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

Tímový projekt – RoboCup 3D

Dokumentácia k inžinierskemu dielu

Študijný odbor: Informačné systémy, Softvérové inžinierstvo
Predmet: Tímový projekt
Vedúci projektu: Ing. Marián Lekavý
Tím: RFC Megatroll
Členovia tímu: Vladimír Bošiak
Samuel Benkovič
Michal Petráš
Martin Adámik
Igor Homola
Michal Čerešnák
Matej Bádal

Obsah

1 Úvod

Tento dokument predstavuje dokumentáciu k inžinierskemu dielu tímového projektu RoboCup. Projekt vypracovávajú členovia tímu č. 4 – RFC Megatroll. Riešenie tohto projektu je obsahom predmetu Tímový projekt.

Dokument vytvoril celý tím a spolu s ostatnými dokumentmi patrí k internej dokumentácii tímu, slúži ale aj pre vedúceho tímu. Taktiež je využiteľný ostatnými aj budúcimi čitateľmi.

1.1 Ciele

1. Celkové odstránenie ruby závislostí z projektu
2. Návrh a implementácia novej architektúry pre strategické a taktické rozhodovanie
3. Refaktoring kódu celého projektu

1.2 Plán

1. Oboznámenie sa s projektom
2. Obnova a aktualizácia inštalačných návodov a Robocup wiki
3. Identifikovanie častí závislých na ruby
4. Zjednodušenie ruby častí
5. Úplné odstránenie ruby častí
6. Analýza diplomových prác, zahraničných tímov
7. Návrh a implementácia novej architektúry

1.3 Plán na letný semester

1. Otestovanie architektúry
2. Návrh a implementácia konkrétnych stratégií
3. Návrh a implementácia konkrétnych taktík
4. Návrh a implementácia konkrétnych situácií
5. Vylepšenie pohybových vlastností robota

2 Šprinty

V tejto časti dokumentu sú opísané všetky šprinty, ktoré prebehli počas semestra. Obsahom každého šprintu sú používateľské príbehy a úlohy k nim prislúchajúce.

2.1 Šprint č. 1

Tabuľka 1 - Backlog šprintu 1

ID	AKO	CHCEM	ABY
1.1	Člen tímu	Analyzovať zdrojové kódy	Zlepšenie prehľadu o závislostiach, prepojeniach a oboznámenia sa s riešením.
1.2	Člen tímu	Spustiť hráča	Hráč aj server fungoval správne bez akýchkoľvek chýb
1.3	Člen tímu	Zmeniť správanie hráča	Oboznámenie s pohybmi a ich riešením.
1.4	Člen tímu	Vytvoriť unit test	Otestovať metódu.
1.5	Člen tímu	Vytvoriť web	Ukladanie materiálov na jedno prístupné miesto.
1.6	Člen tímu	Získať prehľad o svetových tímoch	Predstava o dosiahnutých úspechoch a hlavne o iných riešeniach, ktoré by mohli byť inšipiratívne.
1.7	Člen tímu	Vytvoriť backlog	Pripraviť a identifikovať úlohy do plánu.

Tabuľka 2 - rozdelenie úloh v šprinte

ID Pp	ID podúlohy	Úloha	Zodpovedný
1.1	1.1.1	Analýza zdrojových kódov (JIM)	Bádal,Bošiak
	1.1.2	Analýza zdrojových kódov (TESTFRAMEWORK)	Petráš, Adámik
	1.1.3	Analýza zdrojových	Čerešňák, Homola

		kódov (ROBOCUPLIBRARAY)	
1.2	-	Spustiť hráča	Všetci
1.3	-	Zmeniť správanie hráča	Všetci
1.4	-	Vytvoriť unit test	Všetci
1.5	-	Vytvoriť web	Homola, Bádal
1.6	-	Získať prehľad o tímoch	Všetci
1.7	-	Vytvoriť backlog	Všetci

2.1.1 Analýza zdrojových kódov

2.1.1.1 Jim

- Jim
  - *AllTests* – spustí všetky testy
  - *Settings* – Inicializuje nastavenie hry, keď nie je určené inak načíta "default-né" nastavenia ()
  - *SettingsTest*– test pre class *Settings*
- Jim.agent
  - *AgentInfo* – uchováva informácie o stave hráča a jeho polohy na ihrisku (oprava komentárov funkcii niektoré nie sú pre javadoc, názvy funkcií ako "loguj" pritom má Jim logovací systém, getPlayerState – refactoring?)
  - *Planner (use RoboCupLibrary)* – nastavuje plánovanie a vykonáva daný plán (ruby)
  - *Side* – vymenovanie strán
- Jim.agent.communication
  - *Communication* – Komunikácia so serverom na najnižšej úrovni
  - *CommunicationThread*– stará trieda pre komunikáciu podľa komentárov odstrániteľná
- Jim.agent.communication.testframework
  - *Message*–komunikácia s testframework?
  - *TestFrameWorkCommunication*– komunikácia s tesframework?
- Jim.agent.models
  - *AgentModel* – Vyššie funkcie pre stav agenta a jeho pozíciu
  - *AgentPositionCalculator* – Približná poloha agenta na ihrisku podľa dostupných bodov
  - *AgentRotationCalculatori*– Približné natočenie hráča podľa dostupných bodov
  - *DynamicObject* – Výpočet polohy pohybujúceho sa objektu
  - *EnviromentModel* – uchováva stav sveta
  - *FixedObject*– uchovávanie a získavanie pozície statických objektov
  - *KalmanAdjuster* – nevieme presne čo, je tam pozícia lopty a pozícia fixných objektov
  - *Player* – Entita - Informácie o hráčovi (spoluhráč, protihráč)

    - *TacticalInfo* – Informácie o hernej stratégie (guru Samo hovorí, že nefunguje)
    - *WorldModel* – Informácie o ihrisku
- Jim.agent.models.prediction
    - *Prophecy* – pravdepodobný stav udalostí
    - *Prophet* – vypočítava najpravdepodobnejší vývoj udalostí
- Jim.agent.moves
    - *EffectorData* – entita – efektoru
    - *Joint* – entita – kĺbu
    - *JointPlacement* – uchováva konfiguráciu kĺbu
    - *LowSkill* – kolekcia fáz
    - *LowSkills*–Správa načítaných low skills
    - *Phase* – reprezentácia fázy
    - *Phases* – cache fáz
    - *SkipFlag* – reprezentácia fázy, ktorá sa má vynechať
    - *SkipFlags*- ?
- Jim.agent.parsing
    - *ForceReceptor* – Informácie o sile pôsobiacej na dolné končatiny
    - *HearReceptor* – Informácie o správach (pokrikoch)
    - *ParsedData* – Implementácia serverovej správy
    - *ParsedDataObserver* – Rozhranie pre objekt spracovania správ
    - *Parser* – Trieda pre transformáciu správ zo servera pre agenta
    - *Perceptors* – Stav hardware-u robota (gyro, natočenie a.i.)
    - *PlayerData* – zapuzdrenie informácií z preceptorov
    - *SeenPerceptor* – transformuje správy z vizuál. preceptoru
    - *SeenPerceptorData* – Zapúzdruje informácie z vizuálneho preceptoru
    - *SeePercetor* – Aktualizuje dáta perceptoru
    - *SExpression*- ?
- Jim.agent.sexp
    - *SArray* – dátová štruktúra
    - *SException* – exception
    - *SObject* – default object
    - *SString*- ?
- Jim.agent.server
    - *TFTPServer*- ?
- Jim.agent.skills
    - *ComplexHighSkill* – Manažér high skill-ov
    - *FakeHighSkill* – high skill pre testovanie
    - *HighSkill* – Wraper pre high skill
    - *I**- interface-y
- Jim.agent.trajectory
    - *Obstacles* – Trieda pre výpočet obchádzky prekážky
    - *Trajectory* – reprezentuje trajektóriu postupnosti pohybov
    - *TrajectoryPlanner* – Plánovanie pohybov pre presun hráča z bodu A do B
    - *TrajectoryRealTime* – plánovanie pohybov podľa aktuálnych informácií
- Jim.annotation

    - Slúži na vytvorenie opisu pohybu
- Jim.annotation.gui
    - Grafické rozhranie pre anotácie
- Jim.gui
    - GUI pre replaning
- Jim.init
    - *ScriptBoot* – načítanie ruby skriptov
    - *SkillFromXmlLoad* – Načítanie low skills z XML súborov
    - *TestframeworkMain*- ?
- Jim.log
    - Logovanie pre hráča
- Jim.tests
    - Testovacie triedy

2.1.1.2 TestFramework

Tento framework slúži na získanie spätnej väzby od hráča. Jeho hlavným zámerom je zostrojiť robotického futbalového trénera. Zatiaľ je vypracovaný len vo fáze pozorovateľa.
Framework umožňuje modelovanie testcasov. Tento testovací framework obsahuje i grafické GUI. GUI je ľahko ovládateľné, dá sa v ňom ľahko zorientovať.
Taktiež vytvára vlákna hráčov, ktorých pridáva rovno do simulácie. Podľa identifikovania vzťahov – framework pridáva inštancie aktuálneho Jima.

2.1.1.2.1 Moduly (Balíky)

1. INIT
   Tento modul (balíček) sa stará o samotné spustenie testovacieho frameworku, inicializujú sa tu základné hodnoty ako porty hráča, monitora, servera , ktoré sú následne prenášané do ďalších častí , napr. do komunikácie s hráčom a serverom.

   Taktiež sa tu inicializuje aj samotný user interface. Vykonáva sa kontrola, či je spustený RoboCup server a Monitor. Bez týchto podmienok sa testframework nespustí.

2. LOGGER
   Slúži na logovanie všetkých vykonaných činností. Je spustený nad každým modulom (balíčkom), takže sa logujú všetky činnosti spojené so samotným testframeworkom.

3. MONITOR

Monitor slúži na monitorovanie stavu agenta a robocup servera. Slúži aj pre prijimanie TCP spojení s novými správami. Slúži taktiež na pridávanie a odoberanie vlákien agenta. Existuje aj trieda robocup, ktorá sa snaží simulovať stavy na serveri.

4. UI
   Grafický interface , v ktorom sa zobrazujú všetky dostupné informácie, dovoľuje pridávať a odoberať agentov.
5. COMMUNICATION
   Rozhranie pre komunikáciu a sledovanie procesov. Rozdelené na agent a robocupserver. Každá z týchto častí sa stará o svoju komunikáciu. Obsahuje rozhranie pre komunikáciu agentov. Kontroluje bežiace procesy a agentov.

6. PARSING
   Slúži na parsovanie správ. Typická dĺžka jednej správy nepresahuje jeden riadok a obsahuje číslo hráča, názov tímu, typ správy a posielané hodnoty . Príklad takejto správy je :
   (1 MEGATROLL LEFT) highskill start rollback 0.0
   Jediná výnimka pri posielaní správ sú správy o stave sveta. Stav sveta je na strane hráča deserializovaný do pola bajtov a následne zakódovaný do textového reťazca pomocou Base64.

7. AGENTTRAINER
   Mal by sa starať o zapisovanie do XML, mal by obsahovať umelú inteligenciu, ktorá by učila agenta nové pohyby.

8. ANNOTATOR
   Veľká trieda na parsovanie a vytváranie XML pohybov. Obsahuje triedu zodpovednú za dynamické vytváranie pohybov.

9. WORLD REPRESENTATION
   Modul(balíček), ktorý reprezentuje okolitý svet, hráča. Taktiež zabezpečuje testovanie pohybov z xml.Stará sa o interpretáciu správ do scény, nastavovanie hráčov, reprezentáciu hráča,

10. MONITOR AGENTA
    Je implementovaný pomocou triedy Agent Monitor. Lokálne sa používa iba jedna inštancia, ktorá sa stará o prijímanie nových spojení od samotných agentov. Každé spojenie je reprezentované vlastným threadom, ktorý ma na starosti spracovanie správ.

Aktuálne existujú typ správ:
INIT - posiela agent pri pripojení, obsahuje jeho číslo, názov tímu a stranu na ktorej hrá, takisto či má hráč zapnutý TFTP server a na ktorom porte

DESTROY - posiela agent pri odpojení

HIGHSKILL - informuje test framework o začatí/skončení high skillu, obsahuje meno high skillu a čas kedy k akcii došlo

WORLDMODEL - posiela model sveta agenta do test frameworku

Tím, ktorý pracoval na projekte pred nami, vytvoril novú triedu AgentMonitor, ktorý zohráva úlohu TCP servera pre prichádzajúce spojenia. Pri príchode nového spojenie je vytvorená inštancia, ktorá ma za úlohu spracovanie správ zo spojenie a ich následne odosielanie ďalej.

Popis balíkov podľa názvov:
sk.fiit.testframework.communication.robocupserver – táto trieda sa snaží posielať príkazy na robocup server, tak aby sa nastavil špecifický stav
sk.fiit.testframework.annotator – dynamické vytváranie xml pohybov
sk.fiit.testframework.init – hlavné nastavenia
sk.fiit.testframework.annotator.serialization – parsovanie pohybov, uložených v xml
sk.fiit.testframework.monitor – pridávanie a odoberanie vlákien agentov
sk.fiit.testframework.communication.agent – cez toho rozhranie komunikujú agenti
Identifikované triedy (Jim i Testframework), ktoré sú spojené s ruby (vyplynulo z analýzy a prepojení) :
RubyTest
ScriptBoot
Settings
ReleaseBuilder

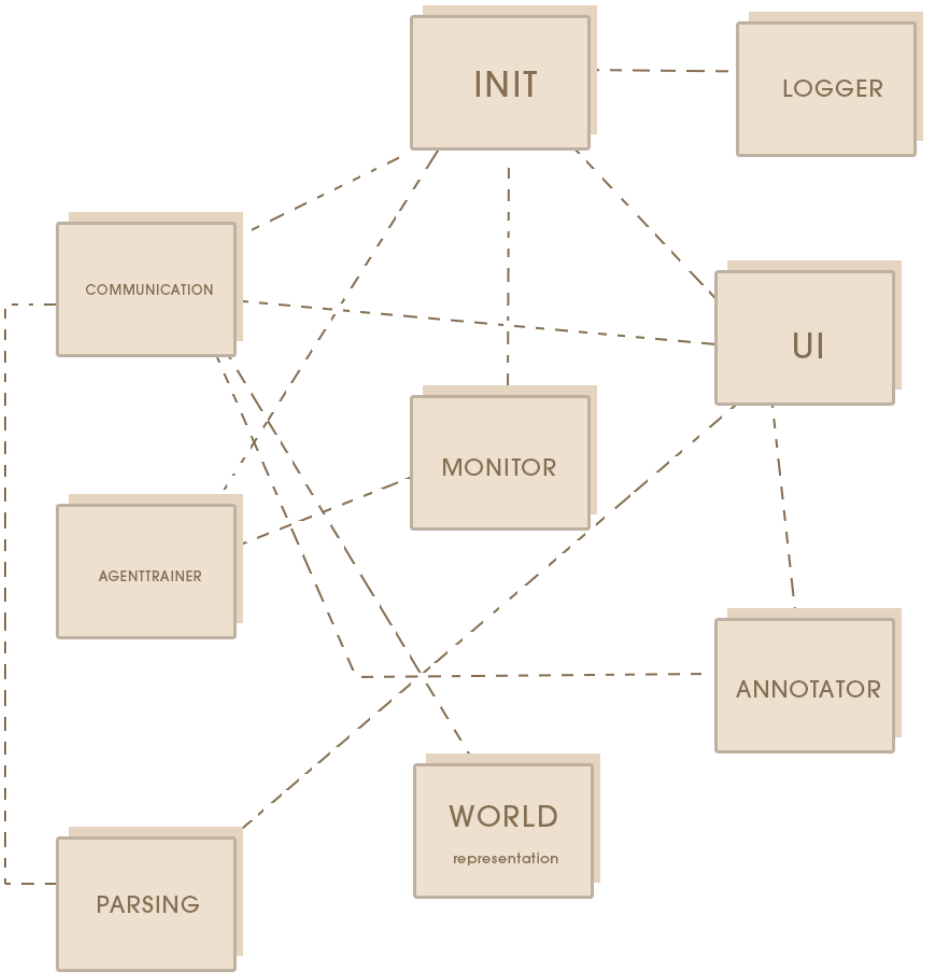
2.1.1.2.2 Nastavenia TESTFRAMEWORKU

robocup.server.command – príkaz na spustenie robocup servera
robocup.server.killcommand – príkaz na vypnutie robocup servera, ak je prázdny, test framework sa pokúsi server vypnúť ukončením jeho procesu
robocup.server.ip - IP adresa robocup servera
robocup.server.port.monitor – monitor port robocup servera pr

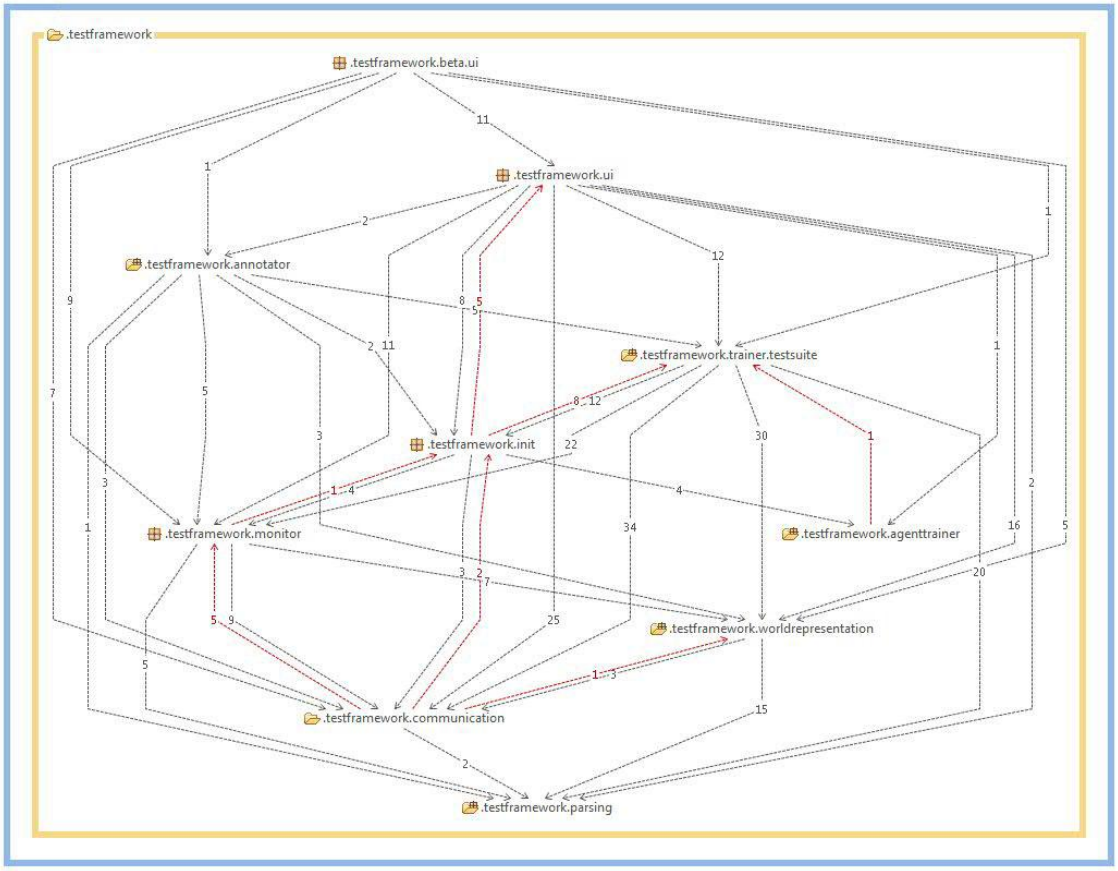
robocup.server.port.player – port na pripojenie hráča k robocup servera
robocup.player.dir – zložka, v ktorej sa nachádza hráč
robocup.player.command – príkaz na spustenie hráča
testframework.monitorAgent.ip – IP adresa, na ktorej počúva testframework na spatnú väzbu
testframework.monitorAgent.port -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

Konfiguračný súbor je prečítaný pri spustení aplikácie.

Obrázok 1- identifikované prepojenia



Obrázok 2 - architektúra testframework-u



2.1.1.3 RobocupLibrary

RoboCupLibrary sa skladá z piatich balíčkov. Dva z nich slúžia len na zavedenie konvencií pri programovaní ako sú anotácie častí kódu. Ďalšie dva balíky obsahujú matematické metódy a posledný balík zabezpečuje integráciu skriptovacieho jazyka s jazykom Java pomocou BeanScriptingFramework-u.

2.1.1.3.1 Balíky

sk.fiit.robocup.library.annotations
Balík obsahuje päťsúborov z ktorýchkaždý jedendefinuje anotáciu k častiam kódu

Bug.java – definuje anotáciu k chybovej časti kódu
Refactor.java – definujeanotáciu kódu ktorý by mal byť prerobený
Reviewed.java – definuje anotáciu kódu, ktorý bol skontrolovaný
TestCovered.java – definuje anotáciu kódu, ktorý bol otestovaný
UnderConstruction.java – definuje anotáciu kódu, ktorý je vo výstavbe

sk.fiit.robocup.library.review
Balík obsahuje jeden súbor, ktorý definuje anotácie
ReviewOk.java - definuje anotáciu k triede alebo metóde, ktorá skontrolovaná, testovaná unittestom a testovaná proti serveru

sk.fiit.robocup.library.init
Balík obsahuje jeden súbor, ktorý zabezpečuje načítanie a spustenie skriptu. Prácu so skriptami zabezpečuje BSF Manager.

