol prerobený highskill beam tak, aby nevolal lowskill rollback. Zakomentovaním týchto riadkov sa nám podarilo dosiahnuť korektné nastavenie hráča na zadanú polohu a hráč nezostal v rohu ihriska. V metóde controll v triede HighSkillPlaner som chcel ošetriť situáciu, kedy hráč leží na zemi - aby si naplánoval highskill na jeho postavenie. Problém je však s metódami agentModel.isOnGround() a podobne i ďalšími, ktoré majú za cieľ zistiť, či agent leží na brušku alebo na chrbáte. Problém v týchto metódach je spôsob, akým zisťujú výsledok týchto metód - z akcelerometra. Jeho hodnoty osí y a z dosť výrazne skáču a v dôsledku podmienky v metóde isOnGround je výsledok metódy true (tj že agent leží na zemi) i v prípade, že agent sa pohybuje chôdzou. V tomto dôsledku nie je úplne spoľahlivé takéto hromadné ošetrovanie, pokiaľ nebudú korektne implementované vyššie spomínané metódy. Táto úloha bola pridaná do backlogu a bude naplánovaná do ďalšieho šprintu.

Proces beamovania je implementovaný momentálne tak, že v konštruktore triedy HighSkillPlanner sa vkladá do obojsmerného radu highskill Beam . To, že je naplánovaný v konštruktore zaručí, že bude to radu vložený ako prvý highskill. Do tohto radu je však vložený iba vtedy, ak je mód hry beamablePlayMode , tj. ak je Jim spustený až po tom, čo bol beamnutý. Zároveň v metóde addHighskillToQueue (metóda, ktorou vyššia vrstva - taktika vkladá highskilly do queue) bolo ošetrené vkladanie highskillov z vyššej vrstvy. Do queue nebudú vkladané highskilly na vykonávanie, pokiaľ nie je hra v móde beamablePlayMode.

Zároveň počas tohto procesu implementácie bola implementovaná funkcionalita na výpis aktuálne spracovávaného highskillu.

# 9 Ôsmy šprint

## 9.1 Build závislosti medzi projektmi

Viacero používateľov používa viaceré IDE - Netbeans, Eclipse, IdeaJ a pre každé IDE sa nachádzali v Gite pomocné súbory. Tieto súbory sme odstránili a zaznamenali do .gitignore súboru. Taktiež sme vytvorili .project.sample súbor - vzorový project súbor pre Eclipse. Pri iniciálnom pullnutí Git repozitáru preto programátor si musí skopírovať .project.sample súbor na .project a až potom spustiť Eclipse. Tento postup sa volil preto, lebo niektorí členovia tímu cez Eclipse mali nainštalovanú inú verziu (iný zdroj) pre knižnicu aspektov aká je použitá v projekte a potrebovali používať tú vzhľadom na ich inštaláciu systému. Takto si ktokoľvek môže prispôsobiť závislosti projektu bez zmeny, ktorá by ovplyvnila nastavenia ostatných členov tímu.

Ďalej sme odstránili závislosti v robocup library, ktoré boli viac-menej duplicitné s Jimom. Odstránili sme i staré závislosti - na Ruby - ktoré už neboli vzhľadom na odstránenie Ruby ďalej v projekte potrebné.

Ďalšou súčasťou upratovania projektu bolo presunutie nepotrebných balíkov - tried, ktoré sa v projekte už nepoužívajú. Nebolo požadované ich zmazanie, nakoľko sa môžu budúcim tímom zísť ako inšpirácia. Išlo hlavne o balík Plan, Build, Review ktoré boli presunuté do balíka Garbage. Odstránený bol i bývalý Planner.

## 9.2 Optimalizácia implementácia highskillov

V rámci šprintu sa ďalej pokračovalo na optimalizácií implementovaných highskillov. Pri tejto úlohe sa podarilo odhaliť závažný problém s lokalizáciou agenta v prípade, že nemá v zornom poli aspoň tri pevné body, podľa ktorých by sa mohol lokalizovať agent pokračuje vo vykonávaní pohybu do nekonečna alebo do skončenia hracieho času. Oprava tohto problému bola naplánovaná na nasledujúci šprint.

## 9.3 Aktualizácia wiki - stránky o anotáciách

Stránky venujúce sa problematike anotácií boli v rámci ôsmeho šprintu zaktualizované nakoľko v predchádzajúcich šprintoch sa vykonali výrazné zásahy do ich implementácie.

# 10 Deviaty šprint

## 10.1 Dokumentácia high skillov

### Nová architektúra

Pri zmene správania hráča bolo nutné prispôsobiť spúšťanie a vykonávanie jednotlivých vyšších schopností(highskills), tak aby bolo umožnené v novo vytvorených taktikách sprístupniť volanie týchto pohybov.

