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

Tímový projekt – RoboCup 3D

Dokumentácia k riadeniu projektu

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

Obsah

1 Úvod

Tento dokument obsahuje zoznam kompetencií tímu RFC Megatroll, dlhodobé rozdelenie úloh. Taktiež obsahuje zoznam podielov autorstva na dokumentácií k riadeniu ale i k inžinierskemu dielu.

Prvá kapitola pojednáva o ponuke nášho tímu a schopnostiach tímu. Druhá kapitola slúži pre prehľad o podiele autorstva v dokumentácií. V ďalších kapitolách sú spísané nami používané metodiky, komunikačné prostriedky. Posledná kapitola obsahuje metodiky vytvorené jednotlivými členmi tímu.

2 Ponuka tímu

2.1 Predstavenie tímu

Spoločné Znalosti: HTML,CSS,Java,UML,XML

Adámik Martin (Absolvent FIIT STU)

- Technológie: JQuery, PHP, SQL
- Práca v tíme: školské projekty

Bádal Matej (Absolvent FIIT STU)

- Technológie: JQuery, PHP (fw. Zend 2, Nette základy), Android, SQL
- Práca v tíme: niekoľko mesiacov (Medium13, s.r.o.)

Benkovič Samuel (Absolvent FIIT STU)

- Technológie: ASP.NET, CMS Umbraco, SQL
- Práca v tíme: niekoľko mesiacov (Sféra, a.s.)

Bošiak Vladimír (Absolvent FIIT STU)

- Technológie: JQuery, PHP (fw. Zend 1,2), Xpath, SOAP, OWL, SQL
- Práca v tíme: viac ako rok (Medium 13, s.r.o.)

Čerešňák Michal (Absolvent VUT FIT)

- Technológie: C#, SQL
- Práca v tíme: školské projekty (5 členov)

Homola Igor (Absolvent FI PEVŠ)

- Technológie: Administrácia Linux, Windows
- Práca v tíme: viac ako rok

Petráš Michal (Absolvent FIIT STU)

- Technológie: JQuery, PHP (fw. Zend), Návrh a anlýza IS, SQL
- Práca v tíme: viac ako rok (Eby, s.r.o.)

2.2 Preferencia tém

2.2.1 Virtuálna FIIT na mobile

V poslednej dobe narastá dopyt po mobilných aplikáciách z dôvodu rozšírenosti mobilných zariadení pre bežných používateľov. Z týchto dôvodov si myslíme, že virtuálna FIIT na mobile poskytne množstvo informácií jednoduchým spôsobom pre používateľov smartphone-ov. Táto mobilná aplikácia hlavne umožní cieľovým používateľom zjednodušiť ich orientáciu v priestoroch našej fakulty.

2.2.1.1 Motivácia

Motiváciou je zdokonalenie sa vo vývoji aplikácií pre mobilné zariadenia implementovaných pomocou štandardných webových technológií (HTML5, CSS, JS) a zároveň sa naučiť, ako tieto technológie zobraziť na mobilnom zariadení.

Ďalší dôvod prečo sa zaujímame o túto tému je uľahčiť dostupnosť dôležitých informácií nielen pre nových študentov, ale aj pre stálych návštevníkov našej fakulty. S tým súvisí aj propagácia fakulty a jej zviditeľnenie a zatraktívnenie širšej verejnosti.

Veríme, že náš tím je dostatočne kvalifikovaný a spĺňa všetky kritéria pre úspešné uskutočnenie tohto projektu, keďže každý jeden člen má skúsenosti s webovými technológiami. Navyše jeden člen má základné skúsenosti s tvorbou aplikácií pre mobilnú platformu Android.

2.2.1.2 Námety

- Informácie o prebiehajúcich udalostiach v miestnostiach(Aj vďaka použiteľnosti QR kódov),
- denné menu v bufete,
- osobný rozvrh na "jeden klik",
- zlepšovať aplikáciu na základe feedback-ov od používateľov.

2.2.2 Analýza výsledkov výskumu

Pri písaní študentských a odborných prác sa autori odkazujú na zdroje z iných publikácií. Vzhľadom na veľký objem zdrojov a nejednoznačnosť v uvádzaní mien autorov, názvov publikácií prípadne kľúčových slov sa ľudia, ktorí v týchto dátach hľadajú, stretávajú s mnohými problémami. Vytvorenie produktu, ktorý by zefektívnil spracovanie, vyhľadávanie, prípadne vizualizáciu informácií by mohlo pomôcť pri celkovom spracovaní bibliografických odkazov.

2.2.2.1 Motivácia

Naša motivácia je práca nad reálnymi dátami z dôvodu efektívnosti pri overovaní výsledkov, ich grafickej reprezentácií vo webovom rozhraní a zapojení nových technológií (MongoDB / Sémantické technológie) do riešenia problému.

Všetci členovia majú skúsenosti s technológiami, ktoré sú dominantné na webe. Viacero členov tímu má skúsenosti s tvorbou webových aplikácií / služieb s použitím databáz. Tieto skúsenosti by sme vedeli naplno využiť pri práci na projekte.

2.2.2.2 Námety

Jedným z možných riešení by bolo vytvorenie hrubého klienta, ktorý by spolu s intuitívnym rozhraním poskytoval pre koncového používateľa jednoduchú prácu s bibliografickými odkazmi.

Využitie existujúcich knižníc napísaných v JS (napr. D3.js) pre zobrazovanie relevantných dát na frontend-e.

2.3.1 Webový komunitný systém otázok a odpovedí

Na internete je v dnešnej dobe dostupných mnoho informácií na rôznorodé témy. Problémom pre koncového používateľa je orientácia vo veľkom množstve výsledkov vyhľadávania. Je prirodzené, že rôzne skupiny používateľov majú problémy, na ktoré hľadajú odpoveď.

Jednou z takýchto skupín sú aj študenti, ktorí často musia riešiť podobné úlohy. Tým pádom hľadajú odpovede na otázky, ktoré už niekto zodpovedal.

V globálnom merítku vyhľadávanie odpovedí na danú tému uľahčujú tzv. Q&A portály.

2.3.1.1 Motivácia

Poskytnúť skupine študentov portál, kde by bolo možné nájsť v kategóriách usporiadané otázky a odpovede. Použitie overených procesov z populárneho portálu Stack Overflow, ktoré by boli prispôsobené potrebám študentov.

Rozšírenie existujúcich konceptov, tak aby bolo čo najjednoduchšie opísať daný problém (prikladanie mediálneho obsahu). V neposlednom rade nám ide o čo najlepší user-experience.

Myslíme si, že sme vhodní kandidáti, pretože každý člen tímu má skúsenosti s technológiami používanými na webe (HTML, CSS, Javascript, PHP). Taktiež viacerí členovia majú skúsenosti s programovaním v tíme a prácou na rozsiahlejších webových aplikáciách.

2.3.1.2 Námety

- hodnotenie prispievateľov
- zatraktívniť GUI
- zhodnotenie kvality príspevku
- implementovanie existujúceho procesu získavania odpovedí

3 Úlohy členov tímu

Táto kapitola obsahuje dlhodobé a krátkodobé úlohy jednotlivých členov tímu. Taktiež obsahuje i autorstvo k jednotlivým častiam dokumentácie.

3.1 Dlhodobé úlohy

V nasledovnej tabuľke (1) sú rozpísané úlohy jednotlivých členov tímu ku dňu 9.10.2013 .

Tabuľka 1- dlhodobé úlohy

Meno	Funkcia
Vladimír Bošiak	Vedúci tímu, Manažér komunikácie
Michal Čerešňák	Manažér rizík
Igor Homola	Manažér plánovania, Webmaster
Matej Bádal	Manažér kvality
Martin Adámik	Manažér monitorovania projektu
Samuel Benkovič	Manažér podpory vývoja
Michal Petráš	Manažér dokumentácie

3.2 Autorstvo v dokumentácii k riadeniu

Tabuľka číslo 2 zobrazuje podiel práce každého člena na tvorbe dokumentácie k riadeniu.

Tabuľka 2- autorstvo v dokumentácii k riadeniu

Autor	Kapitola	Podiel práce
Michal Petráš	Úvod	100%
	Ponuka tímu	14%
	Úlohy členov tímu	100%
	Zápisnica č.5	100%
	6.7	100%
	7.7	100%
Michal Čerešňák	Ponuka tímu	14%
	Zápisnica č.2	100%
	Zápisnica č.10	100%
	6.5	100%
	7.5	100%
Igor Homola	Ponuka tímu	14%
	Zápisnica č.3	100%
	Zápisnica č.9	100%
	6.1	100%
	7.1	100%

Martin Adámik	Ponuka tímu	14%
	6.6	100%
	7.6	100%
	Zápisnica č.7	100%
Matej Bádal	Ponuka tímu	16%
	Zápisnica č.1	100%
	Zápisnica č.8	100%
	6.4	100%
	7.4	100%
Vladimír Bošiak	Ponuka tímu	14%
	Zápisnica č.6	100%
	6.2	100%
	7.2	100%
Samuel Benkovič	Ponuka tímu	14%
	Zápisnica č.4	100%
	6.3	100%
	7.3	100%

3.3 Autorstvo v dokumentácii k inžinierskemu dielu

V tabuľke 3 je zobrazený podiel na vypracovaných častiach dokumentácie k inžinierskemu dielu.

Tabuľka 3 - autorstvo v dokumentácii

Autor	Kapitola	Podiel práce
Michal Petráš	2.1.1	14%
	2.1.2	14%
	2.1.3	14%
	2.1.4	100%
	2.1.7	16%
	2.2.1	16%
	2.2.5	25%
	2.2.6	100%
	2.2.8	100%
	2.2.9	100%
	2.3.1	33%
	2.3.2	100%
	2.4.3	20%
	2.5	100%
	3.2	100%
	2.6.1	100%
	2.6.3	100%
	2.6.4	100%
	2.7.3	100%
	2.7.5	100%
	2.7.8	100%
	2.8.1	100%
	2.8.2	100%
	2.8.3	100%
	2.9.4	100%
	2.10.3	100%
	2.10.5	100%
	2.10.6	100%

Michal Čerešňák	2.1.1 2.1.2 2.1.3 2.1.7 2.2.1 2.2.5 2.3.1 2.8.4 2.8.5	14% 14% 14% 14% 16% 25% 33% 100% 100%
Igor Homola	2.1.1 2.1.2 2.1.3 2.1.7 2.2.1 2.2.5 3.1	14% 14% 14% 14% 16% 25% 100%
Martin Adámik	2.1.1 2.1.2 2.1.3 2.1.7 2.2.1 2.2.5 2.2.7 2.3.1 3.4.1 2.9.1 2.10.1	14% 14% 14% 14% 16% 25% 50% 33% 100% 100% 100%
Matej Bádal	2.1.1 2.1.2 2.1.3 2.1.5 2.1.7 2.2.1 2.2.2 3.4.2 2.7.4 2.7.7 2.9.5 2.10.2	14% 14% 14% 100% 14% 16% 100% 100% 100% 100% 100% 100%
Vladimír Bošiak	2.1.1 2.1.2 2.1.3 2.1.7 2.2.1 2.2.7 2.4.5 2.7.1 2.7.2 2.9.6 2.9.7	14% 14% 14% 14% 16% 50% 100% 100% 100% 100% 100%
Samuel Benkovič	2.1.1 2.1.2	16% 16%

	2.1.3 2.1.6 2.1.7 2.2.3 2.2.4 2.3.3 2.4.1 2.4.2 2.4.3 3.4.3 2.6.2 2.7.6 2.8.6 2.9.2 2.9.3	16% 100% 14% 100% 100% 100% 100% 100% 80% 100% 100% 100% 100% 100% 100%

4 Podporné prostriedky

4.1 Komunikácia

Na komunikáciu používame skype, mobilné telefóny, taktiež máme vytvorenú facebook skupinu pre informovanie všetkých členov. Máme založený aj tímový email, ktorý automaticky rozosiela správy všetkým členom tímu, tak aby bol informovaní ihneď. V prípade potreby sa stretávame osobne a riešime problémy spoločne.

4.2 Manažment projektu

Pre manažment projektu sme si vybrali nástroj Jira. Tento nástroj umožňuje sledovať prácu na všetkých zadaných úlohách, podporuje agilný vývoj formou "Agile" pluginu. Taktiež umožňuje monitorovanie času a lepší odhad na dokončenie jednotlivých úloh, prípadne celého šprintu. Ku každému šprintu sú vytvárané úlohy, ktoré sa následne prideľujú jednotlivým členom tímu.

5 Používané metodiky

5.1 Metodika Scrumovania

ScrumMaster na každý šprint sa zvolí poďla abecedy. Vychádzame z abecedného zoradenia podľa priezviska. Teda v tomto poradí:

1.Adámik

2.Bádal

3.Benkovič

4.Bošiak

5.Čerešnák

6.Homola

7.Petráš

5.2 Metodika pre určenie zapisovateľa

Každý týždeň je zapisovateľom iný člen tímu. Poradie , v ktorom sa na pozícii zapisovateľa členovia tímu striedajú je nasledovné:

1.Bádal

2.Čerešňák

3.Homola

4.Benkovič

5.Petráš

6.Bošiak

7.Adámik

5.3 Metodika pre odovzdávanie dokumentácie používateľských príbehov

Každý člen tímu odovzdá požadované časti dokumentácie na DropBox do priečinka nazvaného podľa šprintu (teda napr. šprint3) a do podpriečinka so svojím menom. Časť dokumentácie nazve podľa názvu príbehu tak aby bolo jasné čoho sa dokument týka. Teda napríklad : OdstranenenieKodu-Adamik.doc

6 Manažment

6.1 Manažment plánovania

6.1.1 Pre koho je metodika určená

Metodika je určená pre členov tímového projektu, každý člen týmu si musí zaznamenávať svoju prácu na projekte. Primárne je metodika určená pre tímy 4 a 9.