Script.java – Script(Stringname): konštruktor triedy
createScript(Stringname): určí, ktorý skript sa má načítať podľa mena
createScriptFrom(Stringfilepath): určí, ktorý skript sa má načítať podľa cesty k súboru
execute(): spustí načítaný script
registerBean(Stringname, Objectvalue): slúži iba na testovanie triedy Script
fetchBeans(String...names): nie je v programe vôbec využitá

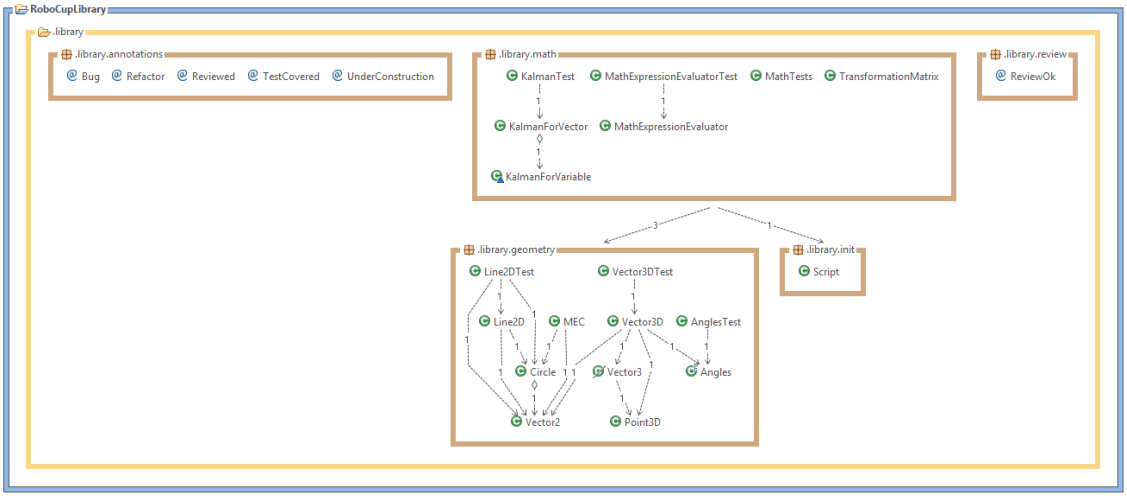
sk.fiit.robocup.library.math
Tento balík obsahuje sedem súborov, ktoré implementujú Kalmanove filtre, prevody matematických výrazov a transformácie matíc.

KalmanForVariable.java – vypočíta Kalmanov filter pre jednu premenné
KalmanForVector.java – slúži na výpočet Kalmanovho filtra pre 3 premenné
KalmanTest.java – trieda, ktorá iba testuje Kalmanove filtre
MathExpressionEvaluator.java – zabezpečuje prevod matematického reťazca na prijatého ako String na číselný výsledok
MathExpressionEvaluatorTest.java – testuje triedu MathExpressionEvaluator
MathTest.java – Spúšťa testovacie súbory AnglesTest.java, KalmanTest.java a Vector3DTest.java
TransformationMatrix.java – obsahuje metódy na prácu s maticami, trieda je využívaná iba programom TestFramework

sk.fiit.robocup.library.geometry
Tento balík obsahuje najmä triedy, ktorériešia výpočty geometrickej matematiky. Zvyšné triedy slúžia iba na testovanie.

Angles.java – knižničná trieda, ktorá rieši operácie s uhlami
Circle.java – trieda reprezentujúca kruh v 2D priestore
Line2D.java – trieda reprezentujúca čiaru v 2D priestore
MEC.java – vypočíta najmenší ohraničujúci kruh pre zoznam bodov
využíva sa pri hľadaní pozície lopty
Point3D.java – trieda reprezentujúca bod v 3D priestore
obsahuje metódy, ktoré sa nevyužívajú alebo nie si implementované
Vector2.java – trieda reprezentujúca bod v 2D priestore pomocou vektoru
Vector3.java – trieda reprezentujúca bod v 3D priestore pomocou vektoru
obsahuje základne operácie ako sčítanie, odčítanie, delenie a vzdialenosť dvoch bodov

Vector3D.java–trieda reprezentujúca bod v 3D oproti triede Vector3 obsahuje prídavné operácie
AnglesTest.java – testuje triedu Angles
Line2DTest.java – testuje triedu Line2D
Vector3DTest.java – testuje triedu Vector3D



Obrázok 3

2.1.2 Spustenie hráča

Spustenie hráča prebehlo na rôznych systémových platformách. Inštalačné problémy a návody boli spísané, taktiež pridané na Wiki.

2.1.2.1 Windows XP Professional 32b Service Pack 3

1. Inštalácia MS Visual C++2008 RedistributablePackage(x86):

   http://www.microsoft.com/en-us/download/details.aspx?id=29

2. Inštaláciasimspark verzia 0.2.4-win32:

   http://sourceforge.net/projects/simspark/files/simspark/

3. Inštalácia rcssserver3d verzia 0.6.7-win32:

   http://sourceforge.net/projects/simspark/files/rcssserver3d/

4. Inštalácia Ruby:

   http://rubyinstaller.org/downloads/

5. Test servera: Server sa spúšťa súbormi v zložke ProgramFiles/rcssserver3d 0.6.7/bin o cez rcssserver.cmd a monitor sa spúšťa cez rcssmonitor.cmd

Inštalácia hráča:

6. Inštalácia eclipseverzia standardkeplerSR1 win32 x86_32
7. Inštalácia AspectJ a EclipseAspectJDevelopmentTools
   7.1. Spustiť nástroj Eclipse
   7.2. Voľbami Help->Install new software otvoriť obrazovku inštalácie pluginov
   7.3. Do adresy zadať URL ktorú podľa verzie eclipse nájde na : http://www.eclipse.org/ajdt/downloads/ a stlačiť klávesu enter
   7.4. V možnostiach je potrebné zvoliť možnosti „AspectJDevelipmentTools (Requires)“ a „AJDTDevelopmentTools“.
   7.5. Potvrdiť voľbu tlačítkom „next“
   7.6. Potvrdiť licenčné podmienky a pokračovať s inštaláciou.
8. Použiť zdrojové súbory z http://labss2.fiit.stuba.sk/TeamProject/2011/team05is-si/download.php.html(rozbaliť)
9. V ecplise vytvoriť Projekt s názvom RoboCupLibrary
   9.1. Importovať ako filesystem danú zložku RoboCupLibrary zo zdrojových súborov Robo cup
10. Vecplise vytvoriť Projekt s názvom Jim
    10.1. Importovať ako filesystem danú zložku Jim zo zdrojových súborov Robo cup
11. Vecplise vytvoriť Projekt s názvom TestFramework
    11.1. Importovať ako filesystem danú zložku TestFrameworkzo zdrojových súborov Robo cup
12. Spustiť robota v eclipse/Jim/src/sk/fiit/jim/init/Main.java (server musí byť pustený)
    12.1. Spustiť TestFramework/TestFramework/src/sk/fiit/testframework/init/Init.java

2.1.2.2 Windows 7 64 bit, Eclipse Helios 3.6.1, Ruby, Java 1.6

1. Inštalácia AspectJ – nutné potvrdiť všetky súčasti, bez potvrdenia niektorej súčasti sa nedoinštalovali správne
2. Aktualizovaná verzia Java, Ruby – pri starších verziách sa vyskytli chyby
3. Import projektu do eclipsu – naimportovanie všetkých súčasti do správnych projektov. Názov musí byť zhodný s názvom projektu. Napr. Jim – Jim.
4. Chyby pri spustení – je potrebné buildnúť všetky projekty
5. V prípade vyhadzovanie errors – potrebné zakomentovať v triede Execution Time Monitor

Po úspešnom nainštalovaní je potrebné spustiť rcserver a monitor. Po spustení je potrebné v eclipse spustiť projekt JIM.

2.1.2.3 Ubuntu 12.04 LTS (Precise Pangolin) 32 bit, Eclipse Juno 4.2.2 a balíky Ruby 1.9.3 a Java SDK 1.6.

Inštalácia AspectJ – nutné potvrdiť všetky súčasti, bez potvrdenia niektorej súčasti sa nedoinštalovali správne

Aktualizovaná verzia Java, Ruby – pri starších verziách sa vyskytli chyby
Import projektu do eclipsu – naimportovanie všetkých súčasti do správnych projektov. Názov musí byť zhodný s názvom projektu. Napr. Jim – Jim.

Chyby pri spustení – je potrebné buildnúť všetky projekty
V prípade vyhadzovanie errors – potrebné zakomentovať v triede Execution Time Monitor
V súbore Goto.rb som zakomentoval posledný riadok GoTo.new
V súbore boot.rb som po riadku include Java pridal riadky load "scripts/high_skills/walk.rb" a load "scripts/plan/plan.rb"

V súbore default.properties v adresári TestFramework-u bolo nutné prepísať bodkočiarky na dvojbodky, pretože classpath premenná pre windows sa oddeľuje bodkočiarkami a pre unix dvojbodkami

2.1.3 Zmena správania

Všetci členovia tímu vykonali zmeny správania hráča pomocou naštudovania plánovača. Následne každý vytvoril svoj vlastný pohyb pomocou xml súboru, ktorý následne použil pre demonštráciu pohybu.

2.1.4 Unit testy

Všetci členovia tímu vytvorili unit test, ktorý ešte systém neobsahoval, tak aby tento test bol vytvorený pre triedu, ktorá sa nebude často meniť.

2.1.5 Web

Web bol vytvorený na CMS systéme Wordpress, pre ľahšiu úpravu a pridávanie nových materiálov. Na webe sú zverejnené zápisnice zo stretnutí, ale aj aktuálny backlog, či kalendár stretnutí. Taktiež tam sú zverejnené dôležité dokumenty, ktoré sa týkajú našej práce, našich metodík.

2.1.6 Prehľad o tímoch

Všetci členova tímu získali prehľad o významných tímov, ktoré sa zúčastňujú medzinárodných súťaži. Medzi najlepšími tímami sme identifikoval tímy ako:

Team DARwIn
NimbRo TeenSize
JoiTech
HuroEvolution AD
Tsinghua Hephaestus

2.1.7 Backlog

Vylepšenie stability hráča po kopnutí do lopty

Chcem	Ako	Prečo
Vylepšiť stabilitu hráča po kopnutí do lopty	Člen tímu	Aby mal robot väčšiu stabilitu po kopnutí a menej padal.

Pozn.: poznáme viacero kopnutí do lopty, rozlišovať ich a zamerať sa na konkrétne

Odstránenie ruby kódu

Chcem	Ako	Prečo
Odstrániť ruby závislosti.	Člen tímu	Aby bol systém prehľadnejší a postavený na jednej technológií

Zlepšenie inštalácie

Chcem	Ako	Prečo
Zlepšiť inštaláciu.	Člen tímu	Zjednodušenie samotnej inštalácie.

Pozn.: Filesystem, export

Editor pohybov – export do XML

Chcem	Ako	Prečo
Dopracovať XML export.	Člen tímu	Pre zjednodušenie vytvárania pohybov pomocou editora.

Pozn.: Iba v prípade, že to naozaj nefunguje

Vylepšenie príchodu k lopte

Chcem	Ako	Prečo
Vylepšiť stabilitu hráča po kopnutí do lopty	Člen tímu	Aby mal robot väčšiu stabilitu po kopnutí a menej padal.

Pozn.: Zamerať sa konkrétnejšie – postaviť ho , alebo dať hneď do pohybu

Zlepšenie kopnutia do lopty

Chcem	Ako	Prečo
Vylepšiť samotný kop do lopty	Člen tímu	Aby robot kopal presnejšie .

Zlepšenie rozhodovania pri kopaní

Chcem	Ako	Prečo
Vylepšiť samotné rozhodovanie pred kopnutím do lopty.	Člen tímu	Pre skrátenie času pred kopnutím.

Analýza minuloročných a zahraničných tímov

Chcem	Ako	Prečo
Analyzovať minuloročné a zahraničné tímy.	Člen tímu	Pre lepší prehľad a prípadne vylepšenia.

Pozn.: Napr. kvôli architektúre.

Analýza diplomových prác

Chcem	Ako	Prečo
Analyzovať diplomové práce na našej fakulte.	Člen tímu	Pre implementovanie výsledkov.

Zlepšenie rozhodovanie pri situácií hráč na hráča

Chcem	Ako	Prečo
Vylepšiť rozhodovanie pri priamom strete dvoch hráčov.	Člen tímu	Pre lepšie rozhodovanie a predchádzanie situácii zbytočných pádov.

Pozn.: Nie je čo zlepšovať – musíme vytvoriť.

Zlepšiť hľadanie pozície lopty

Chcem	Ako	Prečo
Zlepšiť hľadanie pozície lopty v prípadoch kedy ju hráč nevidí.	Člen tímu	Pre zrýchlenie reflex hľadania.

Zlepšiť predkopávanie lopty pred hráčom

Chcem	Ako	Prečo
Zlepšiť predkopávanie lopty pred hráčom z mini krokov na väčšie	Člen tímu	Pre zrýchlenie chôdze s loptou.

Pozn.: Dribbling

Implementovať príkaz na reštart hry po skončení

Chcem	Ako	Prečo
Pridať príkaz na reštart hry.	Člen tímu	Pre ľahší reštart hry.

Pozn.: Iba v prípade, že sa to reálne dá.

Refactoring kódu

Chcem	Ako	Prečo
Vykonať refactoring kódu.	Člen tímu	Odstránenie nepotrebných tried, sprehľadnenie kódu.

2.2 Šprint č. 2

ID	AKO	CHCEM	ABY
2.1	Člen tímu	Analyzovať zahranične tímy	Zakompovanie architektúry a už overených riešení.
2.2	Člen tímu	Dať web na školský server	Aby web bol prístupný stále a na jednom mieste.
2.3	Člen tímu	Vytvoriť balíky pre Ruby	Logické rozdelenie podľa balíkov
2.4	Člen tímu	Prepísať plánovanie z ruby do javy	Odstránenie ruby.
2.5	Člen tímu	Analyzovať diplomové práce	Implementovanie nových súčastí.
2.6	Člen tímu	Spojazdniť GIT	Práca nad jedným repozitárom
2.7	Člen tímu	Pripraviť návody na inštaláciu	Jednoduchšia inštalácia v budúcnosti.

2.8	Člen tímu	Vytvoriť spoločný backlog	Všetky úlohy pre oba tímy v jednom backlogu.
2.9	Člen tímu	Vytvoriť dokumentáciu	Zdokumentovanie posledného šprintu.

Tabuľka 3

ID Pp	ID podúlohy	Úloha	Zodpovedný
2.1	-	Analyzovať zahranične tímy	Všetci
2.2	-	Dať web na školský server	Bádal
2.3	-	Vytvoriť balíky pre Ruby	Benkovič
2.4	-	Prepísať plánovanie z ruby do javy	Všetci
2.5	2.5.1	Analyzovať diplomové práce (Durčák)	Petráš, Adámik
	2.5.2	Analyzovať diplomové práce (Hudec)	Čerešňák, Homola
2.6	-	Spojazdniť GIT	Všetci
2.7	-	Pripraviť návody na inštaláciu	Bošiak,Adámik
2.8	-	Vytvoriť spoločný backlog	Všetci
2.9	-	Vytvoriť dokumentáciu	Petráš

2.2.1 Analýza zahraničných tímov

2.2.1.1 Hamburg Bit-Bots

Hamburg Bit-Bots je nemecký robocupový tím, ktorý je zostavený zo študentov z oddelenia informatiky Hamburgskej univerzity.
Funguje od roku 2011 a používajú *DarwinOP* roboty vyrobené firmou *Robotics and Mechanisms Laboratory*.

2.2.1.1.1 Výskum

Bakalárske práce:

- *Evolving Locomotion for the DarwIn-OP* - vývoj chôdze pomocou evolučných algoritmov.
- *Team coordination in RoboCup soccer based on natural language* - stratégia a komunikácia robotov.
- *Ball recognition based on probability distribution of shapes, Estimation of optical-*

ow fields in multispectral images - rozpoznávanie lopty v hre (použitie pri nesimulovanom robocup-e)

Iný výskum:

- Výmena kamery v robotovi *DarwinOP*
- Vývoj chôdze pomocou evolučných algoritmov
- Pozícia robota na základe bodov
- Pozícia robota na ihrisku (nepoužívajú ešte komplexnú lokalizáciu)
- Komunikácia počas hry
- Vývoj integračného systému medzi simuláciou a realitou

2.2.1.1.2 Úspechy

V roku 2012 sa umiestnili 3. na RoboCup German Open a postupili do druhého kola v svetovom šampionáte.

2.2.1.2 FC Portugal 3D

- Projekt, na ktorom sa podieľajú 3 portugalské univerzity: Aveiro, Porto, Minho
- Hlavným cieľom tímu je adaptácia metodológií, ktoré vyvinul rovnaký tím zameraný ale na 2D simuláciu
- Momentálnym cieľom tímu je práca a vylepšenie high-level rozhodovanie a spolupráca agentov navzájom

Tím v roku 2013 vyhral RoboCup German Open, ktorý prebiehal 26 – 28 aprílav Magdeburgu v

Nemecku a stal sa tak európskym šampiónom. Tým vo finále porazil nemecký tím magmaOffenburg v penaltovom rozstrele

2.2.1.2.1 Ciele výskumu

- Architektúra agenta
- Model humanoida a s ním spojené odmedzenia dynamiky, vnímania a rozhodovania
- Časť výskumu sa zameriava na vývoj stratégie a spojenie humanoidov pochádzajúcich z rôznych tímov
- Modelovanie súperov
- Inteligentnejšie vnímanie

2.2.1.2.2 Architektúra agenta

Architektúru má tím rozdelenú do viacerých balíkov nasledovne nasledovne:

WorldState – Triedy, ktoré uchovávajú informácie o prostredí => statické predmety na ihrisku,
vlastnosti hry
AgentModel – Triedy uchovávajúce informácie o agentovi, štruktúre jeho tela
Geometry – Triedy, slúžiace na definovanie geometrických entít => čiary, kružnice...
Optimization – Triedy používané na proces optimalizácie
Skills – Rozdeľuje sa na dve časti => Reactive Skills – základné správanie, Talent Skills => schopnosť agenta premýšľať a predvídať napr. pohyb objektov schopných pohybu
Utils – Triedy potrebné na fungovanie agenta => komunikačné triedy, parsery, debugery
Strategy – High-level funkcie agenta

2.2.1.2.3 Optimalizácia

Jednou z hlavných úsilí tímu bolo vytvorenie optimalizačného nástroja, ktorý by mohol upravovať správanie agenta. Táto optimalizácia funguje tak, že je načítanie správanie hráča z XML súboru. Následne je vytvorená reprezentácia tohto XML súboru, ktorá sa mení na základe optimalizačného algoritmu a optimalizovaná verzia sa následne vykonáva. Týmto spôsobom tím optimalizoval viaceré skilly hráča, ako napríklad schopnosť postaviť sa.

2.2.1.3 Analýza svetového tímu magmaOffenburg

Je nemecký tým z UniversityofAppliedscienceOffenburg, ktorý sa venuje RoboCup 2D Simulationleague od roku 1999 a RoboCup 3D Simulationleague sa venuje od roku 2008.

2.2.1.3.1 Architektúra

Je založená na päť vrstvovej component-basedarchitecture. Vrstvy sú navrhnuté tak aby sa zabránilo závislosti z nižšej do vyššej vrstvy.Rozhranie slúži na vytvorenie voľnéhoprepojenia medzi komponentmi každej vrstvy a zodpovedajúce vyššej vrstvy.