Celý proces úprav spočíval v analýze existujúceho kódu a jeho úprave do vhodnejšieho tvaru a následnom vytváraní nového kódu slúžiaceho na obalenie existujúcich štruktúr do univerzálnejších konštrukcií.

### Plánovač vyšších schopností

V prvom rade bolo nutné upraviť plánovač vyšších schopností. Jeho úpravou sme docielili oddelenie plánovania schopností od ich vykonávania.

V pôvodnej verzii fungovalo plánovanie schopností iba v tom prípade ak zoznam naplánovaných položiek bol prázdny. Vďaka tejto vlastnosti plánovača sa dala naplánovať práve iba jedna položka, ktorá sa následne vykonala.

V novej verzii plánovača sme vykonali úpravy spojené s možnosťou plánovania viacerých položiek do zoznamu a následne postupným vykonávaním naplánovaných akcií. Popri tejto úprave bolo nutné pridať možnosť automatického zastavenia aktuálne vykonávanej schopnosti a vyčistenia zoznamu naplánovaných akcií. Ďalším upravením plánovača bolo vloženie implicitného vstávania hráča zo zeme. Ak nastal stav, v ktorom sa agent nachádzal na zemi, tak automaticky bolo zrušené vykonávania aktuálnej schopnosti a bolo naplánované vstávanie zo zeme.

### Modul chôdza

Táto časť architektúry vznikla ako úplne nový celok, ktorý zjednocuje spôsob výberu chôdze zo všetkých dostupných druhov chôdze pre agenta v najaktuálnejšej verzii.

V pôvodnom riešení z ktorého vychádzame bolo nutné vyberať ručne najvhodnejší pohyb a ten implementovať v plánovači. Vďaka tomuto sa zabúdalo na vylaďovanie zvyšných pohybov, ktoré sa nepoužívali. Ďalším problémom bol aj výber nevhodného typu pohybu pri určitých situáciách.

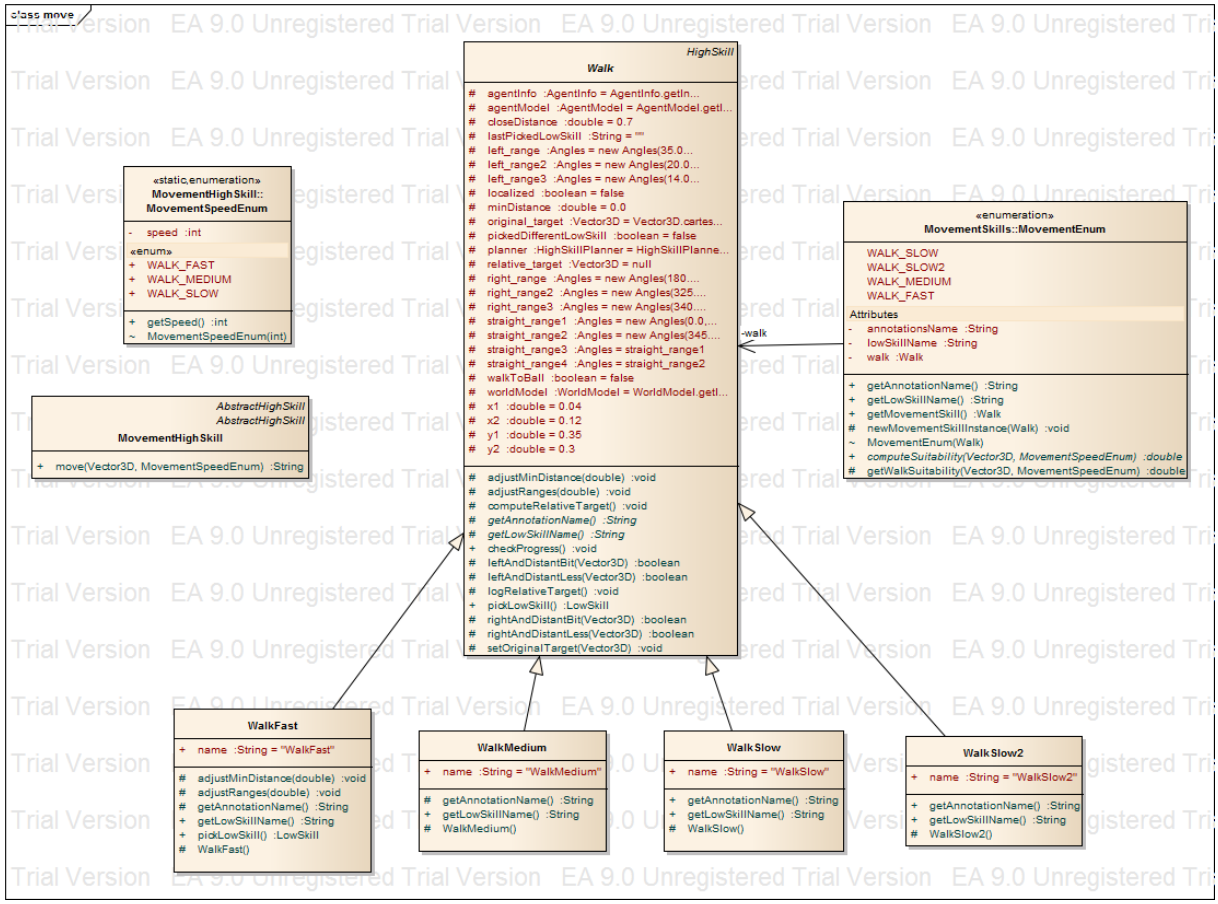
V našom novom riešení má agent zadefinovaný konečný zoznam pohybov, ktoré vie vykonávať. Z tohto preddefinovaného zoznamu pohybov si vyberá najvhodnejší pohyb na základe atribútov poskytnutých z taktiky a na základe anotácií (štatistík) jednotlivých