6.1.2 Obsah metodiky

Metodika je určená pre prácu so systémom JIRA. Systém je určený na evidenciu, sledovanie a plánovanie úloh vo vývojárskych tímoch.

6.1.3 Zoznam nadväzujúcich metodík a dokumentov

Metodika pre tvorbu dokumentácie (Michal Petráš)

6.1.4 Definícia pojmov

Task – úloha
Story / Userstory –používateľská požiadavka
Log work – zaznamenávanie práce
Sub-task – pod úloha
Issue/issues–otázka, požiadavka, úloha, záznam(v slovenskom jazyku neexistuje vhodný preklad slova preto sa v dokumente sa používa označenie issue / issues pre množné číslo)
Backlog – zoznam všetkých issues
Query - dotaz

6.1.5 Role a zodpovednosti používateľov

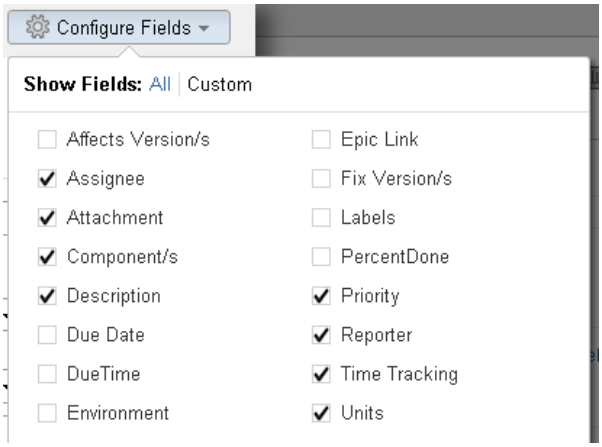
Správca systému JIRA za tímvytváranie a ukončenie šprintov
Scrummaster tvorba reportov, priraďovanie issues
Člen tímuzaznamenávanie času, tvorba issue,získavanie a ukončovanieissue
Vedúci tímu (Reporter)priraďovanie a uzatváranie issues

6.1.6 Práca s issues

6.1.6.1 Nastavenie dotazníka pre tvorbu novej issue

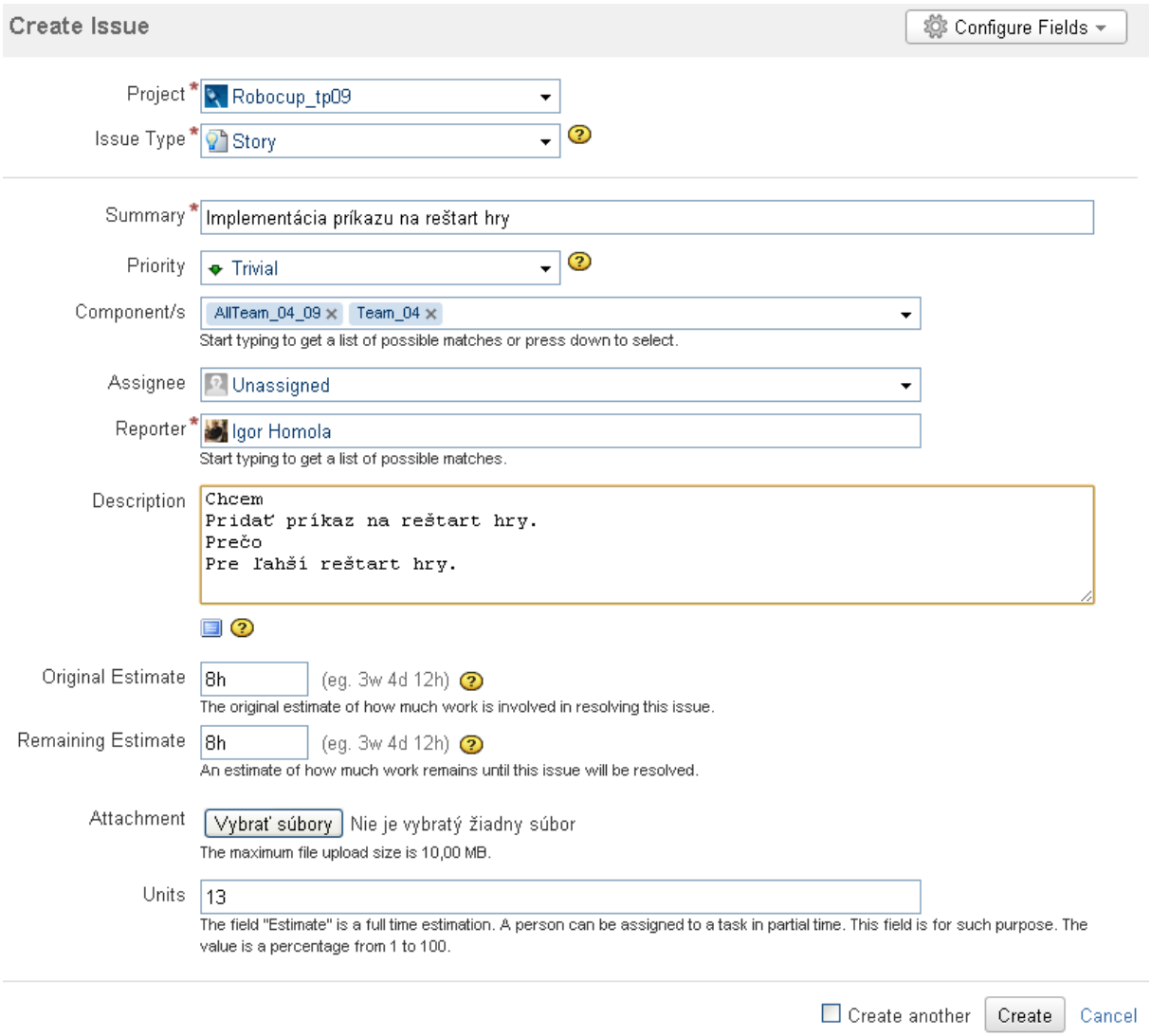
Pred vytváraním issue treba nastaviť v dotazníku vybrané polia. PO kliknutí na tlačidlo ConfigureFields treba vybrať možnosť Custom a zvoliť nasledovné polia pre dotazník podľa obrázku 5.

Obrázok 1 - Polia pre zobrazenie v dotazníku pre tvorbu issue



6.1.6.2 Vytváranie novej issue

Obrázok 2 - vytváranie novej issue



Obsah polí:
Project: Musí byt zvolený projekt Robocup_tp09
Issue Type: Story-základný typ issue vytváraný pri novej požiadavke na vyvíjaný systém

Task– používa sa pri požiadavkách, ktoré priamo nesúvisia s vyvíjaným systémom (napr. Tvorba dokumentácie, úprava webovej stránky ...)
Sub-task – issue využívaná pri potrebe tvorby menších pod úloh, keď je potreba rozdeliť Story prípadne Task na menšie časti.

Vzhľadom na rozsah a účel projektu nepoužívať ostatné tipy issues.
(Bug Epic Improvement New Feature)

Summary: Názov issue , názov musí byť dostatočne konkrétny aby sa z neho dal poznať účel a cieľ issue
Priority: Priorita issue, priorita sa určuje po vzájomnej konzultácii tímu, ak konzultácia neprebehla priorita sa nastavuje na hodnotu Trivial

Component/s: Nastavuje na ktorej tabuli sa issue bude zobrazovať, issue sa musí vždy zobrazovať na spoločnej tabuli AllTeam_04_09, a potom na jednej alebo oboch tabuliach tímu (Team_04 – tabuľa tímu 4, Team_09 – tabuľa tímu 9)
Assignee:Člen týmu ktorému je issue priradená, primárne sa Assignee nastavuje na hodnotu Unassigned (nepriradená)
Reporter: Člen týmu ktorý bude informovaný o zmenách v issue, primárne sa táto hodnota nastavuje na vedúceho tímu.
Description: Podrobnejší popis issue, v prípade typu issuetask a sub-task sa do tohto poľa píše podrobnejší popis úlohy/pod úlohy. V prípade Story sa do poľa vpisuje informácia podľa metodiky scrum pre tvorbu user-story (Chcem – popis požiadavky, Prečo – dôvod požiadavky)
OriginalEstimate: Predpokladaný čas dĺžky riešenia issue *
RemainingEstimate: Zostávajúci čas dĺžky riešenia issue
Attachment: Prílohy potrebné na riešenie issue
Units: Počet bodov pridelený issue (podľa metodiky scrum) po tímovej konzultácii, ak konzultácia neprebehla pole sa necháva nevyplnené

6.1.7 Práca so šprintmi

6.1.7.1 Vytváranie šprintu

1. V záložke Agile treba zvoliť spoločnú tabuľu AllTeam_04_09. Pri práci s tabuľou treba zvoliť možnosť Plan Work Report).
2. Následne dostávame možnosť vytvoriť nový šprint pomocou tlačidla createsprint.
3. Metodoudrag and drop pridávame issues do šprintu zo zoznamu úloh - Backlogu
4. Začať šprint kliknutím na odkaz StartSprint
5. Zadať názov šprintu, začiatočný a konečný čas šprintu
6. Potvrdiť stlačením tlačidla Start

6.1.7.2 Pridávanie issues počas priebehu šprintu

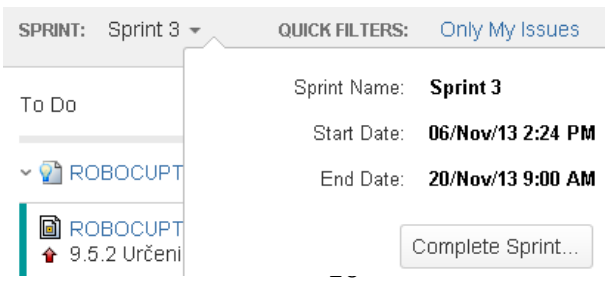
Pridávanie issues je možne v priebehu celého šprintu

1.V záložke Agile treba zvoliť spoločnú tabuľu AllTeam_04_09. Pri práci s tabuľou treba zvoliť možnosť Plan Work Report).
2.Metodou drag and drop pridávame issues do šprintu.

6.1.7.3 Ukončenie šprintu

1.V záložke Agile treba zvoliť spoločnú tabuľu AllTeam_04_09. Pri práci s tabuľou treba zvoliť možnosť W Plan Work Report).
2.Kliknutím na prebiehajúci šprint získavame možnosť ukončiť šprint (CompleteSprint)

Obrázok 3 - ukončenie šprintu



6.1.8 Priraďovanie, zaznamenávanie a ukončenie práce

*Formát zadávania času: 1w 3d 4h 5m (w-týždne, d-dni, h-hodiny, m-

Priradiť prácu používateľovi možno dvoma spôsobmi.
1. Nájdením záznamu v záložke Issues
2. Nájdením záznamu na tabuli AllTeam_04_09 v záložke Agile
V oboch prípadoch sú možnosti Assign (priradiť) a Assign To Me (priradiť mne).
Každý člen tímu si priraďuje issues sám (Assign To Me (priradiť mne)).
Priraďovať prácu inému členovi tímu môže vedúci tímu, scrummaster, alebo správca systému iba po vzájomnej dohode s daným členom tímu.

6.1.8.2 Zaznamenávanie práce (Log Work)

Zaznamenať prácu možno dvoma spôsobmi.
1. Nájdením záznamu v záložke Issues kliknutím na menu More Actions
2. Nájdením záznamu na tabuli AllTeam_04_09 v záložke Agile
Pri zaznamenaní práce je nutné zadať:
TimeSpent: Odpracovanú dobu
DateStarted: Začiatok práce
WorkDescription: Popis práce ktorá bola vykonaná.

6.1.8.3 Ukončenie práce (zatvorenie issue)

Pred ukončením práce je nutné pripojiť k issue súbor s dokumentáciou (Metodika pre tvorbu dokumentácie)
Ukončiť prácu možno dvoma spôsobmi
1.Nájdením záznamu v záložke Issues prácu ukončujeme kliknutím na tlačidlo ResolveIssue.
2. Nájdením záznamu na tabuli AllTeam_04_09 v záložke Agile a prenesením issue do záložky Done na polohu ResolveIssue
Pri ukončení práce je nutné zadať:
Resolution: Done (Vzhľadom na rozsah a účel projektu nepoužívať iné možnosti)
TimeSpent: Poslednú odpracovanú dobu v formáte: 1w 3d 4h 5m
DateStarted: Začiatok poslednej práce
Comment: Popis práce ktorá bola vykonaná.

Ukončenú úlohu následne musí akceptovať a zavrieť Reporter (vedúci tímu) pomocou tlačidla CloseIssue.

