# Slovenská technická univerzita v Bratislave

**FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ**

Ilkovičova 3, 812 19 Bratislava


---

## Tvorba softvérového/informačného systému v tíme

# RoboCup – nové stratégie

(Dokumentácia k projektu)

---


**Tím č. 12 – 12. hráč**

Bc. Juraj Ligocký, Bc. Michal Hrubý, Bc. Gabriel Pán
Bc. Vladimír Oravec, Bc. Ján Hric, Bc. Marek Polák
Pedagogický vedúci: Ing. Ivan Kapustík


2008/2009

# OBSAH


1 Úvod ..... 4
1.1 Účel dokumentu ..... 4
1.2 RoboCup [7] ..... 4
1.3 Zadanie ..... 5
2 Analýza ..... 6
2.1 Prostredie simulovaného futbalu ..... 6
2.1.1 Senzory ..... 6
2.1.2 Režimy hry ..... 7
2.1.3 Akcie hráča ..... 8
2.1.4 Heterogénni hráči ..... 8
2.1.5 Rozhodca ..... 8
2.1.6 Kouč a tréner ..... 8
2.2 Analýza tímov ..... 9
2.2.1 Nexus ..... 9
2.2.2 DAInamite ..... 10
2.2.3 Oxsy ..... 13
2.2.4 Brainstormers ..... 14
2.2.5 Jahodoví princovia ..... 15
2.2.6 Loptoši ..... 16
2.2.7 Sklo ..... 18
2.2.8 FIITMedia ..... 19
2.2.9 UTTP ..... 22
2.2.10 Zhodnotenie analýzy tímov ..... 22
2.3 Možnosti logovania ..... 23
2.4 Štruktúra hráča Jahodových princov ..... 23
2.4.1 Organizácia kódu ..... 24
2.4.2 Najzákladnejšie triedy ..... 25
3 Špecifikácia ..... 28
3.1 Nedostatky hráča tímu Jahodoví princovia ..... 28
3.2 Agresívne správanie hráča ..... 28
3.2.1 Agresívne správanie ..... 28
3.2.2 Modelovanie problému ..... 29
3.2.3 Prevzatie správania ..... 31
3.3 Využitie programovacieho jazyka Java ..... 31
4 Hrubý návrh ..... 32
4.1 Agresívne správanie ..... 32
4.2 Úroveň dostupného kódu tímu DAInamite ..... 34
5 Prototyp ..... 36
5.1 Server verzie 13 ..... 36
5.1.1 Server verzie 13 a tím Jahodoví princovia ..... 36
5.1.2 Server verzie 13 a fakultné tímy ..... 36
5.1.3 Zhodnotenie ..... 36
5.2 Hráč tímu Jahodoví princovia ..... 37
5.2.1 Časté strieľanie z diaľky ..... 37
5.2.2 Úprava rozohrávania brankára ..... 37
5.2.3 Zhodnotenie ..... 38
5.3 Hráč tímu DAInamite ..... 38

5.3.1 Testovanie nárokov hráča tímu DAInamite ..... 38
5.3.2 Funkčnosť hráča DAInamite ..... 39
5.4 Testovanie prototypu ..... 41
5.5 Zhodnotenie prototypu ..... 42
6 Riešenie ..... 43
6.1 Server verzie 13 ..... 43
6.1.1 Spolupráca so serverom ..... 43
6.1.2 Hra na serveri ..... 43
6.1.3 Stabilita servera ..... 44
6.2 Rozhodovanie hráča, úžitok jednotlivých stavov ..... 44
6.3 Formácie a strategická pozícia ..... 46
6.3.1 Návrh a implementácia formácií ..... 46
6.3.2 Algoritmus na získanie strategickej pozície ..... 49
6.3.3 Nový stav pre formácie ..... 49
6.4 Zameriavanie sa na zvukové správy ..... 49
6.5 Zvuková komunikácia ..... 53
6.6 Obranná hra ..... 54
6.7 Rozohrávanie štandardných situácií ..... 54
6.7.1 Fáza čakania ..... 55
6.7.2 Fáza rozohrávania ..... 55
6.8 Ďalšie menšie úlohy ..... 57
7 Testovanie ..... 58
7.1 Spôsob testovania ..... 58
8 Zhodnotenie ..... 60
Použitá literatúra ..... 61
Používateľská príručka ..... 2

**Prílohy**

Príloha A – Používateľská príručka k produktu
Príloha B – Spustenie hráča zo zdrojových kódov
Príloha C – Dátové médium s prototypom a dokumentáciou

# 1 ÚVOD

## 1.1 Účel dokumentu

Tento dokument vznikol v rámci predmetov Tvorba softvérového systému v tíme a Tvorba informačného systému v tíme a zaoberá sa problematikou simulovaného robotického futbalu – RoboCupu [1]. Cieľom projektu je priniesť do sveta RoboCupu na našej fakulte kvalitného hráča, ktorý je výsledkom vylepšenia už existujúceho hráča pomocou rôznych algoritmov spolupráce agentov, rozhodovacích stromov, neurónových sietí a pod. Výsledkom nášho úsilia bude regionálne kolo turnaja RoboCupu, ktoré sa uskutoční na pôde FIIT STU v Bratislave.

Prvá kapitola dokumentu sa stručne zaoberá simulovanému robotickému futbalu.

Druhá kapitola je venovaná podrobnejšej analýze. Opisuje architektúru RoboCupu, predovšetkým simulujúci server. Okrem toho prezentuje poznatky získané jednotlivými členmi hlbšou analýzou fakultných a zahraničných tímov, zameriavajúc sa na ich jedinečné vlastnosti, výhody a nevýhody. Ďalej sú v kapitole zahrnuté výsledky z testovania a skúmania dvoch tímov.

Tretia kapitola obsahuje špecifikáciu riešení, ktorým sa budeme venovať najmä v rámci prototypu.

## 1.2 RoboCup [7]

RoboCup je medzinárodný projekt, ktorý má za úlohu podporovať výskum umelej inteligencie, robotiky a podobných oblastí. RoboCup sa to snaží pomocou futbalu, keď futbal sa stáva hlavnou témou výskumu.

Hlavným cieľom RoboCupu je do roku 2050 vytvoriť tím plne autonómnych humanoidných robotov, ktorí dokážu poraziť najlepších futbalistov (ľudí). Tento cieľ sa môže zdať príliš ambiciózny, ale je to výzva pre umelú inteligenciu na najbližších 42 rokov. Napríklad prešlo len 50 rokov od vynájdenia počítača k vytvoreniu superpočítača Deep Blue, ktorý dokázal poraziť aktuálneho svetového šampióna v šachu.

Na dosiahnutie tohto cieľa je potrebné vytvoriť nové technológie vo viacerých oblastiach ako napr. nové princípy autonómnych agentov, spolupráca v multiagentových systémoch, reagovanie agentov v reálnom čase, výskum senzorov a pod. Jednou z hlavných oblastí, kde sa dajú využiť technológie RoboCupu, je napr. hľadanie a záchrana zranených pri veľkých katastrofách. Touto oblasťou sa zaoberá projekt RoboCupRescue.

RoboCup má tri hlavné kategórie: RoboCupSoccer (simulácia futbalového zápasu), RoboCupRescue (hľadanie a záchrana ľudí pri katastrofách) a RoboCupJunior (určený na vzdelávanie študentov do 19 rokov). Tieto kategórie sa ešte delia na ďalšie. Náš tím sa bude zaoberať kategóriou RoboCupSoccer Simulation.

RoboCupSoccer Simulation slúži na výskum a vzdelávanie sa v oblasti umelej inteligencie a multiagentových systémov. Umožňuje dvom tímom jedenástich hráčov simulovať autonómnych robotov pri hraní futbalu.

Vlastná simulácia futbalového zápasu prebieha na serveri, keď na server sú pripojení jednotliví hráči (agenti). Agenti získavajú zo servera rôzne informácie a na základe nich reagujú. Prebiehajúci zápas sa dá prezerať pomocou tzv. monitora, ktorý na obrazovku zobrazuje prebiehajúcu (alebo uloženú) hru.

## 1.3 Zadanie

Téme RoboCup, presnejšie lige simulovaného robotického futbalu, sa naši študenti venujú už deviaty rok. Tímy študentov, či už v rámci umelej inteligencie alebo tímového projektu, sa snažia vytvárať a vylepšovať programy, ktoré simulujú správanie sa futbalového hráča. Každý tím sa v rámci obmedzení určených pravidlami futbalu a špecifikami simulačného prostredia snaží vytvoriť čo najlepšieho hráča. Mužstvo vytvorené z takýchto hráčov by malo vyhrať nad mužstvom súpera. O súťaži a doterajšej činnosti je dosť popísané aj na stránke STU turnaja v simulovanom robotickom futbale[1].

V rámci fakulty sa realizovalo viacero súťaží a posledné ročníky už boli oficiálnymi turnajmi iniciatívy RoboCup. Množstvo pozitívnych ohlasov priviedlo organizátorov k vyhláseniu ďalšieho regionálneho turnaja RoboCup v simulovanej lige, opäť na záver akademického roka. Práve množstvo nových prístupov a riešení, ktoré predviedli nielen študenti tímového projektu, ale aj študenti umelej inteligencie, ukázali, že možnosti na vylepšovanie hráča nie sú zďaleka vyčerpané a dokonca sa stále rodia prekvapujúce úspešné riešenia. Preto má téma RoboCupu podnázov „Nové stratégie“. Znamená to všeobecne hľadanie nových prístupov a stratégií nielen pre hráča, ale aj vo svojej práci, v úpravách zdrojového kódu, podporných aplikáciách, základných aj vyšších schopnostiach hráča, spôsobe učenia a ladenia počas simulácií. Nové stratégie sú komplexnou výzvou do nového kola víťazstiev!

Na spresnenie je vhodné povedať, že v tomto tímovom projekte budeme rozširovať možnosti a vylepšovať správanie hráčov, vytvorených vo vlaňajších tímových projektoch. Využije sa existujúci zdrojový kód, dokumentácia a aj vytvorené podporné aplikácie. Musí sa zachovať (a podľa možností aj zlepšiť) aj modularita a tým aj rozšíriteľnosť hráča. Zimný semester je vyhradený na oboznámenie sa s celým prostredím, najmä existujúcimi hráčmi a návrhu a prototypovej realizácii jeho vylepšení. Očakáva sa najmä návrh nových prístupov a stratégií vo všetkých už spomenutých oblastiach. Vybrané prístupy sa overia vytvorením jedného alebo viacerých prototypových rozšírení existujúceho kódu. Dôležitou súčasťou bude vytvorenie plánu implementácie a overovania nových stratégií v nasledovnom semestri. V letnom semestri nás čaká realizácia navrhnutých prístupov a stratégií a ich overovanie. Produkt by mal byť dohotovený v deviatom až desiatom týždni semestra, potom je potrebné venovať sa ladeniu a optimalizácii hráča na súťaž, ktorej výsledky idú do celkového hodnotenia tohto projektu.

[1] www.fiit.stuba.sk/robocup

# 2 ANALÝZA

Kapitola Analýza uvádza poznatky nadobudnuté naším tímom o prostredí simulovaného futbalu. V prvom rade ide o mechanizmus, na akom je celý systém postavený. Ďalej sa zaoberá situáciou vývoja v oblasti RoboCupu vo svete, no najmä na našej fakulte. Úlohou analýzy je získanie predstavy o možnostiach prínosu nášho tímu do sféry simulovaného robotického futbalu.

## 2.1 Prostredie simulovaného futbalu

Soccer server je systém predstavujúci simulačné prostredie pre virtuálne ihrisko. Celý mechanizmus simulácie je typu klient-server, hráči sa pripoja na server a komunikujú s ním pomocou protokolu UDP. Server poskytuje hráčom informácie o stave hry a senzorické informácie o objektoch na ihrisku. Hráči posielajú serveru svoje požiadavky na akcie, ktoré chcú vykonať. Na externé sledovanie simulácie možno použiť program Soccer monitor. Existuje viac druhov s rozličnými vlastnosťami, základom je zobrazenie plochy ihriska, hráčov a lopty a možnosť spustenia zápasu.

### 2.1.1 Senzory

Senzory umožňujú hráčovi vnímať okolie. Sú ekvivalentom zmyslov u ľudí. Hráči majú tri druhy senzorov, na základe nimi prijatých informácií si vytvárajú predstavu o svete. Opis senzorov je prebratý z [7].

#### *Zrak*

Zrakový senzor dáva hráčovi informácie o objektoch, ktoré sa nachodia v jeho zornom poli. Informácie sú posielané pomocou funkcie `(see (Time) (ObjInfo) (ObjInfo) ... )` periodicky v stanovenom kroku. `ObjectInfo` je objekt, ktorý obsahuje napríklad informácie o relatívnej vzdialenosti (`Distance`), smere (`Direction`), natočení (`FaceDir`), čísle hráča (`UNum`) a mene tímu (`TeamName`) objektu. Informácie, ktoré hráč vidí, sú zašumené tým viac, čím je objekt ďalej od hráča. S väčšou vzdialenosťou je aj množstvo informácií menšie, teda postupne hráč prestane rozoznávať číslo dresu a príslušnosť k tímu. Kvalitu pohľadu hráča možno zmeniť pomocou funkcie `(change_view (ANGLE_WIDTH) (QUALITY))`, kde `ANGLE_WIDTH` môžu byť hodnoty `normal`, `wide`, `narrow` a `QUALITY high` a `low`. Po hracom poli je viacero značiek, ktoré hráč môže vnímať a podľa ktorých sa môže orientovať (bránky, rohy, stred ihriska).

#### *Sluch*

Sluchový senzor slúži na počúvanie správ od iných hráčov, kouča alebo rozhodcu. Samotné počúvanie je realizované funkciou `(hear (Time) (Direction) (Message))`, kde `Time` je aktuálny čas, `Direction` je smer zdroja. Ak chce hráč poslať správu ostatným hráčom, je mu to umožnené prostredníctvom funkcie `(send (Message))`. Táto správa nesmie byť dlhšia ako desať bajtov. Ak hráč bude môcť počuť

naraz viacero správ, tak začuje náhodnú. Správu od rozhodcu začuje vždy. Správy povedané hráčmi majú obmedzený dosah. Hráč si však môže nastaviť, ktorého hráča chce prednostne počúvať. V prípade, že bude počuť viacero správ naraz, rozozná správu práve od tohto hráča. Ak ten hráč nič nezakričal, tak opäť začuje náhodnú správu.

#### *Pocit tela*

Telový senzor umožňuje hráčovi zistiť svoj fyzický stav pomocou funkcie `sense_body`. Dal by sa prirovnať k rôznym pocitom u ľudí. Server poskytuje hráčovi informácie o jeho rýchlosti, výdrži, šírke pohľadu, počte poslaných správ a pod. Hráč by mal vedieť tieto informácie správne využiť a zaradiť ich do vyhodnocovacieho procesu svojej akcie (napr. nebude bežať plnou rýchlosťou, keď má málo energie).

### 2.1.2 Režimy hry

Podľa režimu hry majú hráči rôzne možnosti akcií. Server pozná tieto režimy hry:

#### *Play on*

Toto je základný režim hry. Nastáva, keď sa lopta začne pohybovať vďaka kopnutiu z iného hracieho režimu (rozohranie lopty, aut, výkop od brány, roh).

#### *Kick off*

Základné rozohranie lopty, pri ktorom musí byť každý hráč na svojej strane. Ak sa na nej nenachádza, server ho umiestni na náhodné miesto na príslušnej polovici hracieho poľa.

#### *Gól*

Nastáva v momente, keď sa lopta dostane svojím celým objemom do brány za bránkovú čiaru. Server preruší zápas na päť sekúnd, presunie loptu do stredu ihriska, pošle správu všetkým hráčom. Počas prerušenia sa hráči môžu presunúť na svoju polovicu pomocou príkazu `move`. V opačnom prípade ich presunie na náhodnú pozíciu server. Server následne zmení hrací režim na `kick_off`.

#### *Aut, roh a výkop od brány*

Nastáva, keď lopta svojím objemom opustí hraciu plochu. Server presunie loptu na príslušnú pozíciu (v prípade autu tam, kde opustila ihrisko; v prípade prekročenia bránkovej čiary buď do rohu, buď na hranicu výkopu od brány). Následne nastaví príslušný herný režim.

#### *Presunutie*

Nastáva pri rozohrávaní lopty (`kick_off`, `throw_in`, `corner_kick`, `goal_kick`, `offside`). Server presunie hráčov tak, aby boli minimálne vo vzdialenosti 9,15 metra od lopty. Hráči sú presunutí na obvod pomyselnej kružnice.

Presun hráča pomocou príkazu `move` je na pozíciu `(X, Y)`. Stred súradnicovej sústavy je v centrálnom kruhu, kladný smer osi `X` je určený smerom k súperovej bráne, kladný smer osi `Y` je určený smerom k pravej čiare (ktorá je určená vzhľadom na kladný smer osi `X`). Tento príkaz je k dispozícii iba v režime `before_kick_off`.

#### *Polčas*

Server preruší zápas v polovici plánovaného času trvania. Štandardná dĺžka polčasu je 3 000 simulačných taktov (okolo päť minút). Ak je po skončení zápasu skóre nerozhodné, predlžuje sa, kým sa nedosiahne gól – tzv. zlatý gól.

### 2.1.3 Akcie hráča

Pod akciami chápeme príkazy, ktoré môže hráč serveru poslať na vykonanie určitej činnosti v nasledovnom simulačnom takte.

- Dash – zrýchlenie hráča v smere jeho tela. Zohľadňuje sa výdrž hráča.
- Kick – kopnutie do lopty určitou silou v určitom smere.
- Catch – chytenie lopty v určitom smere, používa len brankár. Server má definovaný interval, ako často možno príkaz chytenia opakovať.
- Say – poslanie krátkej správy spoluhráčom i protihráčom.
- Pozornosť na hráča – prednostné počúvanie určitého hráča. Ak daný hráč dačo povie súčasne s inými, práve jeho správu hráč zachytí.
- Otočenie hráča – zmena smeru hráčovho tela o určitý moment. V prípade stojaceho hráča je moment rovný uhlu otočenia. Zvyšovaním rýchlosti behu sa uhol zmenšuje.
- Otočenie hlavy – zmena smeru otočenia hlavy hráča relatívne k telu.
- Tackle – pokus o obratie súpera o loptu. Je to tzv. kĺzačka, hráč sa snaží z celej sily kopnúť do lopty. Desať taktov po vykonaní príkazu je hráč nečinný.

### 2.1.4 Heterogénni hráči

Server umožňuje hru s heterogénnymi hráčmi, čiže s hráčmi s rôznymi vlastnosťami. Na začiatku hry sa okrem základného hráča vygeneruje ďalších šesť náhodných typov. Podstatou heterogénnosti hráčov je zmena daktorých vlastností proti základnému hráčovi. Celkovo však platí, že čím lepšia jedna vlastnosť, tým horšia iná. Takto sa otvárajú nové možnosti na obsadenie rol v tíme. Napr. hráč s veľkou výdržou a nízkou presnosťou kopu je vhodným obrancom, no ťažšie sa uplatní v útoku.

### 2.1.5 Rozhodca

Rozhodcu predstavuje „obsluha“ simulácie, čiže je to externý zásah do systému. Rozhodca má právo zmeniť režim hry v prípade, ak nastala neštandardná situácia. Príkladom je faul. Faul síce nie je v simulácii definovaný, no sú určité dohody „správnej“ hry (napr. nesmie sa vytvoriť obranný múr z hráčov pred bránou), ktorých porušenie môže priniesť tímu penalizáciu. Rozhodca vie hráčom poslať zvukové správy. Hráči tieto správy zachytia v každom prípade.

### 2.1.6 Kouč a tréner

Kouč aj tréner slúžia na organizáciu hráčov, majú však rôzne úlohy. Kouč aj tréner dostávajú – na rozdiel od hráčov – neskreslené informácie o svete. Kouč je prítomný aj počas riadnych zápasov. Vie hráčom poslať informácie v podobe zvukových správ, ich frekvencia je však omnoho nižšia než komunikácia medzi hráčmi

samými. Jeho hlavnou úlohou je analyzovať hru a na základe analýzy vyhodnotiť najlepšiu stratégiu na najbližšiu etapu zápasu. Okrem dávania pokynov má možnosť vystriedať hráčov. Tréner, na druhej strane, slúži na trénovanie hráčov mimo riadnych zápasov. Veľmi vhodný je na učenie situácií.

## 2.2 Analýza tímov

Najprv predstavíme niekoľko súčasných zahraničných tímov. Keďže k zahraničným tímom často nie je dostupná kvalitná dokumentácia, ani plný zdrojový kód, ich analýza nie je hĺbková. Má poskytnúť najmä inšpiráciu pre našu prácu na vylepšení domáceho tímu. Fakultné tímy produkujú každoročne viac-menej podrobné dokumenty a zdrojové kódy sú plne dostupné, preto ich rozoberieme dôkladnejšie.

### 2.2.1 Nexus

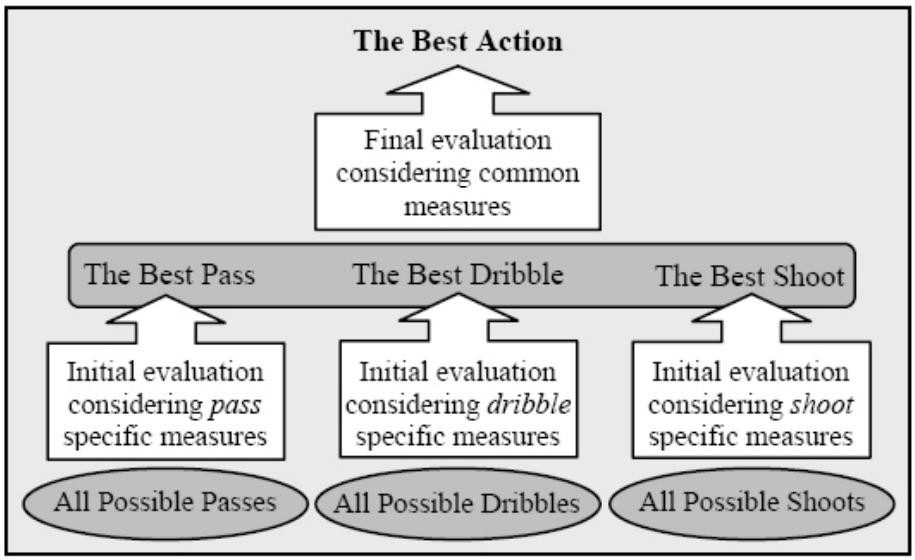
Tím Nexus [2] je arabského pôvodu. Zúčastnil sa minulého svetového ročníka v robotickom futbale a vypadol v druhom kole. Pri tomto tíme je zaujímavé, ako sa vyberá najlepšia nasledovná akcia pre určitého hráča. Pri algoritme sa používa špeciálna dvojfázová metóda, ktorú autori prebrali od tímu TsinghuAeolus – víťaza svetového turnaja v RoboCupe v rokoch 2001 a 2002. V robotickom futbale môže hráč urobiť s loptou tri kroky: nahrať, strieľať alebo driblovať. Informácie si delia na tie, ktoré sú špecifické alebo potrebné pre danú činnosť, a tie, ktoré sú spoločné pre všetky druhy akcií.

| Parameter | S | D | P |
|---|---|---|---|
| Ball distance from the center of the opponent's goal | y | n | n |
| Relative direction of the goalie and the ball motion | y | n | n |
| Dribbling length | n | y | n |
| Relative angle of player's body and the dribbling direction | n | y | n |
| Distance between the passer and the ball | n | n | y |
| Movement direction of the pass receiver | n | n | y |
| Sensitivity of the target area | y | y | y |
| The density of the opponents in the target area | y | y | y |
| Probability of the ball interception by the opponents | y | y | y |
| Updating degree of the received information as to target area | y | y | y |

**Obr. 1.** Rôzne parametre hráča tímu Nexus

```
00:  max_priority = 0
01:  selected_action = no_action
02:  for each feasible action (FA) do
03:    priority = 0
04:    for each evaluation measure (EM) do
05:      priority +=  EM.weight
06:    end for
07:    if priority > max_priority then
08:      max_priority = priority
09:      selected_action = FA
10:    end if
11:  end for
```

**Obr. 2.** Algoritmus pre jednofázovú metódu tímu Nexus



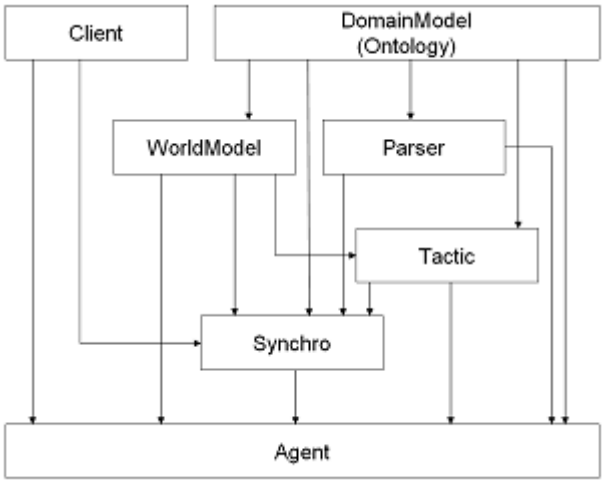

**Obr. 3.** Algoritmus pre dvojfázovú metódu tímu Nexus

### 2.2.2 DAInamite

Tím DAInamite [3] vyvíja svojho vlastného hráča od nuly. Ako programové prostredie si zvolili Javu. Výsledný framework je komplexný produkt, ktorý zahŕňa všetky potrebné časti pre správny chod. Zaujímavými sú vylepšenia tohto frameworku v aktuálnej verzii.