pohybov. Pri výbere pohybu sa vyhodnocuje vhodnosť všetkých dostupných pohybov a na základe výsledkov sa vyberá pohyb, ktorý dosiahol najvyšší výsledok.

Všetky z preddefinovaných pohybov zo zoznamu majú spoločnú rodičovskú triedu, v ktorej sú zadefinované spoločné atribúty a metódy. Táto rodičovská trieda implementuje rozhranie, ktoré definuje správanie každého vyššieho pohybu. V tejto triede je taktiež implementovaná všeobecná metóda na výber nižšej schopnosti, ktorá je použiteľná pre všetkých potomkov. Následne v jednotlivých triedach pohybov je dôležité zadefinovať názov pohybu, slúžiaci pri vyhľadávaní anotácií a nižších pohybov. V každej triede je následné možné preťažiť metódu výberu pohybov a zadefinovať vlastné vhodnejšie správanie.

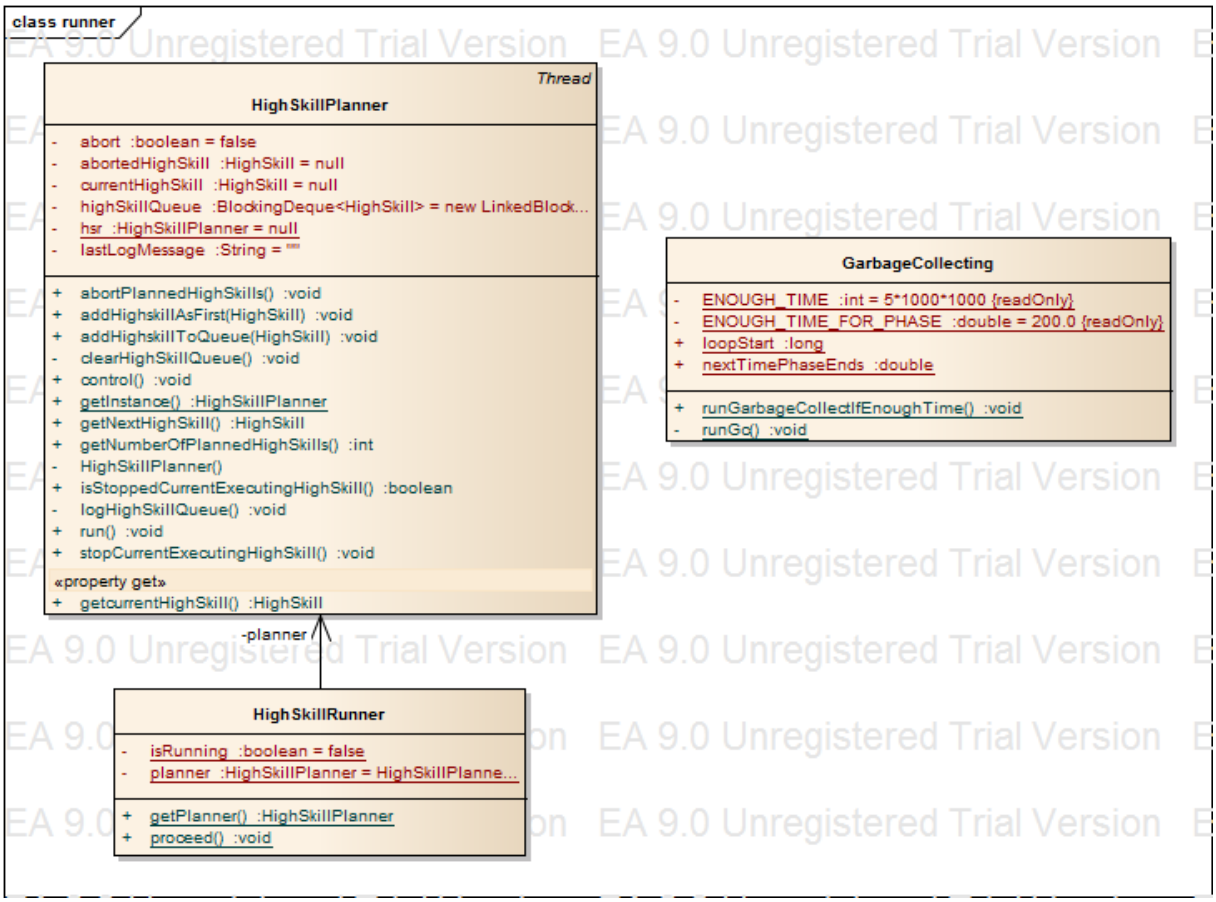
## Architektúra - diagramy tried

Na obrázkoch 10.1 – 10.3 sú zobrazené diagramy tried architektúry pre high skill kop (kick), chôdzu (move) a plánovac týchto high skillov.




**Obrázok 10.1** architektúra chôdze


**Obrázok 10.2** architektúra kopu




**Obrázok 10.3** architektúra plánovača high skillov

## 10.2 Oprava výpočtu či agent leží na zemi

V projekte sme identifikovali problém, že agent nesprávne identifikuje, či leží. V metóde načítaval údaje z akcelerometra , konkrétne *z-ovú* os. Tú porovnával s polovicou gravitačného zrýchlenia a ak bola aktuálna hodnota z *os-y* menšia, tak prehlásil že agent leží. Problém však nastával, že časo i keď agent stál, metóda prehlásila, že leží na zemi. Údaje z akcelerometra skáču veľmi rozdielne a nedajú sa spoľahlivo použiť na získanie či leží, alebo nie. Preto som volanie *isOnGroundJudgedByAccelerometer()* v metóde *isOnGround()* odstránil a nechal iba výpočet v *else* vetve, ktorí zisťuje polohu na základe rotácií vektorov. Výpočet je výrazne spoľahlivejší a pokiaľ agent stojí, alebo beží, tak ho už ďalej neidentifikuje ako ležiaceho na zemi.