6.1.9 Práca s tabuľou

Pri práci s JIRA sa využíva tabulaAllTeam_04_09 pre prehľad, a tabuľa konkrétneho tímu (Team_04 ,Team_09) pre vykonávanie potrebných operácii s issues. Tieto tabule sú typu scrumboard a zobrazujú sa na nej iba issues z konkrétneho šprintu.

Issue pridelené konkrétnemu členovi tímu, si daný člen sám prenáša do záložiek na tabuli tímu, podľa toho v ktorej fáze práce sa konkrétna issue nachádza To Do (práca sa nezačala), In Progress(na issue sa pracuje) Done(práca skončená).

Zoznam všetkých issues(Backlog) možno získať v záložke Issues použitím vhodného Query dotazu.

6.2 Manažment podpory vývoja

6.2.1 Úvod

Perconik je nástroj na pridávanie rôznych značiek do zdrojového kódu programu, ktorými dokáže zabezpečiť väčšiu prehľadnosť o tom ktorú časť kódu treba modifikovať, pridať či odstrániť.

6.2.2 Pojmy

Eclipse - je vývojárske prostredie ktoré slúži najmä na programovanie v jazyku java a taktiež v ňom používame plugin PerConIK

Značka - komentár špecifického charakteru ktorý označuje určitú časť kódu.

6.2.3 Proces pridávania značiek

Pre správne vytvorenie značky je potrebné aby ste si najprv označili kód ktorý chcete označiť, následne pravým kliknutím zobrazíme lištu z ktorej vyberieme PerConIK tool -> Insert Tag.

Zo zoznamu si vyberieme požadovanú značku ktorú si následne upravíme do potrebného formátu. Opis použitia značiek sa nachádza v podkapitolách 3.1 až 3.6.

Po kliknutí na uloženie sa nám značka z validuje a označí na ľavej lište modrou farbou.

6.2.4 Pridanie značky Todo

Každá nedokončená časť kódu musí byť označená značkou Todo. Značka obsahuje atribút Task do ktorého je vhodné zadať meno úlohy. **Značka musí obsahovať atribútu Priority** ktorá nadobúda hodnoty Low, Normal, High, Urgent, Immediate podľa toho o akú dôležitú úlohu sa jedná. Pomocou atribúty Solver je možné prideliť úlohu jednému riešiteľovi. **Todo sa musí používať ako párová značka.**

Príklad (Správne)

```
....
//Todo #Task(Implementation) #Solver(sppred@gmail.com) #Priority(low)  |
sppred@gmail.com 2013-11-17T11:07:21.4660000Z
...
//@<Todo | sppred@gmail.com 2013-11-17T11:07:21.4660000Z
```

Príklad (Nesprávne)

```
....
//Todo #Task(Implementation) #Solver(sppred@gmail.com) | sppred@gmail.com 2013-11-
17T11:07:21.4660000Z
...
```

Príklad (Nesprávne) - jedná sa o smell (viď nižšie)

```
//Todo #Priority(low) this is only temporary | sppred@gmail.com 2013-11-
17T10:50:28.3930000Z
...
```

```
//@<Todo | sppred@gmail.com 2013-11-17T10:50:28.3930000Z
```

6.2.5 Pridanie značky Bug

Označuje nejakú časť kódu v ktorej sa nachádza chyba. Obsahuje rovnaké atribúty ako značka Todo. **Musí obsahovať atribútu Priority**. **Musí sa používať ako párová značka.** Špeciálny prípad značky Todo.

Príklad (Správne)

```
//Bug #Task(returns the wrong value) #Solver(sppred@gmail.com) #Priority(High) | sppred@gmail.com 2013-11-17T11:40:20.1280000Z
....
//@<Bug | sppred@gmail.com 2013-11-17T11:40:20.1280000Z
```

Príklad (Nesprávne)

```
...
//Bug #Task(returns the wrong value) #Solver(sppred@gmail.com) #Priority(High) | sppred@gmail.com 2013-11-17T11:40:20.1280000Z
```

6.2.6 Pridanie značky Ranking

Značka ktorá slúži na ohodnotenie zdrojového kódu čomu zodpovedajú aj atribúty tejto značky ktoré nadobúdajú hodnoty Low, Normal, High. Atribút Comprehensibility hodnotí ako ľahko, ťažko sa dá kód pochopiť. Na druhej strane atribút Testability hodnotí kód z pohľadu testovateľnosti. **Každá značka Ranking musí obsahovať aspoň jeden atribút a musí byť použitá ako párová značka.**

Príklad(Správne)

```
//Ranking #Comprehensibility(High) #Testability(High) | sppred@gmail.com 2013-11-17T12:17:14.1690000Z
...
//@<Ranking | sppred@gmail.com 2013-11-17T12:17:14.1690000Z
```

Príklad(Nesprávne)

```
//Ranking | sppred@gmail.com 2013-11-17T12:17:14.1690000Z
...
```

6.2.7 Pridanie značky CodeReview

Označený kód značkou CodeReview neprešiel posudzovaním pretože porušil nejakú z definovaných konvencií**.** Porušená konvencia sa zadáva do atribútu Violation. **Značka musí obsahovať aspoň jeden atribút Vilation. CodeReview musí byť použitý ako párová značka.**

Príklad(Správny) - Variable Naming Conventions

```
//CodeReview #Violation(VNC) | sppred@gmail.com 2013-11-17T12:42:16.5270000Z
...
//@< | sppred@gmail.com 2013-11-17T12:42:16.5270000Z
```

Príklad (Nesprávny)

```
//CodeReview | sppred@gmail.com 2013-11-17T12:42:16.5270000Z
...
//@<CodeReview | sppred@gmail.com 2013-11-17T12:42:16.5270000Z
```

6.2.8 Pridanie značky Recommendation

Značka Recommondation predstavuje odporúčania ktoré by boli vhodné aplikovať na časť kódu. Jediným atribútom tejto značky je Type. **Každá takáto značka musí obsahovať aspoň jeden takýto atribút. Za každým atribútom sa musí nachádzať text ktorý bližšie špecifikuje daný problém.** Atribút Type nadobúda hodnoty :

- Efficiency - Odporúčanie na zvýšenie efektívnosti kódu.
- Technology - Odporúčanie na použitie technológie ktorá sa ma použiť v danom kóde.
- Reuse - Odporúčanie na to aby bola metóda znovu používala v kóde.
- Other - Ostatné odporúčania.

Príklad (správny)

```
//Recommendation #Type(Reuse) You can call it in method methodName|
sppred@gmail.com 2013-11-17T15:40:56.2240000Z
...
//@<Recommendation | sppred@gmail.com 2013-11-17T15:40:56.2240000Z
```

Príklad (nesprávny)

```
//Recommendation #Type(Reuse) | sppred@gmail.com 2013-11-17T15:40:56.2240000Z
...
//@<Recommendation | sppred@gmail.com 2013-11-17T15:40:56.2240000Z
```

6.2.9 Pridanie značky Smell

Každá časť kódu označená touto značkou obsahuje takzvaných pach. **Každá značka musí obsahovať atribút SmellType** za ktorým sa môže ale nemusí nachádzať text ktorý ho bližšie špecifikuje. Dôležitá je tiež nasledujúca Tabuľka hodnôt[1] ktoré môže nadobúdať atribút SmellType:

Hodnota	Popis	Hodnota	Popis
DupCode	duplikovaný kód	LazyClass	Trieda s malou funkcionalitou
LongMethod	dlhá metóda	SpecGener	Špekulatívna generalizácia
LargeClass	veľká trieda	TempField	Dočasné pole
LongParamList	dlhý zoznam parametrov metódy	MesgChains	Reťazenie volaní
DiverChange	Časté zmeny v kóde	MiddleMan	Nevhodné použitie sprostredkovateľa
ShotSurgery	Veľa malých zmien z jednej požiadavky	InappropIntim	Nadmerné využívanie privátnych polí inej triedy

FeatureEnvy	veľa volaní metódy inej triedy	AlterClasses	Exitencia triedy s alternatívnou funkcionalitou, ale rozdielnym rozhraním
DataClumps	Zhluky dát	IncomLibClass	Nekompletná trieda knižnice
PrimitObses	Nevhodné používanie základných typov	DataClass	Dátová trieda
SwitchStat	Nevhodné použitie vetvenia	RefBequest	Dedenie nepotrebných dátových polí
ParInherHier	Paralelne hierarchie dedenia	Comments	Používanie komentárov namiesto samovysvetľujúceho kódu.

Príklad(správne)

```
//Smell #SmellType(DupCode) | sppred@gmail.com 2013-11-17T17:02:11.8000000Z
...
//@< | sppred@gmail.com 2013-11-17T17:02:11.8000000Z
```

Príklad(Nesprávne)

```
//Smell | sppred@gmail.com 2013-11-17T17:02:11.8000000Z
...
//@<Smell | sppred@gmail.com 2013-11-17T17:02:11.8000000Z
```

6.3 Manažment podpory vývoja

6.3.1 Úvod

Táto metodika popisuje prácu s distribuovaným verizonovacím systémom GIT s cieľom zabezpečiť jednoduchší vývoj a uchovávanie zdrojového kódu. Metodikou by sa mali riadiť programátori.

6.3.2 Súvisiace dokumenty

- Pro Git

  http://git-scm.com/book
- Sémantické verzionovanie

  http://semver.org/

6.3.3 Slovník pojmov

Branch - vývojová vetva

Merge - spojenie vývojových vetiev

6.3.4 Postupy

Vytváranie commit-u

Scenár:

- Ukončenie vývoja na časti funkcionality
- Zdieľanie kódu s iným vývojárom

Postup:

1. Zistenie zmien v aktuálnej lokálnej branch-i.

   Zmeny, ktoré sme vykonali v aktuálnej branch-i a nie sú v stage-i zistíme pomocou príkazu:

   ```
   git status
   ```

2. Pridanie súborov do stage-u

   a. určitý súbor alebo všetky súbory v danom adresári

   ```
   git add <cesta k súboru/adresáru>
   ```

   b. všetky súbory

   ```
   git add .
   ```

3. Vytvorenie commit-u

   a. Veľká zmena: použijeme príkaz

   ```
   git commit
   ```

Popis commit-u vztvárame pomocou textového editoru.

```
1.    <task z Jira / časť projektu> - <jednoduchý popis>
2.
3-x.  <podrobný popis commit-u>
```

b. Malá zmena (dá sa popísať kratšou správou)

```
git commit –m "<task z Jira / časť projektu> - <jednoduchý
popis>"
```

4. Odoslanie commit-u na server

   Stiahneme aktuálne zmeny

```
git pull
```

Ak sú konflikty, vyriešime konflikty.
Odošleme commit do vzdialeného repozitára

```
git push origin <názov branch-e>
```

Zmena aktuálneho branch-u

Scenár:

- Práca na inej feature.
- Bugfix
- Vykonanie zmien v zlej branch-i

Postup:

1. Uvedenie aktuálnej branch-e do konzistentného stavu
   a. Commit-neme aktuálne zmeny podľa postupu 1
   b. Revert-neme zmeny

```
git reset --hard HEAD
```

   c. Zmeny uložíme do stash-u (ak chceme zmeny preniesť)

```
git stash
```

2. Zmena branch-e

   Aktualizácia lokálneho repozitára

```
git pull
```

Zmena aktuálnej branch-e

```
git checkout <názov branch-e>
```

3. Aplikovanie zmien z predošlého repozitára

   (prípad: vykonanie zmien v zlej branch-i)

```
git stash pop
```

Vytváranie branch-ov

Scenár:

- Vytvárame novú feature

  (nový branch vytvárame z development branch-e)
- Rozsiahlejší bugfix

  (nový brnach vytvárame z master branch-e)

Postup:

1. Príprava východzej branch-e

   Zmeníme branch na master / development podľa postupu 2.
2. Vytvorenie novej branch-e

   Názov branch-e bude vo formáte:

```
<Jira-story/task>-<meno zodpovedného>-<popis feature/bug>
Príklad:
TASK-1-Mrkvicka-vylepsenie_kopu_do_lopty
```

Vytváranie tagov

Vytváranie tagov nám umožňuje rozlišovať rôzne verzie aplikácie.

Používa sa sémantické verzionovanie.

Verzia sa zapisuje: *<verzia API>.<feature>.<fix>*.

Scenár:

- Označenie verzie aplikácie

Postup:

1. Vytvorenie tagu
   a. Aktuálny commit

```
git tag -a <verzia> -m '<správa>'
```

   b. Určitý commit

```
git tag -a <verzia> -m '<správa>' <číslo commit-u>
```

Merge-ovanie branch-í

Scenár:

- Ukončenie vývoja určitej feature
- Ukončenie bugfix-u

Postup:

1. Uvedenie aktuálnej branch-e do konzistentného stavu (postup 2 krok 1)
2. Merge nadradenej branch-e do feature branch-e

   Dôvod je, že v prípade konfliktu riešime konflikt v branch-i s nižšou prioritou.