Obrázok 4 - architektúra Magma-AF

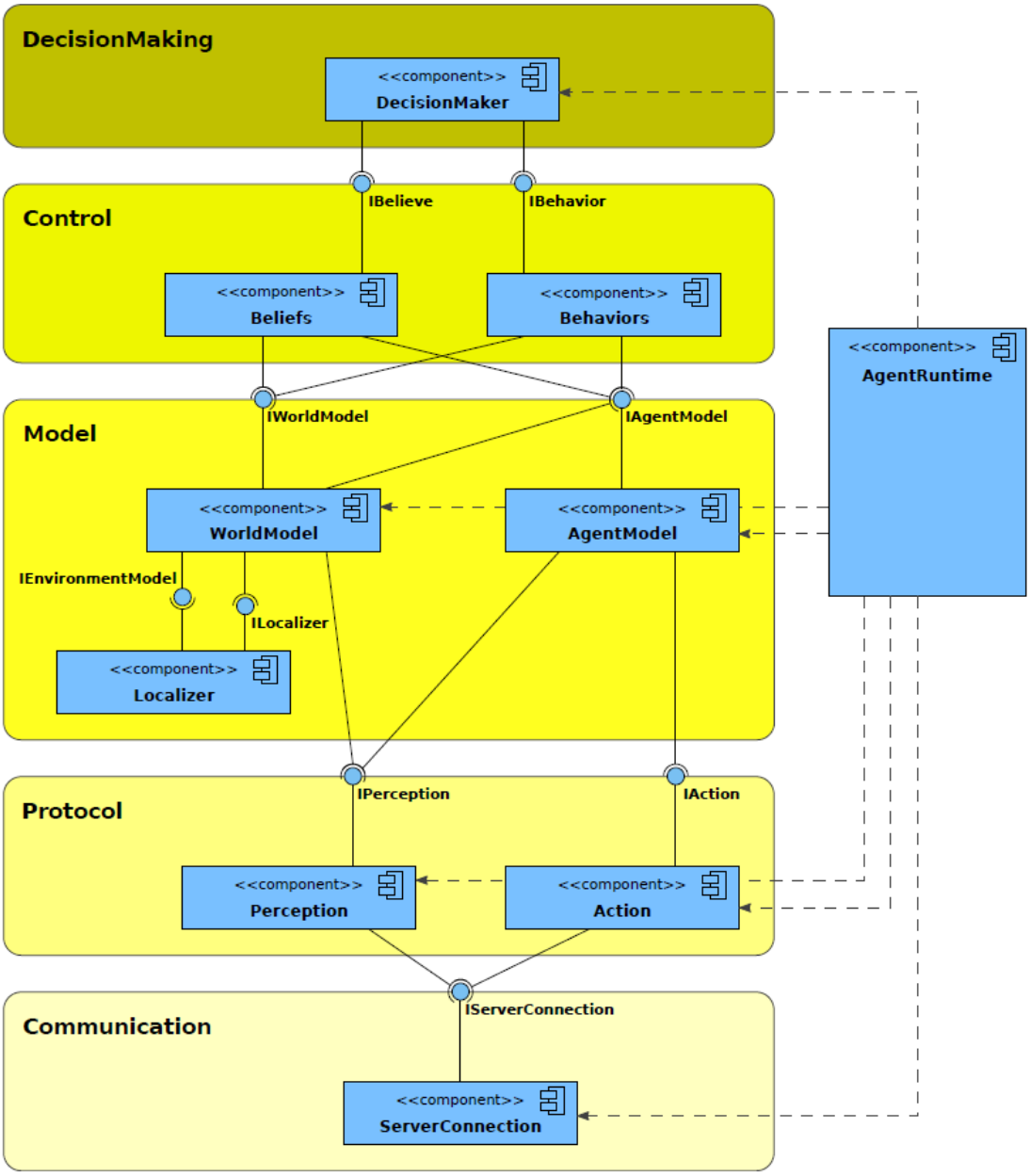


Figure 1: Layered, component-based architecture of the magma-AF.

AgentRuntime je centrálnyprvok v pozadí, ktorý riadi proces po prijatí správy zo serveru. Poradie vykonávania činností je nasledovné:

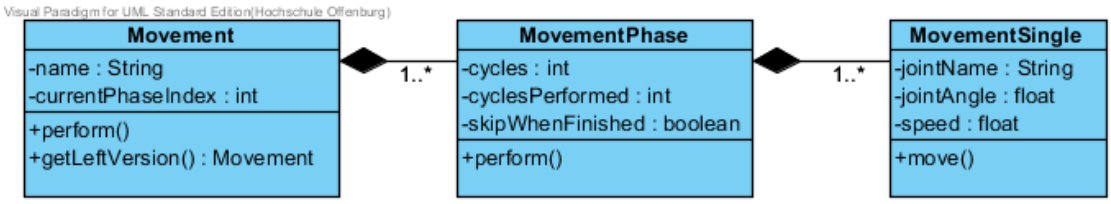
1. Analýza novej správy(Perception)
2. Aktualizácia vnútorného stavu(AgentModel)
3. Aktualizácia stavu prostredia (WorldModel)
4. Rozhodnutie o ďalšom vykonávanom pohybe (DecisionMaker)
5. Priradiť konkrétne pohyby efektorom ktoré sú uložené v AgentModel (AgentModel)

6. Preložiť a poslať správu s akciami serveru (Action)

2.2.1.3.2 Pohyby

Movementframework je najzákladnejší prvok, ktorý využívajú na tvorbu správania sa robota. Na obrázku číslo 3 je vidno model tohto frameworku, ktorý znázorňuje, že každý jeden pohyb je delený na pohybové fázy, ktoré sú zasa delené na jednoduché pohyby.

Obrázok 5 - pohyby



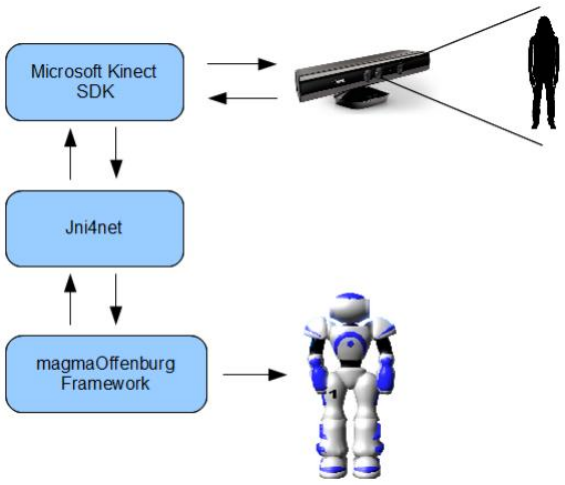
Obrázok 2 - Model "Movementframework" tímu MagmaOffenburg

Strategydeveloperplugin – je plugin pomocou ktorého dokázali pozorovať reakcie agentov na rôzne situácie v zápase.

2.2.1.3.3 KinectControl

Ďalšou zaujímavosťou tímu ma magmaOffenburg je ovládanie robota pomocou skenovania pohybov človeka. Na skenovanie pohybov je využité zariadenie Kinect. Tieto pohyby sú následne transformované na pohyby ktoré je schopný vykonávať robot. Tento framework je napísaný v jazyku C#, magmaOffenburgframework je napísaný v Jave, prístup k dátam vykonáva pomocou JNI Bridge.Ovládanie spoľahlivo pracuje na pohyby rúk, ale pri pohybe nôh robot padá pretože Kinectnie je schopný sledovaťrotácie a uhly nôh a rúk.

Obrázok 6 - kinect



2.2.1.4 Analýza zahraničného tímu UT Austin Villa

Tím UT Austin Villa vznikol na ústave výpočtovej techniky na Texaskej univerzite v Austine. Už krátko po svojom vzniku sa zúčastňovali medzinárodných súťaží v 3D RoboCup-e. Ich hlavným cieľom za posledný rok bolo zlepšenie techniky vstávania hráča a kopania.

2.2.1.4.1 Optimalizácia vstávania

Veľmi dôležitou vlastnosťou každého hráča v 3D Robocupe je to, ako sa čo najrýchlejšie postaviť po páde a zapojiť sa opäť do hry. Tím UT Austin Villa prišiel s nápadom iterovať sériu polôh, ktoré prejdú z jednej na druhú počas určitého času. Tieto polohy sú zoskupením sérií špecifických uhlov.

Pre optimalizáciu vstávanie bol použitý algoritmus CMA-ES (Covariance Matrix Adaptation Evolution Strategy). Keďže zistili, že tento algoritmus je najlepší na optimalizáciu parametrov pre vlastnosti robotov ako je chôdza a otáčanie. CMA-ES je prehľadávací algoritmus, ktorý postupne vytvára a vyhodnocuje sady kandidátov z multivariačného Gaussovského rozdelenia. Postavenie by malo byť dostatočne rýchle, ale aj dostatočne stabilné, pretože pokiaľ sa hráč rýchlo postaví, no hneď spadne, je nepoužitelný. Robot na určenie stability využíva akcelerometer. Pokiaľ nie je plne stabilný, pokračuje vyrovnávanie stability, až kým nebude plne stabilný. Pri hodnotení stabilizácie robot zaznamenáva spotrebovaný čas, z ktorého sa potom určuje, ktoré zoskupenia pohybov sú najvýhodnejšie.

Po optimalizácií sa rýchlosť vstávania z oboch strán zvýšila skoro trojnásobne.

2.2.1.4.2 Optimalizácia kopania

Tím UT Austin Villa vytvoril hybridný systém na kopanie, ktorý využíva fixed pose keyframe (lepšia rovnováha) a inverse kinematics (viac robustný) based kick. Parametre pre všetky kopy boli optimalizované pomocou CMA-ES algoritmu spomenutého vyššie.

Fixed pose keyframe

Táto skupina kopov zahŕňa 3 hlavné druhy kopov (KickLong,KickMedium,KickQuick). Pre každý z nich robot v prvom rade presunie nekopajúcu nohu blízko k lopte a presunie

naň svoju váhu kôli stabilite. Potom zdvihne kopajúcu nohu načiahne ju dozadu a následne švihne a odkopne loptu.
Ako som už spomenul vyššie, tím UT Austin Villa využíva 3 druhy kopov, ktorými sú:

- KickLong - Je využívaný len pri rozohrávaní na začiatku polčasu, taktiež tento druh kopu letí vysoko vo vzduchu, takže loptu nik nemôže zblokovať ani jej zabrániť.
- KickMedium - Tento kop dosiahne menšiu vzdialenosť ako KickLong, ale robot je viac stabilnejší po odkopnutí.
- KickQuick - Zatiaľ čo kop KickMedium trvá približne 2 sekundy, kop KickQuick menej ako sekundu. Tento kop bol vytvorený z dôvodu, že oponent je blízko a robot musí rýchlo reagovať. Taktiež je nestabilný, takmer ako KickLong.

2.2.1.4.3 Inverse kinematics

Slabou stránkou fixed pose keyframe bola potreba veľmi precízneho postavenia k lopte. Alternatívou je zadefinovať si relatívnu cestu k lopte, ktorú by mala robotova noha nasledovať a potom použiť inverznú kinematiku na pohyb nohy pozdĺž určenej cesty. Hlavnou výhodou takéhoto kopu je, že strela je schopná prispôsobiť sa polohe lopty a teda nevyžaduje tak presné polohovanie robota k lopte. Takýto kop sa nazýva KickIK špecifikovaný relatívnymi waypontami , ktorými noha prechádza pomocou interpolácie medzi týmito bodmi pomocou Hermitovej kubickej krivky určujúcu trajektóriu cesty nohy počas kopu.

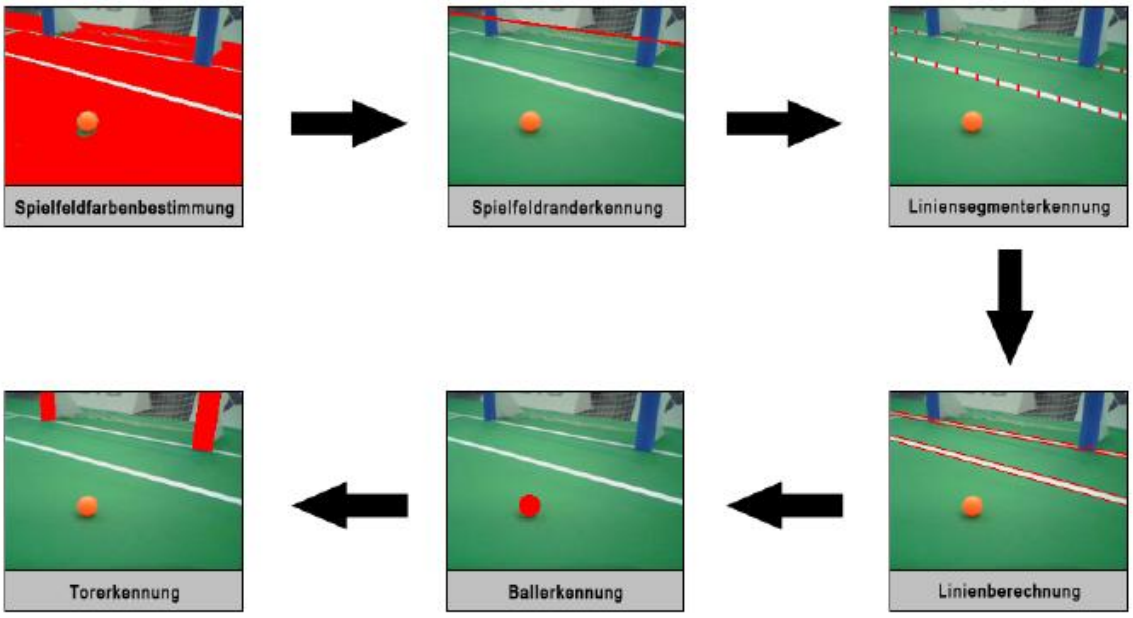
2.2.1.5 Analýza zahraničného tímu Nao-Team HTWK Leipzig

Nao-Team HTWK je nemecký RoboCup tým z Univerzity Aplikovaných Vied v Leipzig-u, ktorý funguje od roku 2009. V roku 2013 sa obsadili druhé miesto na RoboCupGermanOpen. V posledných rokoch sa vývojári tímu zameriavali hlavne na zdokonalenie počítačového videnia, lokalizácie hráča a chôdze.

2.2.1.5.1 Počítačové videnie

Najväčším problémom v oblasti počítačového videnia je vysporiadať sa so zmenami svetelných podmienok (napr. denné svetlo a umelé osvetlenie). Vďaka vedomostiam o tvaroch objektov vyvinuli algoritmus na rozpoznávanie objektov, ktorý si vie poradiť so zmenami svetelných podmienok. Algoritmus sa skladá z niekoľkých detekčných a filtrovacích fáz. V prvej fáze sa určuje dominantná farba obrázku (najčastejšie to je farba ihriska). Na základe tejto informácie je možné nájsť ostatné objekty. V ďalšom kroku sú určené pozície čiar aby bolo možné odstrániť objekty mimo ihriska. Potom je možné určiť pozíciu lopty a a pomocou detekcie vertikálnych hrán aj tyčky bránky. Tento proces je naznačený na obrázku 1.

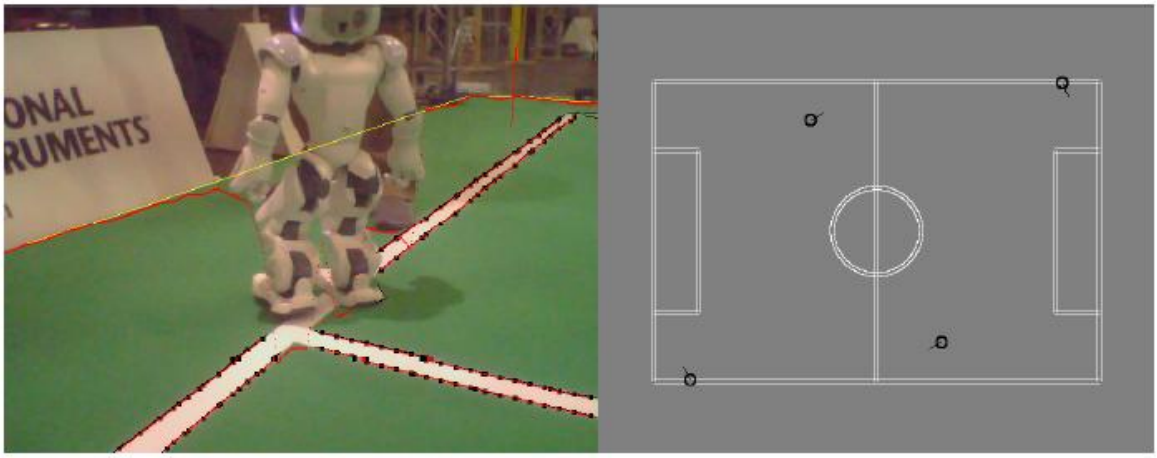
Obrázok 7 - Detekcia a rozpoznávanie objektov



2.2.1.5.2 Lokalizácia

K lokalizácii robota využívajú detekciu pozícií čiar v obraze. Na základe analýzy tohto obrazu vie robot odhadnúť v ktorej časti ihriska sa nachádza. Tento spôsob lokalizácie je výhodný, pretože nie je nutné prerátavať pozíciu pri každom otočení hlavy. Odhad lokalizácie hráča demonštruje obrázok 2.

Obrázok 8 – Odhady možného postavenia hráča na základe analýzy obrazu.



V roku 2010 prvý krát predstavili svoj nový "walkingengine" na súťaži RoboCup. Tento "engine " je založený na parametrickej chôdzi a je podporený o novo vyvinutý stabilizačný algoritmus. Výhodou tohto systému je, že podporuje "omni-directional" chôdzu a je možné rýchlo meniť smer chôdze. Tento systém bol vyvinutý hlavne so zameraním na stabilitu a rýchlosť, kde je mžné dosiahnuť rýchlosti až 300 mm/s. Zaujímavá je hlavne poloha rúk za chrbtom pri chôdzi, ktorá zabezpečuje lepšiu stabilitu robota. Táto chôdza je naznačená na obrázku 3.

Obrázok 9 - Pozícia rúk robota pri chôdzi

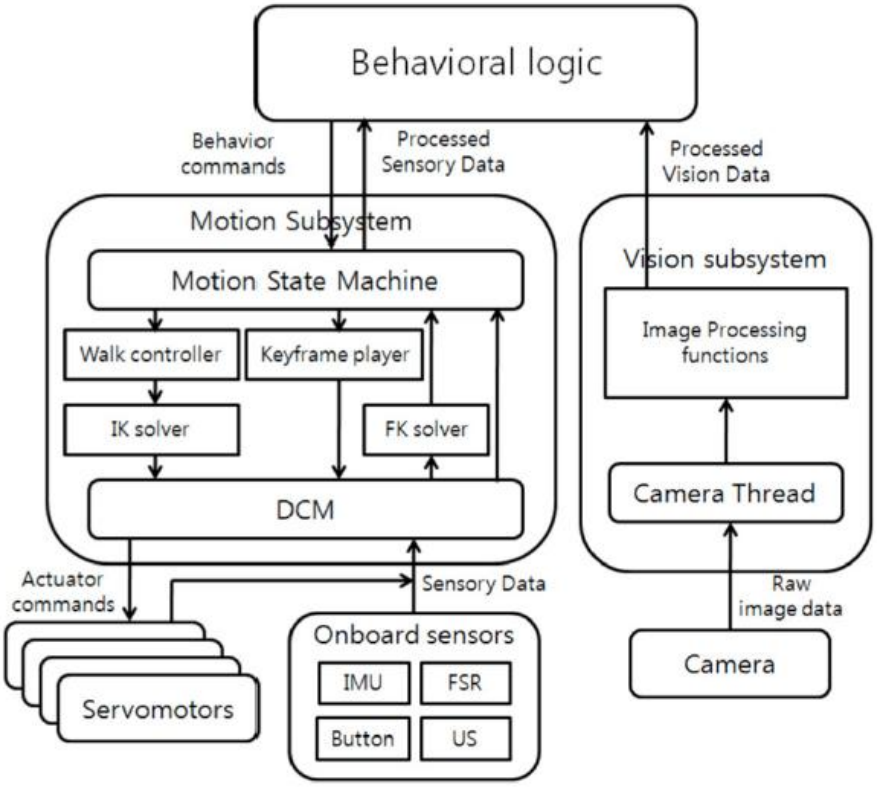
2.2.1.6 Tím Upennalizers

Tento tím vyvíja dlhodobo robotického futbalistu na univerzite University of Pennsylvania's. Vývoj tohto tím trvá od roku 1999. Tento tím sa niekoľko krát kvalifikoval do semifinále celosvetovej robocup ligy, taktiež sa dostal aj do finále.

2.2.1.6.1 Súčasná architektúra

Súčasná architektúra tímu je zložená z architektúr vyvíjaných počas viacerých rokov. Pomocou programovacieho jazyka LUA je implementovaná komunikácia medzi všetkými modulmi, ktoré sú nasledné pospájané s hardvérovým ovládačom NAO alebo s vlastnými regulátormi. Robot v reálnej verzii obsahuje zmyslovú spätnú väzbu, ktorá umožňuje zbierať dáta z rôznych zdrojov ako sú palubné kamery, čidlá alebo inerciálne meracie jednotky a ultrazvukové mikrofóny.
Na obrázku je zachytená celá architektúra hráča ako takého.

Obrázok 10 - logika správania



Na obrázku je zachytená kompletná architektúra, ktorá pozostáva z :

1. Senzory na doske
2. Kamera
3. Servomotori
4. Systém pohybu

5. Systém videnia
6. Chovanie – logika

Každá z týchto častí pozostáva z ďalších podčastí.
Prvým zaujímavým momentom, ktorý by mohol byť prínosom alebo ponaučením pri vývoji nášho robota je ich spracovanie vnímania sveta. Dôležité informácie ako vzdialenosť lopty, pozície hráčov, stav hry, pozície bránok, aktuálna pozícia hráča – **všetky tieto premenné sú premenné uložené v zdieľanej pamäti, ku ktorej môže pristúpiť akýkoľvek modul a následne si údaje prečítať ale ich aj zapísať do tejto zdieľanej pamäte.**

Túto zdieľanú pamäť má tento tím implementovanú ako samostatný modul, za pomocou ktorého potom následne vedia robiť aj realtime on-the-fly debugovanie hráča a analýzu (Na analýzu používajú MATLAB)

2.2.1.6.2 Softvérové moduly a ich rozdelenie

Detailnejšie boli rozpísané len moduly relevantné pri našom vývoji.

1.Kamera

2.Videnie

Interpretuje aktuálnu prítomnosť elementov ako je lopta, obranné posty, útočné posty, hranice ihriska, pozíciu iných robotov

3.Svet

Modeluje robotov stav na ihrisku, vrátané pózy robota.

4.Telo

Číta údaje zo senzorov, zdrojov a spracúva ich, ale taktiež vie nastaviť pozíciu kĺbov hráča.

5.Pohyb

Diktuje hlavné pohyby hráčovi ako postavenie, sadnutie a pod.

6.Chôdza

"walk engine", ktorý sa stará o samotnú chôdzu, rieši konflikty a preberá zodpovednosť za kĺby, ktoré pracujú s chôdzou.