### *Architektúra hráča*

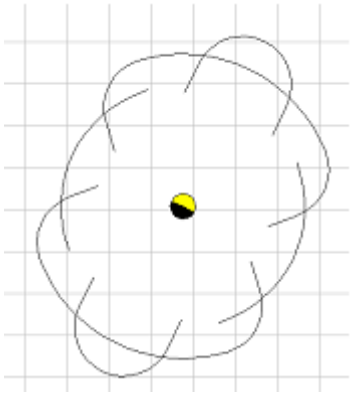
Na zachovanie čo najväčšej flexibility bol použitý Spring-Framework na konfiguráciu štruktúry hráča za behu. To im dovoľuje uloženie závislostí do súboru formátu XML namiesto ich uloženia priamo v kóde, takže výmena komponentov je iba jednoduchá záležitosť konfigurácie. Každý komponent má svoju vlastnú konfiguráciu, ktorá môže byť tiež rozšírená. Prehľad existujúcich modulov a ich závislosti možno vidieť na obrázku.



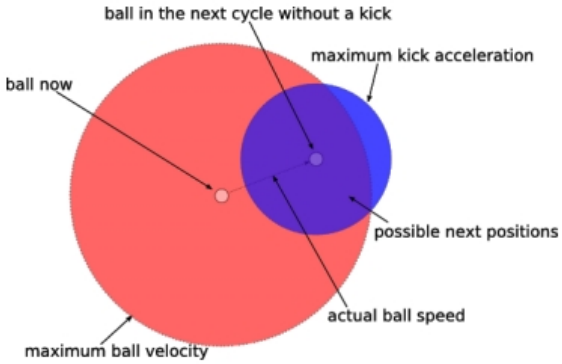

**Obr. 4.** Závislosti medzi jednotlivými modulmi architektúry hráča tímu DAInamite

### *Model akcie*

Predchádzajúca metóda výberu akcie s loptou bola rozdelená na rôzne akcie ako driblovanie, podržanie, prihrávka, strela. Tieto akcie autori rozanalyzovali a zistili nedostatky. Po prvé bolo ťažko porovnať a následne rozhodnúť, ktorá akcia je najlepšia v danom okamihu. Po druhé nevieme povedať, či sme nezabudli na nejakú vhodnejšiu akciu. Na vyriešenie týchto problémov autori implementovali triedy `ReachableArea` a `BallArea`. `ReachableArea` je geometrický model plochy, ktorú môže hráč dosiahnuť za daný čas. `BallArea` reprezentuje, kde sa môže lopta nachádzať po náhodnom kope hráča.



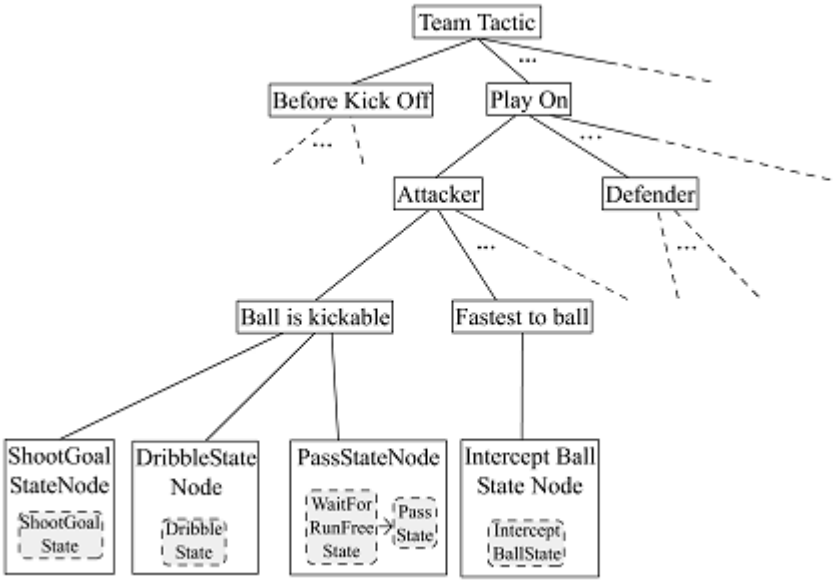
**Obr. 5.** Plocha, ktorú dokáže hráč dosiahnuť za čas – tím DAInamite




**Obr. 6.** Lopta s rýchlosťou a pozíciou – tím DAInamite

### *Taktika*

Nový mechanizmus výberu akcie je zodpovedný za kalkuláciu najlepšej akcie na základe dostupného pohľadu. Tento mechanizmus je realizovaný pomocou rozhodovacieho stromu, ktorý obsahuje taktiku každej roly ako podstrom. Takže môžeme modelovať individuálne správanie sa každého spoluhráča a dynamické prideľovanie rol. Vrcholy stromu predstavujú podmienené varianty a listy predstavujú, čo sa stane. Listy obsahujú tzv. stavy (angl. states), ktoré sú zodpovedné za výpočet špecifickej akcie. Prechádzaním cez strom možno určiť najlepšiu akciu, ktorá sa má vykonať.




**Obr. 7.** Rozhodovací strom tímu DAInamite

### *Zhodnotenie*

Pri tomto tíme nás zaujala ani nie tak jeho štruktúra či špeciálne vlastnosti ako jeho implementačné prostredie. Väčšina členov tímu má slabé skúsenosti s programovacím jazykom C++. Predpokladáme, že budúce roky sa situácia bude

opakovať. Java predstavuje v súčasnosti jeden z najrozšírenejších programovacích jazykov, preto je prirodzené, že aj v oblasti RoboCupu sa začínajú objavovať hráči v ňom napísaní. Na našej fakulte sa zatiaľ používa výhradne C++. Pokúsime sa zistiť, do akej miery sme schopní a či je účelné tento stav zmeniť.

### 2.2.3 Oxsy

Tím Oxsy [4] pochádza z Rumunska. Vznikol ako diplomový projekt študenta Mariana Sebastiana a autor sa mu stále venuje. Na tohtoročnom turnaji skončil piaty.

#### *Konaj prv, než udalosť nastane*

Autor považuje túto metódu za základ fungovania tímu. Cieľom je, aby celý tím konal ako jedna myseľ, aby jednotliví hráči predpokladali akcie spoluhráčov i protihráčov bez nutnosti komunikácie.

#### *Daj a bež*

Jednoduchým príkladom predpovedania je akcia daj a bež, keď sa dvaja hráči snažia prejsť cez súperovho obrancu. Hráč nahrá spoluhráčovi a začne bežať na voľné miesto alebo jedným smerom, kde čaká spätnú prihrávku. Čiže musí predpokladať budúcu prihrávku od daného hráča a začne bežať skôr, než prihrávka nastane, dokonca ešte pred zachytením lopty spoluhráčom.

#### *Výmena rol*

Túto akciu využíva tím v obrane. Ak sa súperovmu hráčovi podarí prejsť cez obrancu po kraji ihriska ako dôsledok neúspešnej kĺzačky (tackle) krajného obrancu, ktorý potom nedokáže istý čas hrať, stredný obranca nečaká, kým súper príde do jeho oblasti. Hneď sa vyberie proti súperovi, aby ho zastavil, čím nahradí krajného obrancu. Pôvodný krajný obranca, keď sa už zregeneruje, beží na miesto stredného obrancu. Krajný a stredný obranca si tak vymenili roly. Je to ďalší príklad predpovedania udalostí. Podstatou je prekvapivý moment pre súpera, ktorý prihrávku nečaká.

Predpovedanie umožňuje tímu reagovať na nečakané udalosti, a na druhej strane, tím sa stáva menej predvídateľným, čo z neho robí silnejšieho súpera.

#### *Postrehy zo zápasov*

- Brankár vybieha aj do stredu ihriska, čím pomáha rýchlo založiť útok. Robí to len vtedy, keď si je istý, že k lopte dobehne prv, než súper.
- Tím je rozdelený na tri skupiny: štyria obrancovia, traja stredopoliari a traja útočníci. Hráči v skupine sa zdržujú takmer výlučne vo vertikálnom zástupe. Je to jednoduchá a pomerne účinná „formácia“.
- Obrancovia sa snažia vždy stáť pri svojom brankárovi bližšie než ktorýkoľvek súper. Sú však situácie, keď sa dostanú na úroveň súpera, pravdepodobne v snahe zachovať zástup, čo môže spôsobiť ťažkosti.
- Obrancovia idú len málo za stredovú čiaru, aj počas útoku sa zdržujú tesne pri čiare. Útočníci – podobne – vždy čakajú v súperovej polke, zriedka prejdú za čiaru.

### *Zhodnotenie*

O kvalite tímu Oxsy svedčí jeho umiestenie na popredných miestach v posledných rokoch. Dostupná dokumentácia k tímu je veľmi slabá. Opísaná metóda predpovedania udalostí by sa dala prirovnať k rozpoznávaniu akcií v návrhu tímu FIITMedia, hráči však medzi sebou nemusia komunikovať. Pozorovanie zápasov tímu Oxsy odhaľuje jednoduchú a účinnú hru: Hráči sú stále zoradení v akejsi formácii, ktorej zakomponovanie do nášho budúceho hráča by stálo za vyskúšanie.

## 2.2.4 Brainstormers

Brainstormers [5] je nemecký tím s desaťročnou tradíciou pôsobiaci na Univerzite Osnabruck v Nemecku. Zároveň je minuloročným víťazov majstrovstiev sveta, ktoré sa konali v čínskom Su-čou. Základné princípy, o ktoré sa opierajú sú:

- sú dva základné moduly: modul sveta a rozhodovací modul
- vstupom do rozhodovacieho modulu je približný celkový model sveta, ako ho poskytuje simulácia
- prostredie futbalu je modelované ako Markovov rozhodovací proces
- rozhodovanie je organizované v komplexných a menej komplexných typoch správania
- moduly správania sa učia pomocou učenia odmenou a trestom
- moderné metódy umelej inteligencie sú aplikované všade, kde je to možné a užitočné

Proces rozhodovania hráča je založený na robotických architektúrach založených na správaní (angl. behavior-based). Skupina viac či menej komplexných správaní sa podieľa na tvorbe rozhodnutia agenta. Do určitého stupňa je možné charakterizovať danú architektúru za hierarchickú, pričom je však možné, aby správanie nižšej úrovne zavolalo správanie vyššej a tým vyvolalo požadovanú akciu.

Tím Brainstormers sa zameral vo svojom poslednom výskume na implementovanie tzv. agresívneho správania. Ide o získanie lopty, ktorú má vo vlastníctve súper, a jej prihranie spoluhráčovi. Pri implementácii využili viacvrstvové perceptrónové neurónové siete, pričom úspešnosť zisku lopty zo všetkých pokusov bola 80%. 5% tvorilo chybné správanie.

Druhým prínosom tímu je vytvorenie a sprístupnenie nástroja nazvaného rcg2swf[2]. Ide o aplikáciu spustiteľnú z príkazového riadka, ktorá z logu zápasu dokáže vytvoriť pútavú animáciu formátu Flash. Program má množstvo nastavení a medzi jeho prednosti patrí realistický vzhľad trávnika, použitie avatarov namiesto kruhov predstavujúcich hráčov a automatickú zmenu strán po polčase. Bolo by výhodné použiť ho pri analýze zápasov, prípadne pri prezentácii výsledkov projektu.

### *Zhodnotenie*

Na hráčovi tímu Brainstormers nás zaujala predovšetkým kvalitná neurónová sieť pre agresívne správanie. Keďže existuje podrobná dokumentácia tejto metódy a je dostupný aj zdrojový kód neurónovej siete, považujeme za reálne pokúsiť sa zahrnúť agresívne správanie aj do nášho budúceho hráča.

[2] http://www.ni.uos.de/index.php?id=1024

### 2.2.5 Jahodoví princovia

Tím Jahodoví princovia [6] je minuloročným víťazom fakultnej súťaže. Pri našej analýze, prototypovaní a implementácii sa pravdepodobne odrazíme od tohto tímu.

Tento tím mal hernú prevahu nad každým súperom, ale niektorým mal problém streliť gól, hoci mal vždy veľké množstvo šancí. Väčšinou strieľal z veľkej vzdialenosti, hoci mal voľný priestor a teda sa mohol presunúť bližšie k bráne, kde by sa šanca na strelenie gólu znásobila. Rozhodne je preto nutné zdokonaliť strieľanie na bránu v tomto zmysle.

Tím Jahodových princov sa inšpiroval hráčom Gang of Six. Hráča tohto tímu si zvolili, lebo dosahoval dobré výsledky. Rozhodli sa nerobiť zmeny v jadre hráča. Ich hlavným cieľom je oprava drobných nedostatkov a implementácia rozširujúcich vlastností, ktoré by mali zdokonaliť celkový výsledok. Ciele, ktoré si vymedzili, sú opísané ďalej.

#### *Problematika zahrávania autových situácii*

V dokumentácii v zimnom semestri opisujú, že túto problematiku budú riešiť, v ich ďalších dokumentoch sme však o tom zmienku nenašli. Túto situáciu sa rozhodli riešiť, keďže hráč, ktorého prebrali, vhadzovanie lopty vôbec neriešil a dochádzalo k stratám lopty, čo bolo závažné najmä v obrane. Podľa nich je nájsť túto chybu veľmi náročné, lebo veľmi ťažko simulovať, kedy k nej dochádza.

#### *Spôsoby zdokonalenia brankára*

Tím vyladil nahrávania brankára. V prípade, že sú všetci hráči obsadení, nahrá ich brankár oproti predchádzajúcemu presnejšie a bezpečnejšie. Aj tak sa však niekedy stáva, že brankár nahrá protihráčovi pre hodnotu fitness. Upravili aj vybiehanie brankára a jeho návrat späť do bránkoviska, chytanie striel smerujúcich popri bránkovisku a držanie defenzívnej pozície.

#### *Zavedenie a implementovanie kouča*

Kouča sa autori rozhodli zaviesť preto, aby hráči nemuseli analyzovať súperových hráčov. Kouč vidí nezašumene celé ihrisko. Podarilo sa im rozbehať kouča tímu Gang of Six a UvaTrilearn. V robotickom futbale existujú heterogénni (nejaké vlastnosti majú potlačené, iné zlepšené) a homogénni hráči (všetky vlastnosti majú rovnaké). Viac sa oplatia heterogénni hráči. Kouč má za úlohu na začiatku zistiť atribúty hráčov a na základe toho ich zatriediť do rol. Roly:

- obranca
- stredopoliar
- stredný útočník
- útočiaci krídelník

Každá z týchto rol má pritom iné atribúty. Napríklad stredopoliar prijíma loptu od stredopoliarov, musí sa s ňou dostať ku bránke a kopnúť gól, preto by mal vedieť vyvinúť väčšiu rýchlosť (zrýchlenie) a mal by mať veľkú silu kopu. Takisto musí byť obratný, aby ho obrancovia ľahko neobrali o loptu. Tieto vlastnosti môžu byť zvýraznené na úkor presnosti prihrávky a takisto výdrže.

### *Implementácia zmeny formácií počas zápasu*

Cieľom zmeny formácií počas zápasu je adekvátnejšie a pružnejšie reagovať na zmeny situácií. Túto funkčnosť sa rozhodli autori realizovať pomocou prebehu stredného hráča. Keď tím útočí, stredný hráč prebehne na útočnú polovicu, keď bráni, prebehne na polovicu obrannú.

### *Algoritmus eliminácie premenných*

Algoritmus je založený na koordinačných grafoch. Cieľom je dosiahnuť, aby sa hráč rozhodoval tak, aby maximalizoval spoločnú výnosovú funkciu a teda robil to najlepšie pre tím. Autori nemenili cieľ tohto algoritmu, iba ho trošku upravili. Algoritmus je založený na výnosových funkciách. Napríklad zastavenie súperovej funkcie.

### *Zhodnotenie*

Tím Jahodoví princovia je najpravdepodobnejším základom nášho hráča. Nadviazal na úspechy svojho predchodcu – tímu Gang of Six. Hoci podľa dokumentácie autori mali spraviť viac úprav, málo z nich sa im podarilo. Ide hlavne o rozohrávanie brankárom, zahrávanie autov a algoritmus eliminácie premenných, ktorý sa v hráčovi v skutočnosti nepoužíva. Po dvoch neúspešných rokoch by sme mohli byť práve my, kto tento algoritmus implementuje.

## 2.2.6 Loptoši

Tím Loptoši [7] si ako základ zvolil hráča predchádzajúceho projektu FIITMedia. Vo fáze prototypovania sa rozhodli zmeniť základ hráča a nadviazali na diplomový projekt FiitBA. Prínos hráča FiitBA je implementovanie vrstevnicovej mapy na rozhodovanie hráča a na reprezentáciu informácií o svete. Loptoši ako prvé vylepšili architektúru a prehľadnosť kódu. Implementovali návrhové vzory, ktoré sprehľadnili a zlepšili čitateľnosť kódu. Upravili niekoľko obsiahlych funkcií na viacero kratších. Odstránili duplicitný kód a abstraktné triedy. Nasleduje opis zmien, ktoré autori tímu navrhli a implementovali.

### *Vedúci tímu*

Úlohou vedúceho tímu je sledovanie okolia, aby mal čo najpresnejšie informácie o dianí na ihrisku a na základe týchto informácii mohol riadiť tím napr. zmenou formácie. Ako kandidát na vedúceho sa určil ten hráč, ktorý je vzdialený minimálne dvanásť metrov od lopty (autori vykonali simulácie troch odlišných metód určenia vedúceho). Tím Loptoši implementoval komunikačný protokol, pomocou ktorého si hráči medzi sebou vymieňajú správy o aktuálnom vedúcom tímu aj o jeho zmene. Protokol je navrhnutý tak, aby sa nestalo, že niektorý z hráčov má ešte starú informáciu alebo je viac vedúcich v jednom čase. Proces zmeny vedúceho je takýto:

Súčasný vedúci sa priblíži ku kandidátovi na vedúceho.

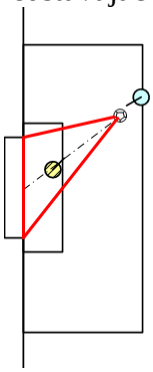
Nastaví si funkciu AttentionTo na kandidáta.

Počká určitý čas a na základe odpovede sa rozhodnutie prijme alebo odmietne.

Problém zmeny vedúceho na vzdialeného hráča sa vyriešil pomocou medzihráča, ktorý sa stane na určitý čas vedúcim a podá tento post tomu vzdialenému hráčovi.

### *Brankár*

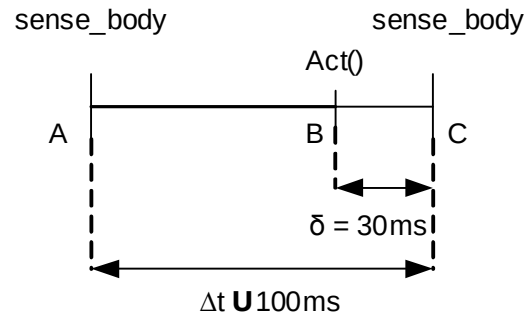
Tím sa v pôvodnom návrhu nezaoberal vylepšením samotného brankára. Avšak zistili, že brankár má výrazné nedostatky, napr. nevidí loptu, zle chytá strely a nedrží defenzívnu pozíciu. Nové riešenie, aby brankár videl loptu, spočíva v tom, že brankár, keď určitý počet taktov (konkrétne dvanásť) nevie, kde sa lopta nachádza, začne sa obzerať okolo seba a hľadať ju. Je zbytočné, aby išiel najprv na svoju domovskú pozíciu. Nová implementácia chytania lopty je taká, že hráč sa rozhodne loptu chytiť, ak vzdialenosť od lopty je parameter servera `catchable_area_l * k`, kde `k` je konštanta z intervalu <0, 1>. Ďalej autori vytvorili funkciu na určenie, či sa lopta môže chytiť o jeden alebo dva takty. Tento nový spôsob chytania lopty významne zlepšil uvedené problémy. Držanie defenzívnej pozície brankára je vyriešené tak, aby sa brankár nachádzal na ťažnici spájajúcej vrchol predstavujúci loptu s jeho protiľahlou stranou.



**Obr. 8.** Príklad dodržania defenzívnej pozície brankára tímu Loptoši

### *Synchronizácia*

Pri skúmaní sa prišlo na problém synchronizácie hráča so serverom. Hráč veľmi často v nejakom takte vykonal akciu pred tým, ako dostal vizuálnu informáciu. Riešenie tohto problému bolo nastavenie času tesne pred koncom taktu, keď hráč vykonal danú akciu (**Obr. 9**).




**Obr. 9.** Znázornenie časovej synchronizácie hráča tímu Loptoši so serverom

Po vykonaní tejto úpravy autori zaznamenali veľké zlepšenie vo volaní funkcie `Act()`, ktorá zabezpečuje konanie hráča.