   ```
   git merge origin/<nadradená branch>
   ```
3. Odoslanie aktuálnej branch-e do vzdialeného repozitára

   ```
   git push origin <feature branch>
   ```
4. Zmeníme aktuálnu branch na nadradenú branch

   ```
   git checkout <nadradená branch>
   ```
5. Merge lokálnej feature branch-e do nadradenej branch-e

   ```
   git merge <feature branch>
   ```
6. Odoslanie aktuálnej branch-e do vzdialeného repozitára

   ```
   git push origin <nadradená branch>
   ```

6.4 Manažment kvality

6.4.1 Úvod

Nasledujúci dokument obsahuje metodiky pre písanie zdrojových kódov s dôrazom na prehľadnosť, čitateľnosť a zrozumiteľnosť. Ďalej obsahuje metodiky pre údržbu kódu a jeho vhodnú dokumentáciu.

6.4.2 Pre koho je metodika určená

Metodika je určená primárne pre členov, tímu, ktorí upravujú kód, teda pre programátorov. Taktiež je určená pre manažéra kvality, ktorý je zodpovedný za celkový stav kódu.

6.4.3 Slovník pojmov

CamelCase – štýl písania, kedy sa slovné spojenia píšu bez medzery, pričom každé nové slovo začína veľkým písmenom. Úvodné písmeno môže byť bu veľké, alebo malé.

6.4.4 Obsah metodiky

Ako už bolo vyššie spomenuté, metodika pozostáva z viacerých častí. Prehľad celých názvov jednotlivých častí je uvedený v tabuľke.

Tabuľka 4 - názvy častí

Názov postupu
Pomenovávanie tried, metód a premenných
Používanie zátvoriek
Zalamovanie kódu
Dokumentovanie pomocou JavaDocu

6.4.5 Postupy pri písaní kódu

6.4.5.1 Pomenovávanie tried, metód a premenných

6.4.5.1.1 Triedy

Všetky triedy musia by pomenované špecifickým spôsobom. Pri písaní názvu triedy je nutné použiť CamelCase. V prípade tried je vhodné, aby aj úvodné písmenko bolo veľké. Názov triedy musí byť v anglickom jazyku a musí vyjadrovať podstatné meno, najlepšie nejakú entitu z reálneho života, ktorú trieda reprezentuje.

Príklad:

```
 public class FastCar {

       /*

        * implementácia triedy

        */

}
```

Nesprávne použitie:

```
public class Fast_Car {

       /*

        * implementácia triedy

        */

}


public class fastCar {

       /*

        * implementácia triedy

        */

}
```

6.4.5.1.2 Rozhrania

Rovnako, ako pri názvoch tried, aj pri názvoch rozhraní sa musí použiť CamelCase s veľkým začiatočným písmenom. Narozdiel od názvu triedy, názov rozhrania musí opisovať vlastnosť, ktorú dané rozhranie vyjadruje. Názov rozhrania teda bude odvodený od anglického názvu pre vlastnosť, ktorá je v trpnom príčastí. Pre ilustráciu je vhodné uviesť rozhranie **Runnable,** ktoré poskytuje možnosť spustenia triedy, ktorá ho implementuje, v

samostatnom vlákne. V prípade, že by sa nedal jednoznačne vymedziť výraz pre konkrétnu vlastnosť, rozhrania by malo byť pomenované s prefixom „I“, za ktorým bude nasledovať podstatné meno entity z reálneho života.

Príklad:

```
public interface  Movable {

	/*

	 * implementácia rozhrania

	 */

}


public interface ICar {

	/*

	 * implementácia rozhrania

	 */

}
```

Nesprávne použitie:

```
public interface CarInterface {

	/*

	 * implementácia rozhrania

	 */

}
```

6.4.5.1.3 Metódy

Opäť, pri názve metódy musí byť požitý CamelCase. Oproti názvom tried a rozhraní však názov metódy bude začínať malým písmenom. Ako názov metódy bude použitý anglický výraz pre sloveso, prípadne viacero slovies, ktoré opisujú, čo daná metóda robí.

Príklad:

```
public int countTotalTravelTime() {

	/*

	 * logika metódy

	 */

}


public boolean isMoving() {

	/*

	 * logika metódy

	 */

}
```

Nesprávne použitie:

```
public int travelTime() {

	/*

	 * logika metódy

	 */

}
```

6.4.5.1.4 Premenné

Premenné sa dajú rozdeliť na dve skupiny – statické premenné a inštančné premenné. Obe skupiny sa musia nazýva podľa toho, na čo sa používajú. Takisto musia byť pomenované v anglickom jazyku. Rozdiel pri pomenovávaní je, že inštančné premenné musia využívať CamelCase a mená statických premenných sa musia písať veľkými písmenami, pričom viacslovné premenné majú jednotlivé slova oddelené podtŕžnikom.

Príklad:

```
public class FastCar {

        //konštanta

        public static final int NUMBER_OF_SEATS;



        //inštančná premenná

        private String carModel;

}
```

Nesprávne použitie:

```
public class FastCar {

       //konštanta

        public static final int number_of_seats;



        //instančná premenná

        private String CarModel;

}
```

6.4.5.2 Používanie zátvoriek

Každý separátna čas kódu (trieda, metóda, riadiace cykly, podmienky) musí byť správnym spôsobom ozátvorkovaná. Otváracia zátvorka sa musí nachádzať na rovnakom riadku, na akom je napísaný daný výraz, ktorý jej predchádza. Táto notácia je zvolená kvôli väčšej prehľadnosti a taktiež úspore miesta, kedy je ušetrený jeden riadok, oproti notácii kde sa otváracia zátvorka píše až na nový riadok.

Príklad:

```
public class Car {

     public static void main(String[] args) {

           int  i = 0;

           if (i == 0) {

                  //logika

           } else {

                 //logika

           }

     }

}
```

Nesprávne použitie:

```
public class Car

{

     public static void main(Strring[] args)

    {

           int  i = 0;

           if (i == 0)

           {

                  //logika

           }

           else

           {

                 //logika

           }

     }

}
```

6.4.5.3 Zalamovanie kódu

Vytvorený kód musí byť správne zalomený do jednotlivých riadkov. Je potrebné rozlišovať dva prípady – metóda nám vráti rovnaký objekt, alebo nám metóda vráti iný objekt.

6.4.5.3.1 Rovnaký objekt

Pri tomto type objektu musí byť každá metóda, ktorá sa pri zreťazení metód volá, zalomená vždy v novom riadku. Takýmto spôsobom je jednoduché rozlíšiť, ktorý objekt sa danými volaniami metód mení. Dobrým príkladom na tento stav sú triedy, ktoré mapujú reálne entity a teda neobsahujú žiadnu výpočtovú logiku, ale len zapúzdrujú dáta.

Príklad:

```
CarEntity car = new CarEntity();

car.setModel( “Volvo “)

   .setMaxSpeed(250)

   .setNumberOfSeats(5);
```

Nesprávne použitie:

```
CarEntity car = new CarEntity();

car.setModel( “Volvo “).setMaxSpeed(250).setNumberOfSeats(5);
```

6.4.5.3.2 Iný objekt

V tomto prípade je nutné, aby bola pri zreťazení metód nasledujúca metóda zalomená do nového riadku len v prípade, že by dĺžka riadku presiahla maximálnu dĺžku riadku.

Príklad:

```
CarRepository repository = new CarRepository();

String model = repository.findCarByMaxSpeed(150).getModel();
```

6.4.5.4 Dokumentovanie pomocou JavaDoc-u

Je nutné, aby bol kód prehľadne a dobre zdokumentovaný. Prostriedkom na to sú komentáre, pričom programovací jazyk Java nám poskytuje podporu dokumentácie pomocou vytvárania komentárov so štandardnou štruktúrou – JavaDoc.

Zdrojový kód musí obsahovať dva typy komentárov – komentáre pre triedy a komentáre pre jednotlivé metódy. Všetky komentáre musia by napísané v anglickom jazyku.

Štandardná hlavička komentáru pre triedu musí obsahovať opis danej triedy, v ktorom bude zhrnuté, na čo daná trieda slúži. Keďže Robocup je projekt, na ktorom do tejto doby pracovala viacero tímov, pričom každý z týchto tímov strávil na projekte určitý čas. Z tohto dôvodu je povinným poľom v štandardnej hlavičke pole @author, kde sa bude nachádza informácia o autorovi zdrojového kódu triedy, respektíve informácia o členovi tímu, ktorý je aktuálne zodpovedný za daný zdrojový kód. Jednotlivé mená sa budú písať za sebou do riadku, pričom budú oddelené čiarkou. Jednotlivé mená budú vo formáte meno, priezvisko oddelené medzerou. Je zakázané zasahovať do mien, ktoré sú už uvedené v tomto poli.

Príklad:

```
/**
 *
 * Class, that represents car and encapsulates data
 *
 * @author Jožko Mrkvička, Matej Bádal
 */
public class Car {
    //implementácia triedy
}
```

Štandardná hlavička pre metódu je pomerne odlišná. Hlavička taktiež musí obsahovať krátky opis. Ďalej musí obsahovať pole @param, kde sa bude nachádza opis konkrétneho vstupu pre metódu. Samozrejme za predpokladu, že metóda nejaký vstup vyžaduje. V neposlednom rade musí hlavička obsahovať pole @return, kde bude opísaný typ návratovej hodnoty, ktorú metóda vracia.

Príklad:

```
public class Travel {

	private long timeOfStart;

	private long timeOfArrival;


	/**
	 *
	 * returns remaining time to arrival to destination
	 * @return remaining time in miliseconds
	 */
	 public long getRemainingTravelTime() {
		return this.timeOfArrival - this.timeOfStart;
	 }
}
```

6.5 Manažment riadenia

6.5.1 Úvod dokumentu

Tento dokument popisuje riadenie a priebeh tímových stretnutí. Opísaný postup je odvodený od scrum modelu pre riadenie vývoju softvérového projektu.

6.5.2 Slovník pojmov

Scrum

Inkrementálna a iteratívna metodika pre agilný vývoj softvérového projektu

Šprint

Základná časová jednotka metodiky scrum. Šprint je časovo ohraničené obdobie, vopredstanovenej dĺžky, v ktorom sú vypracovávané úlohy. Dĺžka šprintu je typicky 2 týždne.

ScrumMaster

Člen tímu zodpovedný za správne fungovanie tímu

ProductOwner

Reprezentuje zákazníka, definuje požiadavky na produkt

Retrospektíva

Obhliadnutie sa za práve dokončeným šprintom. Členovia tímu sa vyjadrujú k tomu čo bolo v danom šprinte dobré a čo treba zlepšiť

BurndownChart

Graf, ktorý vyjadruje čas práce strávený na projekte a čas práce, ktorá ostáva na dokončenie v rámci

šprintu

Backlog

Zoznam úloh, ktoré je nutné spraviť pre dokončenie projektu

Review

Prehľad dokončenej a nedokončenej práce v rámci šprintu

6.5.3 Tímové stretnutia

Tímové stretnutia delíme na formálne a neformálne. Neformálne stretnutia sa konajú výnimočne ak sa členovia tímu potrebujú iba na niečom dohodnúť. Tieto stretnutia majú voľný priebeh kde nie je nutnosť zaznamenávať žiadne dokumenty o priebehu stretnutia. Táto metodika sa ďalej venuje už iba formálnym stretnutiam, ktorých priebeh ma presné postupy. V rámci jedného šprintu sa konajú tri formálne stretnutia:

- Tímové stretnutie na začiatku šprintu
- Tímové stretnutia v priebehu šprintu
- Tímové stretnutie na konci šprintu

Spoločným prvkom všetkých formálnych stretnutí je tvorba zápisu. Zápis tvorí vždy jeden člen týmu, ktorého určí vedúci tímu. V nasledujúcich kapitolách sú jednotlivo popísané stretnutia. Všetky veci, ktoré sú na stretnutiach stanú alebo dohodnú musia byť evidované v zápise, preto ďalej zápis už nie je spomínaný pri jednotlivých procesoch.

6.5.3.1 Tímové stretnutie na začiatku šprintu

6.5.3.1.1 Popis stretnutia

Tento typ stretnutia sa koná na začiatku každého šprintu. Tento proces definuje spôsob, akým sú plánované úlohy na ďalší šprint. Zaoberá sa výberom úloh, hodnotením náročnosti a rozdeľovaním úloh medzi členov týmu.

	Krok	Kapitola
1.	Vyberanie úloh z backlog-u	3.2
2.	Hodnotenie náročnosti úloh – Planning poker cards	3.3
3.	Rozdelenie úloh medzi členov týmu	3.4

6.5.3.1.2 Vyberanie úloh z backlog-u

Predpokladom pre tento proces je mať v backlog-u zoznam všetkých úloh, ktoré je potrebné spraviť pre dokončenie projektu. Výber úloh má na starosti ProductOwner.

Proces:

1. ProductOwner vyberie úlohy pre aktuálny šprint

2. ProductOwner určí priority úloh:

- Minor
- Major
- Critical

6.5.3.1.3 Hodnotenie náročnosti úloh –PlanningPoker Cards

V tomto procese členovia tímu určujú náročnosť úloh, ktoré vybral ProductOwner, pomocou Planning Poker Cards. Hodnoty týchto úloh sa potom pripisujú ako body členom tímu, ktorí danú úlohu vypracujú.