## 10.3 Výpočet pozície agenta

Pri preimplementovaní highskillov chôdze agenta pre účely vylepšenia softvérovej architektúry rozhodovania agenta, sme narazili na problém s výpočtom absolútnej pozície agenta, ktorá je esenciálna pre každý typ chôdze na určenú absolútnu pozíciu ihriska. Každá chôdza potrebuje mať jasne stanovený relatívny cieľ vzhľadom na pozíciu k agentovi. Chôdza na určitú pozíciu si tento relatívny cieľ vypočíta na základe absolútneho cieľa, zadaného z vyššej vrstvy riadenia, a spomínanej absolútnej pozície agenta.

Tento problém bol objavený až v tejto fáze z toho dôvodu, že väčšina pôvodných plánovačov, používala iba chôdzu za loptou, ktorá nie je ovplyvnená problémovým výpočtom pozície agenta, kedže si nemusí vypočítavať potrebný relatívny cieľ. Relatívny cieľ, teda lopta, je priamo získaný z údajov preceptoru videnia, bez dodatočného prepočítavania.

Konkrétny problém spočíval v tom, že po istej dobe sa agentovi úplne prestala aktualizovať pozícia. Z toho dôvodu sa práve vykonávaná chôdza na určitú pozíciu nikdy nie len že nedosiahla cieľovú pozíciu, ale ani sa neukončila. Okrem tohto konkrétneho problému sa často vyskytoval problém s rôzne veľkými odchýlkami vypočítanej pozície agenta.

Prvotným pokusom o zlepšenie výpočtu pozície bolo pridanie dodatočných lokalizácií (teda otočení hlavou na obe strany) počas vykonávania chôdze, kedy by agent mal spozorovať viacero vlajok (záchytných bodov, pomocou ktorých vypočítava svoju pozíciu). Tento pokus sa však ukázal ako neúspešný, pretože agent síce spozoroval viacero vlajok, pozícia sa však korektne neprepočítala.