#### *Situácie*

Tím implementoval nové rozhranie, pomocou ktorého sa môžu doimplementovať nové situácie. Funkcia `LPT_processSituation` slúži na vykonávanie situácií na základe aktuálnej situácie, ktorá je uložená v parametri `LPT_CurrentSituation`. Pre každú úlohu existuje zvláštny kód. Loptoši implementovali aj obrannú situáciu, v ktorej sa obranca z druhej strany postaví k vzdialenej žrdi, aby zabránil gólu v prípade, že by nasledovala prihrávka popred bránku na druhú stranu.

#### *Zhodnotenie*

Hlavným prínosom tímu Loptoši bol vylepšený brankár a celkovo silnejšia obrana vďaka zahrnutej obrannej situácii. Dobrým a odskúšaným nápadom je koncepcia vedúceho hry. Tímu sa nepodarilo realizovať skutočné postavenie hráčov vo formáciách, ale v prípade pokračovania v hráčovi sú na to v ňom vytvorené predpoklady. Koncepciu vedúceho by bolo vhodné prípadne zahrnúť aj do iného hráča, možno sa necháme inšpirovať.

### 2.2.7 Sklo

Tím sklo [8] dobre analyzoval vtedajší stav systému, hráčov a predchádzajúce vylepšenia, ktoré mu poskytli odrazový mostík do ďalšej práce. Ako základ si autori zvolili zdrojový kód ich predchodcov – tímu Stjupit Dox. Pôvodne chceli použiť zdrojové kódy tímu Deravá kopačka, ale napokon sa im zdal Stjupit Dox vhodnejší pre prehľadnosť kódu a oddelenie hráča od vnútorného sveta. Zmeny, ktoré by museli spraviť v tíme Deravá kopačka, by boli väčšie ako pri tomto tíme. Ďalej opíšeme hlavné vylepšenia:

#### *Hráč*

- Zmenený spôsob ukladania informácií o spoluhráčoch a súperoch vo vnútornom modeli sveta.
- Rozšírenie vnútorného modelu o možnosti na prihrávanie. Informácie o možných cieľoch prihrávky sa prenášajú zvukovou správou.
- Využitie heterotypov.
- Predpovedanie pozície lopty kvôli zníženiu času výpočtu.

#### *Zvuková komunikácia*

- Je založená na existujúcej komunikácii tímu Stjupit Dox.
- Pribudla tzv. urgentná správa.

#### *Špeciálne taktické funkcie*

- Dynamické určovanie cieľa hráča driblujúceho s loptou. Berie do úvahy rozloženie súperov.
- Výber miesta, kam hráč kopne loptu, keď už nemôže driblovať.
- Plánovanie výkopu.

#### *Špecializácia taktiky hráčov*

- Špecializácia hráčov na útočníka, stredopoliara a obrancu, pričom každý má svoju vlastnú taktiku.
- Hráč volí taktiku za behu a rozdeľuje stav s loptou a bez lopty.
- Šetrenie energie hráča, ak nemusí bežať naplno.

#### *Brankár*

- Vylepšený spôsob výkopu – brankár najprv vyhodnotí situáciu na ihrisku, zvolí pre seba najlepšiu pozíciu a až potom vykopne

#### *Kouč*

- Sleduje hráčov súpera a zisťuje, aké majú heterotypy. Následne túto informáciu odovzdáva svojmu tímu.
- Je to pomerne zložitá úloha, ale podarilo sa im implementovať rozpoznávanie niektorých heterotypov, a to najmä: rýchlosť, zrýchlenie, rýchlosť otáčania.
- Veľký problém robil šum.

V závere sa hovorí o tom, že najužitočnejší sú rýchli a vytrvalí hráči, preto voľba takýchto heterotypov je výhodná.

#### *Odporúčané vylepšenia*

- Plánovanie postupností akcií hráča.
- Predpoklad správania sa a umiestenia hráčov na základe ich umiestnenia vo formácii a stavu hry.
- Lepšia práca s údajmi o formácii.
- Koordinácia pri útoku.
- Uvoľňovanie spoluhráčov pri výkope a počas hry.

### 2.2.8 FIITMedia

Domáci tím FC Farmári spravil neurónovú sieť na vyhodnotenie vhodnosti prihrávky. Sieť sa inicializovala zo súboru s formátom riadka: d1 d2 d3 y, kde d1 je vzdialenosť od cieľového hráča prihrávky, d2 je vzdialenosť protihráča v smere prihrávky, d3 je kolmá vzdialenosť protihráča a y určuje, či je prihrávka úspešná. Pri vyhodnocovaní vkladá hráč na vstup siete popri spoluhráčovi polohy všetkých protihráčov, ktorých vidí. Prihrávku má hráč uskutočniť, len ak je pravdepodobnosť pre každý vstup vyššia než medzná hodnota. Autori FIITMedie [9] však zistili, že to tak nie je a hráč prihráva takmer vždy, keď nájde spoluhráča vzdialeného 5 až 30 metrov, ktorý je k bráne bližšie než on. Dokonca sa často prihrávalo aj cez protihráčov.

FIITMedia nahradila pre nedostatočnú funkčnosť a ťažkosť trénovania pôvodnú neurónovú sieť vlastnou.

#### *Prihrávanie, kopanie*

Autori FIITMedie pridali k vstupom neurónovej siete na vyhodnocovanie úspešnosti prihrávok parametre štýl kopania (na miesto alebo do behu) a stranu, z ktorej

je postavený súper. Do vektora neurónovej siete vstupujú len určití spoluhráči, ktorý sa určujú algoritmom na zistenie okolia ohrozenia súpera. Okrem toho zjednodušili trénovanie prihrávok.

FC Farmári nedokázali driblovať dostatočne dlho na naviazanie súperových hráčov. FIITMedia to rieši zavedením tzv. nakopávaného driblovania. Hráč si v prípade dostatočne veľkého voľného priestoru nakopne loptu pred seba. V algoritme kopania autori tiež našli a odstránili daktoré chyby, keď hráč kopal loptu iným smerom, než má. Trénovaniu prihrávok a kopania venovali autori najviac času.

Oprava chýb v zdrojovom kóde a implementácia neurónovej siete tvoria základ práce tímu FIITMedia. V dokumentácií sú spomenuté ďalšie zaujímavé nápady, ktoré autori realizovať nestihli.

### *Viac neurónových sietí*

FIITMedia navrhla použitie viacerých neurónových sietí. Každá sieť by bola iná, a preto aj pri rovnakých vstupoch by vznikli iné výsledky. Hráč by mal možnosť použiť všetky siete a na základe presného algoritmu by sa rozhodol, ktorú bude považovať za smerodajnú. Každá sieť by sa natrénovala na inú formáciu súpera, celkove šesť. Viac než šesť formácií videli autori ako neschodné pre nákladnosť a neefektívnosť. Ako alternatívu uvažovali s vytvorením neurónových sietí pre každú pozíciu hráča v tíme. Každý hráč by si podľa svojej pozície vyberal neurónovú sieť len z určitej množiny.

### *Navrhnuté algoritmy na výber vhodnej siete*

- Na základe informácii od kouča
- Kouč posiela informácie len každých tristo taktov. Ak sa pomedzi rapídne zmení pozícia súpera, použitá sieť môže byť veľmi nevhodná.
- Na základe rozhodnutia sa hráča
  - *Pseudonáhodné rozdelenie*

    Hráč vyberie neurónovú sieť. Ak mu určí, že má vykonať prihrávku, vykoná ju. Podľa (ne)úspechu hráč zvýši/zníži dôveryhodnosť danej siete.
  - *Adaptívna kombinácia lokálnych expertov*

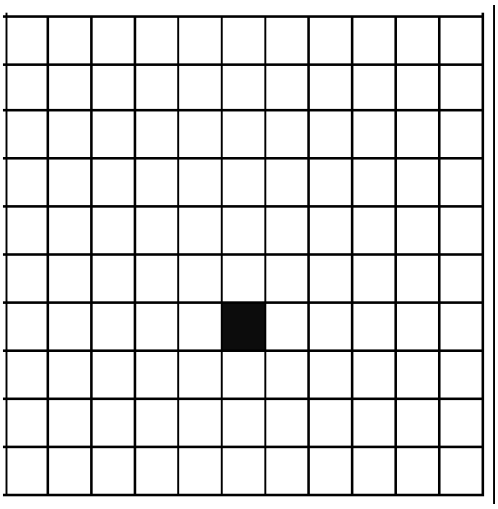
    Myšlienkou je mať nad danými sieťami rozhodovaciu sieť. Tá by sa učila pomocou spätného šírenia chýb (error backpropagation).

### *Akcie*

Ďalším nápadom bolo pridanie akcií, t. j. postupností krokov, ktoré majú vykonať viacerí hráči. Hráč najbližšie k lopte by rozoznal akciu a oboznámil o tom zvukovou správou ostatných hráčov. Určil by pritom aj účastníkov akcie. Na rozoznanie akcie by sa používali vzory resp. masky. Každá maska by určovala rozmiestenie spoluhráčov, voľné miesto (tam nesmú byť protihráči, aby nemohli prerušiť prihrávku) a pozíciu brankára. Každý hráč prijímajúci prihrávku počas akcie by mohol v prípade neúspechu akciu zrušiť alebo zmeniť. Jednotlivé akcie by boli jednoduché, zložitejšia akcia by vznikla kombináciou jednoduchých (akcie by boli hierarchické).

### *Navrhovaný algoritmus porovnávania vzorov*

Pred porovnaním si hráč rozdelí ihrisko na štvorce, čím vznikne mriežka. V nej bude hľadieť smerom nahor a bude sa nachádzať v jednej tretine odspodu. Do mriežky doplní zo svojich informácií o svete spoluhráčov. Mriežku potom prirovná k uloženým vzorom. Vyhovujú len vzory, pri ktorých sa všetky vyplnené pozície kryjú s vytvorenou mriežkou. Pre každý vzor existujú aj iné mriežky, určujúce prítomnosť súperových hráčov. Z tohto sa vyberú možné situácie. Navyše musia byť splnené ďalšie podmienky: pozícia brankára, pozícia v ofsajde a pod. Vyhovujúce situácie sa ohodnotia a vyberie sa najvhodnejšia.



**Obr. 10.** Umiestenie hráča v mriežke pri porovnávaní vzorov – návrh tímu FIITMedia

### *Libero*

Libero je „futbalový obranca voľne zasahujúci do hry po celom ihrisku“. Nemá pevné miesto vo formácii, vyberá si pozíciu sám podľa vývoja hry. FIITMedia navrhla zakomponovanie libera do tímu. Pri ohrození by sa stiahol do obrany, pri útoku by pomáhal. Aby sa nevyčerpal behaním, obranné a útočné libero by boli dvaja rôzni hráči. Ich heterotyp by mal rýchlejšie doplňanie výdrže (staminy).

### *Štatistická analýza rozloženia súperových hráčov*

Návrh bol nechať kouča vyhodnocovať pravdepodobnosť výskytu súperových hráčov na rôznych miestach ihriska záplavovým alebo výškovým algoritmom. Dakoľkokrát za zápas by kouč informoval hráčov o pravdepodobných voľných miestach. Hráči by voľné miesta využívali ako miesta prihrávok.

### *Prihrávka obsadeným hráčom*

Hráč beží s loptou vpred, no pred sebou má len protihráčov. Musí odkopnúť, ináč o loptu takmer isto príde. Zbadá spoluhráča na krídle, ten je však tiež obsadený. Zakričí mu, nech beží vpred, a kopne loptu na vhodné miesto záchytu prihrávky.

FM Farmári prihrávky do behu vôbec nevyužívali.

### *Zhodnotenie*

FIITMedia priniesla do hráča prepracovanú neurónovú sieť s možnosťou jednoduchého trénovania. Vďaka tomu sa zlepšili prihrávky a tie tvoria základ pri (simulovanom) futbale. Mnohé z navrhovaných vlastností hráča nestihli zakomponovať, no pre náš tím môžu byť vhodným zdrojom nápadov.

## 2.2.9 UTTP

UTTP [10] bol tím pôsobiaci na fakulte v roku 2007. Ich hlavnou snahou bolo zlepšiť kvalitu pôvodného hráča tímu Loptoši, z ktorého vychádzali. Kvôli zvýšeniu kvality pristúpili k modularizácii správania, refaktorizácii kódu, dokumentovaniu správania a dopĺňaniu komentárov. V projekte analyzovali správanie hráčov, od najvyššej až po základnú úroveň, ktorá zahŕňala komunikáciu so serverom.

Navrhli spôsob modularizácie správania, pomocou ktorého je možné rôzne druhy správania separovať a následne ich využívať v iných druhoch správania alebo v pôvodných triedach. Ako základ slúži trieda `ModuleManager`, ktorá poskytuje správu a rozhranie pre prácu s jednotlivými modulmi. Ďalej vytvorili rozhranie `ModuleInterface`, ktoré musia implementovať všetky vytvorené druhy správania a triedu `DataStorage` slúžiacu ako centralizované úložisko dát, ktoré potrebujú jednotlivé druhy správania. Nakoniec vyextrahovali samotné druhy správania.

Hráč tímu UTTP obsahuje niekoľko modularizovaných druhov správania. Vytváranie nového modulu realizovali na základe analýzy zdrojových kódov hráča a identifikácie druhov správania.

Tím sa v podstatnej miere venoval oprave chýb, ktoré našli v pôvodnom kóde. Pôvodný hráč nebol kompilovateľný a po určitom čase vždy padol. Projekt následne konvertovali do prostredia Visual Studio 2005. Ďalej hráča optimalizovali aj z hľadiska veľkosti. Objavili a odstránili nepoužívané funkcie, prípadne časti kódu, ktoré nedávali požadované výsledky.

Na účely testovania tím vytvoril vlastný framework. Slúži na testovanie robocupového hráča. Aby ho mohli použiť ďalšie tímy, je nutné doladiť ho (ukončenie testu, kvalitné štatistiky, zber a vyhodnocovanie údajov z testov, a podobne).

### *Zhodnotenie*

Jadrom práce tímu UTTP bola úprava štruktúry existujúceho hráča. Výsledkom je prehľadný zdrojový kód s možnosťou jednoduchého rozšírenia o nové vlastnosti hráča, predovšetkým o nové druhy správania. Na otestovanie nových vlastností tím vyvinul testovací nástroj, hoci nie celkom dokončený, čo ďalej zjednodušuje rozšíriteľnosť hráča. Cieľom tímu bolo, aby sa pokračovatelia mohli sústrediť len na dôkladný návrh nových vlastností, ich zakomponovanie by nemalo byť problém.

## 2.2.10 Zhodnotenie analýzy tímov

Po oboznámení sa so situáciou na fakulte i v zahraničí, sme sa rozhodli v rámci prototypu odskúšať viac ciest:

- Zdokumentovať hráča tímu Jahodoví princovia
- Dôkladné oboznámenie sa s hráčom tímu Jahodoví princovia je dlhodobou úlohou pre všetkých členov. Aby nebolo v budúcnosti (inými tímami) nutné

nazrieť do zdrojového kódu na pochopenie štruktúry hráča, pokúsime sa vytvoriť akúsi používateľsky prívetivú príručku s názornými diagramami.

- Zhodnotiť vhodnosť použitia hráča napísaného v Jave
- Či sa pokúsime o implementáciu v Jave my, alebo až ďalšie tímy, preskúmanie tejto oblasti bude určite prínosom pre fakultný RoboCup.
- Implementovať agresívne správanie podľa vzoru tímu Brainstormers
- Agresívne oberanie o loptu je novinkou vo svete RoboCupu. Ak sa nám ho podarí zahrnúť do hráča, môže sa úroveň domácej ligy posunúť bližšie k svetovej.

## 2.3 Možnosti logovania

Odhaľovanie chýb je súčasťou vývoja každého softvéru. V krátkosti uvádzame, aké možnosti nám v tejto oblasti poskytuje hráč tímu Jahodoví princovia.

### *Logger*

V rámci implementovaného hráča tímu Jahodoví princovia existuje trieda `Logger`. Táto trieda umožňuje logovanie informácií na rôznych úrovniach abstrakcie. Všetky správy sú odovzdané logovacej metóde `log` s indikáciou úrovne jednotlivých správ. Ak je dané, že by sa mali logovať správy s úrovňou n, použijeme metódu `addLogLevel` alebo `addLogRange`, ktorá zaznamená správy úrovne n a ostatné zahodí. Toto nám umožní výpis len určitých správ, ktoré nás zaujímajú. Existuje jedna globálna premenná `Log`, ktorú používajú ostatné triedy využívajúce triedu `Logger` a jej metódy. Inštancia triedy `Logger` je umiestnená v súbore `Logger.cpp` a jej názov je `Log`. Všetky triedy, ktoré ju chcú používať, by si mali vytvoriť referenciu pridaním riadka `extern Logger Log;` a potom môžu používať jej metódy pomocou `Log.log( ... )`. Okrem toho trieda `Logger` obsahuje aj časovač, ktorý umožní výpis času od posledného reštartu časovača.

### *SituationsLog*

Táto trieda slúži na zaznamenávanie situácií, ktoré nastali počas zápasu. Tvorcom je tím GangOfSix, autori programu SituationMonitor umožňujúceho graficky znázorniť jednotlivé situácie, ktoré sú zaznamenané v súboroch XML. Pre každého hráča sa generuje samostatný súbor XML v predpísanej forme. Máme tri typy objektov, ktoré sa zaznamenávajú: bod, čiara a kruh. Bod predstavuje hráčov v okolí. Sú dva typy bodov, a to spoluhráči a protihráči, ktorých odlišujeme farbou bodu. Čiary predstavujú vektory, ktoré nám ohraničujú priestor napr. na vykonanie prihrávky. Kruh predstavuje rádius, v ktorom je hráč schopný prebrať prihrávku. V programe SituationMonitor si môžeme otvoriť jednotlivé súbory XML. Ak sa v tom súbore nachádzajú nejaké záznamy, tak nám program ponúkne výber situácií, ktoré boli zaznamenané. Po ich výbere sa nám vykreslí konkrétna situácia.

## 2.4 Štruktúra hráča Jahodových princov

Cieľom tejto kapitoly je podať prvé priblíženie pripravovaného dokumentu na jednoduché a rýchle vysvetlenie koncepcie, organizácie a hlavných tried robotického

hráča tímu Jahodoví princovia naprogramovaného v programovacom jazyku C++. Dokument bude slúžiť najmä budúcim tímom robotického futbalu, ktorí sa rozhodnú pokračovať v implementovaní tohto hráča a nemajú dostatočné skúsenosti s programovacím jazykom C++.

### 2.4.1 Organizácia kódu

Zdrojové kódy robotického hráča obsahujú dva druhy zdrojových súborov. Prvým je typ súboru s príponou `.h` a druhý s príponou `.cpp`. Na čo slúžia a čo obsahujú jednotlivé súbory v týchto zdrojových kódov, si teraz vysvetlíme.

*Súbory s príponou .h* – obsahuje definíciu triedy. Nachádzajú sa v nej atribúty tejto triedy, metódy a konštruktory. Vo väčšine prípadov obsahuje iba prázdne metódy (definuje sa iba signatúra metódy, metódy nemajú telo). Tieto prázdne metódy potom implementuje súbor s príponou `.cpp`. Súbory s príponou `.h` môžu obsahovať aj viac tried, nie len jednu, ako sa používa vo väčšine iných objektovo-orientovaných jazykoch.

Nasleduje príklad krátkeho úryvku z triedy `BasicPlayer`. Červenou farbou sú zvýraznené poznámky k významu častí kódu.

```
#ifndef _BASICPLAYER_
#define _BASICPLAYER_

#include "ActHandler.h"  // includovanie potrebných súborov

#include "fuzzyobj.h"   // includovanie potrebných súborov

extern Logger Log;

/*! Tu sa nachádza komentár k danej triede*/
class BasicPlayer
{
//jednotlivé atribúty (premenné) tejto triedy
protected:
  ActHandler      *ACT; /*!< ActHandler to which commands can be sent
*/
  WorldModel      *WM;  /*!< WorldModel that contains information of
world   */
  ServerSettings  *SS;  /*!< All parameters used by the server
*/
  PlayerSettings  *PS;  /*!< All parameters used for the player
*/

  ////////////////////// LOW-LEVEL SKILLS//////////////////////////

//jednotlivé metódy (funkcie) tejto triedy
  SoccerCommand   alignNeckWithBody       (
);
  SoccerCommand   turnBodyToPoint         ( VecPosition   pos,
                                            int           iPos = 1
);

} ;

#endif
```

*Súbory s príponou .cpp* – obsahujú implementácie jednotlivých metód, ktoré boli zadefinované v súbore s príponou `.h`. Taktiež obsahujú konštruktor.

```
#include "BasicPlayer.h"   //includovanie potrebných súborov
#include "Parse.h"         //includovanie potrebných súborov

/********************** LOW-LEVEL
SKILLS*******************************/

/*! Tu sa nachádza komentár k danej metóde (funkcie)*/
//implementácia metódy, ktorá je zadefin. v súbore s príponou .h
SoccerCommand BasicPlayer::alignNeckWithBody( )
{
  return SoccerCommand( CMD_TURNNECK,
                        WM->getAgentBodyAngleRelToNeck( ) );
}

/*! Tu sa nachádza komentár k danej metóde (funkcie)*/
//implementácia metódy, ktorá je zadefin. v súbore s príponou .h
SoccerCommand BasicPlayer::turnBodyToPoint( VecPosition pos, int
iCycles )
{
  VecPosition posGlobal = WM->predictAgentPos(iCycles, 0);
  AngDeg angTurn        = (pos - posGlobal).getDirection();
  angTurn              -= WM->getAgentGlobalBodyAngle();
  angTurn               = VecPosition::normalizeAngle( angTurn );
  angTurn               = WM->getAngleForTurn( angTurn,
                                               WM->getAgentSpeed(),
                                               WM->getAgentObjectType() );

  return SoccerCommand( CMD_TURN, angTurn );
}
```

Všetky zdrojové súbory sa nachádzajú v jednom zdrojovom adresári, čo nie je ideálne. Budúce tímy, ktoré preberú tohto hráča, by sa mohli zamerať na lepšie zorganizovanie kódu, aby všetko nebolo pokope.

### 2.4.2 Najzákladnejšie triedy

V tejto kapitole sú opísané najzákladnejšie triedy, predstavujúce hrubý pohľad na štruktúru hráča. Pri každej triede je uvedených len pár metód ako príklad.

`BasicPlayer` – toto je základná trieda hráča. Obsahuje základné metódy (funkcie), ktoré hráč môže urobiť. Môžeme ich rozdeliť na tri úrovne. Sú to:

- *Low level skills* – sú to akoby najnižšie funkcie, ktoré hráč ovláda. Ide o funkcie, ktoré umožňujú otočiť telo k nejakému bodu, komunikovať s ostatnými hráčmi a počúvať ich, nájsť loptu ap.
- *Intermediate level skills* – ide o funkcie, ktoré umožňujú driblovať s loptou rýchlejšie alebo pomalšie, natočiť telo k danému objektu, pohnúť sa k danej pozícii po čiare, kopnúť do lopty ap.
- *High level skills* – ide o funkcie, ktoré umožňujú driblovať s loptou, dať prihrávku dopredu a do voľného priestoru, obohrať súpera ap.

Väčšina týchto metód vracia objekt typu `SoccerComand` (nachádza sa v súbore `SoccerTypes.h`). `SoccerComand` obsahuje všetky informácie o príkaze, ktoré sa majú poslať serveru. Tieto príkazy sú nezávislé od implementácie servera a predtým, ako sa pošlú, sa skonvertujú na reťazec znakov.