Proces:

1. Každý člen si pripraví karty (buď papierové alebo môže použiť mobilnú aplikáciu)

2. ScrumMaster určí, ktorá úloha sa ide hodnotiť

3. Každý člen podľa seba ohodnotí kartou náročnosť úlohy (čím vyššie číslo tým náročnejšia úloha)

4. Na výzvu ScrumMaster-a všetci ukážu hodnotu svojich kariet

5. Členovia tímu, ktorí majú najnižšie a najvyššie číslo musia svoje hodnotenie zdôvodniť

6. Opakujú sa kroky 3,4 a 5 až kým sa všetci členovia tímu nezhodnú na rovnakom čísle

6.5.3.1.4 Rozdelenie úloh medzi členov tímu

Riadenie rozdeľovania úloh medzi členov tímu má na starosti ScrumMaster. Konkrétne úlohy, na ktorých chcú jednotliví členovia pracovať, si však vyberajú sami.

Proces:

1. ScrumMaster určí ktorá úloha sa ide prideľovať

2. Členovia tímu hovoria jeden po druhom či majú o úlohu záujem alebo nie

3. Prideľovanie danej úlohy končí keď jeden z členov úlohu príjme

4. Ak o úlohu nemá záujem nikto, vedúci tímu určí člena, ktorý túto úlohu vypracuje

5. Kroky 1,2,3 a 4 sa opakujú až pokým nie sú všetky úlohy pridelené

Dodatok: Každý člen tímu musí mať pridelenú aspoň jednu úlohu na každý šprint aby nenastala situácia, že mu na konci šprintu bude udelených 0 bodov, čo by znamenalo opustenie tímu a nespravenie predmetu Tímový projekt I.

6.5.3.2 Tímové stretnutie v priebehu šprintu

6.5.3.2.1 4.1 Popis stretnutia

Tento typ stretnutia sa koná vždy v polovici šprintu teda týždeň po začiatočnom stretnutí. Hlavnou náplňou stretnutia je sledovanie pokroku a zistenie čo robí najväčšie problémy pri riešení.

	Krok	Kapitola
1.	Sledovanie progresu prác	4.2
2.	Problémy pri práci	4.3
3.	Preloženie úlohy na iného člena tímu	4.4

6.5.3.2.2 Sledovanie progresu prác

Cieľom sledovania progresu prác je aby ProductOwner zistil ako sa vývoj produktu posunul. Tu sa tiež dozvie kto na akej úlohe pracuje a s akou úspešnosťou.

Proces:

1. ScrumMaster odovzdá vytlačený BurndownChart spolu so zoznamom úloh na daný šprint ProductOwner-ovi

2. Každý člen tímu prezentuje slovne na čom týždeň pracoval a aký má plán na ďalší týždeň

3. Na základe prezentovaných výsledkov ProductOwner môže spresniť zadanie úlohy

6.5.3.2.3 Problémy pri práci

Pri prezentácii doposiaľ odvedenej práce sa venuje čas aj na rozobratie problémov pri riešení. Úlohou tohto sedenia je predísť situácii, že na konci šprintu nebudú splnené všetky úlohy. Tu nastáva aj možnosť preloženia úlohy z jedného člena na druhého.

Proces:

1. ScrumMaster vyzýva člena tímu na diskusiu o problémoch, ktoré nastali pri vývoji

2. Člen tímu prezentuje problémy

3. ProductOwner spresní konkrétne požiadavky

4. Kroky 1,2 a 3 sa opakujú pre všetkých členov, kým nie sú odstránené všetky problémy

6.5.3.2.4 Preloženie úlohy na iného člena tímu

Počas šprintu môže nastať situácia kedy člen tímu nie je schopný danú úlohu vypracovať. V tomto prípade je možné preložiť úlohu z jedného člena tímu na iného.

Proces:

1. Člen tímu uvedie dôvod prečo nie je schopný úlohu dokončiť

2. Vedúci tímu vyzýva ponúkne úlohu ostaným členom týmu

3. Ak o úlohu nie je záujem pridelí ho členovi, ktorý pracuje na úlohách s najnižším ohodnotením

4. Body za prenášanú úlohu sa presunú z člena, ktorý sa vzdal úlohy na člena, ktorý ju prevzal

6.5.3.3 Tímové stretnutie na konci šprintu

6.5.3.3.1 Popis stretnutia

Toto stretnutie prebieha vždy na konci šprintu čiže po dvoch týždňoch. V tomto type tímového stretnutia sa vyhodnocuje, či sa úlohy podarilo dokončiť, prideľujú sa body členom týmu a hodnotí sa spätne priebeh celého šprintu.

	Krok	Kapitola
1.	Review - BurndownChart	5.2
3.	Prideľovanie bodov členom tímu	5.3
4.	Review a Retrospektíva	5.4

6.5.3.3.2 Review – BurndownChart

Proces kontroly splnenia úloh na konci šprintu pomocou BurndownChart-u je podobný ako v priebehu. Tu už by však všetky úlohy mali byť splnené a kždý člen týmu musí prezentovať svoju dosiahnutú robotu.

Proces:

1. ScrumMaster odovzdá vytlačený BurndownChart spolu so zoznamom úloh na daný šprint ProductOwner-ovi

2. Každý člen tímu prezentuje výsledok svojej práce

3. Typy prác a požiadavky pre úspešné splnenie:

- Analýza

    - musí odovzdať dokument s vypracovanou analýzou
    - prezentovať dôležité prvky použiteľné pre náš projekt
- Kódovanie
    - musí prezentovať kód s popisom
    - ukázať funkcionalitu
    - odovzdať dokumentáciu
- Ostatné úlohy
    - každá úloha musí byť popísaná v dokumente
    - odovzdanie dokumentu s popisom

6.5.3.3.3 Prideľovanie bodov členom týmu

Na základe predvedených výsledkov prideľuje ProductOwner body jednotlivým členom. Za splnenú úlohu získava počet bodov, ktorými bola úloha ohodnotená pomocou Planning Poker Cards. Ak na úlohe pracovalo viac členov, body sa rovnomerne rozdelia.

Proces:

1. ProductOwner pridelí body každému členovi týmu

2. ProductOwner zapíše body do systému kde sa každému členovi kumulujú

6.5.3.3.4 Retrospektíva

V rámci retrospektívy sa riešia otázky ohľadom riadenia prebehnutého šprintu.

Proces:

1. ScrumMaster vyzýva členov týmu aby prezentovali svoje postrehy k riadeniu

2. Každý člen prezentuje čo v šprinte prebiehalo dobre, čo treba zlepšiť a čo by zrušil

6.6 Manažment testovania

6.6.1 Úvod

Predmetom metodiky dolnej úrovne je spôsob vytvorenia a spustenia testovacej triedy za pomoci frameworku JUnit v prostredí Eclipse a programovacom jazyku Java.

6.6.2 Dedikácia metodiky

Metodika je primárne určená pre tím č. 4 na predmete Tímový Projekt I. a II. v akademickom roku 2013/2014, ale taktiež aj pre programátorov (testerov), využívajúcich programovací jazyk Java a vývojové prostredie Eclipse.

6.6.3 Použité pojmy

Tabuľka 5 - použité pojmy a ich vysvetlenie

#	Pojem	Vysvetlenie
1	jednotkový test	nízkoúrovňový test, zameraný na testovanie funkčných jednotiek (zväčša tried) zdrojového kódu
2	JUnit	framework pre testovanie zdrojových kódov písaných v jazyku Java
3	framework	programový balík nástrojov pre riešenie komplexných úloh
4	Eclipse	vývojové prostredie v ktorom využijeme framework Junit.
5	Testovacia trieda	trieda vykonávajúca testy nad testovanou triedou
6	Testovaná trieda	trieda, ktorá je testovaná testovacou triedou
7	TestCase	základná trieda z knižnice JUnit, od ktorej je odvodená jedna testovacia trieda vykonávajúca test(y) nad testovanou triedou
8	TestSuite	základná trieda z knižnice JUnit, od ktorej je odvodená jedna testovacia trieda zoskupujúca množinu testovacích tried

6.6.4 Vytvorenie jednotkového testu v prostredí JUnit

V tejto časti metodiky sú pokryté procesy, ktoré sú znázornené v tabuľke č. 5.

Tabuľka 6 - pokryté procesy

#	Názov procesu	Kapitola
1	Proces prípravy testu	5
2	Proces spustenia a vyhodnotenia testu	6

6.6.5 Proces prípravy testu

Tabuľka 7 - jednotlivé kroky v procese prípravy testu

#	Krok	Kapitola
1	Vytvorenie zdrojového priečinka a testovacieho balíka	5.1
2	Vytvorenie testovacej triedy	5.2
3	Zápis anotácie v testovacej triede	5.3
4	Vytvorenie testovacej metódy	5.4
5	Vytvorenie vhodných komentárov	5.5

6.6.6 Vytvorenie zdrojového priečinka a testovacieho balíka

Kvôli väčšej prehľadnosti testovacích tried, ktoré testujú správnosť zdrojového kódu, je nutné vytvoriť nový zdrojový priečinok a v ňom „testovací“ balík, kde sa budú triedy zoskupovať.

6.6.6.1 Vytvorenie zdrojového priečinka

V okne *Package Explorer* kliknúť pravým tlačidlom myši na projekt, v ktorom sa má testovať a v sekcii *New* zvoliť možnosť *Source Folder*. V novootvorenom okne do textového poľa *Folder Name* vložiť názov *test*.

6.6.6.2 Vytvorenie testovacieho balíka

Na novovytvorený *Source Folder* kliknúť pravým tlačidlom myši a v sekcii *New* zvoliť možnosť *Package*. V novootvorenom okne do textového poľa *Name* vložiť názov taký istý ako má balík, ktorý je testovaný.

6.6.7 Vytvorenie testovacej triedy

Každej testovanej triede prislúcha práve jedna testovacia trieda, ktorá overuje jej správnosť.

6.6.7.1 Určenie rozsahu testovania

Samotný rozsah testu je definovaný podľa scenárov testovania a tým pádom sa testuje buď trieda alebo cely balík.

6.6.8 Testovanie triedy

Kliknúť pravým tlačidlom myši na balík, v ktorom sa nachádza trieda na otestovanie a v sekcii *New* zvoliť možnosť *JUnit Test Case*. V novootvorenom okne do textového poľa *Name* vložiť názov testovacej triedy vo formáte *<názov testovanej triedy>Test* . Pokiaľ je vložený názov testovacej triedy v správnom formáte, je nutné stlačiť na tlačidlo *Finish*, po ktorom sa vytvorí testovacia trieda (dedí od triedy TestCase).

6.6.9 Testovanie balíka

Kliknúť pravým tlačidlom myši na balík, ktorý má byť otestovaný a v sekcii *New* zvoliť možnosť *JUnit Test Suite*. V novootvorenom okne do textového poľa *Name* vložiť názov testovacej triedy vo formáte *PackageTest*. Pokiaľ je vložený názov testovacej triedy v správnom formáte, je nutné stlačiť na tlačidlo *Finish*, po ktorom sa vytvorí súhrn testovacích tried (dedí od triedy TestSuite).

6.6.10 Zápis anotácie v testovacej triede

V testovacej triede slúži anotácia na popis autora, verzie a informácií o teste. Ešte pred deklaráciou samotnej testovacej triedy sa musí vytvoriť anotačný komentár.

6.6.10.1 Meno autora a názov tímu

Meno autora testu spolu s názvom tímu, v ktorom pôsobí sa do zdrojového kódu vloží vo formáte:

@author: <meno autora testu>/<názov tímu v ktorom autor pôsobí>

6.6.10.2 Verzia kódu

Verziovanie testovacích tried je určené troma číslicami oddelených bodkou. Do zdrojového kódu sa zapisuje vo formáte:

@version: <prvé číslo>.<druhé číslo>.<tretie číslo>

- prvé číslo – sa inkrementuje, keď sa spravia nekompatibilné API zmeny
- druhé číslo – sa inkrementuje, keď sa pridá funkcia, ktorá je spätne kompatibilná
- tretie číslo – sa inkrementuje pri opravení chýb, ktoré sú spätne kompatibilné

6.6.10.3 Opis testu

V priebehu písania tela testu sa opierať o konvenciu komentovania zdrojového kódu [1].

6.6.11 Vytvorenie testovacej metódy

Každá testovacia metóda prislúcha práve jednej testovanej metóde a overuje jej správnosť na viacerých odlišných vstupoch.

- Nad deklaráciou novej metódy vložiť anotáčný komentár podľa zásad JUnit [2].
- Vytvoriť testovaciu metódu s deklaráciou public void *Test<názov testovanej metódy>*
- Napísať telo testovacej metódy podľa konvencie zápisu zdrojového kódu [3].
- Použiť metódy *Assert<typ>* na porovnanie reálneho a požadovaného výstupu. Namiesto *<typ>* je nutné dosadiť funkciu, ktorá je dostupná vo frameworku JUnit [4].

6.6.12 Proces testovania triedy

Tabuľka 8 - jednotlivé kroky v procese testovania triedy

#	Krok	Kapitola
1	Spustenie testu	6.1
2	Vyhodnotenie testu	6.2
3	Zápis výsledkov	6.3

6.6.13 Spustenie testu

V zdrojovom priečinku pre testy, v testovacom balíku kliknúť pravým tlačidlom myši na testovaciu triedu a v sekcii *Run as* zvoliť možnosť *JUnit Test*.