Pri hlbšom preskúmaní problému sa zistilo, že už od začiatku riešenia tímového projektu je pri prepočítavaní pozície agenta využívaná podmienka

*Math.abs(pos.getXYDistanceFrom(agent.position)) < 1,*

ktorej cieľom je odfiltrovať príliš vysoké odchýlky prepočítanej pozície. Jej princípom bolo, že ak je aktuálne vypočítaná pozícia rozdielna o viac ako jeden meter od pôvodne vypočítanej pozície, tento výpočet sa zahodí. Ľahko sa však napríklad pri rýchlej chôdzi mohlo stať, že

novo-vypočítaná pozícia bol viac krát po sebe zahodená a agent už bol príliš ďaleko na to, aby sa aspoň raz splnila uvedená podmienka. Ako riešenie sme teda zabezpečili, aby sa po vopred stanovenom počte zahodení výrazne vychýlených pozícii, táto pozícia aktualizovala. Ako najúčinnejší sa javí počet 10-tich povolených zahodení pozície.

Na eliminovanie problému, kedy agent nie je otočený ani na súperovu stranu, ani na domácu stranu, čo znamená že vidí najmeší možný počet vlajok, bolo pridaná kontrola času, kedy bola naposledy videná vlajka. V prípade, že je prekročený vopred stanovený čas, agent sa otočí o 45° doprava, až dokým neuvidí aspoň jednu vlajku. Pre tieto účely bolo do metódy, v ktorej sa prepočítava pozícia agenta, pridané aj aktualizovanie času poslednej videnej vlajky.

Popri riešení bolo ešte zabezpečené, aby sa pozícia neprepočítavala počas "beamovacie" módu. Pozíciu je totiž možné aktualizovať priamo z highskillu Beam, ktorý ma presné informácie o určenej pozícii. Rovnako bola ešte pridaná metóda MovementHighSkill.move(), ktorá príjma iba jeden parameter požadovanej rýchlosti chôdze, a na rozdiel od rovnomennej metódy s dvoma parametrami, slúži na chôdzu za loptou (nie na konkrétnu pozíciu).

# A Celkový pohľad

## A-1 Úvod

Táto téma, respektíve projekt RoboCup, je vyvíjaný na našej fakulte niekoľko rokov rôznymi študentmi, a teda je už vo funkčnom stave, čím sa podstatne líši od ostatných tém tímových projektov. Práca viacerých ľudí sa odzrkadlila na konzistentnosti implementácie a jej dokumentácie.

Pre doterajších riešiteľov, či už v podobe jednotlivcov alebo viacerých tímov, nebolo jednoduché detailnejšie sa oboznámiť s rôznymi rozsiahlymi implementovanými princípmi, ktorých nie je málo, a ktoré zasahujú do viacerých vedných oblastí. Samotná logika rozhodovania a ovládania hráča v robotickom futbale je rozsiahla a náročná úloha, priamo úmerne rastúca s ambíciami čo najviac sa priblížiť podobe reálneho futbalového zápasu. Ambície každého z riešiteľov pochopiteľne postupne rástli, čím vznikali rôzne vylepšenia daného riešenia, ktoré sa malými krokmi stále viac a viac približuje spomínanej najvyššej ambícii. Rovnako ambiciózne ciele nášho tímu však narazili na prekážku, ktorú stavia architektúra súčasnej implementácie.

Hlavným cieľom projektu sa teda po prvých dvoch šprintoch (zameraných na oboznámenie sa s riešením) stala komplexná revízia súčasnej architektúry za účelom dosiahnutia vylepšení v podobe strategickejšieho a taktickejšieho plánovania a rozhodovania hráča (*tabuľka A.1*). Konkrétne vylepšenia, ktoré poháňajú vznik novej architektúry sú popísané v ďalších častiach. Revízia architektúry má však za účel nielen vylepšenie hráča, ale jej nepriamym cieľom je aj odstránenie spomínanej nekonzistentnosti implementácie a jej dokumentácie.

Pred začiatkom riešenia tímového projektu boli identifikované rôzne ciele projektu (viď. kapitola *1 Úvod*). Niektoré úlohy komplexnejšie, rozsiahlejšieho charakteru, od ktorých závisia ďalšie úlohy a niektoré menej komplexné, nenáročné, nezávislé od tých komplexných. Vzhľadom na dôležitosť komplexných úloh, bola väčšine menej komplexných znížená priorita.