`Player` – táto trieda dedí od `BasicPlayer`. V jednoduchosti sa dá povedať, že táto trieda obsahuje komplexnejšie metódy na výber nasledujúcej akcie. Obsahuje metódu `mainLoop`, kde sa rozhoduje, ktorú akciu najbližšie vykoná.

`WorldModel` – obsahuje funkcie, ktoré umožňujú vrátiť informácie o súčasnom alebo budúcom stave sveta (futbalové ihrisko). Zároveň sa stará aj o obnovenie informácií. Tieto informácie môžeme rozdeliť do viacerých kategórii:

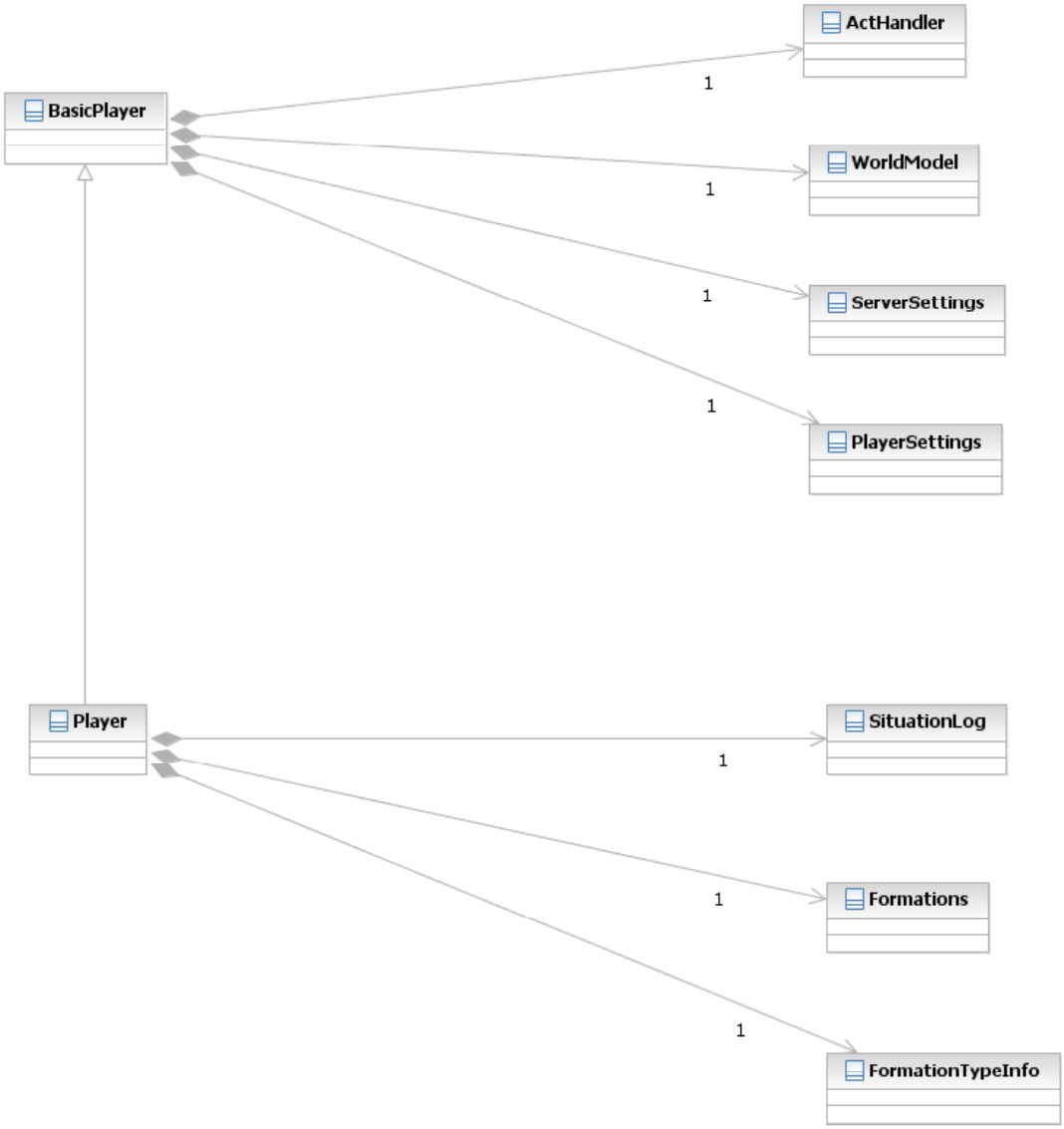
1. Enviromentálne (informácie o serveri)
2. Informácie o zápase (základné informácie o súčasnom stave zápasu)
3. Informácie o objektoch na ihrisku
4. Informácie o akciách, ktoré hráč urobil

`ActHandler` – posiela jednotlivé príkazy na server. `ActHandler` obsahuje rad týchto príkazov. Keď príde signál, prekonvertujú sa tieto príkazy na textové reťazce a pošlú sa na server.

`SituationLog` – táto trieda vytvára logy pre situácie. Obsahuje napríklad metódy, ktoré umožňujú pridávať objetky, ktoré maju byť logované (kapitola 2.3).

`ServerSettings` – táto trieda obsahuje všetky parametre Soccer servera. Ide o paramatre pre hráča ako jeho váha, maximálna rýchlosť, parametre pre únavu, parametre pre loptu, kouča a i. Všetky atribúty majú nastavenú nejakú konkrétnu hodnotu.

`PlayerSettings` – táto trieda obsahuje atribúty, ktoré sú dôležité pre hráča vykonať ďalšiu akciu. Atribút tejto triedy je napríklad rýchlosť lopty pri rýchlej nahrávke. Všetky atribúty majú nastavenú nejakú konkrétnu hodnotu.




**Obr. 11.** Základný diagram tried hráča Jahodových princov

# 3 ŠPECIFIKÁCIA

V tejto kapitole uvádzame, akej činnosti sa budeme venovať v rámci prototypu a aj neskôr v priebehu letného semestra. Za základ nášho hráča pre prototyp použijeme minuloročný tím Jahodoví princovia a zahraničný tím DAInamite.

## 3.1 Nedostatky hráča tímu Jahodoví princovia

Po prečítaní dokumentácie a pozorovaní zápasov tohto hráča sme identifikovali jeho slabšie miesta. V prípade pokračovania v hráčovi by bolo vhodné pokúsiť sa o tieto úpravy:

- Skúsiť vyriešiť autové situácie, lebo tam zbytočne dochádza k strate lopty.
- Vylepšiť hráča, aby miesto strely z väčšej vzdialenosti išiel bližšie k bráne.
- Lepšie prispôsobovanie hráčov pri útoku (postavenie sa na nahrávku a pod.).
- Urobiť diagramy UML štruktúry hráča (kvôli budúcim tímom).
- Zahrnúť algoritmus eliminácie premenných.

Tieto úpravy majú všeobecný charakter a vyžadujú hlbšie preskúmanie zdrojových kódov. Zároveň potrebujeme pohodlný spôsob ich testovania, čo v prípade simulovaného futbalu predstavuje problém pre nemožnosť zastavenia simulácie.

## 3.2 Agresívne správanie hráča

Jedným z cieľov nášho tímového snaženia je implementácia agresívneho správania do hráča tímu Jahodoví princovia na zlepšenie obranných schopností. Ako vzor slúži viacnásobný víťaz RoboCupu 2D – nemecký tím Brainstormers. Výsledkom práce bude navrhnutá, vytvorená a naučená neurónová sieť plus nevyhnutné rozhranie na jej použitie v zápasoch. Nasledujúce riadky obsahujú opis a vysvetlenie problému, ako aj návrh riešenia.

### 3.2.1 Agresívne správanie

Agresívnym správaním sa rozumie snaha hráča prerušiť rozohrávku súperiaceho tímu tak, aby zabránil vzniku útočnej situácie a – naopak – získal loptu na budovanie vlastného útoku. V praxi ide o obranné správanie: pristúpenie k hráčovi s loptou a jej odobratie povoleným spôsobom.

Väčšina tímov robotického futbalu rieši proces získania lopty naivným spôsobom: hráč najbližšie k protivníkovi s loptou sa snaží dostať na dosah lopty, aby ju mohol nahrať spoluhráčom. Táto stratégia často býva neúspešná, hlavne ak súperiaci hráč je z tímu s dobre vyvinutou schopnosťou driblovať [5].

Získanie lopty je vysoko netriviálna úloha, pričom jej zložitosť závisí vo veľkej miere od správania súpera. Pri postupe klasickými programátorskými metódami môže nastať problém vytvorenia špecializovaného hráča, ktorý by fungoval dobre pri niektorých tímoch, ale pri iných vôbec. Okrem toho musí hráč pri procese získania

lopty dbať aj na svoje postavenie a postavenie ostatných spoluhráčov, aby pri možnej strate lopty nedošlo k prečísleniu a súperiaci tím nezískal útočnú výhodu.

Riešenie, ktorým sa zaoberal viacnásobný víťaz RoboCupu 2D, nemecký tím Brainstormers, spočíva vo využití neurónových sietí, presnejšie metódy „reinforcement learning“. Reinforcement learning je metóda učenia sa vďaka interakcii s prostredím. Agent sa učí na základe výsledkov jeho predchádzajúcej činnosti. Nasledujúcu akciu vyberá z množiny predchádzajúcich výsledkov a nových možností, čiže postupuje metódou pokus-omyl. Učiacim signálom je numerická hodnota určujúca úspech akcie, pričom agent sa snaží vyberať v ďalších krokoch akcie, ktoré maximalizujú kumulovanú odmenu v priebehu času [12].

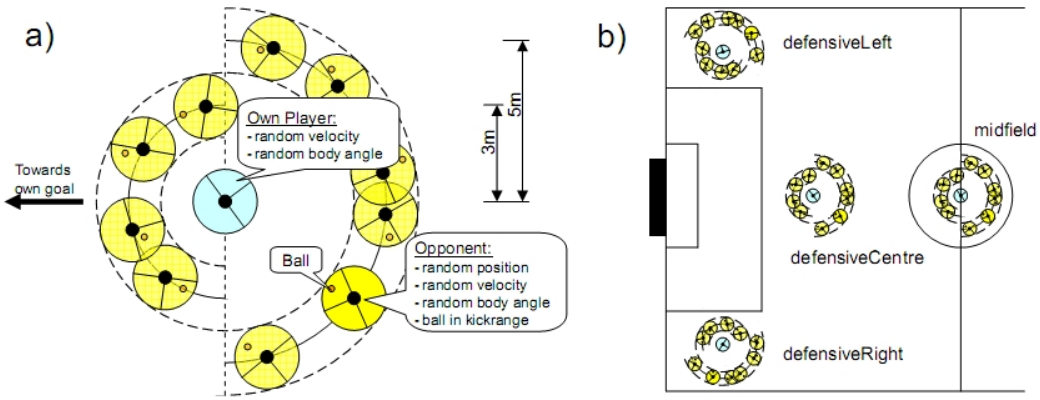
### 3.2.2 Modelovanie problému

Stavový priestor robotického futbalu je mimoriadne rozsiahly a mení sa v každom takte hry. Z toho dôvodu je nutné určiť, ktoré parametre sú nutné na identifikáciu vhodnosti agresívneho správania. Tím Brainstormers sa rozhodol obmedziť problém na deväť dimenzií [5]:

- Vzdialenosť $d$ medzi naším a súperiacim hráčom s loptou
- Rýchlosť nášho hráča ($v_x$ a $v_y$ zložky pohybu)
- Absolútna hodnota $v_{opp}$ – rýchlosť súperiaceho hráča s loptou
- Pozícia lopty na mape ($b_x$ a $b_y$)
- Natočenie tela hráča relatívne k súperovej pozícii (uhol $\alpha$)
- Natočenie tela útočiaceho hráča vzhľadom na našu bránu (uhol $\beta$)
- Hodnota strategického uhla $\gamma = \angle$ GOM (kde G je pozícia našej bránky, O pozícia súpera a M pozícia nášho hráča)

Hráčovi je povolené použiť príkazy `dash(x)` a `turn(y)` z ohraničených intervalov tak, že existuje 76 vykonateľných akcií. Intervaly pre jednotlivé príkazy sú:

- $x \in \langle -100,100 \rangle$
- $y \in \langle -180°,180° \rangle$



**Obr. 12.** a) Vymedzený priestor v okolí hráča s vyznačenými príkladmi umiestnení súpera (žlté kruhy), b) tréningové regióny na mape [11]

Tréningové stavy, slúžiace na učenie hráča, boli modelované podľa nasledujúceho modelu. V okolí hráča si možno predstaviť dva polkruhy, jeden s polomerom 3 m, druhý 5 m. Polkruh s menším polomerom je orientovaný smerom k našej bránke, polkruh s väčším k súperovej. Do takto vymedzeného priestoru je umiestnený súperiaci hráč a lopta, pričom lopta je v blízkosti súpera a má nulovú rýchlosť. Rýchlosti hráča a súpera, ako aj ich vzájomné natočenie, sú zvolené náhodne.

Okrem toho tím Brainstormers vybral štyri tréningové regióny na mape. Jeden v strede ihriska, dva v jednotlivých rohoch našej polovice ihriska a posledný v strede našej časti ihriska. Pri návrhu týchto pozícií vychádzali z idey, kadiaľ súperiace tímy vedú útok (všeobecne stredom, alebo po krídlach plochy).

Navrhovaná sieť pozostáva z 28 neurónov pričom obsahuje jednu skrytú vrstvu (architektúra po vrstvách: 9-18-1). Ako aktivačná funkcia je použitá sigmoidná funkcia a učiacim algoritmom je RPROP algoritmus (Resilient backpropagation). Za rýchlosť učenia α bola zvolená hodnota 1,0.
Chybová funkcia RPROP algoritmu je:

$$E = \sum (d_k - out_k)^2 + 10^{-\alpha} \sum w_{ij}^2$$

Pritom $out_k$ je výstupom siete, $d_k$ je požadovaným výstupom, $w_{ij}$ je j-tou váhou neurónu $i$ a $\alpha$ parametrom váh.

Pri trénovaní siete sa jednotlivé stavy nevymazávajú, ale prebieha ich pretrénovanie po 250 spustených inštanciách.

### *Špecifikácia podľa tímu Brainstormers*

Tím pri vyhodnocovaní výsledkov identifikoval päť možných koncových stavov:

1. *úspech* – Lopta sa dostala do hrateľnej oblasti nášho hráča, hráč obral súpera a môže nahrať voľnému spoluhráčovi.
2. *neúspech* – Neúspešná epizóda môže vzniknúť v dôsledku straty lopty hráčom, príp. keď lopta opustí hraciu plochu a pod.
3. *súper spanikári* – Prejavuje sa nezmyselným odkopnutím lopty súpera mimo svojho hracieho priestoru, často dopredu alebo do voľného priestoru. Takúto epizódu možno považovať aj za remízu.
4. *chyba* – Protihráč prešiel cez nášho atakujúceho hráča a je od neho vzdialený viac ako sedem metrov.
5. *time-out* – Počas trvania epizódy (35 krokov) nenastala ani jedna z vyššie uvedených situácií.

Tím dosiahol po naučení úspešnosť siete presahujúcu 80%, pričom množstvo chybových epizód sa držalo pod úrovňou 5%.

V priebehu klasických zápasov (6 000 taktov) mali hráči približne 66 epizód, v ktorých použili agresívne správanie. Pri konzervatívnom odhade úspechu 50% to znamená viac ako 30 získaní lopty, čím zabránili gólovým situáciám a – naopak – mohli prejsť do útoku.

### 3.2.3 Prevzatie správania

Náš tím nemá ambíciu navrhnúť vlastnú neurónovú sieť ani vlastný algoritmus, lebo to by sme z časových dôvodov nestíhali. Zameriame sa na preskúmanie dokumentácie a voľne dostupného kódu tímu Brainstormes a jeho nasadenie do nášho hráča. V rámci prototypu nás zaujíma predovšetkým kompatibilita riešenia s kódom tímu Jahodoví princovia. Je potrebné zahrnúť do hráča neurónovú sieť tak, aby sme boli schopní trénovať ho na agresívne správanie.

## 3.3 Využitie programovacieho jazyka Java

Doterajší hráči simulovaného futbalu na našej fakulte sú postavení výlučne na programovacom jazyku C++. Čoraz viac sa do popredia v svete objektovo orientovaných jazykov dostáva Java. Väčšina členov nášho tímu nemá skúsenosti s C++. V ďalších rokoch predpokladáme na fakulte podobný vývin, preto chceme overiť možnosť prejsť na programovací jazyk Java.

Ako prvé si treba uvedomiť náročnosť hráča napísaného v Jave. Oproti jazyku C++ sú vyššie nároky najmä na pamäť. Navyše pravidlá RoboCupu neumožňujú komunikáciu medzi procesmi predstavujúcimi jednotlivých hráčov tímu. Na zabezpečenie tejto podmienky sa každý proces spúšťa vo vlastnom virtuálnom stroji (Java Virtual Machine), čo ďalej zvyšuje pamäťové nároky. Preto prvým naším krokom je otestovanie nárokov hráča DAInamite a overiť možnosť nasadenia v podmienkach fakultného turnaja.

Ďalším aspektom je samotné použitie hráča. Keďže nejde o fakultného hráča, nevieme určiť, na akej úrovni z hľadiska štruktúry je. Potrebujeme ukázať jeho vhodnosť ako základ pre pokračovanie, t. j. musíme si byť istý, že obsahuje potrebné funkcie (hlavne nižšej úrovne) použiteľné na vystavanie vyššej rozhodovacej logiky.

# 4 HRUBÝ NÁVRH

V tejto kapitole rozpisujeme, ako máme v pláne zahrnúť špecifikované požiadavky do hráča.

## 4.1 Agresívne správanie[3]

Navrhnuté agresívne správanie (opísané v kapitole 3.2) sa skladá zo štruktúry obsahujúcej vstupný stav, množinu možných stavov a ich ohodnotení a spájaného zoznamu štruktúr s údajmi stav, ohodnotenie, akcia.

Vstupný stav predstavuje 9 hodnôt (d, $v_x$ a $v_y$ zložky pohybu, $v_{opp}$ ,$b_x$ a $b_y$, uhol $\alpha$, uhol $\beta$, uhol $\gamma = \angle$ GOM ), ktoré vstupujú do vyhodnocovacej funkcie tvorenej neurónovou sieťou. Tieto hodnoty sú získané pri identifikácii agresívneho správania rozhodovacou logikou hráča.

Množina možných stavov je pole o veľkosti 76 prvkov. Obsahuje pre vstupný stav vygenerovaných 76 možných akcií, ku každej akcii prislúchajúci stav vyjadrujúci pozíciu hráča, súpera a lopty po akcii a ohodnotenie akcie získané po prechode neurónovou sieťou.

Spájaný zoznam je kolekcia maximálne 35 prvkov. Každý prvok obsahuje stav, ktorý bol na vstupe funkcie (vstupný stav), akciu, ktorá prislúchala danému stavu (t. j. bola vybraná zo 76 možností), a jej ohodnotenie získané po prechode neurónovou sieťou.

Implementácia predpokladá vytvorenie nového správania v kóde Jahodových princov v súbore *Player.cpp* menom *DepriveBall* („získaj loptu“). Je volané po identifikácii správania v rozhodovacej logike hráča. Na starosti má získanie vstupných hodnôt a výber najvhodnejšej akcie pre dané hodnoty. Súčasťou správania musí byť práca s neurónovou sieťou, menovite prečítanie súboru, ktorý ju obsahuje, volanie a učenie siete a spracovanie hodnôt, ktoré vracia. Prečítanie súboru musí byť kvôli rýchlosti realizované len pri prvom volaní správania, potom sa bude už len pristupovať k referencii na daný objekt.

Proces Agresívneho správania možno rozdeliť do šiestich fáz:

1. Identifikácia agresívneho správania v hre.
2. Generovanie možných stavov.
3. Ohodnotenie stavov prechodom cez neurónovú sieť.
4. Výber akcie a zapamätanie si stavu, ktorý ju generoval.
5. Ak akcia neskončila, pokračujem bodom 1.
6. Ak akcia skončila, vyhodnotím jej výsledok.

[3] Agresívne správanie sme nakoniec vôbec neimplementovali

### *Identifikácia agresívneho správania*

Identifikácia prebieha v rozhodovacej logike hráča. Menovite v logike hráča Jahodových princov v súbore *Player.cpp :: mainLoop()* a následne *PlayerTeams.cpp :: deMeer5()*. Agresívne správanie je vhodné začleniť do tejto časti (od riadka *709* v metóde *deMeer5()*):

```
Ak som najbližšie k lopte:
      Idem k lopte; zohľadní sa výdrž.
      Pohľad kmitá popri lopte.
```

Po zmene:

```
Ak som najbližšie k lopte:
      Ak má loptu súper a som blízko neho:
            Oberiem súpera.
      Inak:
            Idem k lopte; zohľadní sa výdrž.
            Pohľad kmitá popri lopte.
```

V algoritme sú navrhnuté dve nové funkcie. Prvá zisťuje, či má loptu súper a podľa toho vracia logickú hodnotu *true* alebo *false*. Súper má loptu, pokiaľ poznáme jeho pozíciu a natočenie ako aj pozíciu lopty a z týchto údajov vyplýva, že lopta sa nachádza v súperovej oblasti kopu (*kickable area*).

Druhá funkcia taktiež vracia logickú hodnotu – podľa toho, či sa hráč nachádza v blízkosti súpera. Byť blízko súpera znamená, že súper je v okruhu maximálne 5 m od čela hráča, prípadne vo vzdialenosti max. 3 m za hráčom.

Ak obe funkcie vrátia hodnotu pravda, hráč identifikoval akciu obratia súpera o loptu a zavolá funkciu *DepriveBall*.

### *Generovanie možných stavov*

Tím Brainstormers identifikoval 76 možných akcií pre vstupný stav, pričom medzi možné príkazy zradili len dash(x) a turn(y) z ohraničených intervalov. Intervaly pre jednotlivé príkazy sú:

$x \in \langle -100,100 \rangle$

$y \in \langle -180°,180° \rangle$

Hráč môže naraz vykonať len jednu akciu, buď zmeniť svoju rýchlosť, alebo natočenie. Rýchlosť je možné meniť po piatich krokoch, čím metóda generuje celkovo 40 možných zmien rýchlosti. Natočenie tela je možné meniť po desiatich stupňoch, čo v súčte dáva 36 možných akcií.

Generovanie pracuje so vstupným vektorom údajov, čiže stavom, ktorý bol aktívny v čase identifikácie agresívneho správania. Na tento vstupný vektor aplikuje vždy inú zo 76 akcií, pričom výstupný vektor spolu s akciou, ktorá ho vytvorila, ukladá do množiny vstupných stavov.

Výsledkom metódy je naplnené pole 76 možných stavov spolu s príslušnými akciami čakajúce na ohodnotenie.

### *Ohodnotenie stavov prechodom cez neurónovú sieť*

Metóda vyhodnocovania stavov predpokladá inicializovaná neurónovú sieť s nastavenými váhami prečítanými z textového súboru. Na vstup dostáva vektor deviatich hodnôt, výstupom je jedna hodnota – reálne číslo vyjadrujúce vhodnosť akcie.