6.6.14 Vyhodnotenie testu

Po spustení JUnit testu sa v okne, v ktorom bol doteraz zobrazený *Package Explorer*, otvorí tab s názvom *JUnit*, v ktorom sa nachádzajú všetky informácie o vykonaných testoch.

Konkrétne sú to informácie o čase vykonania, počte vykonaných testov, počte chybných testov a chýb v zdrojovom kóde. Od predošlých informácií závisí sfarbenie lišty, ktoré môže byť:

- Zelené – všetky testy zbehli bez chýb.
- Bordové – testovacia metóda dostala na výstup rozdielnu hodnotu ako očakávala.
- Červené – testovacia metóda nebola správne naimplementovaná.

Pod lištou sa nachádza zoznam testov, ktoré je možné expandovať a prezerať si jednotlivé testovacie metódy. V prípade, že lišta nie je zelená, sa pri teste a následne pri metódach, ktoré sú chybné zobrazí ikonka „x“. Kliknutím na konkrétnu chybnú metódu sa v okne *Failure Trace* zobrazí výpis o chybe. Pokiaľ sa objaví chyba alebo sa požadovaná hodnota liši od reálnej, je nutné navrhnúť spôsob riešenia týchto problémov.

6.6.15 Zápis výsledkov

Po vyhodnotení testov musí zapisovateľ uskutočniť zápis nasledujúcich poznatkov do zápisnice:

- Chybné testy spolu s informáciami, ktoré sú zobrazené v okne *Failure Trace* (viď. kapitola 6.2 Vyhodnotenie testu).
- Odlišnosť medzi požadovanými a reálnymi hodnotami spolu s návrhom spôsobu riešenia tohto problému a jeho postup.
- Vytvoriť a zapísať štatistiku testovaných modulov alebo balíčkov. Porovnať počet správnych a chybných testov s celkovým počtom testov.

6.6.16 Zoznam nadväzujúcich metodík dokumentov

[1] Konvencia komentovania zdrojového kódu

[3] Konvencia zápisu zdrojového kódu

6.7 Manažment dokumentovania

6.7.1 Dokumentovanie používateľského príbehu

Tento dokument opisuje metodiku pre tvorbu dokumentácie používateľského šprintu. Metodika obsahuje 2 procesy :

- Proces 1 – Tvorba písomnej dokumentácie
- Proces 2 – Dokumentácia pre RoboCup Wiki

6.7.1.1 Pojmy

- Dokumentácia šprintu – dokument, ktorý obsahuje všetky informácie o danom šprinte
- Wiki – portál založený na softvéri MediaWiki

6.7.1.2 Skratky

- PP – používateľský príbeh
- PpID – identifikačné číslo používateľského príbehu

6.7.1.3 Tvorba písomnej dokumentácie

Táto časť obsahuje konvencie pre tvorbu písomnej dokumentácie – konkrétne dokumentácie používateľského príbehu.

6.7.1.3.1 Nástroj na tvorbu písomnej dokumentácie

Na tvorbu dokumentácie sa použije program Microsoft Word 2010 alebo 2013.

6.7.1.3.2 Konvencie

Pre zjednodušenie spájania dokumentácie každého šprintu do jedného konzistentného dokumentu bola vypracovaná šablóna, ktorá obsahuje základné konvencie pre písanie dokumentu.

6.7.1.3.3 Názov dokumentu

Pri odovzdávaní dokumentu na spájanie je potrebné aby bol názov dokumentu nasledovný:

Autor_SprintCislo_ppID_datum

Autor: Meno autora

SprintCislo : Číslo aktuálneho šprintu

ppID : Identifikačné číslo používateľského príbehu

datum: Dátum odovzdania dokumentu

6.7.1.3.4 Štýly

Hlavná požiadavka je dodržanie zadefinovaných štýlov, ktoré sú uvedené v tabuľke 17.

Tabuľka 9 - zoznam zadefinovaných štýlov

Názov štýlu	Použitie
Normálny	Bežný text
Bez riadkovania	Tabuľka
Nadpis1	Nadpis úrovne 1
Nadpis2	Nadpis úrovne 2
Nadpis3	Nadpis úrovne 3
Nadpis4	Nadpis úrovne 4
Nadpis5	Nadpis úrovne 5
Bold	Bežný text – dôležité vyznačenie

6.7.1.3.5 Vkladanie obrázkov

Obrázky vkladať v nasledovných prípadoch:

- Zmena štruktúry projektu – diagramy
- Návrh nových súčastí systému – diagramy
- Analýza – získané obrázky z iných zdrojov
- Špecifické prípady, ktoré si vyžadujú obrázok

6.7.1.3.6 Vkladanie tabuliek

Tabuľky vkladať v nasledovných prípadoch:

- Štruktúrovaný popis nových funkcií
- Prehľad tried vytvorených v používateľskom príbehu
- Zoznam vyskytnutých problémov a návrh ich riešenia
- Špecifické prípady, ktoré si vyžadujú tabuľku

6.7.1.3.7 Štruktúra dokumentu

6.7.1.3.7.1 Úvod

Obsahuje:

- Krátky opis používateľského príbehu
- Zoznam používateľských príbehov, s ktorými je previazaný

6.7.1.3.7.2 Analýza
Obsahuje:

- Podrobná analýza pre daný používateľský príbeh

6.7.1.3.7.3 Návrh
Obsahuje:

- Návrh riešenia – diagramy, obrázky, zmeny
- Opis riešenia – podrobný opis navrhovaného riešenia

Možnosť vynechania: v prípade príbehu, ktorý sa týka iba analýzy

6.7.1.3.7.4 Implementácia
Obsahuje:

- Opis implementácie – podrobný opis implementovaného riešenia
- Tabuľka zmien – zoznam ovplyvnených tried, balíkov

Možnosť vynechania: v prípade príbehu, ktorý sa týka iba analýzy alebo návrhu

6.7.1.3.7.5 Zhodnotenie - výstup
Obsahuje:

- Krátke zhodnotenie používateľského príbehu
- Tabuľka výsledkov – celkový zoznam vykonaných zmien
- Tabuľka so zoznamov nových problémov – opísane objavené problémy

6.7.1.4 RoboCup Wiki

6.7.1.4.1 Konvencie
Pre potreby dokumentovania aktuálneho stavu na jednom mieste bol vypracovaný postup pre pridávanie relevantných častí dokumentácie na RoboCup Wiki.

6.7.1.4.2 Formátovanie

6.7.1.4.2.1 Text
- Písanie do nového riadku – vynechať jeden prázdny riadok

- Vypnutie formátovania wiki – tagy <nowiki>, </nowiki >
  Používať iba v špeciálnych prípadoch, kedy nevyhovuje formátovanie Wiki.

- Zdrojový kód – tagy <code>, </code>
  Používať pre všetky zdrojové kódy vkladané do Wiki.

6.7.1.4.2.2 Úrovne nadpisov

- Úroveň 1 : =Názov úrovne 1=
  Názov kapitoly
- Úroveň 2: ==Názov úrovne 2==
  Názov podkapitoly
- Úroveň 3: ===Názov úrovne 3===
  Názov dôležitej časti podkapitoly

6.7.1.4.2.3 Odkazy

- Externý odkaz – [http://www.ukazka.com názov odkazu]
  Používať pre všetky externé odkazy, ktoré sú zahrnuté v dokumentácii

- Interný odkaz – [[názov podstránky na wiki]]
  Používať na miestach, kde je potrebné sa odkázať na inú podstránku

- Odkaz na súbor - [[Súbor: obrázok.jpg]]

6.7.1.5 Práca s Wiki

6.7.1.5.1 Prihlásenie

- Kliknúť na systémové prihlásenie
- Zadať pridelené prihlasovacie údaje

6.7.1.5.2 Rozhodnutie

- Existujúca dokumentácia a jej úpravy – kapitoly 1.4.3.3 , 1.4.3.4
- Nová dokumentácia – kapitola 1.4.3.5

6.7.1.5.3 Výber kapitoly

- Vybrať správnu kapitolu prislúchajúcu k PP
- Vybrať podstránku pre konkrétny problém

6.7.1.5.4 Editácia stránky

- Kliknúť Upraviť pri konkrétnej časti stránky
- Pridať dokumentované časti
- Editovať pole zhrnutie úprav podľa aktuálneho stavu
- Uložiť

6.7.1.5.5 Vytvorenie novej podstránky

- Zadať novú url - wiki/index.php?title=NazovPodstranky
  Názov podstránky – musí prislúchať k dokumentovaniu používateľského príbehu
- Vybrať z ponuky hornej lišty možnosť vytvoriť
- Vykonať potrebné úpravy na stránke – pridať dokumentáciu
- Vyplniť zhrnutie úprav
- Uložiť

7 Manažment – horné metodiky

7.1 Manažment plánovania

7.1.1 Obsah metodiky

Metodika je určená pre manažment úloh, šprintov a projektov pomocou systému Jira. Systém slúži na evidenciu, sledovanie a plánovanie úloh, šprintov a projektov vo vývojárskych tímoch riadiacich sa agilnou metodikou scrum.

7.1.2 Dedikácia metodiky

Metodika je určená pre programátorské tímy, ktoré na zaznamenávanie a plánovanie práce využívajú systém Jira. A administrátorov udržujúcich systém Jira v prevádzke.

7.1.3 Nadväzujúce metodiky a dokumenty

Metodika práce so systémom Jira, Metodika dokumentácie, Metodika k riadeniu a priebehu tímových stretnutí, Metodika riadenie tímu , Metodika Scrum, Metodika administrácie systému Jira

7.1.4 Vymedzenie pojmov

Scrum– agilný spôsob vývoja v tímoch (viac v Metodike Srum)

Úloha - požiadavky zákazníka, implementačné a technické požiadavky ktoré majú byť riešené v projekte

Issue– reprezentácia úlohy v systéme Jira

Vlastník produktu – -Productowner (podľa metodikyScrum)

Zoznam úloh – zoznam všetkých zaznamenaných úloh -backlog (podľa metodiky Scrum)

7.1.5 Roly

Scrummaster - tvorba reportov, priraďovanie úloh, vytváranie a ukončovanie šprintov

Systémový administrátor – inštalácia systému, zálohovanie systému, správa verzií systému

Jira administrátor – manažment používateľských kont, riešenie používateľských problémov, udržovanie konzistentnosti obsahu systému

Vlastník produktu – pridávanie a úprava zoznamu úloh, plánovanie šprintu, prioritizácia úloh

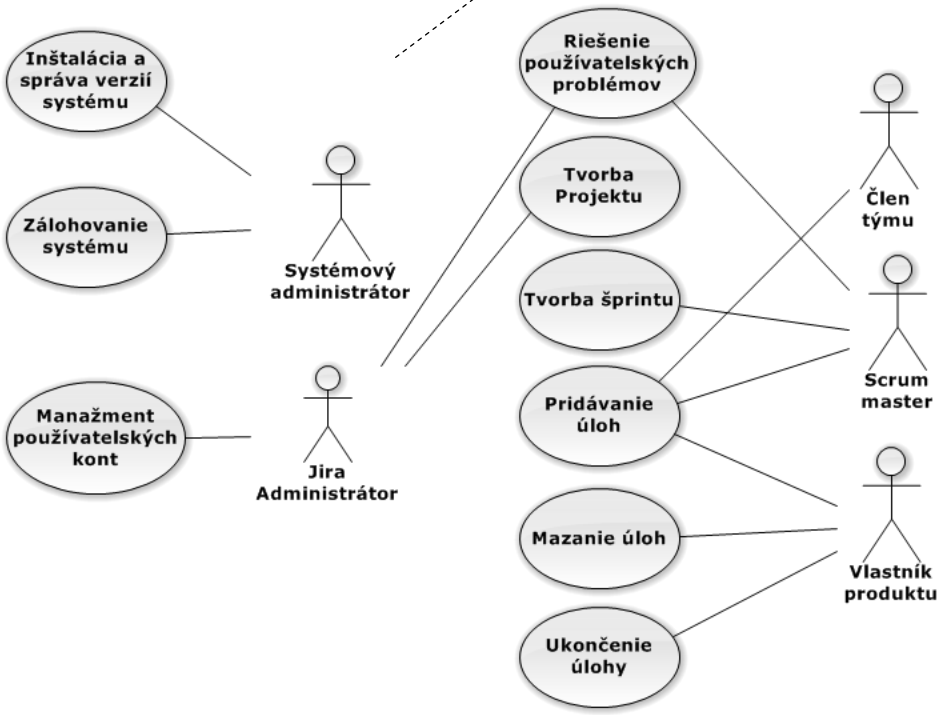
Člen týmu – práca s úlohami, zaznamenávanie času a komentárov

7.1.6 Procesy

7.1.6.1 Administrácia systému Jira

Proces administrácie systému Jira zachytáva inštaláciu, udržiavanie systému, správu verzií, používateľských kont a pokyny pre pravidelnú archiváciu databáz.