7.Kop

Upravuje stabilitu hráča počas pohybov, ktoré sa týkajú kopania do lopty. Pomocou tohto modulu vedia nahrať rôzne nastavenia pre kopanie tak aby sa dosiahli rôzne druhy kopov.

8.Keyframes (kľúčové snímky)

Obsahuje zoznam skriptov, pre konkrétne pohyby. Niektoré pohyby sú však vykonávané stále napr. v module Telo, ktoré sa stará o vstávanie.

9.Stav hry

Získava a spracúva stav hry.

10.Stav hlavy

Kontroluje pohyby hlavy, rozhoduje kedy sa prepnúť do módu hľadania lopty, sledovania lopty alebo iba jednoduchého rozhliadania.

11.Stav tela

Rozosiela inštrukcie o pohybe tela, podmienky z predošlých modulov rozhodujú o tom, či robot bude driblovať, hnať sa za loptou alebo vykonávať kopy.

2.2.1.6.3 Videnie

Algoritmus použitý na samotné videnie je podobný tým, ktoré boli použité už v minulosti. Pre lepšie sledovanie a naháňanie lopty ale tento tím implementoval "landmarky", ktoré umožňujú robotovi sa rýchlejšie sa rozhodnúť. Použili napríklad metódu pixelov, kedy rozoznávajú prvý farebný pixel ihriska od bielej čiary.

2.2.1.6.4 Lokalizácia

Problém lokalizácie robota na ihrisku a spoznanie jeho aktuálnej pozície je vyriešený pravdepodobnostným modelom pre predstavenie odhadu pomocou Kalmanoveho filtra, ale aj Markovho modelu a Monte Carlo filtra. Implementácia tohto pravdepodobnostného modelu je ale pomerne zložitá

2.2.1.6.5 Pohyb

Pohyb je kontrolovaný dynamický modulom chôdze, ktorý je skombinovaný s preddefinovanými skriptami. Modul ráta optimálne položenie chodidla tak aby bola chôdza čo najplynulejšia a zároveň bez chýb (pády, zlý krok). Tím používa pomerne často parametrickú chôdzu, ktorú vedia ľahko zmeniť.
Ukážka parametrov je priložená na obrázku.

Obrázok 11 - parametre chôdze

```
------------------------------------------
-- Stance and velocity limit values
------------------------------------------
walk.stanceLimitX={-0.10,0.10};
walk.stanceLimitY={0.09,0.20};
walk.stanceLimitA={-0*math.pi/180,40*math.pi/180};

walk.velLimitX={-.04,.05};
walk.velLimitY={-.02,.02};
walk.velLimitA={-.4,.4};
walk.velDelta={0.02,0.02,0.15}

--Foot overlap check variables
walk.footSizeX = {-0.04,0.08};
walk.stanceLimitMarginY = 0.035;
```

2.2.1.6.6 Kopanie

Kopanie je realizované preddefinovanými skriptami, ktoré robot používa na kopanie rôzneho štýlu. Tieto kopy musia byť špecificky a opatrne vyvíjané , nakoľko je potrebné všetky otestovať a zároveň vyladiť stabilitu, rýchlosť. Taktiež sa tímu podarilo vytvoriť špeciálny kop, ktorý je mixom preddefinovaných skriptov ale i pohybového enginu, vďaka tomu je tento kop rýchlejší ako kopy preddefinované.

2.2.1.6.7 Keyframing

Tento modul obsahuje tzv. snapshoty jednotlivých pohybov a ich náväznosti – ako tieto pohyby musia byť vykonávané v poradí za sebou .To znamená, že snapshot, ktorý má zadefinované z akého stavu sa k nemu dostane pohyb a kde má pokračovať.
Tieto snapshoty tím optimalizoval hlavne na kopy a vstávanie robota.

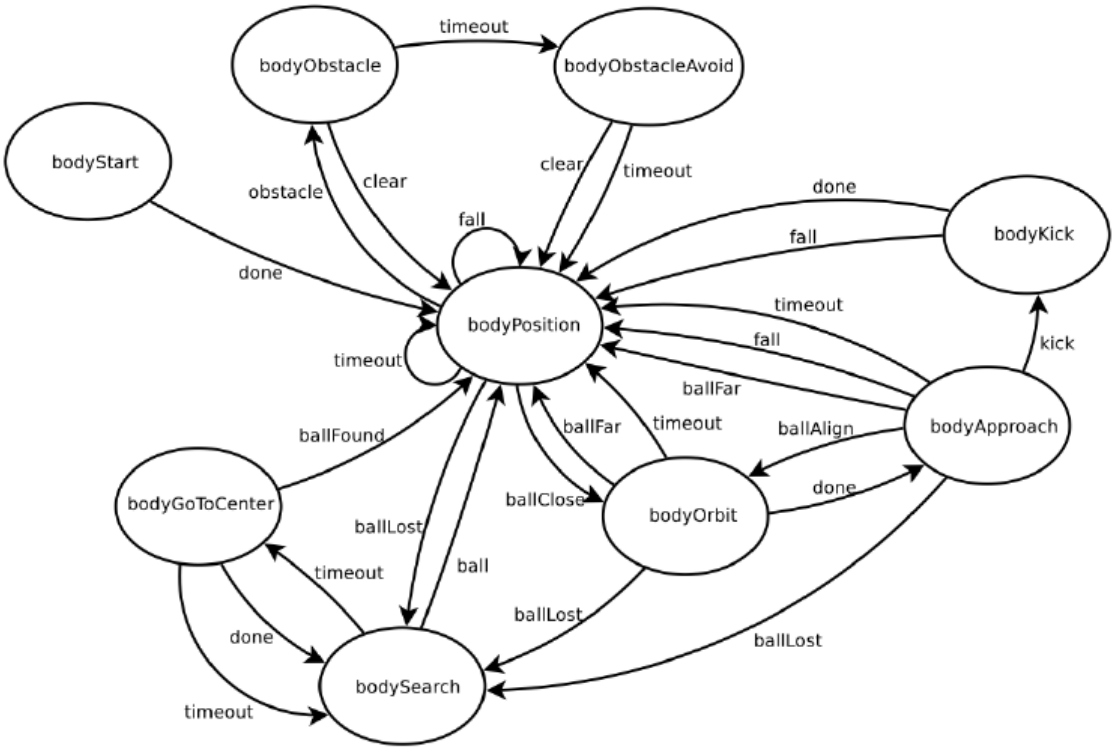
2.2.1.6.8 Kĺby

Robot tohto tímu taktiež obsahuje 22 kĺbov, ktoré plne využíva

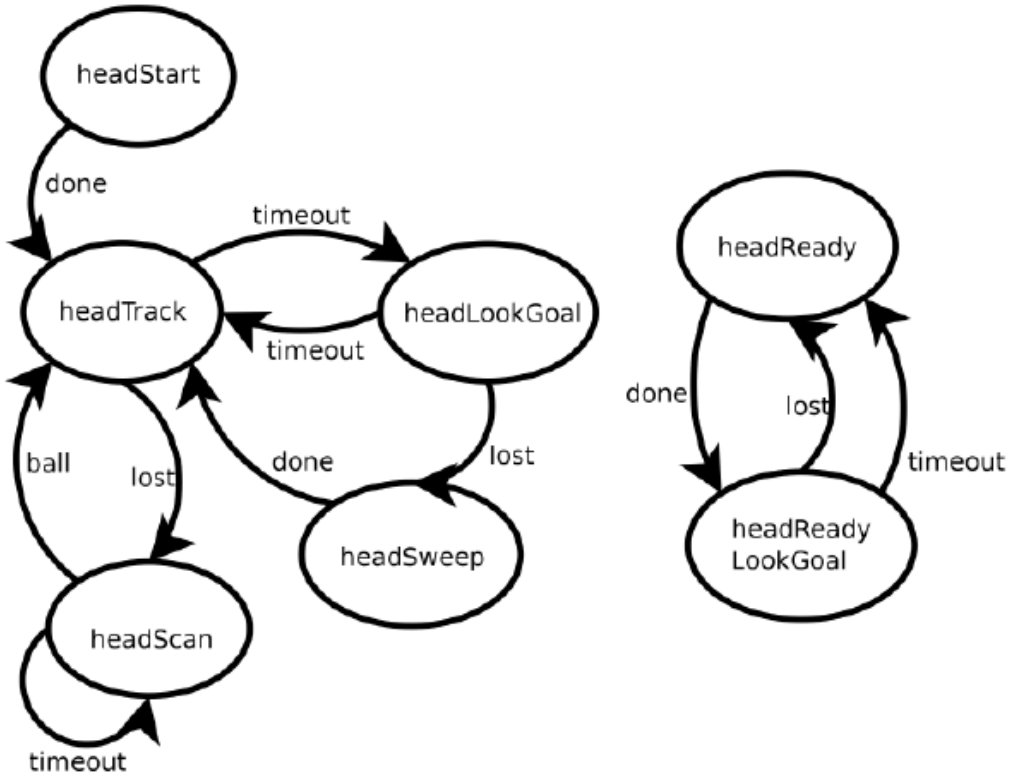
2.2.1.6.9 Správanie

Správanie diktuje konečný stavový automat. Automat má zadefinované stavy napr. pre telo, hlavy. Správanie obsahuje sériu veľkých súborov, ktoré definujú konkrétne stavy. Na obrázku je priložená ukážka stavového automatu, ktorý využíva tím UPennalizers.

Obrázok 12 - deterministický automat pre rozhodovanie



Obrázok 13 - deterministický automat



2.2.1.6.10 Zmena správania

Zmenu správania má tento tím vyriešenú priam excelentne. Ich celá architektúra je postavená tak, že zmena správania sa dá dosiahnuť deklarovaním nového stavu, jeho podmienok a náväzností priamo v jednom konkrétnom súbore, kde sú zadefinované i ostatné stavy.

2.2.2 Web na školskom serveri

Web bol nahratý na školský server a je plne prístupný. Naďalej je web spravovaný na open source systéme wordpress.

2.2.3 Balíky pre ruby

Do agenta Jim boli pridané dva balíčky a to sk.fiit.jim.highskills ktorý je určený na highskilly ktoré boli pôvodne implementované v ruby v priečinku scripts/high_skills. Ďalší balíček sk.fiit.jim.plan obsahuje plány prepísané z ruby z priečinku scripts/plan.

2.2.4 Prepísanie plánovania

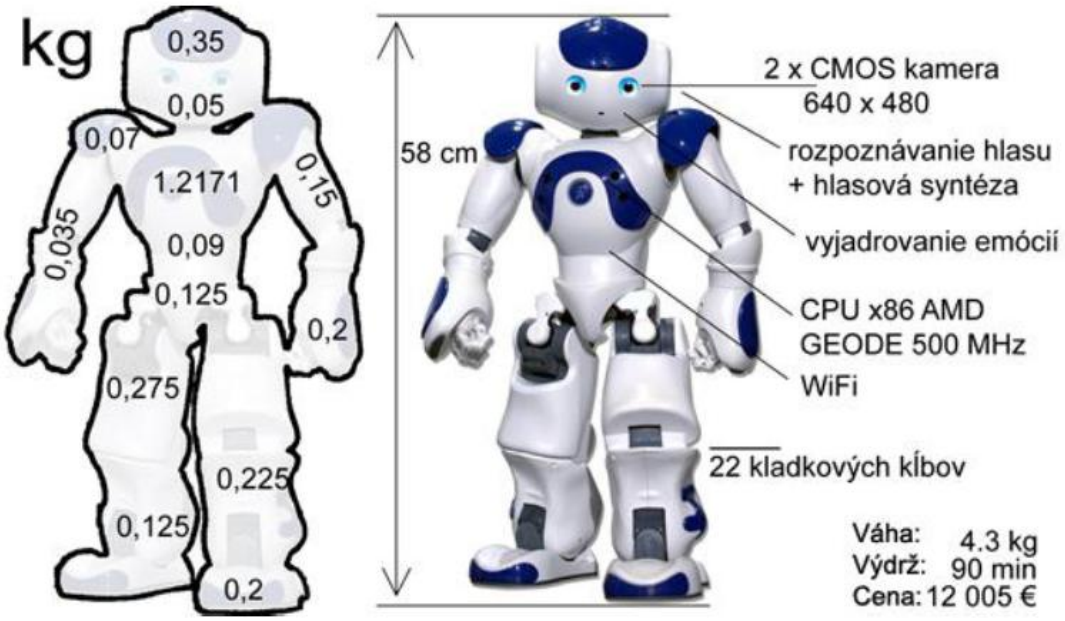
sk.fiit.jim.plan.Plan.java
Je implementovaná trieda ktorá má totožnú funkciu v Jimovy ako mal súbor plan.rb. Súčasťou tejto triedy je veľmi dôležitý rad highskillov s ktorým pracuje metóda control, ktorá buď zabezpečí vykonanie prvého z highskillov alebo v prípade ak máme rad prázdny vykoná metódu replan. Celá trieda je určená najmä nato aby rozširovala ostatné plány preto aj metóda replan neobsahuje žiaden zdrojový kód. Aby sme zamedzili duplicite kódu, táto trieda obsahuje aj metódy See_ball , ball_unseen, turned_to_goal, straight ... ktoré využívajú už spomínané konkrétne plány.
sk.fiit.jim.agent.Planner.java
Táto trieda bola upravená tak aby už nevykonávala ruby ale volala konkrétny plán v jave ktorý sa dá nastaviť v triede sk.fiit.jim.Setting.java napríklad pre nastaveniu plánu PlanTactic použijeme príkaz settings.put("Planner", "PlanTactic").

2.2.5 Analýza diplomových prác

2.2.5.1 Motorika hráča simulovaného robotického futbalu

Práca sa zameriava na zdokonalenie chôdze hráča. Hlavným cieľom je zrýchlenie chôdze, ktorá bola 24 m/s. Aby bola chôdza rýchla a stabilná je potrebné mať znalosti o stavbe tela Nao robota. Obrázok 1 definuje Nao robota aj s jeho charakteristikami.

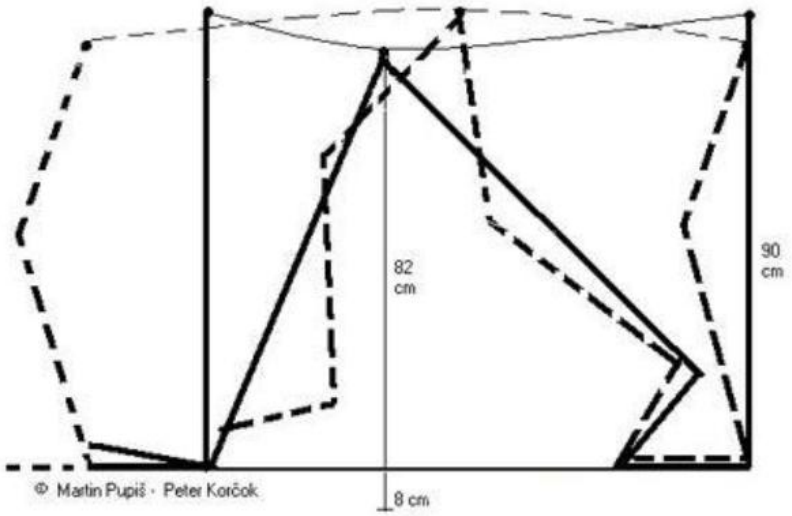
Obrázok 14 - NAO robot



2.2.5.1.1 Biomechanika chôdze a behu

Ľudská chôdza slúži ako vzor pri tvorbe chôdze robota, preto ju autor v práci podrobne analyzoval. Porovnanie trajektórií pohybu chôdze a behu sú znázornené na obrázku 2.

Obrázok 15 - Trajektórie pohybov – chôdza(plná čiara), beh(prerušovaná čiara)



2.2.5.1.2 Možnosti Nao robota

Nao robot nebol vytvorený iba na účel hrania futbalu preto jeho vlastnosti sú komplexnejšie a tým pádom chýbajú iné vlastnosti, ktoré sú vo futbale nevyhnutné. Spoločnosť AldebaranRobotics, ktorá vyvinula Nao robota udáva, že robot nie je schopný behu. Zároveň taktiež udávajú maximálnu konštrukčnú rýchlosť 1 km/h.

Druhou nevýhodou robota je, že nemá ohybnú chrbticu, ktorá by pomáhala pri stabilizácii pohybov tak ako je tomu u človeka.

Poslednou nevýhodou z hľadiska chôdze je, že robot nedokáže ohýbať chodidlo. Z tohto dôvodu nie je možné kopírovať detailne ľudskú chôdzu.

2.2.5.1.3 Algoritmy pre vývoj dynamickej chôdze

Pri chôdzi dvojnohých robotov je nutné riešiť dve otázky. Kam položiť nohu aby bol krok stabilný, optimálny a viedol k cieľu. Druhá otázka je, ktoré kĺby treba zapojiť aby bolo daný pohyb možné vykonať.

- **Zero-moment point (ZMP)**
  Princíp ZMP spočíva v zaistení rovnováhy robota, pri ktorej nie je potrebné mať ťažisko v zóne stability. ZMP počíta aj s pôsobením iných síl ako gravitačná alebo odstredivá. Pomocou ZMP je možné generovať mapu miest kam môže robot stúpiť aby ostal stabilný.

- **Fuzzy logika**

  Fuzzy logika sa využíva na vytvorenie kontrolóra pre pohyby do rôznych strán. Skladá sa z 3 základných častí:

  - **Fuzzy generátor kroku**

    Určuje dĺžku kroku podľa princípu ZMP pričom musí obmedziť výrazné kolísanie dĺžky kroku. Ďalej zabezpečuje aby bolo ťažisko v zóne stability a riadi zrýchlenie.

  - **Fuzzy generátor sklonu tela**

    Ako napovedá názov, fuzzy generátor sklonu tela zabezpečuje nakláňanie tela čo slúži pre rýchlu zmenu orientácie. Naklonením tela sa mení ťažisko, čím je možné meniť ciele chodidla.

  - **Generátor primitívnej chôdze**

    Generátor primitívnej chôdze rozpočítava uhlové rýchlosti jednotlivých kĺbov a tým vytvára výsledný pohyb.

- **Neurónové siete**

  Sama3D je jeden z humanoidných robotov využívajúci neurónové siete pri chôdzi. Hlavným problémom neurónových sietí je veľká časová náročnosť pre ich učenie.

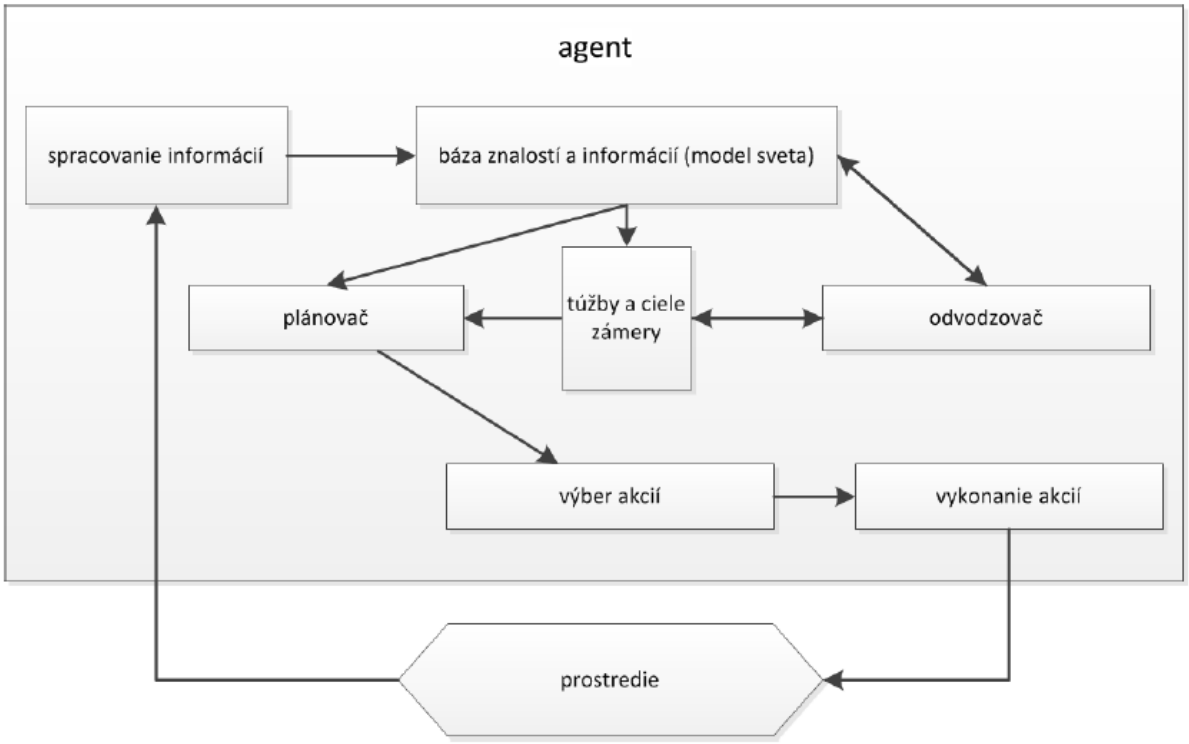
Riešené funkcie pre skvalitnenie chôdze

- Výpočet polohy ťažiska
- Zistenie pozície tela Nao robota
- Výpočet Zero-Moment Pointu
- Inverzné kinematika kĺbov
- Umožnenie hráčovi meniť smer počas chôdze

2.2.5.1.4 Model agenta

Jim je vytvorený na princípe BDI(Belief – Desire - Intention) architektúry. Schéma architektúry BDI je na obrázku 3.