Ďalšie časti celkového pohľadu obsahujú zhrnutie obsahu šprintov vrátane konkrétnych dôvodov pre zmenu návrhu architektúry, vrátane opisu pôvodného a súčasného stavu architektúry.

| Šprint | Hlavné úlohy šprintu |
|---|---|
| **1. a 2.** | Oboznámenie sa s riešením a refaktoring |
| **3., 4., a 5.** | Návrh a implementácia prototypu architektúry |

**Tabuľka A.1** Hlavné úlohy šprintov

## A-2 Oboznámenie sa s riešením

Úlohy prvých dvoch šprintov boli zamerané na oboznámenie sa s aktuálnym stavom projektu. Okrem inštalovania potrebných nástrojov a študovania dostupnej wiki dokumentácie projektu, šprinty obsahovali ďalšie úlohy na overenie získaných poznatkov vytvorením unit testov zvolených častí implementácie ako aj zmenením správania hráča, napríklad úpravou jeho rozhodovania, alebo jeho pohybu. Poznatky pri inštalovaní potrebných nástrojov sú odzrkadlené v doplnenej, resp. upravenej wiki dokumentácii.

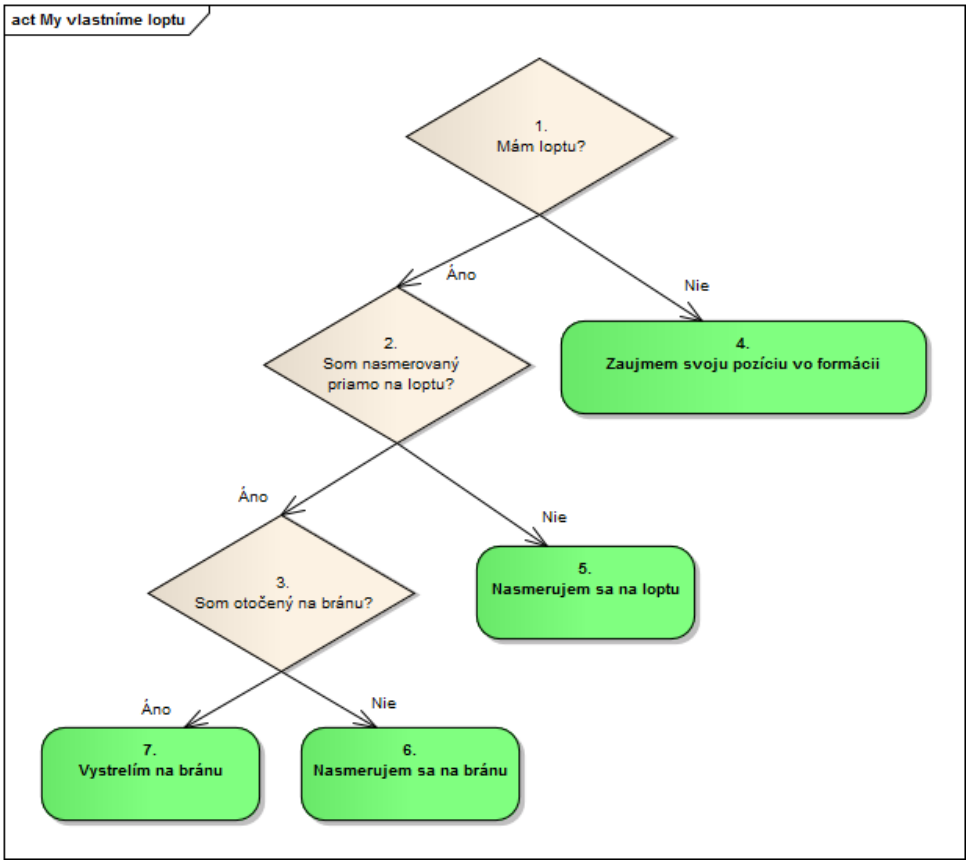
Takto získané poznatky však ešte nie sú postačujúce na širšie oboznámenie sa s projektom a RoboCup-om ako takým, preto bolo v rámci úvodných šprintov naplánované aj analyzovanie posledných úspešne ukončených diplomových prác, ktorých zaujímavé a užitočné výsledky ešte neboli zakomponované do súčasnej implementácie, ktorú sme mali k dispozícii. Rovnako bolo ešte potrebné oboznámiť sa s existujúcimi riešeniami iných zahraničných tímov, ktoré riešia niektoré problémy lepšie a istým spôsobom nás tak navádzajú na správnu cestu pri vylepšovaní vlastného riešenia.