Obrázok 4 Kompetencie používatelov systému Jira



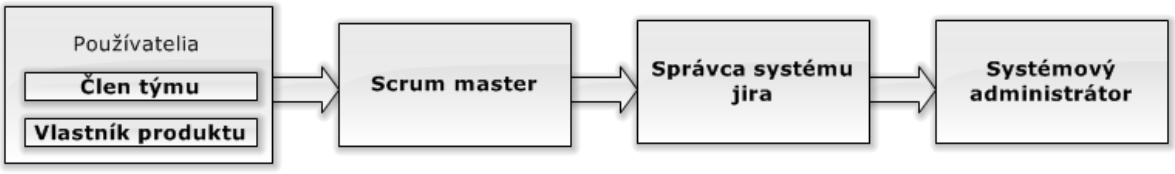
Inštaláciaa správa verziísystému – všetky inštalácie a úpravy verzii, doplnkov systému má na starosti systémový administrátor. O nastávajúcich zmenách systému je Systémový administrátor povinný informovať Jira administrátora.

Zálohovanie systému (databáz) – vykonáva Systémový administrátor v pravidelných intervaloch (Intervaly zálohovania sú stanovené v Metodike administrácie systému Jira)

Manažment používateľských kont –zmeny privilégií a tvorbu kont vykonáva Jira administrátor na základe oficiálnych požiadaviek ktoré prešlischvaľovacím procesom

Riešenie používateľských problémov -postup riešenia vzniknutého problému (ak nemá daný používateľ kompetencie riešiť problém postupuje problém ďalej)

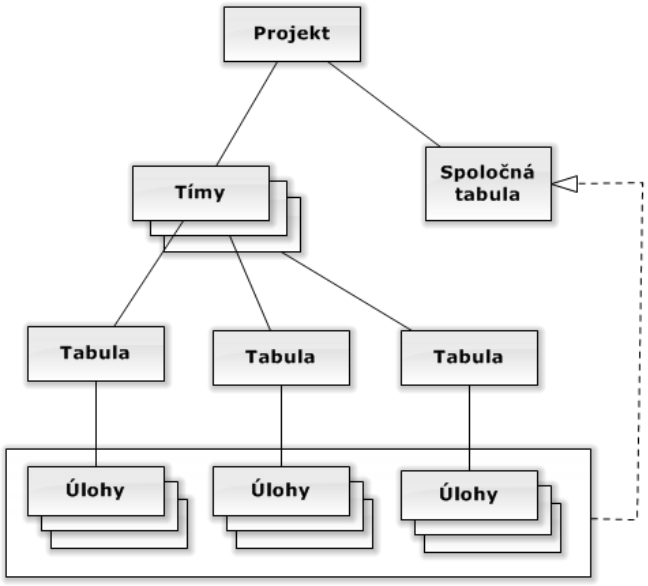
Obrázok 5 - Postup riešenia používateľských problémov



7.1.6.2 Priebeh projektu

Proces popisuje priebeh projektu a kompetencie používateľov pri jeho tvorbe a ukončení v systéme Jira.

Obrázok 6 štruktúra projektu v systéme Jira



Vytvorenie projektu, tímov a tabúl- je v kompetencii Jira administrátora na základe podkladov od vlastníka produktu

Tvorba úloh - je v kompetencii členov tímu a vlastníka produktu

Ukončenie projektu –projekt ukončuje Jira administrátor po súhlase vlastníka produktu, ktorému bola dodná potrebná dokumentácie. Dáta z projektu zálohuje Jira administrátor podľa Metodiky administrácie systému Jira.

7.1.6.3 Šprint

Proces popisuje priebeh šprintu a kompetencie používateľov pri jeho tvorbe a ukončení v systéme Jira.

Vytváranie šprintu - je v kompetencii scrummastra na základe podkladov od vlastníka produktu

Pridávanie úloh do šprintu –úlohy pridáva do šprintu scrummaster po vzájomnej do členov tímu a vlastníka produktu

Ukončenie šprintu - šprint ukončuje scrummaster po súhlase vlastníka produktu, ktorému bola dodná potrebná dokumentácie. Dáta zo šprintu zálohuje Jira administrátor podľa Metodiky administrácie systému Jira.

7.1.6.4 Priebeh úlohy

Proces priebehu úlohy zachytáva všetky stavy úlohy v ktorých sa úloha môže nachádzať. Rozhodnutie o zaradení úlohy do procesu má vlastník produktu, sledovanie napredovania a riešenia úloh má na starosti scrummaster. O akceptovaní riešenia rozhoduje vlastník produktu.

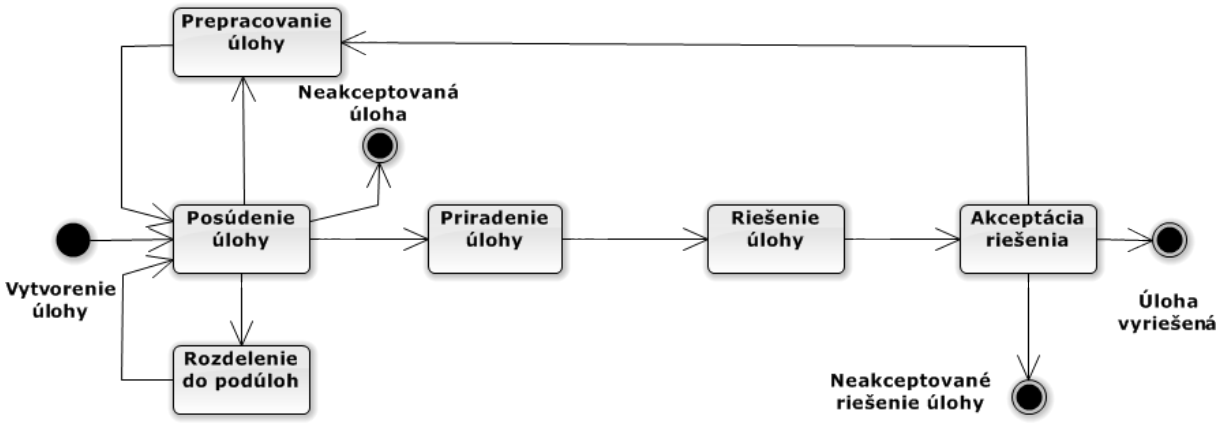
7.1.6.5 Vstupy

-požiadavky zákazníka
- implementačné požiadavky
- technické požiadavky

7.1.6.6 Výstup

- vyriešená / nevyriešená úloha

Obrázok 7 - možné stavy úloh



Vytvorenie úlohy – úlohy sa vytvárajú na základe požiadaviek a potrieb zákazníka, vlastníka produktu alebo členov týmu, tieto požiadavky a potreby sa zadávajú do systému formou issue do zoznamu úloh.

Posúdenie úlohy– úlohy posudzuje výhradne vlastník produktu, ktorý úlohám následne prideľuje prioritu pre ich riešenie.

Prepracovanie úlohy – neakceptovateľné požiadavky pre riešenie úlohy dáva vlastník produktu na prepracovanie členom týmu, prípadne sám konzultuje zmeny so zadávateľom úlohy.

Rozdelenie do podúloh – úlohy ktoré nemožno vyriešiť v jednej iterácií šprintu prípadne sú príliš zložité by mali byť rozdelené na menšie a prehľadnejšie podúlohy

Priradenie úlohy–členovia tímu si úlohy výhradne prideľujú samy podľa priorít stanovených vlastníkom produktu, vo výnimočných prípadoch môže úlohy priradiť vlastník produktu

Riešenie úlohy – scrummaster má dohlaď nad stavom a napredovaním úlohy

Akceptácia riešenia – vlastník produktu po vyriešení úlohy a dodaní všetkých potrebných dokumentov vyhodnotí stav riešenia a následne úlohu akceptuje, dá na prepracovanie, alebo zamietne

7.2 Manažment podpory vývoja (verziovanie)

7.2.1 Obsah metodiky

Metodika je určená pre manažment verzií vo verzionovacom systéme GIT. Systém GIT slúži na ukladanie (a zálohovanie) verzií projektov, dokumentov, zdrojových kódov.

7.2.2 Dedikácia metodiky

Metodika je určená pre tímy vývojárov, ktorí vytvárajú rozsiahle systémy, moduly a.i., ktoré je nutné verzionovať pre zaistenie archivácie a zálohovania kódu.

7.2.3 Slovník pojmov

Pojem	Význam
Branch	vývojová vetva
Merge	spojenie vývojových vetiev
Repozitár	úložisko zdrojového kódu
Feature	Menšia (menej zložitá) funkcionalita
Master branch	hlavná vývojová vetva, obsahuje jednotlivé verzie produktu
Development branch	vývojová vetva, ktorá obsahuje vývoj zložitejšej funkcionality
Feature branch	vývojová vetva, ktorá obsahuje menej zložitú funkcionalitu (väčšinou podúloha, fix, hotfix)
Fix	Oprava chybnej funkcionality
Hotfix	Oprava chybnej funkcionality na produkčnej verzií

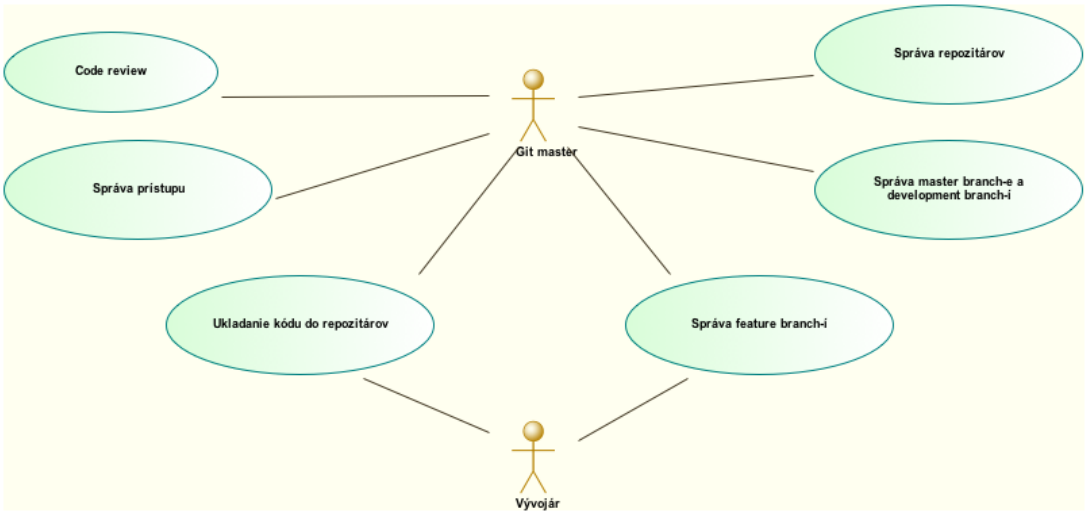
7.2.4 Roly

Roly, ktoré sme definovali sú popísané v tabuľke 4 a graficky znázornené v diagrame na obrázku 8.

Tabuľka 10 - Popis úloh pre definované roly

Rola	Úlohy
Git Master	• Správa repozitárov • Správa master branch-e a development branch-í • Review kódu (Podľa metodiky) • Označovanie branch-í a commit-ov • Úlohy vývojára viď. nižšie • Správa prístupu k repozitárom
Vývojár	• Ukladanie kódu do repozitárov (vývoj, hotfix, fix) • Správa feature branch-í

Obrázok 8 - Use case diagram pre definované roly



7.2.5 Procesy

7.2.5.1 Príchod nového vývojára

Pri príchode nového vývojára do vývojového tímu je potrebné aby si nainštaloval verzionovací systém GIT a poslal Git master-ovi prístupoví SSH kľúč.

Git master vytvorí prístup do repozitárov, na ktorých bude nový vývojár pracovať.

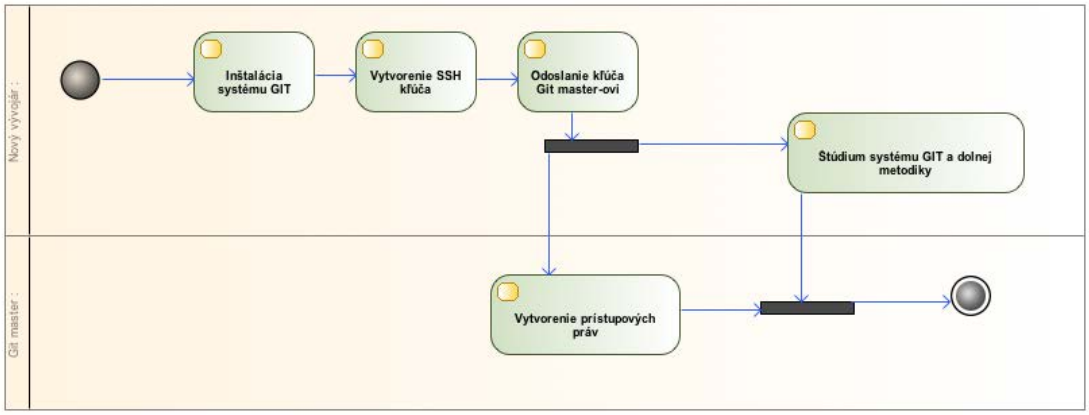
Vývojár má povinnosť oboznámiť sa s prácou s verzionovacím systémom GIT a to formou návodov, dokumentácie systému a dolnej metodiky práce so systémom GIT.

Vstup: nový vývojár.