Obrázok 16 - architektúra BDI



2.2.5.1.5 Implementované riešenie

Zero-Moment Point

K výpočtu ZMP boli použité nasledujúce rovnice:

Obrázok 17 - vzorec

$$X_{ZMP} = \mathrm{X} - \frac{\mathrm{Z}\,\ddot{\mathrm{X}}}{\ddot{Z} + g_z}$$

$$Y_{ZMP} = \mathrm{Y} - \frac{\mathrm{Z}\,\ddot{\mathrm{Y}}}{\ddot{Z} + g_z}$$

Kde (X,Y,Z) predstavujú súradnice v karteziánskom súradnicovom systéme, veličiny označené s dvoma bodkami predstavujú 2. Deriváciu polohy na základe času a veličina g predstavuje gravitačné zrýchlenie.
Na výpočet stabilizácie je využitá funkcia FR perceptoru.

2.2.5.1.6 Upravované triedy

`FixedObject.java` – prispôsobenie hráča na nový server 0.6.5
`BodyPart.java` – trieda zavádzajúca podrobné informácie o stavbe tela hráča

`AgentModel.java` – doplnenie výpočtov hybnosti, ťažiska, ZMP súradníc a výstupov z FR perceptorov
`Joint.java` – doplnenie pokročilých údajov o kĺboch, úprava rozsahu efektorov

2.2.5.1.7 Vytvorené plány

`DynamicWalkPlan` – plán pozostáva z volania vyššej schopnosti `WalkFast` (`walk_fast.rb`) s parametrami: *lowskill* a procedúrou vracajúcou uhol otočenia.

2.2.5.1.8 Odporúčania do budúcna

Preprogramovanie agenta do jedného programovacieho jazyka
Naviazanie "informovaného" správania do všetkých pohybov

2.2.5.1.9 Záver

Výsledkom diplomovej práce bolo zrýchlenie chôdze agenta z pôvodných 15 cm/s na 27 cm/s pričom agent bol stabilný, nepadal a presnosť chôdze bola dostačujúca.

2.2.5.2 Rozhodovacia logika

Cieľom tejto práce bolo vylepšiť rozhodovanie hráča, vylepšiť zhodnotenie situácie na ihrisku a nájsť adekvátne riešenie tohto problému.
V tomto riešení bol vytvorený systém na predchádzanie kolíziám s inými hráčmi, taktiež sa pracovalo na schopnostiach viesť tímovú hru – výpočet strategických pozícií, kopnutie do lopty a pohyb vo formáciách. Hráčovi v tomto riešení pribudli schopnosti rozpoznať smer iných hráčov ale aj komunikácia s ostatnými hráčmi.
V tejto práci sa riešiteľ rozhodol venovať hlavne :
Plánovaniu trasy a to akým spôsobom sa vyhľadáva pozícia, na ktorú sa chce dostať, ale tiež aj samotným priebehom presunu hráča

Riešenie možných kolízií s inými hráčmi
Výpočet miesta, kde ma kopnúť loptu

Komunikácia s ostatnými hráčmi

Tímové formácie

Vytvorenie nástroja na testovanie

2.2.5.2.1 Rozhodovanie hráča

Schopnosti rozhodovať sa boli identifikované ako jednoduché – lineárne – pomerne na nízkej úrovni. Úroveň tímovej hry bola identifikovaná ako "takmer žiadna".
Spomenuté lineárne rozhodovanie spočíva v tom, že hrá
, ktorý je najbližšie k lopte sa k nej presunie, aby ju mohol odkopnúť priamo na bránu. Túto
jednoduchú taktiku je možne ľahko odhaliť, preto nie je vhodná pre zúčastnenie sa akejkoľvek súťaže.

Riešiteľ sa rozhodol na vylepšenie použiť analytickú geometriu. Zdalo sa to ako vhodné riešenie, pretože pri plánovaní môžeme upustiť od 3D priestoru a každú situáciu jednoducho vyjadriť pomocou jednoduchých útvarov na 2D ploche. Každý hráč by predstavoval bod, ktorý bude tvoriť stred útvaru, pomocou ktorého vyjadríme dosah hráča. Pre tento účel bola využitá kružnica a elipsa – najlepšie opísateľná akčná oblasť v okolí hráča.

Ďalším útvarom pri riešení problému bude priamka, ta predstavuje najlepší spôsob ako sa dostať z bodu A do bodu B.

Rozhodovanie vytvorené len na základe analytickej geometrie ale nedosahovalo také dobré výsledky ako boli očakávané.

Preto bolo navrhnuté vylepšenie obídenia prekážky pomocou grafu a dijkstrovho algoritmu.

2.2.5.2.2 Kolízny model hráča

Slúži na zhodnotenie aktuálnej situácie na ihrisku z pohľadu hráča a jeho zámerov, či mu pri ich Vykonávaní hrozí kolízia s iným hráčom. Ak dôjde k odhaleniu možnej kolízie, v rámci kolízneho modelu sa navrhnú všetky potrebne opatrenia, aby sa tomu včas zabránilo.

2.2.5.2.3 Rozhodovanie pomocou grafov

V riešení sa rozhodlo pozrieť na problém z trochu iného uhla a to tak, že namiesto optimálnej cesty budeme hľadať najkratšiu cestu ohodnoteným grafom. Hrany grafu budú nadobúdať rôzne hodnoty na základe prítomnosti protihráčov na ihrisku. Penalizáciou takýchto hrán docielime želaný efekt vyhýbaniu sa prekážkam. Graf vytvorili ako sieť vrcholov a hrán, pričom prepojene budú iba susediace vrcholy. Vzdialenosť medzi vrcholmi stanovili na 0,5 m, Čím získali na osi x 42 vrcholov a na osi y 28 vrcholov, čo dohromady znamená graf s 1176 vrcholmi. Najkratšiu cestu hľadali pomocou dijkstrovho algoritmu.

Na obrázku 4 je zobrazená jedna zo situácií, ktorú použili v diplomovke.

Obrázok 18 - situácia z diplomovky

2.2.5.2.4 Predikcia pohybu hráča

Riešiteľ sa snažil o čo najviac údajov o polohe hráča v určitom časovom rozmedzí. Nad údajmi následne chcel vykonať jednoduchú lineárnu regresiu pomocou metódy najmenších štvorcov. Výstupom lineárnej regresie má byť rovnica priamky, z ktorej je možné dosadením konkrétnych .

Pri použití lineárnej regresie je dôležité sledovať koeficient korelácie ***r***. Korelácia je miera závislosti medzi dvoma alebo viacerými premennými. Korelačný koeficient nadobúda hodnoty od -1 po 1, pričom hodnota 0 vypovedá o žiadnej korelácii. To znamená, že premenne sú od seba nezávisle. Vo všeobecnosti sa snažíme získať čo najvyšší koeficient korelácie (-1 alebo 1), pretože nám vypovedá o sile relácie medzi premennými. Pokiaľ sa koeficient blíži k -1 alebo 1 mohli povedať, že sledovane hodnoty sú na sebe závisle.

2.2.5.2.5 Pozícia hráča vo formácii

Realizácia tímového správania sa riešiteľovi javila ako dobrý štart pre tímové formácie. Hráči by tak boli schopní hrať taktickú hru, zaberať strategické pozície na ihrisku. Formácia by mala pozostávať z troch jednotiek – obrany, útoku a stredu.
Navrhnuté boli štyri typy formácií : 2-1-1, 1-2-1, 1-1-2 a 2-2.
Výpočet pozície hráča sa pri návrhu rozhodli odvodiť od aktuálne pozície lopty na ihrisku. Každý hráč bude mať na základe svojej role stanovenú základnú pozíciu. Táto

pozícia bude vychádzať zo situácie, kedy sa lopta nachádza priamo v strede jednotlivých časti ihriska. Do úvahy sa berie x a y súradnica lopty.

Formácia 1-1-2 –útočná formácia. Dvaja útočníci sú schopní pri útoku zakladať kombinácie a v prípade slabej súperovej obrany je veľká pravdepodobnosť k úspešnému ukončeniu útoku gólom. Nevýhoda tejto formácie spočíva v možnom prečíslení nášho obrancu pri obrane, čim sa zvyšuje riziko inkasovania gólu.

Formácia 1-2-1 – neutrálna formácia. Ma posilnený stred, čo by malo umožniť ľahko preniesť hru na súperovu polovicu. Tuto formáciu by sme mohli označiť za vyrovnanú, pretože stredopoliari sú schopní rovnakým spôsobom podporiť obranu ako aj útok a do hry by sa po väčšinu času mali zapájať až traja hráči.

Formácia 2-1-1 – obranná formácia. Tato formácia sa sústredí na zastavenie súpera pri streľbe na bránu. Dvaja obrancovia sú schopní pokryť dvoch hráčov a nie je ich preto jednoduché oklamať jednoduchými kombináciami.

Formácia 2-0-2 – pri tejto formácii vychádzame z rozloženia hráčov pri futsale. Hráč ma pri tejto formácii definovanú pomerne veľkú domovskú oblasť a preto je viac v pohybe oproti iným formáciám.

2.2.5.2.6 Komunikácia

Server dovoľuje hráčovi v ľubovoľnom okamihu odoslať správu s maximálnou dĺžkou 20 znakov. Ak sú intervaly medzi odosielanými správami menšie ako 0,04s , budú tieto správy zahodené. Pri riešení komunikácie je potrebné navrhnúť parser pre hear preceptor.
Navrhovaná štruktúra správy:
· presný identifikátor (číslo alebo písmeno).
· Identifikátor odosielateľa – číslo hráča.
· Identifikátor prijímateľa- číslo hráča alebo 0 ak je sprava určená všetkým.
· Ďalšie informácie – text samotnej správy.

Bolo navrhnutých osem typov správ:
oznámenie pozície – hráč odosiela svoju globálnu pozíciu. V tele tejto správy hráč odosiela svoje súradnice. Tieto sú zaokrúhlené na 1 desatinne miesto. Keďže najväčšie možne takto odosielane číslo je -10,5, čo sa dna zapísať ako -105, každá súradnica vyžaduje 4 miesta. Napríklad číslo 5 sa transformuje na +050. V správe sa udávajú súradnice x, y , namiesto súradnice z sa odošle iba jedno číslo 0-ak hráč spadol a leží na zemi alebo 1 ak je hráč postavený. Posledný parameter, ktorý hráč odošle, je veľkosť uhla natočenia hráča vzhľadom na ihrisko. Táto hodnota sa udáva v stupňoch od 0 po 360, čiže vyžaduje 3 miesta. Takáto sprava ma spolu s hlavičkou dĺžku 15,

· ***žiadosť o oznámenie pozície hráča*** – ak hráč potrebuje vedieť pozíciu niektorého zo spoluhráčov, odošle tento typ správy spolu s číslom hráča, o ktorého pozíciu ma záujem,

· ***žiadosť o oznámení pozície lopty*** – ak hráč nevie, kde je lopta, môže o tuto informáciu požiadať svojich spoluhráčov. Nemá definovane žiadne telo správy,

· ***informácia o získaní lopty*** – ak hráč získa loptu informuje o tom svojich spoluhráčov. Tato informácia slúži na uvedomenie si spôsobu hry. V tomto prípade by sa spoluhráči mali stavať na ofenzívne posty. Keďže od hráča s loptou sa odvíja hra, je potrebne, aby ostatní spoluhráči vedeli na koho sa majú sústrediť. Nemá definovane žiadne telo správy,

informácia o strate lopty – ak hráč príde neúmyselné o loptu, zavolá tuto správu buď pri páde, alebo ju o ňu pripraví protihráč. Tato sprava ma za úlohu nabudiť najbližšieho hráča pri lopte, aby sa ju pokúsil získať. Nemá definovane žiadne telo správy,

· ***o výzve na prihrávku*** – pri prihrávke hráč s loptou informuje svojho spoluhráča o úmysle mu nahrať loptu. V tom prípade sa spoluhráč pripraví na prijatie prihrávky. V tele správy sa posielajú globálne súradnice x a y, na ktoré chce hráč prihrať. Jedna súradnica sa podobne ako pri posielaní pozície hráča zapisuje pomocou 4 znakov,

· ***odpoveď na výzvu o prihrávku*** – hráč muší na výzvu o prihrávku odpovedať. V odpovedi potvrdí prijem alebo odmietnutie prihrávky. V tele správy hráč posiela 0 ak nechce prijať prihrávku alebo 1 ak prihrávku akceptuje,

· ***idem k lopte*** – ak hráč zisti, že sa nachádza najbližšie klopte, odošle o tom informáciu všetkým hracom, ktorou dáva najavo svoj úmysel získať loptu. V tele správy odosiela jeho vypočítanú vzdialenosť od lopty, aby boli zvyšní spoluhráči schopní porovnať ich vzdialenosť od lopty. V prípade, ak iný hráč zisti, že sa nachádza pri lopte bližšie, odošle opäť tuto správu so svojou vzdialenosťou. Prvý hráč po prijatí tejto správy mal by zmeniť svoje rozhodnutie a namiesto chôdze k lopte prejsť na ofenzívnu pozíciu.

Riešiteľ sa rozhodol definovať presné časové úseky v ktorých je presné určené okno, v ktorom môžu jednotliví hráči posielať správu. Vzhľadom na to, že každý hráč pozná aktuálny čas, dĺžku intervalu, svoje číslo a veľkosť tímu, je určenie okamihu jednoduché.

2.2.5.2.7 Nástroj na testovanie

• **Model sveta** obsahuje informácie o hráčovom modeli sveta, ako je pozícia lopty, pozície protihráčov a spoluhráčov a vlastnú pozíciu. Okrem toho sa v tomto okne majú nachádzať aj informácie o aktuálne vykonávaných činnostiach z plánu. Z trénera od tímu androids chceli v tomto okne využili možnosť meniť polohu lopty a získanie reálnej pozície lopty a hráča zo serveru. Súčasťou tohto okna je okrem textovej informácie aj grafické znázornenie sveta,

• **správy** v tomto okne by sa zobrazovali prijate správy, ale taktiež aj pripravene správy, ktoré sa hráč chystá odoslať. Pomocou tohto okna chceli hráčovi vkladať správy, ktoré ma odoslať,

• **kolízie** vzhľadom na to, že riešenie kolízii tvorí značnú časť tejto prace, chceli vytvoriť okno, v ktorom sa budú nachádzať informácie o aktuálnych kolíziách spolu s ich navrhovaným riešením. Textové informácie by boli doplnene o grafičke znázornenie situácie spolu s jej riešením,

• **manipulácia** keďže si stanovili ako cieľ manipuláciu s vnútorným svetom hráča, Potrebovali k tomu špeciálny panel na ktorom budú mať k dispozícii funkcie ako pridanie a odobranie spoluhráča/protihráča a nastavenie cieľa pre chôdzu a kopnutie,

• **upravené JCC** keďže pôvodný nastroj využívaný hracom JIM je pomerne kompaktný a hodí sa k navrhnutým komponentom, chceli ho len mierne upraviť a integrovať k nášmu riešeniu. V JCC je dobre riešene vypisovanie logov hráča. Jedinú úpravu, ktorú si to vyžaduje, je doplnenie automatických aktualizácii bez nutnosti klikania pre získanie najnovších údajov.

Požadované funkcie

Možnosť zadávať presne globálne a relatívne súradnice
Vyber typu hráča, ktorého chceme pridať (spoluhráč, protihráč)
Možnosť meniť pozíciu vloženého hráča a taktiež ho aj odobrať
Možnosť pevne nastaviť hráčovi cieľ pre kopnutie alebo chôdzu

2.2.5.2.8 Implementácia rozhodovania

2.2.5.2.8.1 Kolízny model

Pre riešenie kolízii sa implementovali triedy v rámci kolízneho modelu, ktoré sa umiestnili do balička agent.models.collision. V tomto modeli sú implementovane triedy, ktoré umožňujú pomocou analytickej geometrie odhaliť a navrhnúť riešenia možných kolízii. Vzhľadom na to, že funkcionalita hráča JIM je rozširovaná prostredníctvom modulov, mohli sme implementáciu realizovať nezávisle od hráča, čo uľahčilo ladenie a testovanie. Počas implementácie vytvorili jednoduchú aplikáciu, do ktorej sa mohli ručne zadávať údaje reprezentujúce model sveta hráča a následne sledovať, či navrhnutý spôsob dokáže odhaliť kolízie a aké ponuka možnosti ich riešení. Vzhľadom na to, že cely kolízny model je založený na analytickej geometrii, bolo celu situáciu jednoducho vykresliť a overiť tak správnosť vypočítaných údajov.

Inštancia triedy CollisionModel sa vytvára pri vykonávaní vyšších schopnosti chôdze alebo kopu. V najvyššej vrstve sa stanoví hráčovi cieľ, kam sa ma dostať. Tento cieľ sa otestuje na možne kolízie, v prípade nájdenia prekážok sa navrhne alternatívna trasa k cieľu, ktorá sa pošle do nižšej vrstvy, kde sa rieši aký pohyb sa bude vykonávať. V prípade odhalenia možnej kolízie tento model upravuje cieľ trasy hráča tak, aby ku kolízii nedošlo.

2.2.5.2.8.2 Výber alternatívneho cieľa

Prvý spôsob výberu alternatívneho cieľa spočíval v nasmerovaní hráča priamo na bod dotyku priamky s kružnicou predstavujúcou prekážku. Problém, ku ktorému pri tomto spôsobe dochádzalo bolo, že hráč po dosiahnutí tohto cieľa ešte stále nemal voľnu cestu k dosiahnutiu cieľa, čo viedlo k viacnásobnému hľadaniu dotyčníc. Takto navrhnute riešenie dosahovalo v porovnaní s predchádzajúcim oveľa lepšie výsledky, pretože hráč najneskôr pri dosiahnutí stanoveného bodu zistil, že pri pohybe z aktuálneho miesta k cieľu nehrozí kolízia a teda úspešné obišiel prekážku

2.2.5.2.8.3 Využitie grafov pri rozhodovaní

Ako vyplýva z návrhu, rozhodovanie hráča o spôsobe riešenia kolízii spočíva v dvoch krokoch. Prvý krok je realizovaný prostredníctvom analytickej geometrie, kedy dochádza k odhaleniu kolízii a navrhnutiu riešenia lokálnej situácie, ktorá berie do úvahy iba hráča a prekážku.

Druhy krok je realizovaný prostredníctvom dijkstrovho algoritmu na grafe, ktorý reprezentuje aktuálnu situáciu na ihrisku. Ako riešenie kolízie sa nakoniec využíva syntéza oboch krokov, kedy sa riešenia vyplývajúce z prvého kroku porovnávajú s riešením v druhom kroku a vyberá sa to lepšie z hľadiska globálnej situácie. Jednoduchšie povedané, z navrhovaných dotyčníc k prekážke vyberáme tu, ktorá sa nachádza bližšie k najkratšej ceste z dijkstrovho algoritmu.

2.2.5.2.8.4 Predikcia pohybu hráča

V návrhu sa zaoberali možnosťami predikcie pohybu hráčov, pričom sa sústredili na využitie lineárnej regresie. Tuto funkcionalitu implementovali prostredníctvom triedy PlayerTracking. Hlavnou myšlienkou navrhnutého riešenia bolo zozbierať množstvo údajov, z ktorého je následné možne pomocou lineárnej regresie získať smer a rýchlosť hráča. Pri každej aktualizácii polohy hráča, sa nová poloha zaznamená spolu s časom pozorovania v triede PlayerTracking. Pozície, ktoré ukladajú do zoznamu, sú vkladane ako globálne súradnice pozície hráča. Pokiaľ mame dostatočne veľkú vzorku aktuálnych dať, čo v terajšom riešení znamená aspoň desať pozícii nie starších ako tri sekundy, vypočítame korelačný koeficient vzorky.

2.2.5.2.8.5 Strategické pozície

Výpočet strategických pozícii sa realizoval upravením používaných metód napísaných v Ruby. Jedna sa o dve metódy z triedy StrategicPositionCalculator a to defensive_position() a offensive_position(). Hráč pritom rozlišuje situáciu, kedy je

aktívne zapojený do riešenia lokálnej situácie a spolupracuje s iným hráčom a snaží sa nabiehať na prihrávku alebo sa snaží pokryť protihráča. Pri výpočte lokálnych pozícii sa využíva trieda CollisionModel. Pri riešení globálnej situácie sa hráč snaží dostať na svoju pozíciu v rámci tímovej formácie. Výpočet pozície vo formácii je implementovaný ako metóda v Ruby. V Ruby scripte formation.rb ma každý hráč zadefinovanú domovskú pozíciu, ktorá vyplýva z roly hráča a metody apply_ball_x() a apply_ball_y(), ktoré transformujú základnú domovskú pozíciu hráča tak, aby zodpovedala aktuálnej pozícii lopty. Formácie je možne aktivovať alebo deaktivovať v súbore settings.rb.

2.2.5.2.8.6 Komunikácia

Implementácia komunikácie spočívala vo vytvorení samostatného modulu, ktorého úlohou bolo spracovať prijaté a generovať odosielané správy. Preto boli vytvorené dve triedy:

- MesageFactory – prijíma podnety na vytvorenie správ, ktoré následne ukladá do zásobníka. Tie sa vyberajú hneď ako hráč má možnosť prehovoriť a upravujú sa do formátu aby im každý porozumel. Taktiež je naviazaná na triedu Communication, ktorá ma na starosti prepojenie so serverom.
- MessageDecoder – má za úlohu rozpoznať prijatú správu a určiť typ správy na základe informácie, ktorú správa obsahuje.