Už počas prvého šprintu sme si však uvedomili veľkú nevýhodu, ktorú súčasný projekt mal. Podstatná časť rozhodovania a pohybovania agenta bola napísaná v inom jazyku, ako väčšina implementácie. Preto bola ešte v rámci oboznamovania sa projektom, v druhom šprinte, táto časť kódu prepísaná. Konzistentnosť riešenia sa zlepšila a z vlastnej skúsenosti veríme, že to napomohlo lepšiemu pochopeniu riešenia nie len našim členom tímu, ale i ďalším budúcim riešiteľom.

Všetky tieto úlohy zvýšili potrebné povedomie tímu na následné zahájenie pokusov o nejaké vylepšenie, ktoré v našom prípade viedlo k spomínanej zmene architektúry.

## A-3 Motivácia k zmene architektúry

Na *obrázku A.1* je ukážka pôvodného stavu rozhodovania sa agenta. Takéto jednoduché rozhodovanie podmienkami “if-then” bolo implementované v tzv. plánovačoch. Svojím spôsobom sa jedná a funkčné riešenie, ktorým je možné dosiahnuť rôzne zložité ciele, avšak komplexnosť rozhodovania hráča v reálnom futbale siaha oveľa vyššie. Je pravda, že pôvodný stav by teoreticky umožňoval realizovať aj komplexnejšie rozhodovanie, no dané rozhranie plánovača jednoducho nebolo pripravené na zložitejšie logické konštrukcie, ktoré vyplývajú z reálneho plánovania akcií hráčov.




**Obrázok A.1** Ukážka rozhodovania agenta v plánovači

Pod komplexným rozhodovaním si môžeme predstaviť rozhodovanie, ktoré je z pohľadu klasického futbalu a iných tímových športov bežné a tým je rozdelenie rozhodovania na vyššie a nižšie úrovne. Pod vyšším rozhodovaním je možné si predstaviť stratégiu, ktorú má daný tím určenú napríklad už pred začatím stretnutia a pod nižším rozhodovaním je možné si predstaviť konkrétne taktiky, ktoré daný tím vykonáva počas hry. Výber taktík tímu (nižšie rozhodovanie) je závislé práve od použitej stratégie (vyššie rozhodovanie).

Takéto štruktúrovane rozhodovania na stratégie a taktiky nielenže uľahčí riešiteľom vytváranie nového a čitateľnosť existujúceho rozhodovania, ale otvorí i oveľa väčšie možnosti pre reagovanie na rozličné situácie hry pomocou rozličných taktík. Logika rozhodovania by už teda nebola reprezentovaná neprehľadnou formou v podobe rozhodovacích stromov, ale v prirodzenejšej forme, zodpovedajúcej najvyššej ambícii (spomenutej aj v úvode pohľadu) ktorou je rozhodovanie skutočných hráčov futbalu. Cieľom je teda vytvorenie menej obmedzujúceho a otvorenejšieho rozhrania pre plánovanie, rozhodovanie a vykonávanie pohybov nášho agenta.

## A-4 Stav architektúry po zimnom semestri

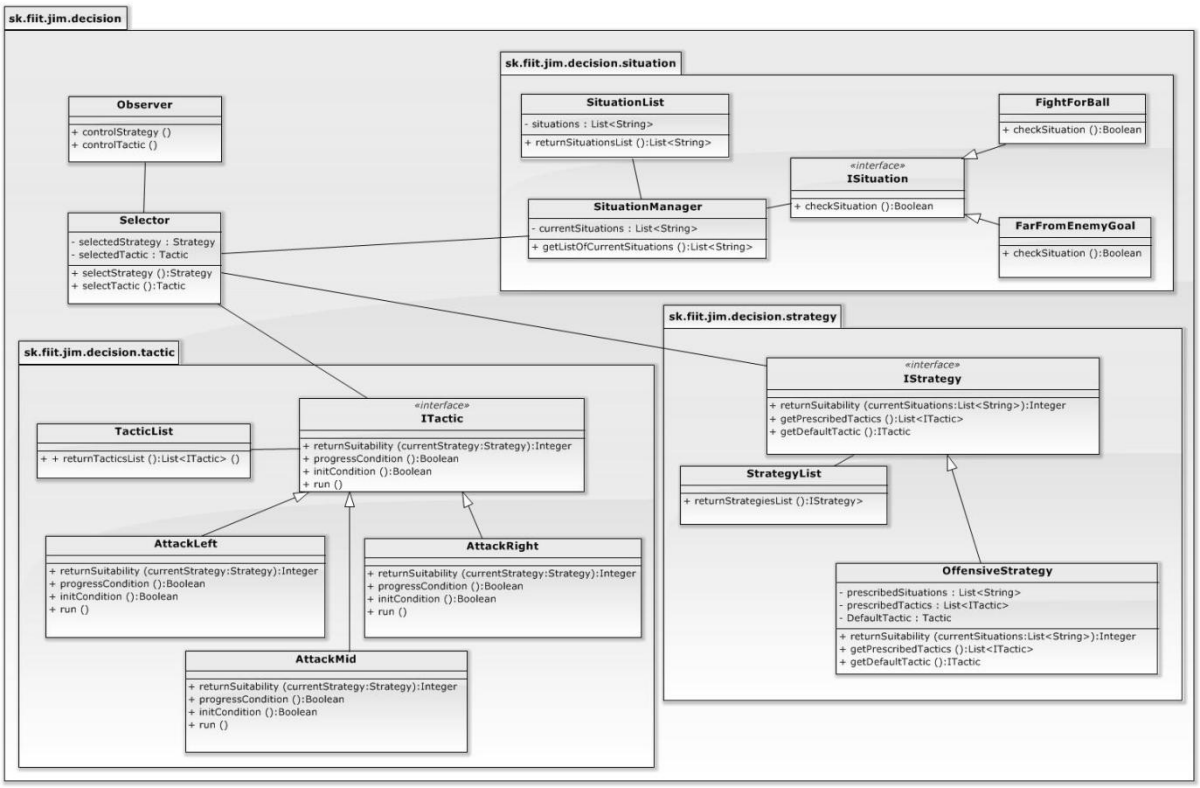
Súčasná architektúra je zobrazená na Obrázku A.2. Diagram tried súčasnej architektúry. Pre priblíženie popíšeme iba jednotlivé balíky.