Metóda ohodnotí všetkých 76 možných stavov, pričom výstup pre každý zapíše do poľa k príslušnej akcii. Po ukončení vyberie stav s najvyššou hodnotou výstupu, vstupný vektor, akciu a jej ohodnotenie zapíše do spájaného zoznamu akcií. Zvýši počítadlo krokov akcie o 1 (maximum je 35) a zavolá danú akciu pre hráča.

### *Ukončenie akcie*

V správaní *DepriveBall* je nutné testovať ukončujúce podmienky po vyvolaní správania, ako aj po vykonaní akcie. Pri vyvolaní je testované počítadlo krokov (<35). Ak počítadlo dosiahlo maximálny povolený počet stavov, nezavolá žiadnu akcie a skúsi prenechať výber akcie rozhodovaciemu algoritmu. Pritom vynuluje počítadlo na 0 a vymaže spájaný zoznam predchádzajúcich akcií.

Druhé testovanie nastáva po vykonaní akcie hráčom. Správanie testuje, či epizóda skončila úspechom alebo jednou z neúspešných možností.

### *Učenie siete*

Učenie siete je založené na metóde učenia odmenou a trestom (angl. reinforcement learning). Princípom je, že sieti povieme, či jej výstup bol dobrý alebo zlý, ale nepovieme jej aký presne mal byť. Využitý je algoritmus čiastkových rozdielov TD (z anglického temporal difference). Požadované výstupy siete, ktoré sa nazývajú odmena (angl. reward), sú určené ako:

$$r = \begin{cases} 1 & \textit{ak sekvencia akcií vedie k výhre} \\ 0 & \textit{ak sekvencia akcií vedie k remíze} \\ -1 & \textit{ak sekvencia akcií vedie k prehre} \end{cases}$$

Na vstup siete dávame postupne všetky možné akcie (vstupné vektory). Nasledujúcu akciu v čase t vyberieme podľa najväčšej hodnoty skutočného výstupu siete *o(t)*.

Učenie siete prebieha na stovkách akcií, pričom je nutné zabezpečiť ich rôznorodosť (t. j. rozličné postavenie na ihrisku, postavenie súpera vzhľadom na hráča a loptu). Časť akcií vedie k výhre, časť k prehre, respektíve k remíze. Po skončení trénovania vie sieť ohodnotiť každú akciu v čase *t* číslom *o(t)*, ktoré je úmerné pravdepodobnosti výhry.

Optimálne váhy siete sú nájdené a uložené do súboru so sieťou pomocou gradientovej metódy najprudšieho spádu. Úprava ľubovoľnej váhy v sieti pri vstupnom vektore $\bar{x}^{(t)}$ je určená TD pravidlom:

$$\Delta w^{(t)} = a \ \ (o^{(t+1)} - o^{(t)}) \sum_{n=1}^{t} \frac{\partial o^{(n)}}{\partial w}$$

## 4.2 Úroveň dostupného kódu tímu DAInamite

Súčasťou prototypu bude overenie vlastností voľne dostupnej verzie zahraničného tímu DAInamite napísaného v Jave. Najskôr otestujeme nároky na hardvér, potom nás bude zaujímať predovšetkým, či sme schopní hráča použiť ako kostru pre náš vývoj.

Voľne dostupná verzia neobsahuje komplexné vyššie správanie. Práve to je sféra, v ktorej by sme pokračovali. Vyššie správanie by sme prebrali z hráča tímu Jahodoví princovia (aj preto je potrebné, aby sme sa s ním podrobne oboznámili). Nemôžeme však predpokladať úspešné zvládnutie tejto úlohy, ak kód tímu DAInamite neposkytuje dobrý základ. Pod základom rozumieme funkcie nižšieho „správania“, ako je komunikácia so serverom, udržiavanie si modelu sveta, schopnosť plniť jednoduché úkony typu kopni, bež a pod. V prototype sa zameriame na postupné odskúšanie týchto funkcií (napr. pomocou trénera):

- *Kop* – Zaujímajú nás možnosti kopnutia istým smerom, určitou silou, na konkrétnu pozíciu.
- *Beh* – Aj pri behu je podstatná schopnosť bežať určitým smerom a na konkrétny bod. Zároveň je zaujímavé narábanie s výdržou (staminou). Beh je činnosť na viac taktov, preto je potrebná aj možnosť prerušiť ho v prípade potreby.
- *Nahrávka* – V podstate ide o špeciálny prípad kopu na jednej strane (prihrávajúcim hráčom) a špeciálny prípad behu na druhej (cieľových hráčom pri prihrávke do behu).
- *Pohľad* – Odskúšame pohľady hráča s rôznou šírkou a rôznym smerom. Overíme aj príjem zrakových správ a ich spracovanie.
- *Komunikácia* – Pod komunikáciou chápeme výmenu krátkych správ medzi hráčmi. Popri schopnosti vyslania správy nás zaujíma aj možnosť počúvať iného hráča (príkaz `AttentionTo`).
- *Prehľad o dianí na ihrisku* – Toto už patrí k stredným až vyšším schopnostiam hráča, no napriek tomu je existencia modelu sveta nutným predpokladom na pokračovanie v kóde.

Úroveň týchto funkcií nám poskytne prehľad o tom, či má význam pokračovať v tíme DAInamite. Ide totiž o základné funkcie, a ak nie sú dostatočne implementované, pravdepodobne by sme neboli schopní nahradiť ich vlastnými, čo by mohlo mať za následok neschopnosť nášho hráča zúčastniť sa turnaja.

# 5 PROTOTYP

Keďže nie sme pevne rozhodnutí o základe nášho hráča, prototyp je rozdelený na dve časti podľa hráča, ktorého sa týka. Prvá časť je venovaná hráčovi tímu Jahodoví princovia, druhá sa týka hráča tímu DAInamite.

## 5.1 Server verzie 13

Dlhší čas sa na fakulte používa server verzie 9.3.6, pričom to nie je zďaleka najnovšia verzia. Dôvodom sú najmä problémy s kompiláciou novších verzií pod operačným systémom Windows, v ktorom server na fakultných zápasoch beží. Počas písania tohto dokumentu sa však na oficiálnej stránke projektu rcssserver[4] (The RoboCup Soccer Simulation Server) objavila nová verzia skompilovaná pre OS Windows. V prototype sme odskúšali jej stabilitu, aby sme si boli istí jej schopnosťou nasadenia pri reálnych zápasoch.

### 5.1.1 Server verzie 13 a tím Jahodoví princovia

Hráč tímu Jahodoví princovia v pôvodnom stave nefungoval s novým serverom. Pripojenie na server prebehlo úspešne, no v zápätí hráč spadol s chybou prístupu do pamäte. Dôkladným skúmaním zmien servera[5] sme identifikovali chybu. Nový server vygeneruje na začiatku hry štandardne osemnásť heterotypov naproti doterajším siedmim. Hráč Jahodových princov nerátal s vyšším množstvom než sedem. Jednoduchá úprava umožnila hráča s novým serverom spustiť. Otestovali sme aj spätnú kompatibilitu upraveného hráča so staršími verziami servera.

### 5.1.2 Server verzie 13 a fakultné tímy

Stabilitu servera verzie 13.0.1[6] sme overili spustením viacerých zápasov rôznych fakultných tímov z minulých rokov. Zaujímala nás hlavne stabilita vlastného servera, teda jeho schopnosť vydržať bežať bez chyby celý zápas. Pozorovali sme však aj stálosť a zmeny v správaní hráčov. Zistili sme zaujímavú skutočnosť: hráč tímu Loptoši [7] ako jediný so serverom nefunguje. Horšie je, že sám hráč sa nejaví ako problémový, v skutočnosti skončí s chybou server. Kuriózne je číslo taktu, v ktorom server spadne – ak server spadne, je to vždy takt 307. Pokračovatelia hráča – tím UTTP tento problém nemá, preto sme jeho príčinu neskúmali.

### 5.1.3 Zhodnotenie

Dostupnosť novej verzia servera umožní ďalší rozvoj RoboCupu na našej fakulte. Zatiaľ nemôžeme nový server vyhlásiť za celkom stabilný. Väčšina fakultných tímov

[4] http://sourceforge.net/project/showfiles.php?group_id=24184&package_id=16551
[5] Stručný zoznam zmien servera možno nájsť v prílohe B k zápisnici č. 7 v dokumentácii k riadeniu
[6] Počas písania dokumentácie vyšla verzia 13.0.2. Výsledky zbežného testovania sa nelíšili od 13.0.1

s ním pracovať dokáže, preto považujeme jeho nasadenie na fakultných zápasoch aj v súčasnom stave za reálne.

## 5.2 Hráč tímu Jahodoví princovia

Pri vylepšení hráča sme sa zamerali na časti, ktoré nám odporučil tím Jahodových princov. Podarilo sa nám upraviť časté strieľanie z diaľky ako aj rozohrávanie brankára.

### 5.2.1 Časté strieľanie z diaľky

Problémom hráča Jahodových princov bola jeho častá streľba z diaľky, hoci mohol ešte postupovať na bránku súpera a tým pádom sa dostať do lepšej pozície, či už streleckej alebo možnosť nahrať lepšie postavenému spoluhráčovi. Pri analyzovaní rozhodovania hráča sa nám podarilo zistiť, kde hráč vykonáva rozhodnutie, kedy voliť strelu a upravili sme túto časť, aby nestrieľal z diaľky príliš často. Pri implementácii sme zmenili čísla, ktorými sa delila veľkosť ihriska (`PITCH_LENGTH`).

```
// try to kick to score if we are near to opponent goal
if (posAgent.getDistanceTo(goalMiddle) <= (PITCH_LENGTH / 4.5))
{
   Log.log(100, "kopem...");
    soc = kickToScore();
}
// or try to pass to teammate
else
{
   Log.log(100, "prihravam...");
   double fit = 0;
   soc = kickToPass(&fit);

// if fitness is too bad then kick to score
if (fit < 5 && posAgent.getDistanceTo(goalMiddle) >=
      (PITCH_LENGTH / 4))
   soc = kickToScore();
}
ACT->putCommandInQueue( soc );
ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
```

### 5.2.2 Úprava rozohrávania brankára

Brankár Jahodových princov sa pri rozohrávaní riadi počtom súperových hráčov a loptu rozohrá do oblasti, v ktorej je najmenej hráčov súpera. Toto rozhodovanie sme zmenili tak, aby sa brankár rozhodoval na základe počtu spoluhráčov. Pri implementácii sme pokusne zmenili správanie z výberu oblasti s najmenším počtom súperových hráčov na výber oblasti s najväčším počtom spoluhráčov (`OBJECT_SET_OPONENTS` v kóde sme zmenili na `OBJECT_SET_TEAMMATES`). Taktiež sme upravili volanie funkcie `getX`, ktoré sa používa na umiestenie brankára na určitú pozíciu. Volanie sme upravili tak, aby sa brankár dostal na menšiu x-ovú súradnicu a mal tým pádom väčšiu možnosť výkopu.

```
if( WM->getTimeSinceLastCatch() == 25 && WM->isFreeKickUs() )
```

```
{
    // move to position with lesser opponents.
   if( WM->getNrInSetInCircle( OBJECT_SET_TEAMMATES,
                               Circle(posRightTop, 15.0 )) >
       WM->getNrInSetInCircle( OBJECT_SET_TEAMMATES,
      Circle(posLeftTop, 15.0 )) )
      soc.makeCommand( CMD_MOVE,posRightTop.getX()-3,
                       posRightTop.getY(),0.0); else
      soc.makeCommand( CMD_MOVE,posLeftTop.getX()-3,
                       posLeftTop.getY(), 0.0);
   ACT->putCommandInQueue( soc );
}
else if( WM->getTimeSinceLastCatch() > 28 )
{
      //soc = kickTo( VecPosition(0,posAgent.getY()*2.0), 2.0 );
    double fitt =0;
    soc = kickToPass(&fitt);
   //soc = kickToPass( );
   ACT->putCommandInQueue( soc );
}
```

### 5.2.3 Zhodnotenie

Pri útoku hráča sme spozorovali mierne zlepšenie. Hráč sa nepokúša strieľať na bránu z takej diaľky ako dosiaľ. Vykopávanie brankára sa zmenilo, no výsledky nemožno označiť jednoznačne lepšie. Rozhodovanie brankára pri výkope by malo zahŕňať komplexnejší prieskum okolia. Množstvo spolu-/protihráčov totiž nie je dostatočným kritériom.

## 5.3 Hráč tímu DAInamite

Hráč tímu DAInamite je prvý potenciálny základ pre hráča na fakulte napísaného v jazyku Java. Úlohou prototypu je ukázať, že použitie jazyka Java je schodným riešením. Venujeme sa testovaniu nárokov i overeniu funkčnosti voľne dostupného hráča DAInamite.

### 5.3.1 Testovanie nárokov hráča tímu DAInamite

Našou úlohou bolo otestovať javovú implementáciu hráča a porovnať výkonnosť s implementáciou iného tímu v C++ na dvoch rôznych strojoch. Prvé testy prebiehali tak, že všetky potrebné súčasti bežali na jednom stroji. To znamená, že RoboCup server, tím Squirell Squadron a aj javový tím DAInamite bežali na jednom počítači naraz. Výsledky boli nasledovné:

#### *Intel Core II Duo E6750 @ 2.66GHz, 2.67GHz, 2 GB RAM – Desktop*

Vykonali sme šesť simulácií, každá prebehla bezchybne až do konca polčasu – 1 000 simulačných taktov. Vykonanie 1 000 taktov trvalo dve minúty aj 40 sekúnd, z čoho vyplýva, že desať taktov zodpovedá 1,6 sekundy reálneho času. Tento čas bol rovnaký pre všetky simulácie.

Zaťaženie stroja, čo sa týka nárokov na procesor, bolo pomerne nízke – zaznamenali sme približne 5% nárast zaťaženia procesora počas behu simulácie oproti pokojovému stavu.

#### *Intel Core II T7200 @ 2.00GHz, 996MHz, 2 GB RAM – Notebook*

Rovnako sme vykonali šesť simulácií, pričom väčšina nebola úspešne ukončená. Simulácie prebiehali nasledovne:

1. 446 taktov za 1,17 minúty
2. 674 taktov za 1,49 minúty
3. 104 taktov za 0,18 minúty
4. 360 taktov za 0,58 minúty
5. 1 000 taktov za 2,40 minúty
6. 1 000 taktov za 2,40 minúty

Spolu 2 584 taktov za 422 sekúnd, z čoho vyplýva priemerný čas 1,63 sekundy na desať taktov. Avšak simulácie vo väčšine prípadov neprebehli do konca a nepodarilo sa nám zistiť dôvod, prečo to tak bolo.

Zaťaženie procesorov kolísalo okolo hodnoty +40% počas behu simulácie. Nároky na pamäť boli na oboch strojoch približne rovnaké:

- RoboCup server približne 11 MB
- jedenásť hráčov tímu Squirell Squadron 10 MB
- jedenásť hráčov v Jave + monitor tímu DAInamite 603 MB

Spoločná záťaž bola približne 625 MB pamäte.

#### *Viac počítačov*

Ďalšie testy prebiehali s použitím silnejšieho stroja ako servera, na ktorom bol spustený RoboCup server a javový tím. Na slabšom stroji bolo spustených len jedenásť hráčov tímu Squirell Squadron. Simulácia na serveri prebiehala približne rovnakým spôsobom ako pri prvých pokusoch (zaťaženie procesora +4% až +5%) a na slabšom stroji spotrebovalo jedenásť hráčov približne 15 MB pamäte a vyťaženie procesora sa pohybovalo okolo +8%.

#### *Zhodnotenie*

Z predchádzajúceho jasne vyplýva, že implementácia v Jave má oveľa vyššie nároky na pamäť aj silu procesora ako C++ implementácia. Vyťaženie procesora ani v jednom prípade nebolo nad 50%. Čas 1,6 sekundy na desať taktov v oboch prípadoch hovorí o tom, že simulácia prebiehala celkovo pomalšie, ako by mala, ale nebolo to spôsobené nedostatkom výpočtovej sily.

### 5.3.2 Funkčnosť hráča DAInamite

Cieľom je overiť funkčnosť voľne dostupného kódu hráča tímu DAInamite. Najprv sme identifikovali dostupné funkcie nižšieho správania hráča, potom sme overili ich funkčnosť.

### *Implementované akcie hráča*

Všetky akcie hráča sú odvodené od abstraktnej triedy *Action*. Implementované a fungujúce akcie v hráčovi tímu DAInamite sú tieto:

- *AttentionToAction* – vyšle príkaz `attentionto` (hráč sa sústredí na zvukový povel od iného hráča).
- *CatchAction* – vyšle príkaz na chytenie lopty, ktorý môže vykonať len brankár.
- *ChangePlayerTypeAction* – príkaz, ktorý dáva tréner, keď chce vystriedať hráča za nového, napríklad za hráča iného typu. Je povolený trikrát za hru.
- *ChangeViewModeAction* – zmení spôsob videnia hráča, ktorý je definovaný kvalitou a uhlom pohľadu (jeho šírkou). Toto zároveň ovplyvňuje, ako často hráč dostane zrakovú informáciu.
- *DashAction* – zrýchľuje hráča (beh).
- *KickAction* – príkaz, ktorým hráč kopne do lopty.
- *PointToAction* – hráč vystrie ruku a ukáže na určité miesto na ihrisku. Táto akcia je veľmi skreslená.
- *SayAction* – slúži na posielanie správ medzi hráčmi na ihrisku. Pomocou tohto si môžu posielať správy.
- *SenseBodyAction* – zavolanie tejto akcie spôsobí, že hráč dostane v ďalšom takte informáciu *SenseBody*. Už sa explicitne nepoužíva, lebo je štandardne nastavené posielanie `sensebody` v každom takte.
- *TackleAction* – akcia, ktorá slúži na oberanie hráčov o loptu. Po použití tejto akcie sa hráč nemôže určitý počet taktov hýbať a môže ju vykonať len v smere, na ktorý je natočený a dopredu alebo dozadu.
- *TurnNeckAction* – otočí hlavou hráča o určitý uhol od -90 do 90 stupňov v závislosti od natočenia tela.
- *MoveAction* – hráč sám sebe nastaví pozíciu na ihrisku. Toto sa dá použiť len pred zápasom v režime *before kickoff*, alebo to môže brankár pred výkopom použiť v rámci svojho pokutového územia.
- *InitAction* – pripojí hráča daného typu do hry.
- *ByeAction* – vyšle príkaz, ktorý spôsobí odpojenie hráča (nepoužíva sa v praxi).
- *ReconnectAction* – agent sa znova pripojí na server.
- *CoachAction* – akcia používaná trénerom na prípravu špeciálnych situácií, ktorá rozostaví hráčov alebo loptu na určené miesto na ihrisku alebo môže zmeniť režim hry.
- *PlayerTypeAction* – slúži na nastavovanie heterogénneho typu hráča.

### *Implementované stavy hráča*

Rozhodovanie tímu je založené na systéme stavov, v ktorých sa hráč môže nachádzať. Ak sú splnené vstupné podmienky niektorého z pripravených stavov, spustí sa vyhodnocovanie výhodnosti pre daný stav a na základe ohodnocovacieho systému sa vyberie stav, ktorý je pre ďalší priebeh zápasu z hľadiska daného hráča najvýhodnejší. Tento systém umožňuje ľahko pridať akýkoľvek nový stav, prípadne zmeniť podmienky

pre už existujúci stav. Ak by sme chceli implementovať nejaký typ nového správania do hráča, bolo by to práve pomocou vytvorenia nového stavu, v ktorom sa vykoná niekoľko základných akcií na najnižšej úrovni s nejakým účelom. Momentálne sú v tíme DAInamite implementované nasledujúce stavy:

- *BeforeKickOffState* – v tomto stave sa hráči nastavia na svoje počiatočné pozície na ihrisku. Vyskytuje sa len v režime `before_kick_off`.
- *DribbleState* – stav, v ktorom hráč má loptu a zisťuje, kade a akou rýchlosťou driblovať, na ktorej strane si má držať loptu a pod.
- *GoalieKickInState* – v tomto stave sa nachádza bránkar po chytení lopty a rozhoduje o svojich ďalších akciách (kam vykopne, kam sa postaví).
- *GoalKeeperState* – v tomto stave sa nachádza brankár, keď nemá loptu. Rozhoduje sa, kam sa postaví, aby chytil strelu a kedy ju má chytať.
- *InterceptBallState* – stav, v ktorom hráč nemá dosah na loptu a snaží sa ju čo najrýchlejšie dostihnúť.
- *InterceptOpponentState* – stav, v ktorom sa nachádza hráč, keď nemôže kopnúť do lopty a je najrýchlejším hráčom, čo sa môže dostať k protihráčovi s loptou. Hráč sa snaží dostihnúť protihráča s loptou a zobrať mu ju.
- *PassState* – stav, v ktorom má hráč loptu a môže prihrávať.
- *SearchBallState* – hráč sa dostáva do tohto stavu, keď stratí loptu a opätovne ju hľadá. V takomto stave sú aj hráči, ktorí nemajú nič lepšieho na práci.
- *ShootGoalState* – stav, do ktorého sa dostáva hráč, keď má možnosť vystreliť na bránu.

Trieda *StateEvaluation* prechádza všetky stavy a vyberie najvhodnejší, ktorý sa hodí pre agenta v danej situácii.

## 5.4 Testovanie prototypu

Rozohrávanie brankára a streľbu na bránu sme testovali pozorovaním zápasov. Nevytvorili sme žiaden testovací nástroj, najmä pre zložitosť vyhodnotenia úspešnosti. Jediný jednoduchý spôsob otestovania by bolo spustenie množstva simulácií a štatistické vyhodnotenie úspešnosti. Ani tento spôsob však nepovažujeme za vierohodný.

Časť protoypu týkajúca sa hráča DAInamite predstavuje v podstate testovanie. Tu išlo skôr o zameranie sa na analýzu možností hráča než o systematické testovanie. Stabilitu sme overili množstvom simulácií.

Stabilitu servera verzie 13 sme tiež testovali spustením mnohých simulácií. Odskúšali sme viaceré fakultné tímy z posledných rokov, čím sme zároveň overili spätnú kompatibilitu hráčov i servera. Až na jednu výnimku, server s hráčmi spolupracuje.

## 5.5 Zhodnotenie prototypu

Prototyp sme postavili na dvoch základoch – domáci hráč napísaný v jazyku C++ a zahraničný hráč napísaný v Jave. Dôvodom tohto postupu bola naša nerozhodnosť, či radikálny skok na použitie prostredia Javy je tým správnym. Java je programovacie prostredie, s ktorým ešte RoboCup na fakulte nemá skúsenosti. Nemohli sme – v prípade nevhodnosti javového hráča – riskovať takmer zbytočne venované úsilie tomuto hráčovi, preto sme preskúmali aj hráča napísaného v C++.

Tím Jahodoví princovia (C++) sa javí ako dobrý základ pre ďalšiu prácu. Kód je pomerne prehľadný (žiaľ, slabo okomentovaný) a prípadné zmeny načrtnuté v prototype i tie, ktoré sme len navrhli, dokážeme do hráča zahrnúť. Prekážkou je neskúsenosť väčšiny nášho tímu s programovacím jazykom C++.