Testovanie tohto riešenia pozostáva z dvoch častí. Prvou je schopnosť hráča odoslať správu a druhá porozumieť prijatej správe.

2.2.5.2.8.7 Nástroj na testovanie

Tento nástroj pozostáva z piatich komponentov a poskytuje všetky potrebné informácie o stave hráča, o jeho cieľoch a samotnom rozhodovaní. Najväčší dôraz sa kládol na riešenie problémov s kolíziami a preto sa jeden z vytvorených komponentov venuje výhradne tejto problematike.

2.2.5.2.9 Testovanie

V tejto kapitole boli predstavené dosiahnuté výsledky z testovania novonadobudnutých schopností hráča.

2.2.5.2.9.1 Testovanie riešenia kolízií

Pre overenia riešenia boli navrhnuté jednoduché situácie, pri ktorých sa sledovalo rozhodnutie hráča. Základom testu bolo vloženie jedného umelého statického hráča medzi pozíciu hráča a lopty. Takýmto spôsobom boli vytvorené štyri modelové situácie.

- **Modelová situácia č. 1 (identifikovanie prekážky a jej riešenie)** - z dosiahnutých výsledkov vyplýva, že hráč je schopný správne identifikovať prekážku a následne ju obísť, pričom dochádza k predĺženiu času potrebného k prechodu na požadovanú pozíciu o 6 sekúnd, čo je ale oproti situácii kedy hráč nemal schopnosť vyhýbania sa prekážok zlepšenie až o 54 sekúnd. Vyhýbanie sa prekážkam má preto určite zmysel použiť pri plánovaní trasy hráča.
- **Modelová situácia č. 2 (obídenie prekážky)** - Ako dokazujú zaznamenané výsledky, hráč bol schopný v 9 z 10 prípadov správne zvoliť smer pre obídenie prekážky. Jedna zlá voľba smeru bola spôsobená nepresným určením pozície prekážky čo viedlo k zlému rozhodnutiu, napriek tomu považujeme dosiahnuté výsledky za dobré.
- **Modelová situácia č. 3 (zložitejšia situácia na ihrisku a jej riešenie)** - Hráč je pri obídení prekážok takmer 2 krát rýchlejší a menej náchylný k pádom, pretože plynule obchádza prekážku bez zastavenia. Navyše zbehnutie použitého algoritmu trvá priemerne 20 ms, čo je približne 50 krát rýchlejšie ako genetický algoritmus, ktorý bol používaný predtým.
- **Modelová situácia č. 4 (dynamický hráč)** - V testoch sa náš až štyri krát rozhodol protihráča obísť v smere jeho pohybu, čo bolo považované za nedostatočné výsledky. Tento problém ale nebol spôsobený zlým kolíznym modelom, ale nepresnými údajmi o polohe hráčov, kedy sa domnieval, že jeho pozícia a smer protihráča boli iné ako v skutočnosti (globálne premenné).

2.2.5.2.9.2 Testovanie predikcie smeru pohybu hráča

Pre testovanie predikcie pohybu hráča bol navhrnutý test, pri ktorom bol umiestnený jeden hráč na pozíciu X a druhý na pozíciu Y. Následne hráč z pozície Y prechádzal na pozíciu X. Počas toho druhý hráč spozoroval pohyb prvého hráča a odhadoval jeho smer a v 59% ho odhadol správne.

2.2.5.2.9.3 Testovanie formácií

Pri testovaní formácií boli použití štyria hráči, ktorí mali počas testu definované rôzne roly (pozícia vo formácii). Formácia bola použitá 1-2-1 a na začiatku bola lopta na súradniciach [0,0]. Hlavným cieľom testov bolo ako sa hráči dokážu prispôsobiť aktuálnej pozície lopty. Počas testovania bolo zistené, že hráči dokážu správne zaujať svoje pozície a taktiež sa výborne pohybovať vo formáciách a prispôsobovať svoju polohu na základe polohy lopty.

2.2.5.2.9.4 Testovanie tímového správania

Testovanie prebiehalo v dvoch fázach, v prvej fáze bolo cieľom či sa daný hráč dokáže zorientovať a nabehnúť si prihrávajúcemu hráčovi na nahrávku. A v druhej fáze, zas či je

hráč schopný nájsť spoluhráča a prihrať mu loptu správnym smerom a s dostatočnou rýchlosťou. Obe fázy testu zbehli úspešne a bolo dokázané, že hráč je schopný vykonať obe činnosti.

2.2.5.2.9.5 Testovanie komunikácie

Testovanie komunikácie prebiehalo taktiež v dvoch fázach ako testovanie tímového správania. V prvej fáze prebiehala komunikácia medzi hráčmi tak, že boli rozmiestnení na ihrisku a jeden kričal druhému žiadosť o výzvu na nahrávku a čaká odpoveď. Treba však podotknúť, že do hráči neberú do úvahy svet. Výsledkom testu bolo dokázané, že komunikácia funguje dostatočne dobre. V druhej fáze sa testovalo či hráč dokáže pochopiť prijatú správu. Test prebiehal tak, že jednému hráčovi boli manuálne zadávané správy a on ich interpretoval druhému hráčovi. Ten pokiaľ správu prijal odpovedal späť a vykonal akciu, ktorú prijal v podobe správy od prvého hráča. Dosiahnuté výsledky dokazujú, že hráči sú schopní medzi sebou komunikovať a taktiež dokážu správne interpretovať obsah správ a vykonať požadované akcie.

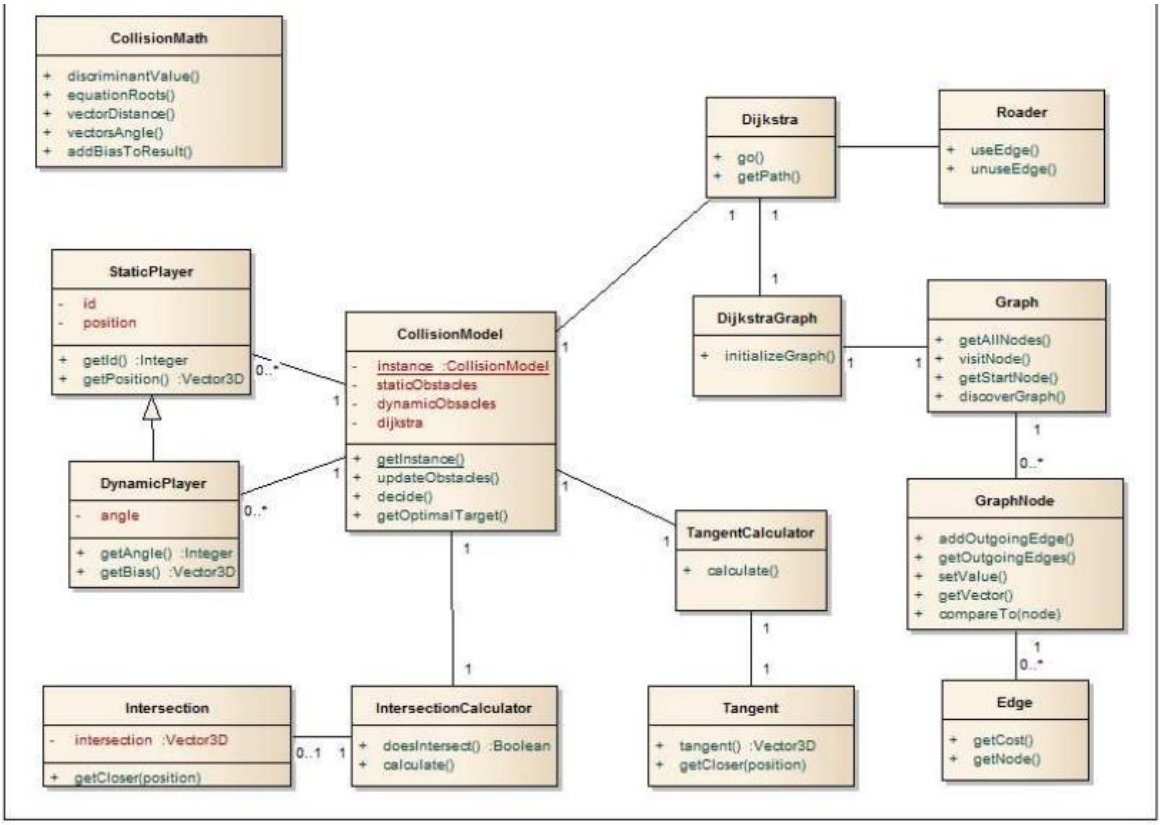
2.2.5.2.10 Vylepšenia

Po dokončení práce boli spísané možné vylepšenia, ktoré by bolo dobré do tejto implementácie zapracovať. Konkrétne to boli:

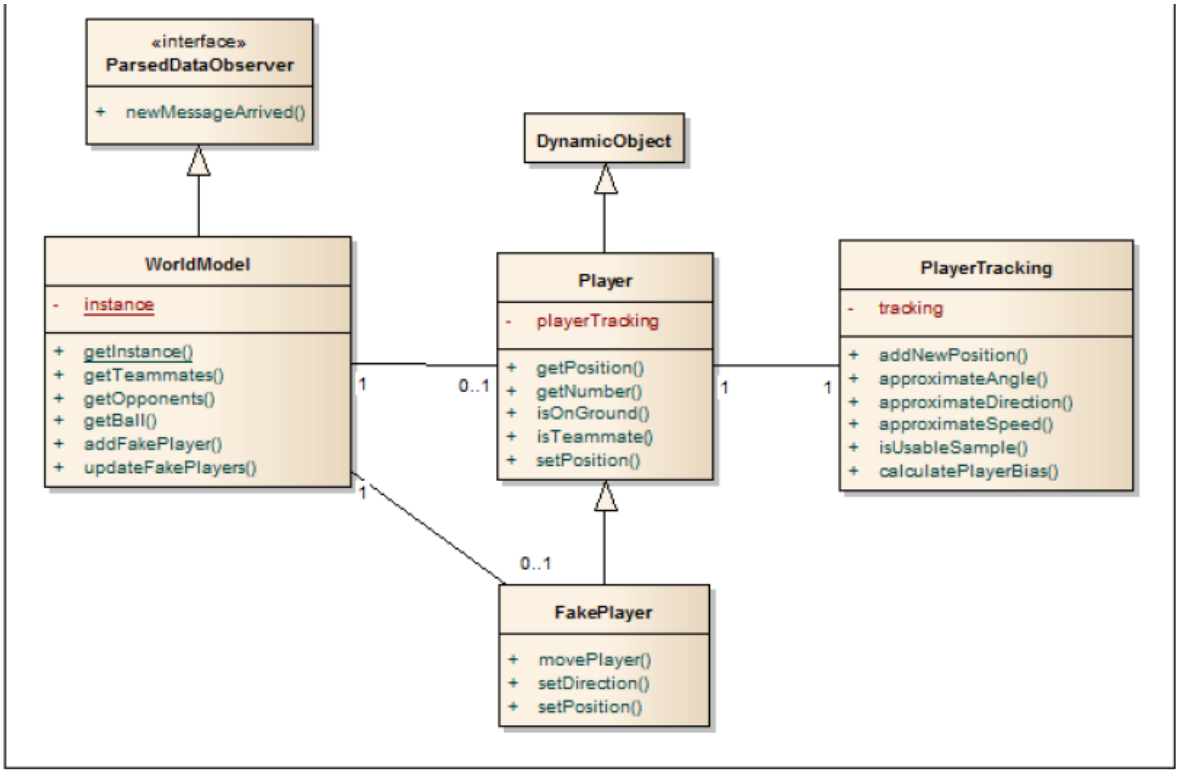
1. Zlepšenie presnosti získaných údajov
2. Predikcia pohybu hráča, tak aby sa zohľadňovala vzdialenosť hráča od prekážky
3. Vymeniť elipsu za dve kružnice, jedna kružnica by bola okolo aktuálnej polohy hráča a druha okolo odhadovanej budúcej polohy
4. Dynamické menenie rozmerov elipsy na základne pohybu prekážky

2.2.5.2.11 Diagramy

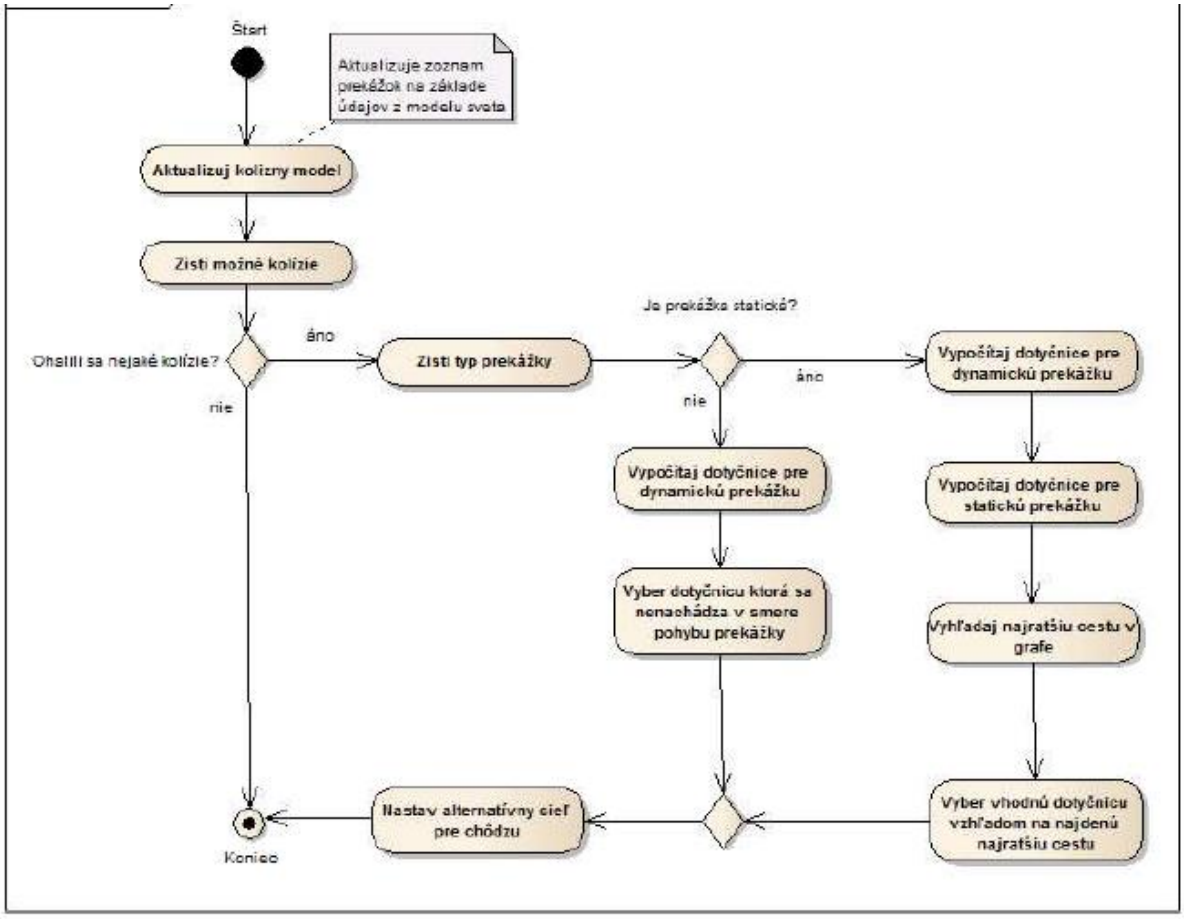
Obrázok 19 - diagram tried pre kolízny model



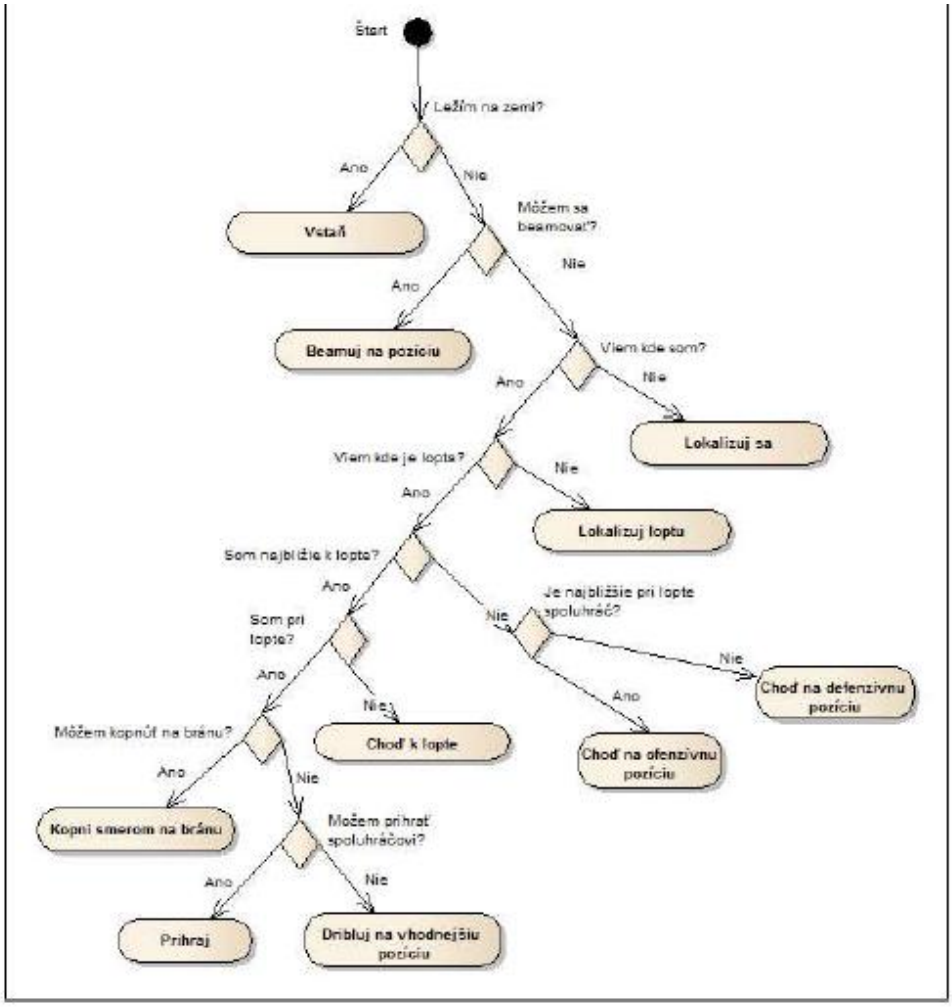
Obrázok 20 - diagram tried pre kolízny model



Obrázok 21 - algoritmus pre rozhodovanie kolízií



Obrázok 22 - Rozhodovanie hráča



2.2.6 GIT

Všetci členovia tímu úspešne spojazdnili GIT. Tím sa prepojil s druhým tímom. Práca sa bude vykonávať nad jedným repozitárom.

2.2.7 Návody na inštaláciu

Na robocup Wiki boli pridané návody na inštaláciu pre MAC a Linux.

2.2.8 Spoločný backlog

V jire bol vytvorený spoločný backlog, ktorý slúži pre oba tímy súčasne.

2.2.9 Dokumentácia

Bola spísaná dokumentácia za celý predošlý šprint. Následne boli tieto dokumenty spojené do jedného celku.

2.3 Šprint č. 3

Tabuľka 4 - backlog šprint

ID	AKO	CHCEM	ABY
3.1	Člen tímu	Určiť či zakomentovaný kód funguje	Sa mohol upratať kód
3.2	Člen tímu	Odstrániť nefunkčný kód	Sa mohol upratať kód
3.3	Člen tímu	Navrhnúť a implementovať taktickú vrstvu	Zlepšenie architektúry a rozhodovania
3.4	Člen tímu	Navrhnúť a implementovať strategickú vrstvu	Zlepšenie architektúry a rozhodovania
3.5	Člen tímu	Vytvoriť architektúru highskills	Zlepšenie pohybov a rozhodovania pre ich využívanie
	Člen tímu	Upraviť dokumentáciu k riadeniu a produktu	Odovzdanie pre kontrolný bod

Tabuľka 5 - rozdelenie úloh

ID Pp	ID podúlohy	Úloha	Zodpovedný
3.1		Určiť či zakomentovaný kód funguje	Petráš,Adámik,Čerešňák
3.2	-	Odstrániť nefunkčný kód	Petráš,Adámik,Čerešňák
3.3	-	Navrhnúť a implementovať strategickú vrstvu	Všetci
3.4	-	Vytvoriť architektúru highskills	Všetci
3.5		Upraviť dokumentáciu k riadeniu a produktu	Petráš

2.3.1 Funkčnosť zakomentovaného kódu

SK.FIIT.JIM.AGENT.COMMUNICATION
Communication.java - metóda restart() - bola implementovaná Androidmi , pre vykonanie reštartu

SK.FIIT.JIM.AGENT.MODELS
AgentModel.java – Obsahuje komentáre typu : test
LastDataReceived – zakomentované kvôli náročnosti prijímaných dát, pamäťová náročnosť – komentár – ak sa začne používať celé treba pridať

AgentPositionCalculator.java - kód, ktorý maže queue na lastposition ak je velkost 21 ...
ANALYZERESULT - len analýza výsledkov

DynamicObject.java - predictPosition() – nefunguje správne, chyba neznáma, je to bez komentára. Vyzerá ako keby niekto zabudol zmazať alebo nevedel čo stým.