Prvým a najdôležitejším balíkom je balík situation obsahujúci všetky situácie, ktoré sú potrebné pre určenie aktuálneho stavu hry. Aktuálny stav hry je určený podmnožinou zo všetkých situácií, ktorú vytvára SituationManager na základe pravdivosti tvrdenia ich nastatia v situáciách.

Ďalším balíkom je balík Strategy obsahujúci všetky stratégie hry. Každá stratégia definuje prípustné situácie, pri ktorých jej vykonávanie má význam. Ďalej definuje zoznam taktík, ktoré sa majú v rámci danej stratégie vykonávať.

Posledným balíkom je balík Tactic. Tento balík obsahuje všetky vhodné taktiky pre definované stratégie. Podobne ako stratégie, taktiky definujú prípustné situácie, pri ktorých má ich vykonávanie význam. Ďalej každá taktika bližšie popisuje vlastnosti vyšších schopností, na základe ktorých sa vyberajú jednotlivé nižšie schopnosti. Príkladom takéhoto popisu je rýchlosť, stabilita a presnosť jednotlivých schopností. Pri výbere jednotlivých vyšších schopností majú veľký význam aj situácie, na základe ktorých sa tieto schopnosti vyberajú.

Dôležitou súčasťou architektúry je trieda Selector, ktorá zabezpečuje výber najvhodnejších stratégií a taktík podľa vyhodnotených situácií. Táto trieda slúži ako pozorovateľ, ktorý sleduje zmeny vo vyhodnotených situáciách a kontroluje, či je aktuálne vybraná taktika alebo stratégia stále platná.




**Obrázok A.2** Diagram tried súčasnej architektúry

## A-5 Používateľská príručka

K projektu je priebežne udržiavaná používateľská príručka s návodmi na inštaláciu a spustenie projektu. Používateľská príručka je dostupná na adrese wiki http://team09-13.ucebne.fiit.stuba.sk/wiki , prípadne zazipovaná v odovzdaných súboroch ako export databázy. Ide o wiki, ktorá je prevzatá od minuloročných tímov, ktorí tvorili tento projekt. Priebežne je nami aktualizovaná, boli do nej zaznamenané postupy, ktorými sme odstránili vyskytujúce sa problémy pri spúšťaní projektu na rôznych platformách a zaznamenali nami vykonané zmeny.

Návod na inštaláciu a spustenie projektu je dostupný na adrese http://team09-13.ucebne.fiit.stuba.sk/wiki/index.php/N%C3%A1vody_a_in%C5%A1tal%C3%A1cie .

Princípy fungovania projektu sú dostupné na url stránke wiki http://team09-13.ucebne.fiit.stuba.sk/wiki/index.php/%C3%9Avod_do_RoboCup_na_FIIT

Pri tvorbe projektu sme sa riadili metodikou písania zdrojových kódov dostupnou na adrese http://team09-13.ucebne.fiit.stuba.sk/downloads/Metodika_pisania_zdrojovych_kodov.docx, ktorou je vhodné sa riadiť i pri pokračovaní vo vývoji ďalšími tímami.