Na pokračovanie v hráčovi DAInamite (Java) bolo potrebné zvážiť všetky výhody a nevýhody, ktoré so sebou prináša. Počas analýzy tímu DAInamite sme sa venovali viacerým častiam implementácie. Ako prvé sme testoval rýchlosť simulácie zápasu javových hráčov a hráča napísaného v C++. Testy potvrdili očakávania, že javoví hráči sú nároční na hardvér a celkovo pomalší. Vzhľadom na výkonnosť dnešných počítačov však nevidíme túto vlastnosť pri zápasoch ako nedostatok. Taktika a rozhodovanie hráča sú na dostatočnej úrovni na použitie ako základ pre ďalší vývoj.

Do začiatku letného semestra sme sa potrebovali definitívne rozhodnúť, v ktorom z týchto hráčov budeme pokračovať. Na Jahodových princoch nás odrádzal programovací jazyk, na DAInamite nutnosť počítača so silným procesorom i množstvom pamäte počas vývoja. Agresívne správanie pomocou neurónových sietí sme mali v pláne zahrnúť bez ohľadu na použitého hráča, nakoniec sme sa však k tomu nedostali.

# 6 RIEŠENIE

Po hlasovaní na začiaku letného semestra sme sa nakoniec rozhodli pre pokračovanie v javovom hráčovi DAInamite. Všetky ďalšie kapitoly sa preto zaoberajú jeho správaním a rozširovaním.

## 6.1 Server verzie 13

Jedna z prvých vlastností, ktoré sme na hráčovi odskúšali, bola funkčnosť so serverom verzie 13. Bolo treba zistiť, či s ním spolupracuje, ako na ňom hrá a či je server stabilný.

### 6.1.1 Spolupráca so serverom

Hráč vo svojej pôvodnej podobe nedokázal so serverom fungovať. Po pripojení sa okamžite vypol. Server ani nevypísal jeho pripojenie. Preskúmaním správ poslaných serveru a odpovedí servera sme zistili príčinu. Formát správy *init*, pomocou ktorej sa hráč prihlasuje na server, nevyhovoval. Súčasťou správy je verzia hráča, ktorá má formát desatinného čísla. Staršie verzie servera nekontrolovali formát dostatočne a umožnili hráčovi DAInamite pripojiť sa, aj keď svoju verziu uvádzal ako 9.4.5 (dve desatinné bodky). Server 13 toto odmieta. Posielanú verziu sme uravili na 9.45 a hráč je schopný pripojiť sa na server.

Ani po tejto úprave hráči nedokázali fungovať. Tento raz spadli s výnimkou chybného prístupu do poľa. Príčina chyby bola taká istá ako pri Jahodových princoch: hráči rátali s maximálnym počtom heterotypov 7, server 13 ich štandardne generuje 18. Tento problém sa tiež podarilo odstrániť. Nedokázali sme však spojazdniť priložený monitor *SoccerScope*. Keďže pre fungovanie hráča nie je potrebný (no veľmi pomocný pri ladení), nebádali sme hlbšie. Monitor funguje po nastavení počtu heterotypov na serveri na 7 (súbor player.conf).

### 6.1.2 Hra na serveri

Hráči boli schopní na serveri hrať, no hra bola iná než na starších verziách. Zdalo sa, že hráči ignorujú možnosť prihrávok smerom dopredu. Po dlhých testovaniach a súčasnom vývoji ďalších funkcií (najmä formácií) sme zistili, že problém je vo vlastnostiach daktorých vytvorených heterotypov. Počas zápasu hráči s týmito heterotypmi nedokážu driblovať. Bližším skúmaním správ poslaných na server sme usúdili, že hráči dokážu loptou pohnúť len po vynaložení určitej sily. To potvrdzuje pozorovanie, že keď sa dostanú k lopte, stoja (pričom by mali driblovať), kým k nim nepríde súper, a potom prihrajú. Pri driblovaní sa totiž nesnažia do lopty kopnúť takou silou ako pri prihrávkach. Je podivuhodné, že server pri slabšej sile loptou ani nehne. Nevylučujeme možnosť, že je to chyba servera. Ak je to však úmyselné, hráčov bude treba upraviť na iný štýl hry – tí, čo to nevedia, nebudú driblovať.

Našťastie vďaka zakomponovaniu formácií do nášho hráča (kapitola 6.3) nepredstavuje táto chyba fatálny problém. Hra je hodne lepšia než pred implementáciou

formácií. Žiaľ, zhoršenie je badateľné, preto bude možno treba v budúcnosti chybu odstrániť.

### 6.1.3 Stabilita servera

Veľa simulácií sme spustili aj na serveri 13 a prebehli bezproblémovo. Všetky simulácie však prebiehali na Linuxe a až na konci semestra sme skúsili simulácie na windowsovom serveri. Testovanie odhalilo, že žiadna z verzií 13[7] nie je dostatočne stabilná na seriózne použitie. Zápasy väčšinou nestihnú skončiť a server spadne. Preto odporúčame použiť buď staršiu verziu, buď server pre Linux.

## 6.2 Rozhodovanie hráča, úžitok jednotlivých stavov

Pred implementáciou vlastných stavov a správania je nutné poznať princípy, na základe ktorých sa hráč rozhoduje a vyberá si jednotlivé akcie. DAInamite framework, na základe ktorého je postavený náš hráč, používa koncept úžitku (angl. benefit) situácií. Architektúra a princípy výberu sú dobre opísané v dokumentácii DAInamite v kapitole 10.

Stručne zhrnuté, výber vhodných akcií funguje na princípe stavového automatu. Rozhodovanie prebieha v dvoch fázach: v prvej sú zo zoznamu dostupných stavov odstránené všetky, ktoré nespĺňajú podmienku na spustenie (predpodmienku, angl. precondition). V druhej fáze sa pre každý zo zvyšných stavov vypočíta celková vhodnosť a následne sa vyberie stav s najvyššou dosiahnutou hodnotou. Vhodnosť je počítaná ako násobok úžitku danej akcie a jej pravdepodobnosti úspechu (pričom pod úspechom sa myslí šanca na úspešné vykonanie akcie definovanej daným stavom).

Hodnoty úžitku a pravdepodobnosti nie sú vždy pevne dané, vypočítavajú sa dynamicky pre každý takt. Ich hodnota závisí od aktuálneho stavu na ihrisku, napr. rozostavenia hráčov, pozície lopty, držania lopty, vzdialenosti od najbližšieho protihráča a pod.

Pri vytvorení novej akcie je vhodné poznať, aké približné hodnoty nadobúdajú ostatné stavy. To nám umožňuje vhodne nastaviť vlastné hodnoty tak, aby sme negatívne neovplyvnili proces výberu: pri nevhodne (vysoko) zvolených hodnotách sa môže stať, že ako najvhodnejší bude vyhodnotený náš nový stav namiesto oveľa vhodnejšieho iného.

Z tohto dôvodu sme počas zápasov logovali hodnoty pre stavy, ktoré sa nachádzajú v základnom frameworku a tieto sme následne vyhodnotili (**Tab. 1**, **Obr. 13**).

**Tab. 1.** Hodnoty úžitku rôznych stavov

| BeforeKickOffState | | |
|---|---|---|
| State | Benefit | Probability |
| robocup.component.tactics.BeforeKickOffState | 1 | 1 |

| GoalkeeperState | | |
|---|---|---|
| State | Benefit | Probability |

[7] V čase písania dokumentu bola najnovšia verzia 13.2.1, testovali sme však len po 13.2.0.

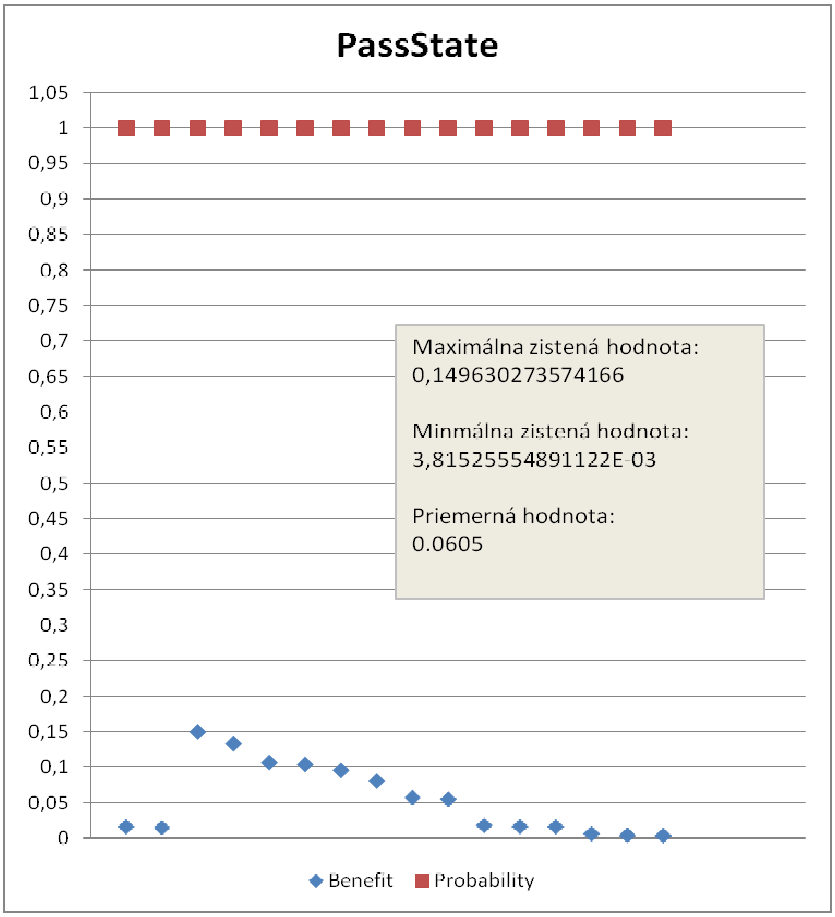
| GoalkeeperState | | |
|---|---|---|
| State | Benefit | Probability |
| robocup.component.tactics.GoalkeeperState | 0,001 | 1 |
| robocup.component.tactics.GoalkeeperState | 1 | 1 |

| SearchBallState & SearchBallWithAttState12 | | |
|---|---|---|
| State | Benefit | Probability |
| robocup.component.tactics.SearchBallState | 0,00001 | 1 |
| robocup.component.tactics.SearchBallState | 0,95 | 1 |
| sk.robocup.dvanastyhrac.tactics.SearchBallWithAttState12 | 0,00001 | 1 |
| sk.robocup.dvanastyhrac.tactics.SearchBallWithAttState12 | 0,95 | 1 |

| StartPositionState | | |
|---|---|---|
| State | Benefit | Probability |
| robocup.component.tactics.StartPositionState | 1 | 1 |

| FormationState12 | | |
|---|---|---|
| State | Benefit | Probability |
| sk.robocup.dvanastyhrac.tactics.FormationState12 | 0,01 | 1 |

| InterceptStates | | |
|---|---|---|
| State | Benefit | Probability |
| sk.robocup.dvanastyhrac.tactics.InterceptBallState12 | 0,95 | 1 |
| sk.robocup.dvanastyhrac.tactics.InterceptOpponentState12 | 0,95 | 1 |
| sk.robocup.dvanastyhrac.tactics.DefensiveInterceptOpponentState12 | 0,95 | 1 |




**Obr. 13.** Hodnoty úžitku rôznych stavov

## 6.3 Formácie a strategická pozícia

Asi najväčší nedostatok pôvodného hráča (voľne dostupnej verzie) bolo, že nedodržiaval žiadne formácie a hráči sa nehýbali, kým lopta nebola v ich blízkosti. Výsledkom bol tím, z ktorého hrali súčasne dva, najviac traja hráči, neschopnosť kombinačných akcií, strelenia gólu ani úspešného bránenia.

Tento problém sme si rozdelili na tri časti:

- Návrh a implementácia viacerých druhov formácií.
- Návrh a implementácia algoritmu, ktorý každému hráčovi povie, na akú pozíciu sa má presunúť.
- Návrh a implementácia nového stavu pre formácie.

### 6.3.1 Návrh a implementácia formácií

Ako prvé sme si vytvorili triedu *PlayerFormationInfo12*, ktorá uchováva informácie o aktuálnej formácii hráča. V tejto triede sme si zadefinovali nasledovné atribúty:

- **Vektor homePosition –** predstavuje vektor pozície hráča podľa aktuálnej formácie.

- **boolean isBehindBall –** ak je nastavená na true, tak daný hráč musí byť za loptou.
- **double maxY –** maximálna možná pozícia hráča na osi Y.
- **double minY –** minimálna možná pozícia hráča na osi Y.
- **double attractionX –** koeficient príťažlivosti k osi X.
- **double attractionY –** koeficient príťažlivosti k osi Y.

Ďalej sme si zadefinovali premennú **FORMATION** typu enum, ktorá obsahuje názvy všetkých formácií. Konštruktor triedy zabezpečí, že pri vytváraní objektu sa priradia aktuálne hodnoty všetkým atribútom.

Ďalším krokom bolo vytvorenie formácií, ktoré sme pridali do existujúcej triedy *FormationData*, ktorá už obsahovala nejaké základné formácie, ktoré sa používali na rozmiestenie hráčov pred rozohrávkou v strede ihriska. Vytvorili sme si *ArrayList formationList*, ktorý obsahuje pole arraylistov jednotlivých formácií. Príklad takého arraylistu:

```
ArrayList<PlayerFormationInfo12> attackList = new ArrayList<PlayerFormationInfo12>();
formationList.add(attackList);
//goalie
attackList.add(new PlayerFormationInfo12(new Vektor(Vektor.XY, 0, -50), false, 0, 0, 0, 0)
// number 2
attackList.add(new PlayerFormationInfo12(new Vektor(Vektor.XY, -10, -16.5), false, 2, -45, 0.25, 0.7));
// number 3
attackList.add(new PlayerFormationInfo12(new Vektor(Vektor.XY, 5, -21), true, 2, -42, 0.2, 0.5));
// number 4
attackList.add(new PlayerFormationInfo12(new Vektor(Vektor.XY, 10, -16.5), true, 2, -47, 0.4, 0.5)) ;
// number 5
attackList.add(new PlayerFormationInfo12(new Vektor(Vektor.XY, -5, -15), false, 2, -45, 0.25, 0.7));
// number 6
attackList.add(new PlayerFormationInfo12(new Vektor(Vektor.XY, -11, 0), false, 20, -26, 0.3, 0.65));
// number 7
attackList.add(new PlayerFormationInfo12(new Vektor(Vektor.XY, 11, 0), false, 20, -26, 0.25, 0.7));
// number 8
attackList.add(new PlayerFormationInfo12(new Vektor(Vektor.XY, 0.5, 20), false, 35, -5, 0.3, 0.5));
// number 9
attackList.add(new PlayerFormationInfo12(new Vektor(Vektor.XY, -0.5, 20), false, 35, -5, 0.25, 0.6));
// number 10
attackList.add(new PlayerFormationInfo12(new Vektor(Vektor.XY, -19, 30), false, 45, -5, 0, 0.7));
// number 11
attackList.add(new PlayerFormationInfo12(new Vektor(Vektor.XY, 19, 30), false, 45,-5, 0, 0.7));
```

Ako môžete vidieť, tento arraylist obsahuje údaje o každom hráčovi v danej formácii. Celkový počet formácií, ktoré sme odskúšali, je osem:

- ATTACK – základná útočná formácia,
- DEFENSE – základná obranná formácia,

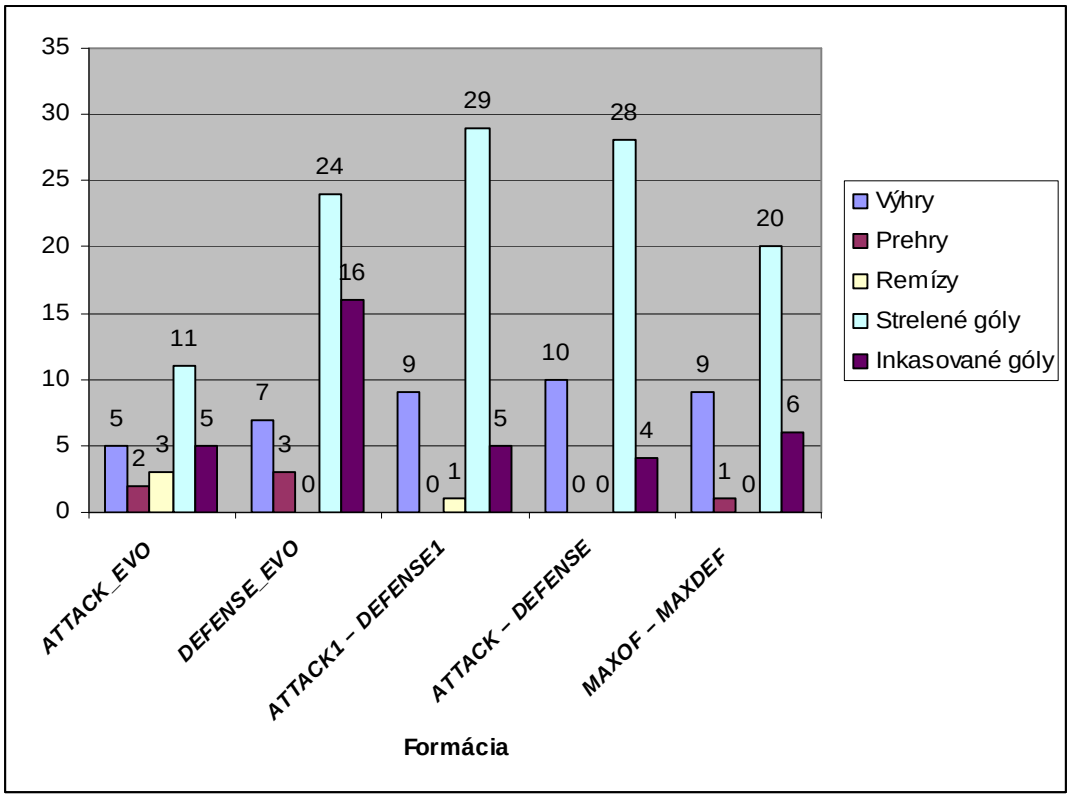
- ATTACK_EVO – iný varianta útočnej formácie prebratý z [1],
- DEFENSE_EVO – iný variant obrannej formácie prebratý z [1],
- MAX_OFFENSIVE – najútočnejšia formácia,
- MAX_DEFENSIVE – najdefenzívnejšia formácia,
- ATTACK1 – vychádza z formácie ATTACK,
- DEFENSE1 – vychádza z formácie DEFENSE.

### *Testovanie formácií*

Aby sme si mohli overiť výhodnosť jednotlivých formácií, museli sme spraviť viacero simulácií. Všetky simulácie prebiehali na výkonnom počítači s operačným systémom Linux a použitý server bol verzie 10.0. Na základe množstva simulácií sme odhaľovali nedostatky, ktoré sme následne opravili. Po finálnych úpravách sme spravili test jednotlivých formácií. Testovacia vzorka pre každú formáciu alebo ich kombináciu predstavovala desať zápasov proti hráčovi Jahodových princov. Výsledky tohto testu sa nachádzajú v tabuľke (**Tab. 2**) a aj vo forme grafu (**Obr. 14**).

**Tab. 2.** Testovanie formácií

| Formácie | Výhry | Prehry | Remízy | Strelené góly | Inkasované góly |
|---|---|---|---|---|---|
| ATTACK_EVO | 5 | 2 | 3 | 11 | 5 |
| DEFENSE_EVO | 7 | 3 | 0 | 24 | 16 |
| ATTACK1 – DEFENSE1 | 9 | 0 | 1 | 29 | 5 |
| ATTACK – DEFENSE | 10 | 0 | 0 | 28 | 4 |
| MAXOF – MAXDEF | 9 | 1 | 0 | 20 | 6 |




**Obr. 14.** Výsledky testovania formácií

### 6.3.2 Algoritmus na získanie strategickej pozície

Tento algoritmus sme prebrali z hráča Jahodových princov. Jeho prepracovaním sme ho úspešne implementovali do nášho hráča. Bolo treba dávať pozor na to, že hráč DAInamite považuje za y-ovú súradnicu smer po dĺžke ihriska (od brány k bráne). Princíp fungovania algoritmu je nasledovný:

- Najskôr sa zistí aktuálna formácia.
- Do x-ovej súradnice sa uloží pozícia vo formácii plus pozícia lopty vynásobená attractionX.
- Do y-ovej súradnice sa uloží pozícia vo formácii plus pozícia lopty vynásobená atractionY.
- Ak *isBehindBall* je true a aktuálna pozícia na y-ovej osi je väčšia ako pozícia lopty, tak nová hodnota sa nastaví na hodnotu y-ovej pozície lopty.
- Ak je aktuálna pozícia na y-ovej osi väčšia ako hodnota maxY, tak sa nastaví na hodnotu maxY. To isté pre minY.
- Návratová hodnota tejto funkcie je nový vektor so strategickou pozíciou.

### 6.3.3 Nový stav pre formácie

Rozhodovanie hráča je založené na stavoch (kapitola 6.1). V jednom takte sa môže vykonávať práve jeden stav, a preto bolo potrebné naimplementovať nový stav, ktorý bude zabezpečovať dodržiavanie formácií (*FormationState12*). Každý stav obsahuje predpodmienku, ktorá určí, či sa daný stav môže vykonať. V našom prípade sa v predpodmienke nachádzajú podmienky, ktoré zabezpečia, aby sa tento stav vykonal iba v prípade, že hráč je ďaleko od lopty a nemá inú vhodnejšiu akciu na vykonanie. Na vyhodnocovanie stavov majú vplyv aj funkcie *successBenefit* a *successProbability*, ktorých návratová hodnota sa musí určiť tak, aby neprevyšovala stavy s vyššou prioritou. Po viacerých testoch sme nastavili tieto hodnoty na 0,01 a 1, čo po vynásobení nám dáva hodnotu 0,01. Táto výsledná hodnota je rozhodujúca pri vyhodnocovaní jednotlivých stavov. V prípade, že sa predpodmienka vyhodnotí ako splnená, nastaví sa aktuálna formácia, ktorá sa má v danom čase dodržiavať. V predpodmienke sa vyhodnocujú aj zmeny formácií počas zápasu, napr. ak je lopta na súperovej polke ihriska, aktuálna formácia sa nastaví na útočnú a naopak. Ak nastane situácia, že náš stav s formáciami je najvhodnejší, vykoná sa hlavná akcia tohto stavu, ktorá pošle hráčovi súradnice strategickej pozície, na ktorú sa má presunúť.