TacticalInfo.java – nedoriešené

WorldModel.java - predictionBall2 - zakomentované lebo treba dorobiť ---> adent.models.prediction.Prophet

double angle = player.getAbsoluteRotation(); - uhol, ktorý sa nikde nepoužíva - pravdepodobne sa niečo testovalo ..

SK.FIIT.JIM.AGENT.MODELS.PREDICTION
Prophet.java - toto je spojené s predictionBall2 - nefunguje
- dole je zakomentovaný kód OLD FROM androids - nikde nieje uvedené či TODO alebo len niečo iné .. je tam ale @problem
- metódu ale treba nechať je to metóda na calculateBallPrediction ...

SK.FIIT.JIM.AGENT.PARSING
ParserTest.java - zakomentovaná časť slúži ako test
Perceptors.java – nezistené príčiny
SeePerceptors.java – zakomentované

SK.FIIT.JIM.AGENT.SERVER
TFTPServer.java – zakomentovaný kód slúži len ako príklad použitia TFTP triedy, ktorá podľa mňa slúžila len ako inšpirácia pri implementovaní triedy TFTPServer.java.

SK.FIIT.JIM.ANNOTATION.DATA
XMLCreator.java – zakomentovaná časť kódu mení spôsob výpisu namiesto do súboru na obrazovku.

SK.JIM.ANNOTATION.GUI
Window.java – zakomentovaná čast sa pokúša o vytvoriť xml súbor s pohybom a následne ho validuje.

SK.FIIT.JIM.GUI
ReplanWindow.java –
this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE) - iba zatvorí okno, proces stále beží na pozadí, namiesto toho sa využíva this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
môže byť odstránené

SK.FIIT.JIM.LOG
LogTest.java - zakomentovana časť slúži ako test

SK.FIIT.JIM.TESTS
GoalieTestCase.java - zakomentovaná časť slúži ako test, po odkomentovaní nie je spustiteľný

GoalieTestCaseTest.java - zakomentovaná časť slúži ako test, nedokončené nespustiteľné

TestJim.java - zakomentovaná časť slúži ako test, po odkomentovaní nie je spustiteľný, zrejme ešte využíva ruby skripty
Celý balík môže byť odstránený

2.3.2 Odstránenie zakomentovaného kódu, nepoužívaných funkcií a premenných

Kód vyhodnotený ako nepotrebný bol odstránený. Veľká časť zakomentovaného kódu bola nepoužiteľná alebo v stave, v ktorom nedávala zmysel.

2.3.3 Architektúra – situácie,taktiky, highskilly

2.3.3.1 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.

2.3.3.2 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 skôre 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

2.3.3.3 Taktiky

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

Kedy sa (ne)mení taktika?

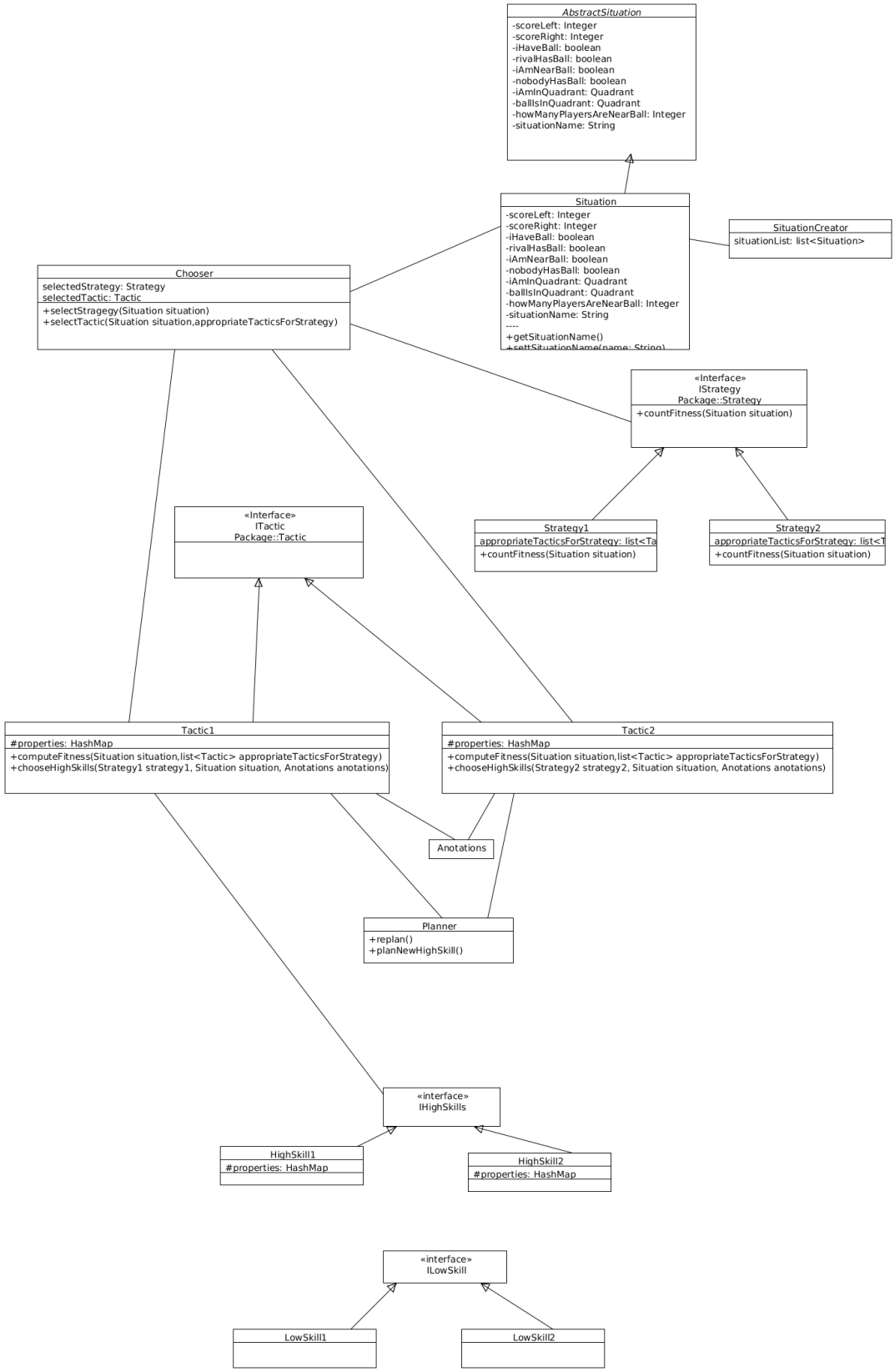
- Pokiaľ nemám oveľa lepšiu taktiku, tak ju nemením
- pokiaľ prejdem do iného kvadranta,

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

2.3.3.4 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.

Obrázok 23 - Návrh architektúry



2.3.4 Úprava dokumentácie

Dokumentácia bola upravená podľa požiadaviek pre kontrolný bod. Do dokumentácie boli pridané chýbajúce časti.

2.4 Šprint č. 4

Tabuľka 6 - šprint č.4

ID	AKO	CHCEM	ABY
4.1	Člen tímu	Navrhnúť a implementovať framework - situácie	Aby sa robot vedel rozhodovať na základe situácii.
4.2	Člen tímu	Navrhnúť a implementovať framework - stratégie	Aby robot vedel vyberať stratégiu.
4.3	Člen tímu	Navrhnúť a implementovať framework - taktiky	Aby robot vedel konať na základe taktiky
4.4	Člen tímu	Dokumentácia – šprint 3	Dopracovať dokumentáciu
4.5	Člen tímu	Selector a observer	Pozorvanie hry a rozhodovanie

2.4.1 Framework – situácie

Každá situácia sa nachádza vo vlastnej triede ktorá obsahuje základnú metódu *checkSituation* ktorá overuje či sú splnené podmienky nato aby daná situácia mohla nastať. SituationManager je trieda ktorá volá túto metódu *checkSituation* a konštruuje požadovaný zoznam všetkých aktuálnych situácií na ihrisku. Tento zoznam poskytuje ostatným balíkom architektúry. Pri pridávaní novej situácií je dôležité aby obsahovala metódu *checkSituation* a taktiež ju uviesť v triede *SituationList.*

2.4.2 Framework – stratégie

Vytvorili sme jednu stratégiu O*ffensiveStrategy* ktorá by mala slúžiť ako príklad. Obsahuje zoznam predpísaných situácií ktoré by mali platiť ak sa táto situácia má vykonávať. Metóda *getSuitability* vracia číslo koľko s týchto situácií sa práve nachádza v zozname aktuálnych situácií. Stratégia taktiež musí obsahovať zoznam povolených taktík ktoré sa môžu vykonávať a jednu základnú taktiku. Pri pridaní novej metódy je dôležité

aby bola taktiež zapísaná do triedy *StrategyList* ktorý používa trieda *selector* spomínaná nižšie.

2.4.3 Framework – taktiky

Rovnako ako pri stratégiách a situáciách musí každá nová taktiky byť umiestnená do vlastnej triedy s požadovaným názvom. Takáto nová taktika musí byť tiež zapísaná do triedy *TacticList*. Interface taktík definuje štyri metódy :

- *getInitCondition* - Podmienka ktorá musí byť splnená nato aby sa mala šancu vybrať daná taktika. Volá ju *selector*.
- *getSuitability* - V prípade ak viac taktík ma splnenú *InitCondition* dochádza k problému kedy treba vybrať jednu konkrétnu. Zložitejším algoritmom vypočítame číslo ku každej spornej taktike podľa ktorého vyberieme jednu konkrétnu ktorá sa má vykonávať. Volá ju *selector*.
- *getProgressCondition* - Aktuálne vykonávaná taktika musí mať splnenú túto podmienku inak dôjde k preplánovaniu a k zmene taktiky. Volaná triedou *SelectorObserver*.
- *run* - metóda ktorá vykonáva samotnú taktiku to znamená volá highskilly.

2.4.4 Dokumentácia

Dokončená dokumentácia a sprehľadnenie niektorých diagramov.

2.4.5 Selector a Observer

Celý balík obsahuje dve dôležité triedy. Jednou z nich je *SelectorObserver*. *Je to* trieda ktorá je volaná vždy keď hráč dostane zo servera akúkoľvek správu. Volá metódy *controlTactics* a *controlStrategy*. V metóde *controlTactics* sa kontroluje *ProgressCondition* aktuálnej taktiky. Pri nesplnení tejto podmienky dôjde k preplánovaniu a to zavolaním metódy selectTactic. Táto metóda sa nachádza už v druhej triede s názvom *Selector*. Selector obsahuje metódy ktoré vyberajú konkrétnu stratégiu a konkrétnu taktiku.

2.5 Šprint č. 5

Tabuľka 7 - šprint 5

ID	AKO	CHCEM	ABY
5.1	Člen tímu	Finalizovať dokumentáciu	Absolvovanie kontrolného bodu
5.2	Člen tímu	Vylepšiť selector	Vylepšiť rozhodovanie
5.3	Člen tímu	Otestovať architektúru	100% istota, že funguje

2.5.1 Finalizovať dokumentáciu

Finalizácia dokumentácie prebehla spájaním všetkých ostávajúcich častí do centrálnej dokumentácie

2.5.2 Vylepšiť selector

Selector bol aktualizovaný. Chybné časti boli odstránené.

2.5.3 Otestovanie architektúry

V spolupráci s druhým tímom boli vytvorené unit testy pre všetky dôležité časti. Architektúra bola otestovaná spustením hráča, servera a následným sledovaním vykonávania testovacích situácií, taktík a stratégií.

3 Celkový pohľad

V kapitole celkový pohľad sa nachádza podrobný popis robotického hráča, jeho architektúry, technológií , pomocou ktorých je hráč vytvorený ale aj popis vykonaných zmien a vylepšení.

3.1 RoboCup

3.1.1 Náplň projektu

Náplňou projektu je vytvoriť futbalového hráča pre 3D simuláciu, ktorý dokáže plnohodnotne využívať možnosti poskytované simulačným prostredím.

3.1.2 Úlohy projektu

Úlohou je získať a analyzovať zoznam prístupov, ktoré boli na našej fakulte použité v predchádzajúcich rokoch, aby bolo možné jednoducho využiť skúsenosti nazbierané predchádzajúcimi riešiteľmi, či už to boli tímové projekty, diplomové alebo bakalárske práce. Pozornosť je potrebné venovať podporným nástrojom, ktoré umožnia rýchlejší a kvalitnejší vývoj. Dôležitým faktorom bude využitie moderných prístupov robotiky a umelej inteligencie. Požiadavkami sú aj prehľadnosť a rozšíriteľnosť na úrovni návrhu aj implementácie.

3.1.3 Ciele projektu

Prevziať hráča vytvoreného na našej fakulte v minulom roku a doplniť ho o komplexnejšie typy správania. Vytvoriť efektívne zložitejšie pohyby hráča, ale pozornosť by sa mala venovať aj rozhodovaniu na vyššej úrovni, taktike a stratégii. Ďalším dôležitým cieľom nášho projektu bude vykonať komplexný refaktoring kódu a prepísanie častí kódu ktoré sú napísané v jazyku Ruby do jazyka Java.

3.2 Technológie

3.2.1 Java

Väčšina zdrojových kódov je napísaná v jazyku Java. Logické rozdelenie tried, metód a funkcií je popísane v kapitole 3.3 Architektúra.

3.2.2 Ruby

Pri prevzatí projektu bola časť plánovania riešená v jazyku Ruby. Táto časť sa podarila úspešne odstrániť a prepísať do jazyka Java. Všetky potrebné funkcie a triedy boli zachované a prepísané čo najpresnejšie. Nové triedy boli zaradené do príslušných balíkov.

3.2.3 Xml

Pohyby hráča sú riešené pomocou xml súborov, v ktorých je daná presná štruktúra pohybov, ktoré hráč podporuje. Xml súbory jednotlivých pohybov sú načítavané v časti implementovanej v jazyku Java.

3.3 Architektúra

3.3.1 Jim

- Jim
  - *AllTests* – spustí všetky testy
  - *Settings* – Inicializuje nastavenie hry, keď nie je určené inak načíta "default-né" nastavenia ()
  - *SettingsTest*– test pre class *Settings*
- Jim.agent
  - *AgentInfo* – uchováva informácie o stave hráča a jeho polohy na ihrisku (oprava komentárov funkcii niektoré nie sú pre javadoc, názvy funkcií ako "loguj" pritom má Jim logovací systém, getPlayerState – refactoring?)
  - *Planner (use RoboCupLibrary)* – nastavuje plánovanie a vykonáva daný plán (ruby)
  - *Side* – vymenovanie strán
- Jim.agent.communication
  - *Communication* – Komunikácia so serverom na najnižšej úrovni
  - *CommunicationThread*– stará trieda pre komunikáciu podľa komentárov odstrániteľná
- Jim.agent.communication.testframework
  - *Message*–komunikácia s testframework?
  - *TestFrameWorkCommunication*– komunikácia s tesframework?
- Jim.agent.models
  - *AgentModel* – Vyššie funkcie pre stav agenta a jeho pozíciu
  - *AgentPositionCalculator* – Približná poloha agenta na ihrisku podľa dostupných bodov
  - *AgentRotationCalculatori*– Približné natočenie hráča podľa dostupných bodov
  - *DynamicObject* – Výpočet polohy pohybujúceho sa objektu
  - *EnviromentModel* – uchováva stav sveta
  - *FixedObject*– uchovávanie a získavanie pozície statických objektov
  - *KalmanAdjuster* – nevieme presne čo, je tam pozícia lopty a pozícia fixných objektov
  - *Player* – Entita - Informácie o hráčovi (spoluhráč, protihráč)
  - *TacticalInfo* – Informácie o hernej stratégie
  - *WorldModel* – Informácie o ihrisku
- Jim.agent.models.prediction
  - *Prophecy* – pravdepodobný stav udalostí
  - *Prophet* – vypočítava najpravdepodobnejší vývoj udalostí
- Jim.agent.moves
  - *EffectorData* – entita – efektoru
  - *Joint* – entita – kĺbu

    - *JointPlacement* – uchováva konfiguráciu kĺbu
    - *LowSkill* – kolekcia fáz
    - *LowSkills*–Správa načítaných low skills
    - *Phase* – reprezentácia fázy
    - *Phases* – cache fáz
    - *SkipFlag* – reprezentácia fázy, ktorá sa má vynechať
    - *SkipFlags*
- Jim.agent.parsing
    - *ForceReceptor* – Informácie o sile pôsobiacej na dolné končatiny
    - *HearReceptor* – Informácie o správach (pokrikoch)
    - *ParsedData* – Implementácia serverovej správy
    - *ParsedDataObserver* – Rozhranie pre objekt spracovania správ
    - *Parser* – Trieda pre transformáciu správ zo servera pre agenta
    - *Perceptors* – Stav hardware-u robota (gyro, natočenie a.i.)
    - *PlayerData* – zapuzdrenie informácií z preceptorov
    - *SeenPerceptor* – transformuje správy z vizuál. preceptoru
    - *SeenPerceptorData* – Zapúzdruje informácie z vizuálneho preceptoru
    - *SeePercetor* – Aktualizuje dáta perceptoru
    - *SExpression*- ?
- Jim.agent.sexp
    - *SArray* – dátová štruktúra
    - *SException* – exception
    - *SObject* – default object
    - *SString*- ?
- Jim.agent.server
    - *TFTPServer*- ?
- Jim.agent.skills
    - *ComplexHighSkill* – Manažér high skill-ov
    - *FakeHighSkill* – high skill pre testovanie
    - *HighSkill* – Wraper pre high skill
    - *I**- interface-y
- Jim.agent.trajectory
    - *Obstacles* – Trieda pre výpočet obchádzky prekážky
    - *Trajectory* – reprezentuje trajektóriu postupnosti pohybov
    - *TrajectoryPlanner* – Plánovanie pohybov pre presun hráča z bodu A do B
    - *TrajectoryRealTime* – plánovanie pohybov podľa aktuálnych informácií
- Jim.annotation
    - Slúži na vytvorenie opisu pohybu
- Jim.annotation.gui
    - Grafické rozhranie pre anotácie
- Jim.gui
    - GUI pre replaning
- Jim.init
    - *ScriptBoot* – načítanie ruby skriptov
    - *SkillFromXmlLoad* – Načítanie low skills z XML súborov
    - *TestframeworkMain*- ?

- Jim.log
  - Logovanie pre hráča
- Jim.tests
  - Testovacie triedy

3.3.2 TestFramework

Tento framework slúži na získanie spätnej väzby od hráča. Jeho hlavným zámerom je zostrojiť robotického futbalového trénera. Zatiaľ je vypracovaný len vo fáze pozorovateľa.
Framework umožňuje modelovanie testcasov. Tento testovací framework obsahuje i grafické GUI. GUI je ľahko ovládateľné, dá sa v ňom ľahko zorientovať.
Taktiež vytvára vlákna hráčov, ktorých pridáva rovno do simulácie. Podľa identifikovania vzťahov – framework pridáva inštancie aktuálneho Jima.

3.3.2.1 Moduly (Balíky)

- INIT
  Tento modul (balíček) sa stará o samotné spustenie testovacieho frameworku, inicializujú sa tu základné hodnoty ako porty hráča, monitora, servera , ktoré sú následne prenášané do ďalších častí , napr. do komunikácie s hráčom a serverom.

  Taktiež sa tu inicializuje aj samotný user interface. Vykonáva sa kontrola, či je spustený RoboCup server a Monitor. Bez týchto podmienok sa testframework nespustí.

- LOGGER
  Slúži na logovanie všetkých vykonaných činností. Je spustený nad každým modulom (balíčkom), takže sa logujú všetky činnosti spojené so samotným testframeworkom.

- MONITOR
  Monitor slúži na monitorovanie stavu agenta a robocup servera. Slúži aj pre prijimanie TCP spojení s novými správami. Slúži taktiež na pridávanie a odoberanie vlákien agenta. Existuje aj trieda robocup, ktorá sa snaží simulovať stavy na serveri.

- UI

Grafický interface , v ktorom sa zobrazujú všetky dostupné informácie, dovoľuje pridávať a odoberať agentov.

- COMMUNICATION
  Rozhranie pre komunikáciu a sledovanie procesov. Rozdelené na agent a robocupserver. Každá z týchto častí sa stará o svoju komunikáciu. Obsahuje rozhranie pre komunikáciu agentov. Kontroluje bežiace procesy a agentov.

- PARSING
  Slúži na parsovanie správ. Typická dĺžka jednej správy nepresahuje jeden riadok a obsahuje číslo hráča, názov tímu, typ správy a posielané hodnoty . Príklad takejto správy je :
  (1 MEGATROLL LEFT) highskill start rollback 0.0
  Jediná výnimka pri posielaní správ sú správy o stave sveta. Stav sveta je na strane hráča deserializovaný do pola bajtov a následne zakódovaný do textového reťazca pomocou Base64.

- AGENTTRAINER
  Mal by sa starať o zapisovanie do XML, mal by obsahovať umelú inteligenciu, ktorá by učila agenta nové pohyby.

- ANNOTATOR
  Veľká trieda na parsovanie a vytváranie XML pohybov. Obsahuje triedu zodpovednú za dynamické vytváranie pohybov.

- WORLD REPRESENTATION
  Modul(balíček), ktorý reprezentuje okolitý svet, hráča. Taktiež zabezpečuje testovanie pohybov z xml.Stará sa o interpretáciu správ do scény, nastavovanie hráčov, reprezentáciu hráča,

- MONITOR AGENTA
  Je implementovaný pomocou triedy Agent Monitor. Lokálne sa používa iba jedna inštancia, ktorá sa stará o prijímanie nových spojení od samotných agentov. Každé spojenie je reprezentované vlastným threadom, ktorý ma na starosti spracovanie správ.

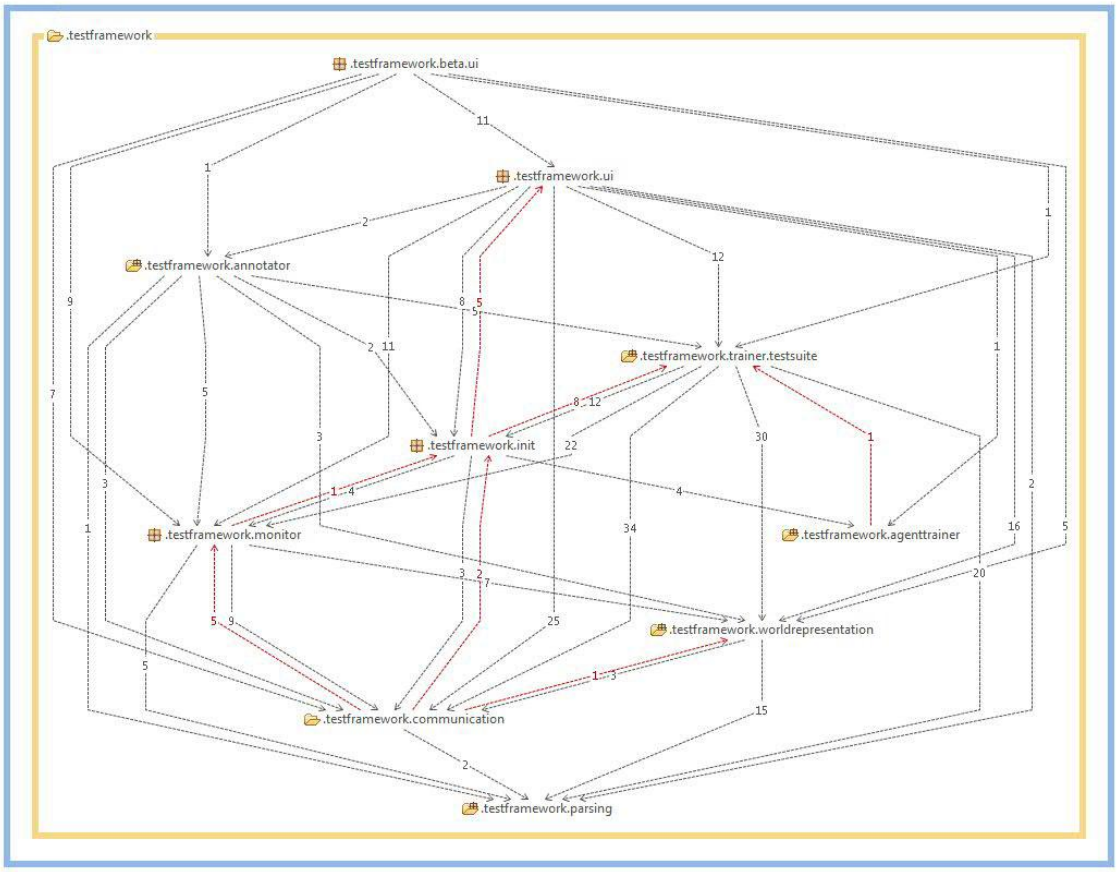
  Aktuálne existujú typ správ:
  INIT - posiela agent pri pripojení, obsahuje jeho číslo, názov tímu a stranu na ktorej hrá, takisto či má hráč zapnutý TFTP server a na ktorom porte

  DESTROY - posiela agent pri odpojení

HIGHSKILL - informuje test framework o začatí/skončení high skillu, obsahuje meno high skillu a čas kedy k akcii došlo

WORLDMODEL - posiela model sveta agenta do test frameworku

Obrázok 24 - TestFramework



3.3.3 RoboCupLibrary

sk.fiit.robocup.library.review
Balík obsahuje jeden súbor, ktorý definuje anotácie
ReviewOk.java - definuje anotáciu k triede alebo metóde, ktorá skontrolovaná, testovaná unittestom a testovaná proti serveru

sk.fiit.robocup.library.init
Balík obsahuje jeden súbor, ktorý zabezpečuje načítanie a spustenie skriptu. Prácu so skriptami zabezpečuje BSF Manager.

Script.java – Script(Stringname): konštruktor triedy
createScript(Stringname): určí, ktorý skript sa má načítať podľa mena
createScriptFrom(Stringfilepath): určí, ktorý skript sa má načítať podľa cesty k súboru

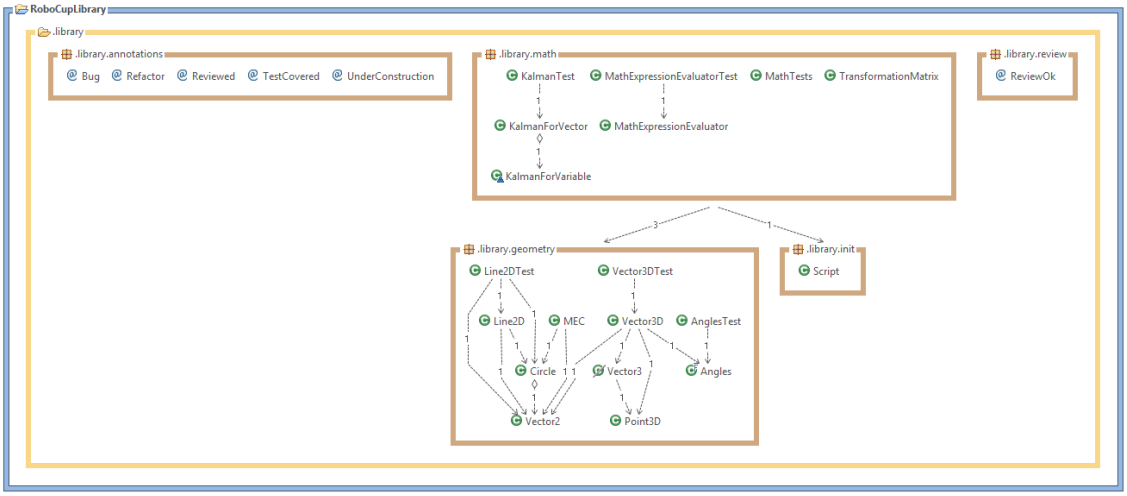
execute(): spustí načítaný script
registerBean(Stringname, Objectvalue): slúži iba na testovanie triedy Script
fetchBeans(String...names): nie je v programe vôbec využitá

sk.fiit.robocup.library.math
Tento balík obsahuje sedem súborov, ktoré implementujú Kalmanove filtre, prevody matematických výrazov a transformácie matíc.

KalmanForVariable.java – vypočíta Kalmanov filter pre jednu premenné
KalmanForVector.java – slúži na výpočet Kalmanovho filtra pre 3 premenné
KalmanTest.java – trieda, ktorá iba testuje Kalmanove filtre
MathExpressionEvaluator.java – zabezpečuje prevod matematického reťazca na prijatého ako String na číselný výsledok
MathExpressionEvaluatorTest.java – testuje triedu MathExpressionEvaluator
MathTest.java – Spúšťa testovacie súbory AnglesTest.java, KalmanTest.java a Vector3DTest.java
TransformationMatrix.java – obsahuje metódy na prácu s maticami, trieda je využívaná iba programom TestFramework
sk.fiit.robocup.library.geometry
Tento balík obsahuje najmä triedy, ktorériešia výpočty geometrickej matematiky. Zvyšné triedy slúžia iba na testovanie.

Angles.java – knižničná trieda, ktorá rieši operácie s uhlami
Circle.java – trieda reprezentujúca kruh v 2D priestore
Line2D.java – trieda reprezentujúca čiaru v 2D priestore
MEC.java – vypočíta najmenší ohraničujúci kruh pre zoznam bodov
využíva sa pri hľadaní pozície lopty
Point3D.java – trieda reprezentujúca bod v 3D priestore
obsahuje metódy, ktoré sa nevyužívajú alebo nie si implementované
Vector2.java – trieda reprezentujúca bod v 2D priestore pomocou vektoru
Vector3.java – trieda reprezentujúca bod v 3D priestore pomocou vektoru
obsahuje základne operácie ako sčítanie, odčítanie, delenie a vzdialenosť dvoch bodov
Vector3D.java–trieda reprezentujúca bod v 3D oproti triede Vector3 obsahuje prídavné operácie
AnglesTest.java – testuje triedu Angles
Line2DTest.java – testuje triedu Line2D
Vector3DTest.java – testuje triedu Vector3D

Obrázok 25 - RobocupLibrary



3.4 Zmeny

3.4.1 Odstránenie ruby

Do agenta Jim boli pridané dva balíčky a to sk.fiit.jim.highskills ktorý je určený na highskilly ktoré boli pôvodne implementované v ruby v priečinku scripts/high_skills. Ďalší balíček sk.fiit.jim.plan obsahuje plány prepísané z ruby z priečinku scripts/plan.

3.4.1.1 sk.fiit.jim.plan.Plan.java

Je implementovaná trieda ktorá má totožnú funkciu v Jimovy ako mal súbor plan.rb. Súčasťou tejto triedy je veľmi dôležitý rad highskillov s ktorým pracuje metóda control, ktorá buď zabezpečí vykonanie prvého z highskillov alebo v prípade ak máme rad prázdny vykoná metódu replan. Celá trieda je určená najmä nato aby rozširovala ostatné plány preto aj metóda replan neobsahuje žiaden zdrojový kód. Aby sme zamedzili duplicite kódu, táto trieda obsahuje aj metódy See_ball , ball_unseen, turned_to_goal, straight ... ktoré využívajú už spomínané konkrétne plány.

3.4.1.2 sk.fiit.jim.agent.Planner.java

Táto trieda bola upravená tak aby už nevykonávala ruby ale volala konkrétny plán v jave ktorý sa dá nastaviť v triede sk.fiit.jim.Setting.java napríklad pre nastaveniu plánu PlanTactic použijeme príkaz settings.put("Planner", "PlanTactic").

3.4.2 Zmena architektúry

3.4.2.1 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.

3.4.2.2 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 skôre 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

3.4.2.3 Taktiky

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

Kedy sa (ne)mení taktika?

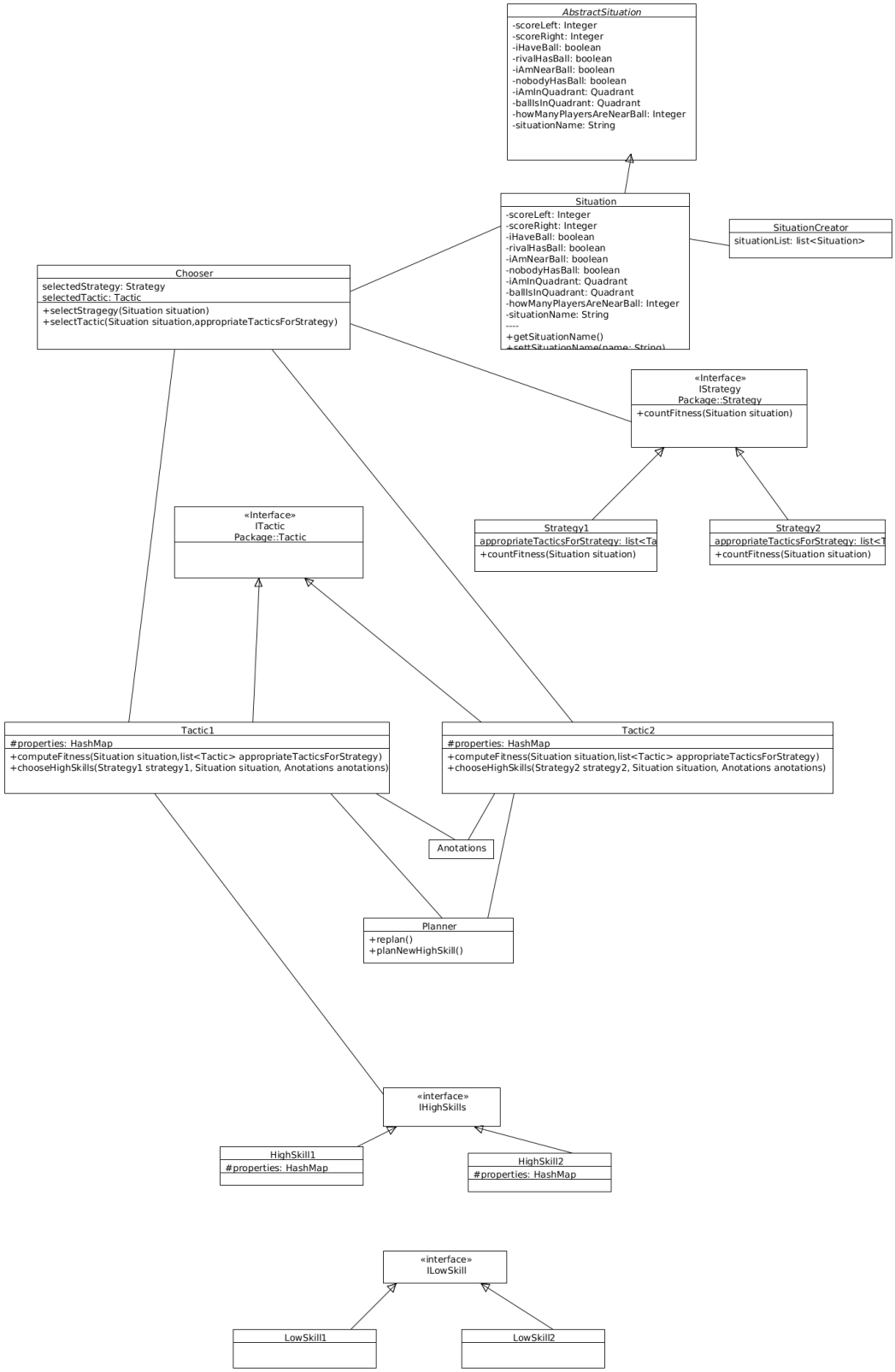
- Pokiaľ nemám oveľa lepšiu taktiku, tak ju nemením
- pokiaľ prejdem do iného kvadranta,

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

3.4.2.4 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.

Obrázok 26 - Návrh architektúry



3.4.3 Revízia architektúry

Do agenta JIM sme implementovali novú architektúru ktorá sa stará o to aby agent vedel používať viacero stratégií a taktík. Architektúra je rozdelená na nasledujúce balíčky:

3.4.3.1 Balík Situácií (sk.fiit.jim.decision.situation)

Každá situácia sa nachádza vo vlastnej triede ktorá obsahuje základnú metódu *checkSituation* ktorá overuje či sú splnené podmienky nato aby daná situácia mohla nastať. SituationManager je trieda ktorá volá túto metódu *checkSituation* a konštruuje požadovaný zoznam všetkých aktuálnych situácií na ihrisku. Tento zoznam poskytuje ostatným balíkom architektúry. Pri pridávaní novej situácií je dôležité aby obsahovala metódu *checkSituation* a taktiež ju uviesť v triede *SituationList.*

3.4.3.2 Balík Stratégií (sk.fiit.jim.decision.strategy)

Vytvorili sme jednu stratégiu O*ffensiveStrategy* ktorá by mala slúžiť ako príklad. Obsahuje zoznam predpísaných situácií ktoré by mali platiť ak sa táto situácia má vykonávať. Metóda *getSuitability* vracia číslo koľko s týchto situácií sa práve nachádza v zozname aktuálnych situácií. Stratégia taktiež musí obsahovať zoznam povolených taktík ktoré sa môžu vykonávať a jednu základnú taktiku. Pri pridaní novej metódy je dôležité aby bola taktiež zapísaná do triedy *StrategyList* ktorý používa trieda *selector* spomínaná nižšie.

3.4.3.3 Balík Taktík (sk.fiit.jim.decision.tactic)

Rovnako ako pri stratégiách a situáciách musí každá nová taktiky byť umiestnená do vlastnej triedy s požadovaným názvom. Takáto nová taktika musí byť tiež zapísaná do triedy *TacticList*. Interface taktík definuje štyri metódy :

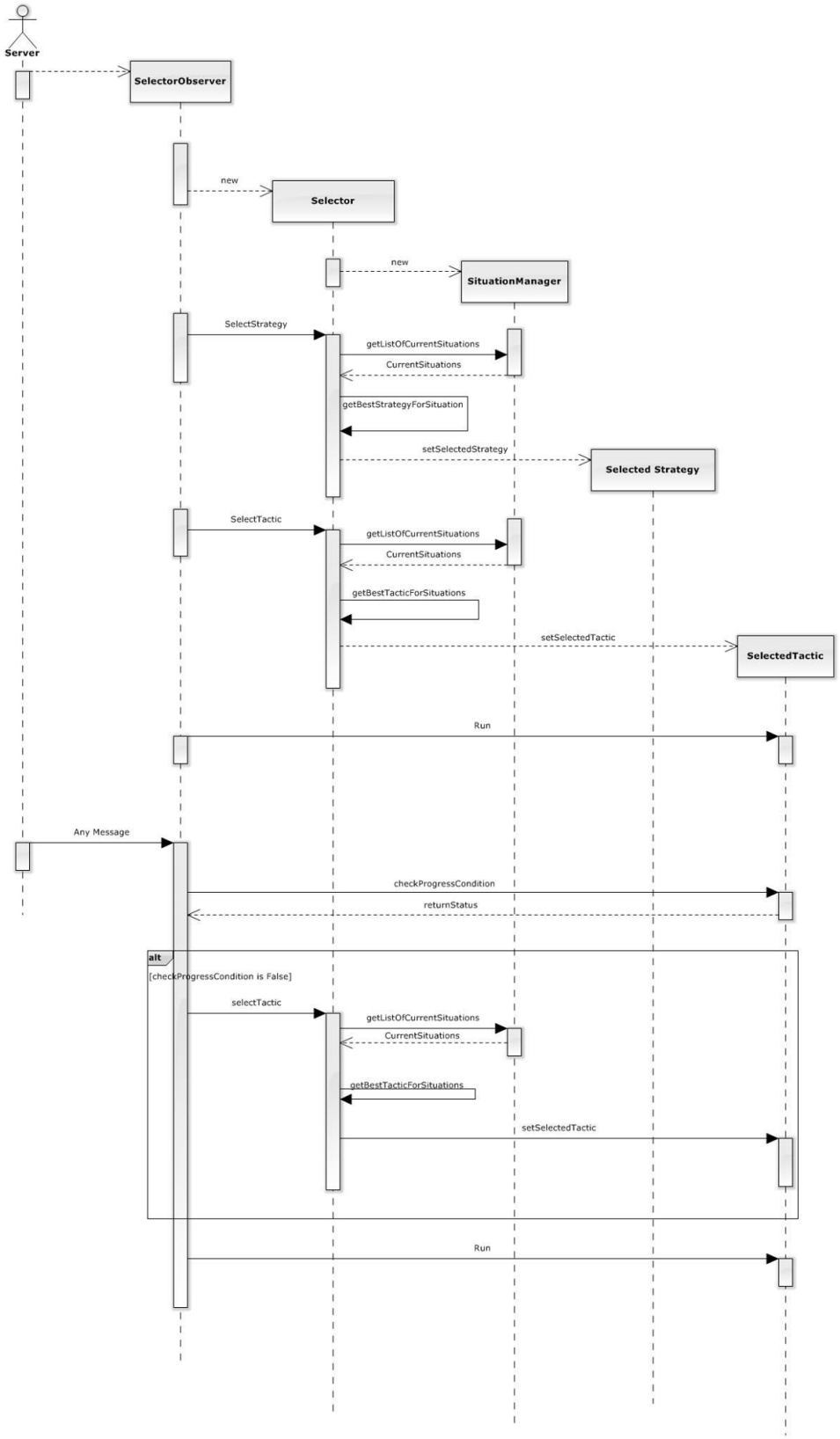
- *getInitCondition* - Podmienka ktorá musí byť splnená nato aby sa mala šancu vybrať daná taktika. Volá ju *selector.*
- *getSuitability* - V prípade ak viac taktík ma splnenú *InitCondition* dochádza k problému kedy treba vybrať jednu konkrétnu. Zložitejším algoritmom vypočítame číslo ku každej spornej taktike podľa ktorého vyberieme jednu konkrétnu ktorá sa má vykonávať. Volá ju *selector.*
- *getProgressCondition* - Aktuálne vykonávaná taktika musí mať splnenú túto podmienku inak dôjde k preplánovaniu a k zmene taktiky. Volaná triedou *SelectorObserver*.
- *run* - metóda ktorá vykonáva samotnú taktiku to znamená volá highskilly.

3.4.3.4 Základný balík rozhodovania (sk.fiit.jim.decision)

Celý balík obsahuje dve dôležité triedy. Jednou z nich je *SelectorObserver. Je to* trieda ktorá je volaná vždy keď hráč dostane zo servera akúkoľvek správu. Volá metódy *controlTactics* a *controlStrategy*. V metóde *controlTactics* sa kontroluje *ProgressCondition* aktuálnej taktiky. Pri nesplnení tejto podmienky dôjde k preplánovaniu a to zavolaním metódy selectTactic. Táto metóda sa nachádza už v druhej

triede s názvom *Selector*. Selector obsahuje metódy ktoré vyberajú konkrétnu stratégiu a konkrétnu taktiku.

Obrázok 27 - sekvenčný diagram novej architektúry



Obrázok 28 - Diagram tried a Balíkov