## 6.4 Zameriavanie sa na zvukové správy

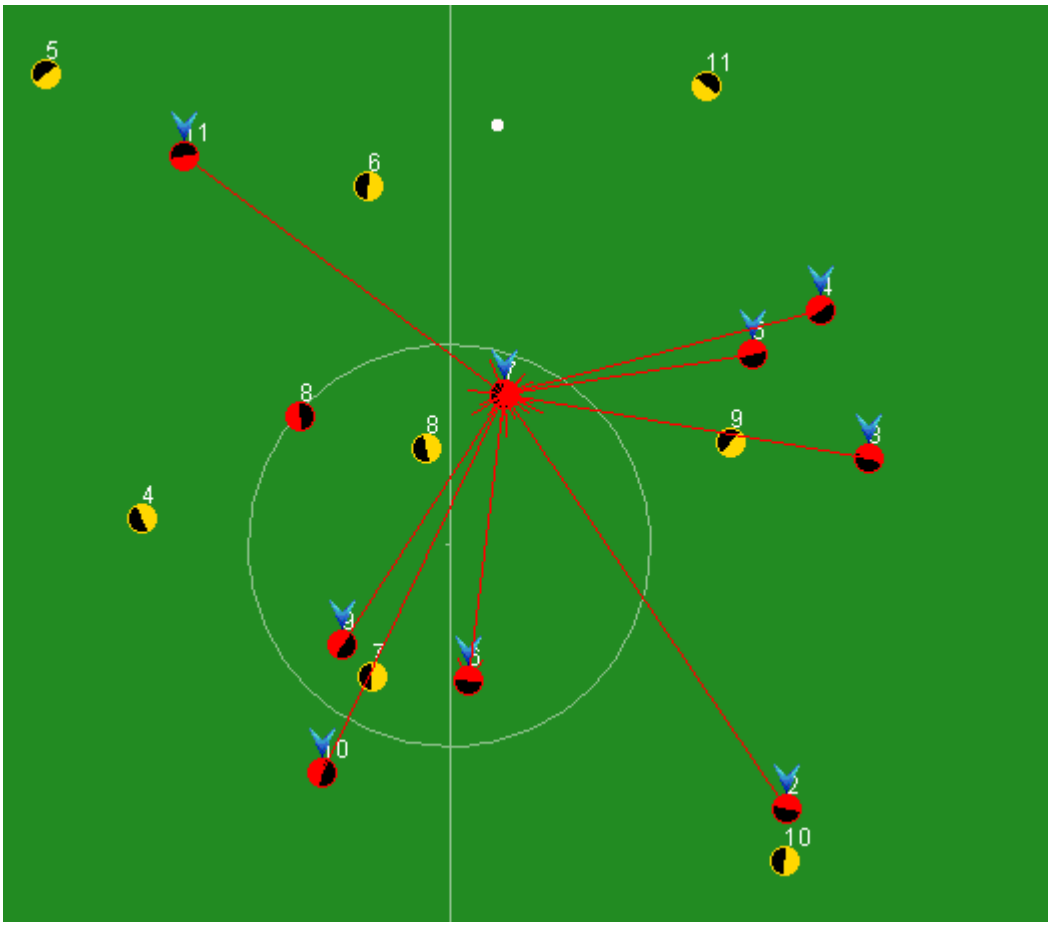
Napriek implementovanej schopnosti posielať správy o pozícii lopty sme pri testovaní zistili, že hráči reagujú neuspokojivo – nepočúvajú daného kričiaceho hráča, ale iných, hlásiacich svoju pozíciu. To malo za následok otáčanie sa iným smerom, ako sa nachádzala lopta, jej hľadanie a tým zbytočné strácanie taktov. Analýzou kódu, zahrnutím pomocných výpisov a sledovaním zápasov sme prišli na to, že hlavný podiel na neefektívnosti našej komunikácie má funkcia zameriavania sa (`Attention to Action`), implementovaná pôvodnými autormi hráča. Zameriavanie sa je jednou z vedľajších akcií, ktorú je možné poslať na server v každom takte. Jej

výsledkom je zameranie sa agenta na jedného špecifického hráča, od ktorého prijme zvukovú správu. Správy od ostatných hráčov na ihrisku ostanú ignorované.

Pôvodná implementácia zameriavania sa pozostávala z cyklického striedania hráčov podľa čísla taktu. Celý algoritmus možno zapísať nasledujúcim spôsobom:

- získaj spoluhráčov,
- zameraj sa na spoluhráča s číslom [číslo taktu modulo 11],
- vyfiltruj – ak som ho počúval už v minulom takte, nastav zameranie sa na null (nenastala zmena, neposielaj znova to isté na server).

Graficky zobraziť, kto od koho prijal v danom takte správu, je možné prostredníctvom monitoru *SoccerScope* dodávaného s hráčom. V pôvodnej implementácii možno vidieť ako napriek tomu, že hráč má loptu a nahráva (pritom kričí pozíciu lopty), hráči počúvajú iného spoluhráč (**Obr. 15**).



**Obr. 15.** Pôvodná implementácia zameriavania sa na hráča

### *Naša implementácia zameriavania sa*

Na zlepšenie výkonu tímu ako celku, rýchlejšie reakcie a zmenšenie času potrebného na nájdene lopty sme sa rozhodli implementovať vlastnú funkciu zameriavania sa na zvukové správy od hráča.

Pri analýze hry sme vychádzali z nasledujúcich premís:

- Pôvodná funkčnosť zabezpečuje cyklické obnovovanie informácií o pozíciách spoluhráčov nezávisle od prijatej zrakovej informácie, je preto žiaduce zachovať ju.
- Správnym výsledkom nami navrhovanej funkčnosti má byť nastavenie pozornosti hráčov, ktorých sa to týka, na hráča s loptou, ktorý bude pravdepodobne prihrávať.
- Hráči, ktorí sú príliš ďaleko alebo loptu nevidia, majú fungovať podľa starého spôsobu (získavať informácie o pozícii spoluhráčov).
- Ak má loptu súper, prípadne ju nemá ani jeden tím, zameriavanie sa by malo fungovať podľa pôvodného algoritmu.

Pôvodný algoritmus je implementovaný v triede `AttentionToActionFactory`, konkrétne v statickej metóde `getAction(WorldModel world)`. Jej návratovou hodnotou je vytvorený objekt akcie na zameranie sa na hráča.

Naša implementácia, v rámci zachovania čistoty kódu, spočívala vo vytvorení vlastnej triedy *AttentionToActionFactory12* dedením od pôvodnej. Trieda obsahuje jedinú statickú metódu `getAttentionToAction(AbstractState state, AttentionToAction ata)`, ktorá vracia akciu na zameranie sa.

Pôvodné herné stavy dedia z triedy `AbstractState`, ktorá používa pôvodný algoritmus výpočtu zameriavania sa. Aby sme mohli používať náš algoritmus a nezasahovali pritom do kódu tímu DAInamite, museli sme vytvoriť pre každý herný stav nový dedením od pôvodného.

Daným spôsobom boli vytvorené tieto triedy:

- `InterceptOpponentState12` – je rozšírením InterceptOpponentState,
- `InterceptBallState12` - je rozšírením InterceptBallState,
- `SearchBallWithAttState12` - je rozšírením SearchBallState,
- `FormationState12` - je novým stavom, dedí od triedy AbstractState.

Nové stavy sú prakticky totožné s pôvodnými, až na prepísanú metódu `calculateAttentionToAction()`, ktorá má nasledujúcu štruktúru:

```
 public void calculateAttentionToAction() {
          this.attentionToAction =
AttentionToActionFactory12.getAttentionToAction(this,
this.attentionToAction);
          //if we got null, compute default
          if (this.attentionToAction == null)
   {
   super.calculateAttentionToAction();
          }
     }
```

V uvedenom kóde je vytvorenie akcie ponechané na statickú funkciu z triedy *AttentionFactory*. Jej návratovou hodnotou je buď hotová akcia, ktorá sa následne zakóduje a pošle serveru, alebo hodnota null. V prípade hodnoty null sa zavolá pôvodná funkcia z rodičovskej triedy fungujúca podľa vyššie opísaného algoritmu.

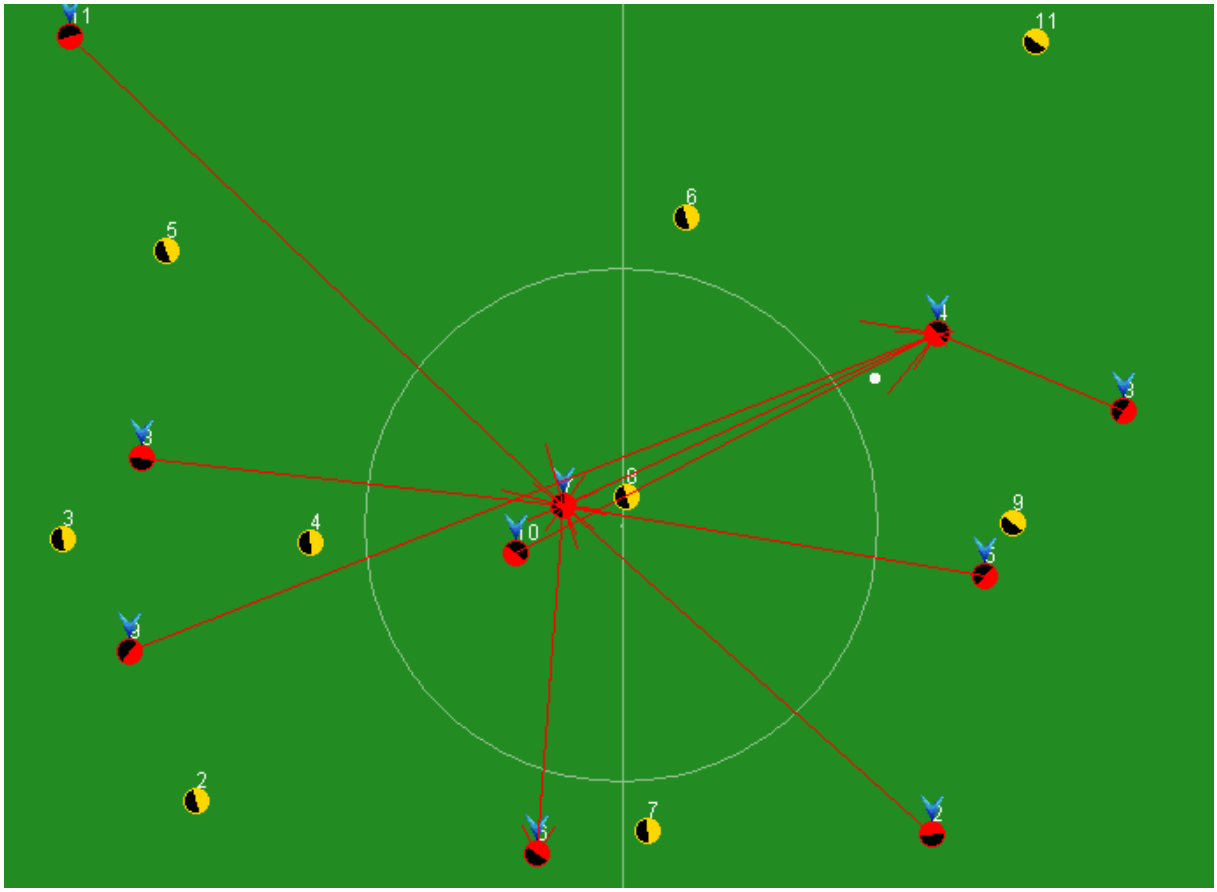
Náš algoritmus funguje podľa nasledujúcej schémy:

- Ak má náš tím loptu a vzdialenosť < MAX_DISTANCE, potom:
  - vráť hráča, ktorý je najbližšie k lopte (ak taký existuje, inak vráť najrýchlejšieho k lopte),
  - zameraj sa na neho (ak som zameraný sám na seba, null).

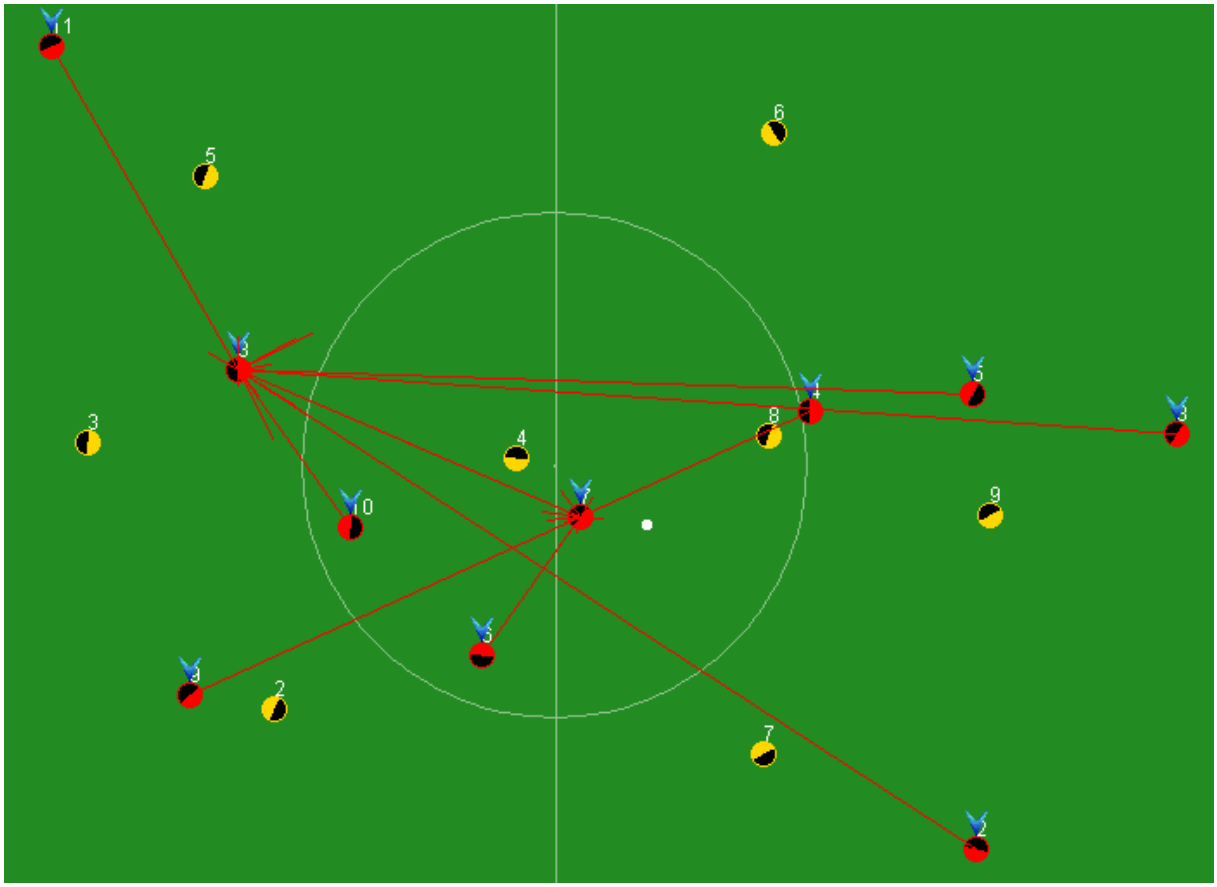
- inak vyber štandardne (ak je lopta od hráča vo väčšej vzdialenosti ako MAX_DISTANCE alebo ju má súper, príp. ju nemá nikto).

Maximálnu vzdialenosť sme testovaním ustálili na hodnote 25 metrov. Zistili sme, že agenti sa naozaj zameriavajú na spoluhráča s loptou aj počas viacerých nasledujúcich cyklov. Algoritmus samotný funguje bez chýb, v priebehu simulácií sa nám však podarilo objaviť viacero nepresností. Hráči sa nie vždy zamerali na spoluhráča s loptou, ale vybrali si iného, prípadne sa na neho zamerali neskoro (napr. až po prihrávke). Dôvodom týchto nepresností je model sveta, ktorým hráči disponujú. Často majú nepresné informácie o pozícii lopty a hráčov, čím je skreslený výsledok rozhodovania. Tieto nepresnosti bohužiaľ nie je možné odstrániť. Nakoniec sme však zistili, že nepresnosť sme čiastočne zavinili sami chybou v kóde pri implementácii formácií. Po jej odstránení sa pozícia lopty v modeli sveta spresnila. Našu implementáciu zameriavania sa považujeme za daných okolností za najlepšiu možnú.

Nasledujúce dva obrázky zobrazujú funkčnosť našej implementácie zameriavania sa. **Obr. 16** zobrazuje hráčov počujúcich správu od hráčov č. 7 (podľa pôvodného algoritmu) a č. 4 (hráč, ku ktorému smeruje prihrávka. Väčšina spoluhráčov vyhodnotila, že bude najbližšie k lopte). **Obr. 17** zobrazuje situáciu o 12 taktov neskôr, po nahrávke hráča č. 4. Hráči s číslami 4, 6 a 9 vyhodnotili, že pri lopte bude ako prvý hráč č. 7, preto od neho počúvajú správy (do momentu prihrávky oznamujúce jeho aktuálnu pozíciu). Hráči fungujúci podľa pôvodného algoritmu sú pritom zameraní na hráča č. 8.



**Obr. 16.** Naša implementácia zameriavania sa (1)



**Obr. 17.** Naša implementácia zameriavania sa (2)

## 6.5 Zvuková komunikácia

Rozhodli sme sa využiť zvukovú komunikáciu medzi hráčmi. Naším cieľom bolo, aby hráč dokázal kričať, kam kope loptu pri prihrávaní. V niektorých situáciách to pomáha jeho spoluhráčom pri reagovaní na prihrávku, lebo presne vedia rýchlosť a pozíciu lopty v momente kopu. Dokážu si vypočítať jej výslednú pozíciu, aj keď loptu v momente prihrávky nevidia. Ďalej je opísaná úloha z pohľadu nahrávajúceho hráča, to znamená kričanie vhodnej správy v cykle nahrávky. Z pohľadu počúvajúcich hráčov toto správanie opisuje kapitola 6.3.

Nové správanie sme naimplementovali v triede *PassStateWithSay12* (číslo 12 sme zvolili na odlíšenie našich tried od pôvodného frameworku), ktorá dedí od pôvodnej triedy *PassState*. Pôvodný stav je rozšírený o volanie metódy *calculateSayAction()*, ktorá zavolá triedu *SayActionFactory12* zodpovednú za vytvorenie korektnej textovej správy o lopte. V *SayActionFactory12* sa využíva trieda *MessageFactory*, ktorá umožňuje kódovanie a posielanie rôznych správ hráčmi pri vykonávaní akcií. Podrobný opis fungovania tejto triedy je možné nájsť v dokumentácií k frameworku DAInamite. Metóda vráti textovú správu, v ktorej je zakódovaná kopnutá lopta so svojimi vlastnosťami v momente kopu. Hráč, ktorý počuje túto správu, si ju automaticky dekóduje a informácie uloží do svojho vnútorného sveta. Framework zabezpečuje, aby sa informácie z počutých správ neprepísali zrakovými informáciami.

Podobným spôsobom sme neskôr upravili aj driblovací stav (trieda *DribbleState*) a vytvorili sme nový *DribbleStateWithSay12*, v ktorom hráč kričí svoju pozíciu počas driblovania. Okolití hráči tak dostávajú presné informácie o pozícii hráča s loptou vždy, keď túto správu počujú. Z tej si sami vypočítavajú aj pozíciu lopty, ktorá musí byť zákonite blízko pri počutom hráčovi.

Po dokončení zvukovej komunikácie (posielanie aj prijímanie správ) sme sa presvedčili, že nový systém dobre funguje a umožňuje získavať ostatným hráčom informácie o zmene polohy lopty, aj keď sa na loptu vôbec nepozerajú. Veľmi dobre sa dal pri testovaní využiť monitor *SoccerScope*, ktorý je súčasťou frameworku a umožňuje sledovať vnútorný svet hráčov ako aj vyslané a počuté správy.

## 6.6 Obranná hra

Ďalším typom správania, ktoré sme v hráčovi implementovali, bolo zvýšenie agresivity obrancov, ak sa lopta nachádza v blízkosti našej šestnástky. Nutnosť zlepšiť obrancov vznikla z dôvodu, že často sa stávalo, že naši obrancovia ustupovali pred súperovými útočníkmi až hlboko do šestnástky. Chceli sa presunúť na svoje pozície, ktoré im vyplývali z používania formácií, a často nestíhali už potom vhodne reagovať na súpera, ktorý ohrozoval našu bránu.

Vytvorili sme preto stav *DefensiveInterceptOpponentState12*, ktorý vychádza z pôvodného *InterceptOpponentState*, ale načítava sa len pre obrancov. Tento aj pôvodný stav sú zodpovedné za napádanie súperiaceho hráča s loptou. Pri uplatnení tohto stavu hráč napadne súpera a pokúsi sa mu vziať loptu. Pôvodnú podmienku, ktorá spúšťala stav len v prípade, že hráč je od protihráča vo vzdialenosti 5 metrov (naša prvá verzia), sme upravili tak, aby hráč – ak je to obranca a lopta sa nachádza v našej šestnástke – reagoval na vzdialenosť 10 metrov. V opačnom prípade sa hráč riadi pôvodnou podmienkou. Neskôr sa ukázalo, že podmienka so šestnástkou nie je dostatočná, lebo keď sa súper dostal do našej šestnástky, bolo už na prípadné reakcie väčšinou neskoro. Preto sme ju zmenili a vytvorili nový priestor, ktorý sme nazvali *DangerArea* a pokrýva priestor šestnástky + 5 metrov (21 metrov v okolí brány). Nový priestor je implementovaný v pôvodnej triede *WorldModel*. Pribudli sem aj metódy, pomocou ktorých zisťujeme, či sa nejaký objekt nachádza v tomto priestore (metóda *inOwnDangerArea()*).

S použitím tohto správania obrancovia reagujú na súpera omnoho skôr a obranná efektivita sa tak v kombinácii s formáciami výrazne zlepšila.

## 6.7 Rozohrávanie štandardných situácií

Pri hre hráča v zápasoch sme si všimli, že v niektorých situáciách má hráč problémy s rozohrávaním štandardných situácií, hlavne s autovým vhadzovaním. Analýzou kódu sme dospeli k záveru, že v existujúcom kóde chýba správanie, ktoré by malo na starosti výber vhodnej akcie pri daných herných módoch. V prípade úspešného rozohrania sa aktivoval stav *PassState*, v prípade neúspechu správanie *SearchBall*, keď sa hráč len obzeral okolo seba a stál na mieste.

Rozhodli sme sa preto vytvoriť správanie, ktoré by sa aktivovalo pri každom rozohrávaní lopty naším tímom. Medzi štandardné situácie, v ktorých malo byť správanie spustené, patria tieto režimy hry (názvy podľa kódu tímu DAInamite):

FREE_KICK_OWN, INDIRECT_FREE_KICK_OWN, CORNER_KICK_OWN, GOAL_KICK_OWN, KICK_IN_OWN, KICK_OFF_OWN.

Akciu rozohrávania sme rozdelili na 2 fázy:

1. fáza čakania,
2. fáza rozohrávania (prihrávky).

### 6.7.1 Fáza čakania

Fáza čakania (tzv. stav *WaitAndLookState*) má za úlohu po pribehnutí k lopte čakať nastaviteľný počet taktov (10), počas ktorých sa hráč obzerá okolo seba a aktualizuje svoje údaje o pozíciách spolu- a protihráčov.

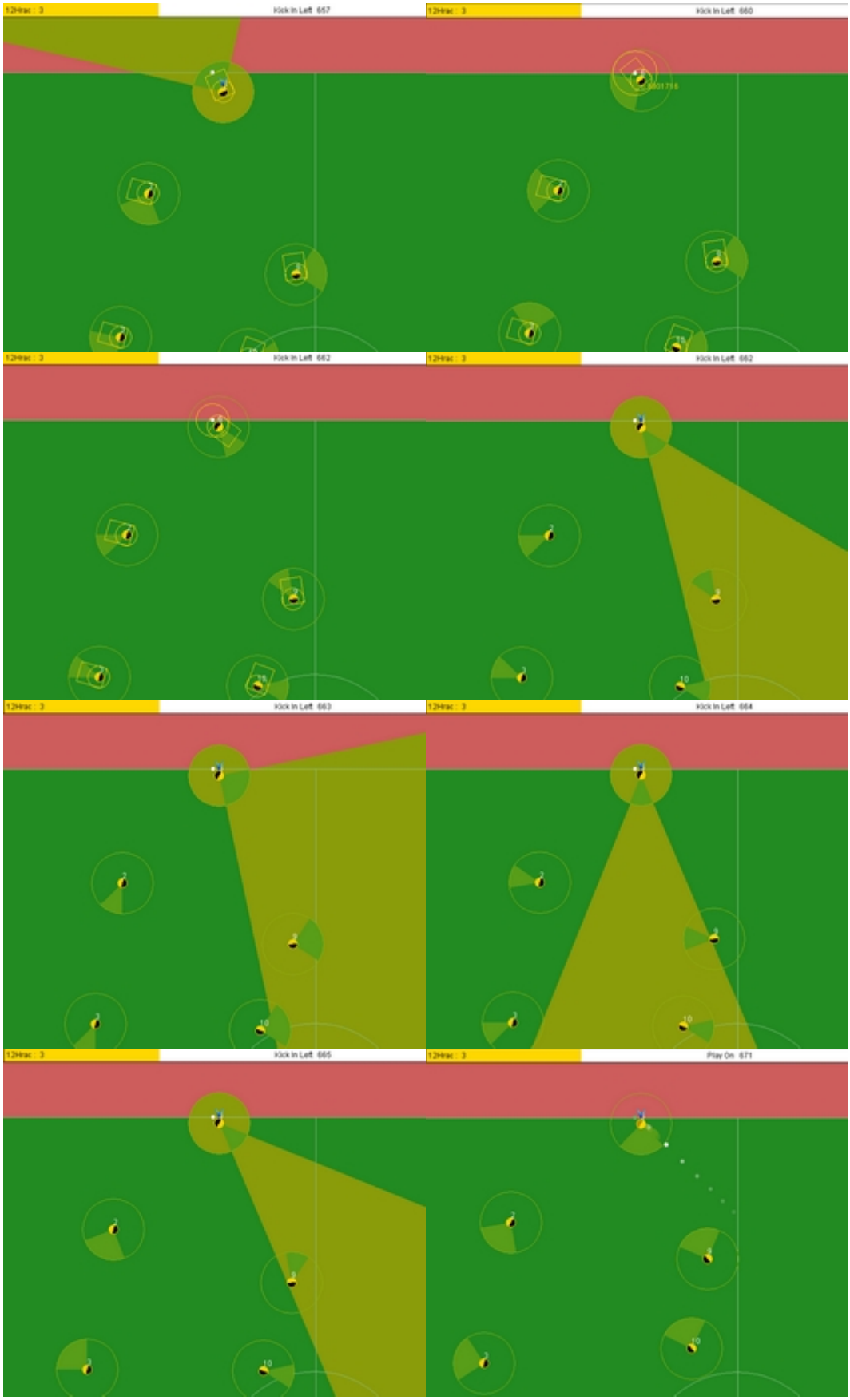
Do implementácie tohto čakacieho stavu bola akcia hráča priamočiara: hráč pribehol k lopte a okamžite rozohral, často nepresne. Vizualizáciou jeho modelu sveta sme zistili, že hráč má o pozíciách ostatných hráčov pomerne staré informácie, ich poloha sa často výrazne líšila od aktuálnej. *WaitAndLookState* má za úlohu túto nepresnosť minimalizovať. Pozície ostatných hráčov sa po odkopnutí autu menia minimálne, hráči buď stoja, alebo sa pomaly vracajú na svoje preferované miesta na ihrisku. Rozohrávajúci hráč má preto po prezretí ihriska čerstvejšie a presnejšie informácie a jeho schopnosť prihrávky sa zlepšila.

*WaitAndLookState* zabezpečuje taktiež natočenie hráča smerom do ihriska. Zaznamenali sme prípady, keď hráč rozohrávajúci päťku dobehol k lopte otočený k našej bráne a v takejto pozícii začal s obzeraním sa po ihrisku. Ak počas hry nastane podobný prípad a uhol hráča je väčší ako 135° alebo menší ako -135°, hráč sa automaticky otočí o požadovaný uhol **do** ihriska.

### 6.7.2 Fáza rozohrávania

Fáza rozohrávania je implementovaná v stave *KickInStateWithSay*, rozširujúcom stav *PassStateWithSay*. Jeho úlohou je prihrať najlepšie postavenému spoluhráčovi, čo zabezpečuje metóda *getPassAction()* z triedy *Prophet*. V prípade, že sa nepodarí nájsť vhodnú prihrávku, hráč má za úlohu odkopnúť loptu do voľného priestoru. Pokiaľ sa nachádzame na našej hernej polovici, hráč odkopáva loptu popri aute najväčšou možnou silou, aby minimalizoval možnosť zachytenia lopty súperom. Pokiaľ rozohrávame na súperovej strane a nemáme komu nahrať, hráč odkopáva loptu smerom na bránu. Vtedy je šanca že buď padne gól, alebo sa nám podarí zachytiť (dobehnúť) loptu a začať útočnú akciu.

Nasledujúca sekvencia obrázkov (**Obr. 18**) zobrazuje beh hráča číslo 6 za loptou v aute, rozhliadanie sa po ihrisku a následne vykoná prihrávku smerom k súperovej bránke. Medzi 2. a 3. obrázkom v rade je možné vidieť otočenie hráča smerom do ihriska.



**Obr. 18.** Rozohrávanie autu

## 6.8 Ďalšie menšie úlohy

Okrem opísaných komplexnejších úloh sme pracovali aj na iných úlohách:

*Analyzovanie problému s hromadením sa dát v pamäti* pri spustení hráča spolu s monitorom – tento jav nie je chybou. Monitor – ak je tak nastavený – zaznamenáva priebeh zápasu do pamäte a umožňuje potom tento záznam uložiť do súboru. Záznam môže obsahovať aj vnútorný model sveta každého hráča, preto jeho veľkosť nie je zanedbateľná.

*Výbere akcie na základe počutej správy* – pri implementácií zvukovej komunikácie sme sa rozhodovali, či vykonávať určité akcie aj po získaní zvukovej správy hráčom. Momentálne sa vyhodnocovanie spúšťa len po získaní zrakovej informácie, ale berú sa do úvahy aj informácie zo zvukovej správy, preto sme usúdili, že nie je potrebné aby sa svet vyhodnocoval po počutej správe.

Prerobili sme pôvodný systém načítavania stavov len výhradne z balíka *robocup.component.tactics* tak, aby sa stavy dali načítavať z ľubovoľného balíka v classpath. Zmeny sa týkali načítavania stavov z konfiguračného súboru.

Zisťovali sme poradie, v akom sa prijímajú obrazové a zvukové informácie v takte. Ako prvé sa prijíma informácia *SenseBody*, potom informácia *Hear* a posledná je vizuálna informácia.

# 7 TESTOVANIE

V tejto kapitole sa venujeme testovaniu hráča. Pokúšame sa zistiť, či nami navrhnuté zlepšenia sú funkčné a či zároveň predstavujú skutočné zlepšenie oproti pôvodnému hráčovi DAInamite.

## 7.1 Spôsob testovania

Ako už z charakteru projektu vyplýva, nie je možné použiť klasické spôsoby testovania. Z tohto dôvodu sa za najvhodnejší spôsob testovania považuje porovnanie nášho novovytvoreného hráča s predchádzajúcim hráčom. Lenže pôvodný hráč nemal prakticky žiaden pohyb (okrem hráčov blízko lopty), takže výsledky by boli takmer bezpredmetné (náš hráč by vždy vyhral). Preto sme sa rozhodli testovať hráča simuláciami proti Jahodovým princom. Na simuláciách sme pozorovali výsledky našich zmien. Mnohé zmeny sa nedali „zmerať“ a bolo ich možné pozorovať len na štýle hry.

Výsledky simulácií testovania formácií možno vidieť v kapitole 6.3. Ostatné návrhy sme overovali pozorovaním hry a vnútorného sveta hráčov (pomocou monitora priloženom pri hráčovi). Kvôli kompletnosti uvádzame tabuľky s výsledkami zápasov proti Jahodovým princom pred zakomponovaním formácií (**Tab. 3**) a po ňom (**Tab. 4**).

**Tab. 3.** Zápasy pôvodného hráča proti Jahodovým princom

| zápas | DAInamite | Jahodoví princovia |
|---|---|---|
| 1 | 1 | 4 |
| 2 | 0 | 0 |
| 3 | 2 | 5 |
| 4 | 0 | 2 |
| 5 | 1 | 5 |
| 6 | 0 | 3 |
| 7 | 0 | 4 |
| 8 | 2 | 2 |
| 9 | 1 | 5 |
| 10 | 0 | 3 |

**Tab. 4.** Zápasy nášho hráča proti Jahodovým princom

| zápas | Dvanásty hráč | Jahodoví princovia |
|---|---|---|
| 1 | 4 | 0 |
| 2 | 5 | 1 |
| 3 | 6 | 0 |
| 4 | 4 | 1 |
| 5 | 7 | 2 |
| 6 | 3 | 0 |
| 7 | 6 | 1 |

| | | |
|---|---|---|
| 8 | 4 | 2 |
| 9 | 5 | 2 |
| 10 | 6 | 1 |

Výsledky naznačujú, že formácie výrazne zlepšili hru nášho hráča. Vidieť to na prvý pohľad pri zápase. Počas semestra sme vytvárali mnoho simulácií. Týkali sa overenia, či nejaká zmena viedla k zlepšieniu alebo k zhoršeniu, a testovania rôznych verzií servera. Často výsledky zápasov neposkytovali rozumnú mierku úspešnosti. Potom sme sa uchýlili k subjektívnemu hodnoteniu úspechu na základe pozorovania štýlu hry.

# 8 ZHODNOTENIE

Náš tím priniesol na fakultu ako prvý hráča napísaného v Jave. Tento krok považujeme za veľmi dôležitý pre budúcnosť RoboCupu na fakulte, pretože v súčasnosti čoraz menej študentov ovláda jazyk C++. Zmena jazyka by však nebola samo osebe veľmi prospešná, ak by hráč nebol schopný konkurovať ostatným tímom fakulty. Hráč DAInamite, ktorého sme si zvolili za základ, je viac než konkurencieschopný a okrem veľmi dobre štruktúrovaný, takže realizácia nových nápadov je veľmi jednoduchá a nezasahuje do pôvodného kódu. Túto vlastnosť si na ňom ceníme najviac.

Hráča sme počas druhého semestra obohatili o viaceré nápady. Najpodstatnejším z nich je dodržiavanie formácií, bez ktorého by sme mali malú šancu uspieť. Plány z prvého semestra (menovite neurónová sieť na agresívne správanie) sme síce nesplnili, no nepovažovali sme to za reálne, pretože podstatnejšie bolo dôkladne sa oboznámiť s novým hráčom a vykonať malé, ale podstatné úpravy.

Práca na tímovom projekte nám priniesla predovšetkým skúsenosti z oblasti vývoja v tíme. Z tohto hľadiska si všetci chválime najmä vedené zápisnice, ktoré nám umožnili kontrolovať naše úlohy. Z technológií, ktoré sme používali (SVN, Maven), sme naštudovali len to, čo sme potrebovali, no neľutujeme to, lebo sme tak mali možnosť hlbšie nazrieť do sveta simulácie robotického futbalu a získať prehľad o multiagentových systémoch. Veľmi dobrý pocit sme získali najmä zistením, že práca na hráčovi viedla k zlepšeniu.

Možností v oblasti RoboCupu je také množstvo, že bude pokračovať ešte dlhé roky. Fakultné tímy výrazne zaostávajú za svetovými. Poteší nás, ak po rokoch zistíme, že aj vďaka nám sa situácia zlepšila. Našim pokračovateľom prajeme veľa úspechov a veríme, že ich bude projekt baviť aspoň tak, ako nás. Ak by nevedeli, čo sa oplatí ďalej implementovať, odporúčame pozrieť nepriradené úlohy zo zápisnice č. 20 (dostupná na našej stránke alebo v dokumentácii k riadeniu).

# POUŽITÁ LITERATÚRA


1. Oficiálna stránka RoboCupu.
   http://www.robocup.org
2. Amin Milani Fard et al.: *Nexus 2D 2008 Team Description*, IEEE Latin American Robotics Competitions - RoboCup Brazil Open 2008, Oct, 2008, Rio de Janeiro, Brazil.
   http://nexus.um.ac.ir/Nexus-2d-2008.pdf
3. Endert et al.: *DAInamite 2008 Team Description*. In Proceedings RoboCup 2008, Suzhou.
   http://www.dainamite.de/fileadmin/Dainamite-Dateien/Papers/DainamiteTDP2008.pdf
4. Marian Sebastian: *OXSY 2006 Team Description*, OXygen-SYstems laboratory, Str. Constantin Noica, Bl.5, Sc.C, Ap.36, C.P. 550169, Sibiu, ROMANIA.
   http://navid.alamati.googlepages.com/Oxsy2006TeamDescription.pdf
5. Riedmiller M. et al.: *Brainstormers 2D — Team Description 2008*, Institute of Cognitive Science, Universität Osnabruck.
   www.ni.uni-osnabrueck.de/fileadmin/user_upload/publications/riedmiller.gabel.trost.bs08tdp.pdf
6. Ladislav Borženský et al.: *RoboCup S – nové stratégie*, Bratislava: FIIT STU, 2008.
   http://labss2.fiit.stuba.sk/TeamProject/2007/team16is-si/dokumenty/odovzdane2/Implementacia.pdf
7. Peter Cséfalvay et al.: *RoboCup – nové stratégie*, Bratislava, FIIT STU, 2007.
   http://labss2.fiit.stuba.sk/TeamProject/2006/team08/public_html/docs/technicka_dokumentacia.doc
8. Serguei Gorbatchev et al.: *RoboCup – simulácia robotického futbalu*, Bratislava, FIIT STU, 2004.
   http://labss2.fiit.stuba.sk/TeamProject/2003/team08/download/doc/final04/v_software_final.pdf
9. Ľubomír Hromádka et al.: *RoboCup – nové stratégie*, Bratislava, FIIT STU, 2006.
   http://labss2.fiit.stuba.sk/TeamProject/2005/team02/DokumentaciaFinal.pdf
10. Peter Kajsa et al.: *RoboCup – nové stratégie*, Bratislava, FIIT STU, 2007.
    http://labss2.fiit.stuba.sk/TeamProject/2007/team06is-si/projektova_dokumnetacia-UTTP_a.doc
11. Riedmiller M. et al.: *Brainstormers 2D — A Case Study on Improving Defense Behavior in Soccer Simulation 2D*, Institute of Cognitive Science, Universität Osnabruck.
    www.ni.uos.de/fileadmin/user_upload/publications/gabel.riedml.trost.robocupsymposium08.pdf
12. Florentin Woergoetter and Bernd Porr: *Reinforcement learning*, Scholarpedia, 3(3):1448, 2008.
    http://www.scholarpedia.org/article/Reinforcement_learning

# Príloha A
# Používateľská príručka k produktu

# Slovenská technická univerzita v Bratislave

**FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ**

Ilkovičova 3, 812 19 Bratislava


---

## Tvorba softvérového systému v tíme

# RoboCup – nové stratégie

(Používateľská príručka k produktu)

---


**Tím č. 12 – 12. hráč**

Bc. Juraj Ligocký, Bc. Michal Hrubý, Bc. Gabriel Pán
Bc. Vladimír Oravec, Bc. Ján Hric, Bc. Marek Polák
Pedagogický vedúci: Ing. Ivan Kapustík



2008/2009

A-1

# POUŽÍVATEĽSKÁ PRÍRUČKA

### *Úvod*

Tento dokument predstavuje používateľskú príručku k produktu. V jednotlivých častiach opisuje produkt, hardvérové a softvérové nároky, postup ako nainštalovať produkt a spôsob, ako sa produkt používa.

### *Hardvérové nároky*

Minimálne hardvérové nároky na produkt sú pomerne vysoké. Odporúčame použiť stolný počítač (desktop), nie laptop, s procesorom 2 GHz a pamäťou 2 GB. Pre činnosť produktu je dôležitý plynný chod, preto je vhodné použiť silnejšiu konfiguráciu, než je minimálna. Navyše, program dokáže efektívne využiť aj lepší procesor. Po kompilácii by mal byť schopný bežať aj na iných typoch počítačov než PC, nie je to však odskúšané.

### *Softvérové požiadavky*

Produkt možno spustiť v ľubovoľnom operačnom systéme, pod ktorým pracuje virtuálny stroj Javy (Java Virtual Machine).

### *Inštalácia produktu*

Produkt vyžaduje na svoju činnosť ďalšie programy. Tie sú dodané spolu s ním. V prvom rade je to *server*, ktorý slúži na simuláciu zápasu. Dodané sú zdrojové kódy verzie 13.0.1, binárna verzia 13.0.1 pre operačný systém Windows 32-bitový a binárna verzia 12.1.3 pre operačný systém Linux 64-bitový (odskúšané na distribúciách Fedora a Mandriva). Na spustenie a sledovanie zápasu je potrebný *monitor*. Dodané sú verzie pre operačný systém Linux 64-bitový a verzia pre operačný systém Windows 32-bitový. Na sledovanie zápasov z logovacích súborov servera (teda nie práve prebiehajúcich) slúži priložený program *LogPlayer*.

Inštalácia týchto programov v operačnom systéme Linux spočíva v použití príkazu rpm na inštaláciu balíkov, t. j. súborov s príponou RPM. V operačnom systéme Windows inštalácia pozostáva z rozbalenia súborov s príponou ZIP, každý do vlastného adresára.

Produkt vyžaduje nainštalovanú Javu verzie 1.5 a vyššie. Sú priložené inštalátory pre operačný systém Linux a Windows.

Inštalácia produktu pozostáva len zo skopírovania súborov do vlastného adresára.

### *Spustenie simulácie*

Spustenie simulácie pozostáva z týchto krokov:

- Spustenie servera – tento krok musí byť prvý. Spustiteľný súbor má názov *rcssserver* v Linuxe a *rcssserver.exe* vo Windowse. Server vytvára logovacie súbory v adresári, z ktorého sa spustil.
- Spustenie monitora – použije sa príkaz *rcssmonitor* v Linuxe a *Soccermonitor.exe* vo Windowse. Windowsový monitor vyžaduje navyše ručné pripojenie k serveru napr. pomocou F2. Možno špecifikovať adresu a port servera. Pre lokálny server necháme štandardné hodnoty (127.0.0.1 a 6000).

- Pripojenie hráčov na server – k produktu je dostupný súbor dvanastyhrac.jar. Nastavíme sa do priečinka, v ktorom sa súbor nachádza. V tomto priečinku musia byť súbory *.conf. Použijeme príkaz: `java -jar dvanastyhrac.jar`. Jeho spustením sa všetci hráči aj kouč pripoja na lokálny server. Možno použiť parameter –conf x na určenie, že hráč má načítať konfiguráciu zo súboru x. Na server je možné pripojiť hráča iného tímu.
- Spustenie polčasu – hráči začnú hrať. Tento krok treba vykonať až po úplnom pripojení všetkých hráčov. V linuxovom monitore na to slúži tlačidlo s nápisom „kick off“, vo windowsovom sa to dá vykonať napr. pomocou F5. Štandardne po skončení prvého polčasu načim vykonať to isté.
- Skončenie simulácie pozostáva z korektného vypnutia servera. V okne konzoly, kde je server spustený, sa to vykoná stlačením Ctrl+C. Hráči sa postupne sami vypnú, monitor vypneme klasickým zavretím okna.

# Príloha B
# Spustenie hráča zo zdrojových kódov

### *Spustenie hráča (frameworku) DAInamite*

Vo všeobecnosti možno povedať, že spustenie Java hráča zo zdrojových kódov nie je triviálnou úlohou. V tejto kapitole uvádzame pomerne podrobný postup, keďže sami sme mali problém s niektorými bodmi – hlavne týkajúcimi sa použitia pre nás nových technológií (SVN, Maven).

Najjednoduchší spôsob, ako importovať projekt, je stiahnuť ho kompletný z SVN. Druhou možnosťou je stiahnuť kompletný projekt ako archív a rozbaliť ho vo vhodnom adresári na disku. Keďže ide o projekt zostavovaný v Mavene, odporúčame mať Maven nainštalovaný a správne nakonfigurovaný. Ako klienta SVN sa nám v prostredí Windows osvedčil open-source nástroj TortoiseSVN[8] alebo špecializované moduly do vývojových prostredí (Eclipse, NetBeans).

**Postup pre Eclipse IDE:**

1. Stiahneme archív so zdrojovými súbormi.
2. Otvoríme Eclipse.
3. Naimportujeme – File/Import/“Existing project into workspace“ - nájdeme cestu k archívu, klikneme na *finish* a projekt je automaticky importovaný. V *Project Exploreri* si môžeme všimnúť kompletný robocup projekt.

**Spustenie hráča v prostredí Eclipse:**

1. nájdeme triedu RoboupAgent v „Project Exploreri / pravý klik / run as / run configurations“,
2. do "program argumentov" vložíme: `MenoTimu localhost goalie.conf defender.conf defender.conf defender.conf defender.conf player.conf player.conf player.conf player.conf player.conf player.conf coach.conf,`
3. do "VM argumentov" môžeme/nemusíme: -Xmx512M ,
4. do "working directory" - ${workspace_loc:Robocup/etc/agents} ...vďaka tomu Eclipse nájde cestu ku konfiguračným súborom
5. dáme apply/ run. Pokiaľ nemáme spustený Robocup server, odozvou hráča bude množstvo chybových hlásení,
6. stiahneme server (9.4.5, 13.0.2) zo stránky RoboCupu,
7. rozbalíme a spustime ho,
8. opäť spustíme Robocup hráča, konfigurácia je už našťastie uložená v Eclipse, takže len spustíme hráčov príkazom "Run",
9. Na obrazovke by sa mal objaviť monitor, po stlačení „kick off“ začínajú hráči hrať.

**Import do NetBeansu**

1. File / Import Project / Eclipse Project -> Import project ignoring dependencies...zadáme cestu k rozbalenému archívu so zdrojovými súbormi a ľubovoľný cieľový adresár (destination folder).

[8] http://tortoisesvn.net/

2. Ak máme nainštalovaný v operačnom systéme Maven a v IDE plugin, ktorý umožňuje pracovať s ním, bod 1 sa zmení na: File / Open Project -> vyberiete adresár, NetBeans by mal automaticky rozpoznať projekt vytvorený v Mavene.
3. Po importe zmeníme konfiguráciu projektu na:
    - Main class: robocup.component.RobocupAgent,
    - Arguments: - ako v Eclipse (bod 2),
    - VM options - ako v Eclipse (bod 3),
    - Working directory: [cesta ku zdrojakom]\etc\agents.
4. Stiahneme a spustime server, následne spustíme samotný projekt.
5. Na obrazovke by sa mal objaviť monitor, po stlačení „kick off“ začínajú hráči hrať.

Najdôležitejšou pomôckou pri pochopení kódu tímu DAInamite okrem vložených komentárov, je dokumentácia k ich frame-worku. Stiahnuť ju je možné napr. na stránke nášho tímu. Spustenie hráča zo zdrojových kódov je podrobne opísané v kapitole číslo 2.4.

# Príloha C
# Dátové médium s prototypom a dokumentáciou