Výstup: vývojár oboznámený s verzionovacím systémom GIT a prístupom do repozitáru

Zodpovedný: Git master

Obrázok 9 - činností pri príchode nového vývojára



7.2.5.2 Vytvorenie projektového repozitára

Pri vývoji na novom projekte alebo module je potrebné vytvoriť repozitár. Git master vytvorí vo verzionovacom systéme GIT nový repozitár, ktorému správne nastaví prístupové práva (Tabuľka 5).

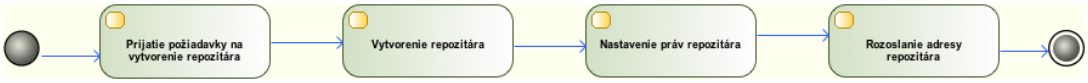
Po vykonaní predchádzajúcich úkonov odošle Git master vývojárom adresu repozitára.

Vstup: Požiadavka na vytvorenie repozitára.

Výstup: Vytvorený repozitár s nastavenými prístupovými právami.

Zodpovedný: GIT master

Obrázok 10 - Diagram činností pri vytváraní nového repozitára



Tabuľka 11 - Práva pre zápis a odvodenie

Rola	Branch					
	Master		Development		Feature	
	Zápis	Odvodenie	Zápis	Odvodenie	Zápis	Odvodenie
Git master	Áno	Áno	Áno	Áno	Áno	Áno
Vývojár	Iba hotfix	Nie	Áno	Áno	Áno	Áno

7.2.5.3 Ukončenie vývoja a deployment projektu

Merge feature branch-í do development branch-e

Vývojári ukončia vývoj a zapíšu zmeny do feature branch-e. Danú feature branch merge-nú do development branch-e, kde svoju funkcionalitu otestujú.

Git master vykoná review kódu (priebežne ju musí vykonávať aby na konci vývoja nemusel robiť rozsiahlejšiu review). V prípade, že odhalí chybu obráti sa na vývojára, ktorý vykoná fix priamo v development branch-i.

Merge development branch-e do master branch-e

Git master po záverečnom otestovaní a vykonaní review kódu, merge-ne danú development branch-u do master branch-e.

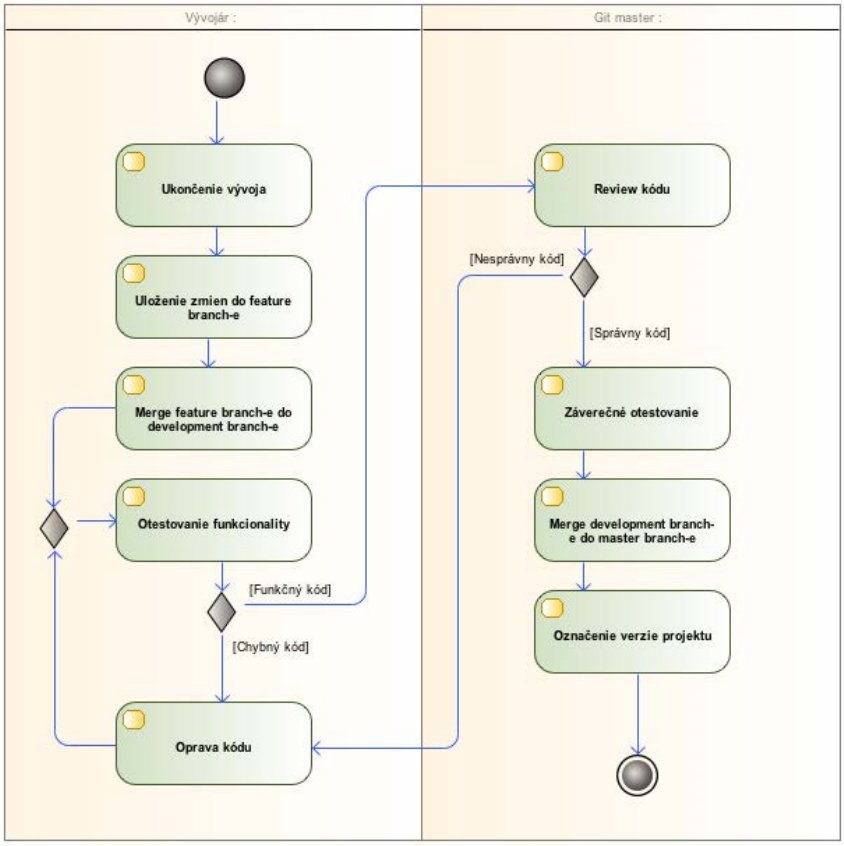
Po merge-i vytvorí tag s príslušnou verziou podľa spodnej metodiky.

Vstup: Požiadavka pre ukončenie vývoja alebo deployment.

Výstup: Funkčné zmeny uložené v master branch-i.

Zodpovedný: GIT master, vývojári

Obrázok 11 - Diagram činností pri ukončovaní vývoja



7.2.5.4 Vytvorenie development branch-e

Pri vývoji na novej (zložitejšej) funkcionalite projektu je potrebné vytvoriť novú development branch-u.

Git master vytvorí vo verzionovacom systéme GIT v príslušnom repozitári novú branch-u, ktorá bude obsahovať názov funkcionality (change request-u / content update-u).

Vstup: Požiadavka pre vytvorenie novej development branch-e.

Výstup: Nová branch pre vývoj novej funkcionality.

Zodpovedný: GIT master

7.2.5.5 Vytvorenie feature branch-e

Pri vývoji na novej (jednoduchšej) funkcionalite, ktorá môže ovplyvniť funkcionalitu inej časti projektu je potrebné vytvoriť novú feature branch-u.

Vývojár vytvorí svoju feature branch-u z development branch-e ktorá zahŕňa danú pod-funkcionalitu.

Vstup: Potreba oddelenia jednoduchšej funkcionality (napr.: kvôli funkčnosti iných komponentov).

Výstup: Nová branch pre vývoj novej (feature) funkcionality.

Zodpovedný: Vývojár

7.3 Manažment podpory vývoja (perconik)

7.3.1 Zavedenie štábnej kultúry

Vstup : *Zdrojový kód programu s chybami ktorý ešte nebol kontrolovaný.*

Výstup: *Zdrojový kód ktorý bude mať identifikované časti ktoré treba nejakým spôsobom doplniť, upraviť alebo len označiť za správne implementované.*

7.3.2 Dedikácia metodiky

Metodika sa zaoberá procesmi ohľadne správneho prechádzania zdrojového kódu. Obsahuje hlavné procesy ako sú :

#	Krok	Kapitola
1	Proces inštalácia PerConIK	7.3.3.1
2	Proces pridania funkcionality.	7.3.3.2
3	Proces vyriešenia chyby v kóde.	7.3.3.4
4	Proces CodeReview	7.3.3.5

7.3.3 Role

Člen tímu - Programátor ktorý pridáva do programu nové časti kódu spolu zo značkami.

Manažér podpory vývoja - Člen tímu zodpovedný za správny vývoj softvéru, snaží sa identifikáciu chýb v kóde, neimplementovaných častí kódu.

Manažér kontrola kvality - Člen tímu zodpovedný za kvalitu kódu, realizuje review kódu.

7.3.3.1 Proces inštalácie PerConIK

Na zlepšenie vývoja softvéru používame nástroj PerConIK ktorý nám umožňuje pridávať značky do zdrojového kódu programu. Preto pri pridávaní nového člena do tímu je potrebné zabezpečiť aby si dobre nainštaloval program PerConIK.

Vstup : *Eclipse bez pluginu PerConIK*

Výstup : *Funkčný PerConIK s validáciu značiek v Eclipse.*

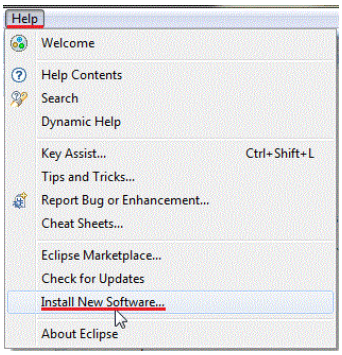
Zodpovedný: *Manažér podpory vývoja.*

#	Krok	Kapitola
1	Proces inštalácie PerConIK-u do Eclipsu	4.1
2	Nastavenie PerConIK-u	4.2
3	Proces zapnutia validácie značiek.	4.3

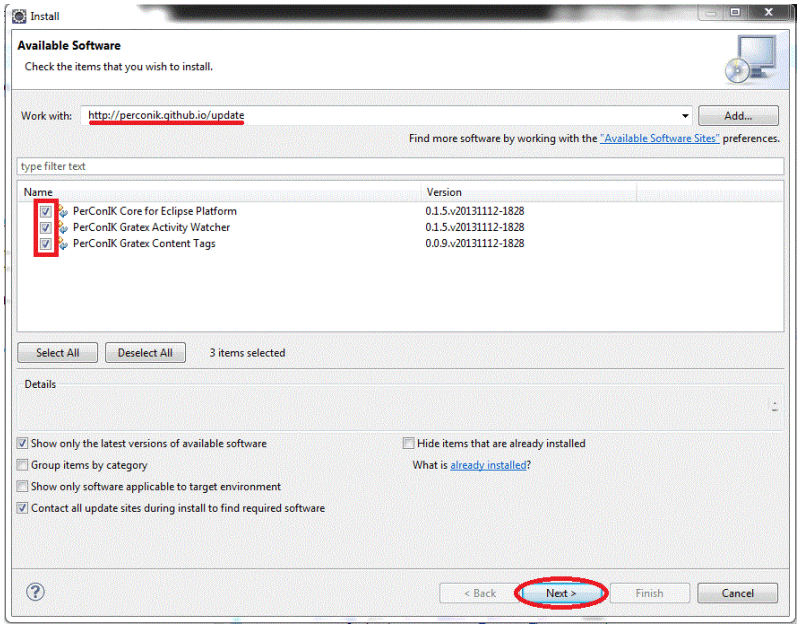
7.3.3.2 Proces inštalácie PerConIK-u do Eclipsu

Pre správnu inštaláciu treba zvoliť v hornej lište Eclipsu položku *Help → Install New Software...*. tak ako je vidieť na obrázku 12 . Zobrazí sa okno. Do položky *Work With* zadajte adresu http://perconik.github.io/update. Zobrazí sa zoznam s ktorého vyberiete všetky položky a kliknete na next. Toto okno je zobrazená aj na obrázku 13 (v pravo). Následne postupujte podľa inštrukcií.

Obrázok 12 - Inštalácia PerConIK-u.



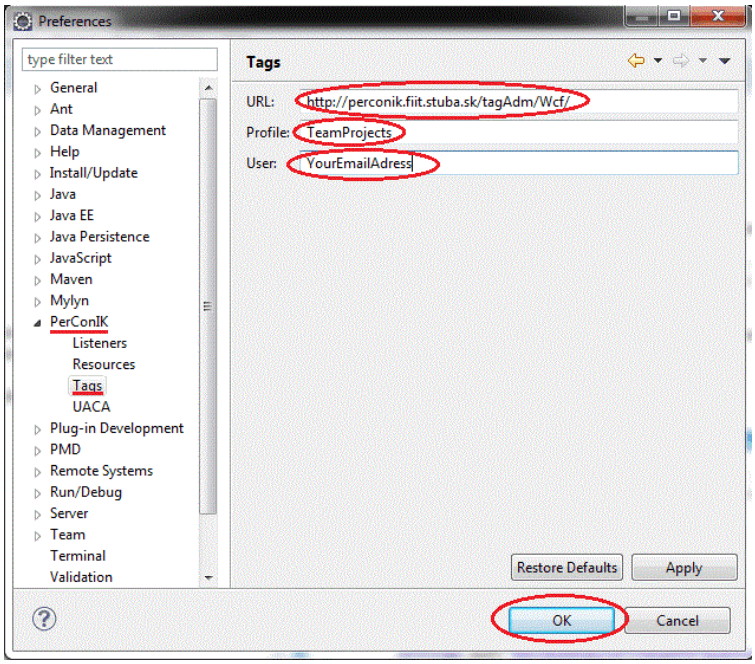
Obrázok 13 - Inštalácia PerConIK-u.



7.3.3.3 Nastavenie PerConIK-u

Pre správne fungovanie programu je potrebné nastaviť prihlasovacie údaje do repozitára PerConIK-u podľa obrázka 14. Toto nastavenie nájdete v menu pod *Window → Preferences.*

Obrázok 14 - Nastavenie údajov do repozitára.



7.3.3.4 Proces zapnutia validácie značiek.

Pravým kliknutím na projekt v ktorom chcete značky validovať zobrazíte lištu s ktorej vyberiete *PerConIK Tools → Enable Tags Nature*. V kóde sa vám označia nesprávne napísané značky.

7.3.4 Proces pridania funkcionality.

Dôležitý proces ktorý vykonáva každý člen týmu ktorý určitú funkcionalitu pri implementácií vynechá, nedokončí. Tento proces sa týka priamo značky TODO a môžeme ho rozložiť na nasledujúce pod-procesy:

#	Krok	Kapitola
1	Identifikovanie neimplementovanie funkcionality.	7.3.4.1
2	Pridelenie solver-a.	7.3.4.2
3	Implementácia funkcionality.	7.3.4.3
4	Posun funkcionality na review	7.3.4.4

Vstup : *Program bez implementovanej funkcionality.*

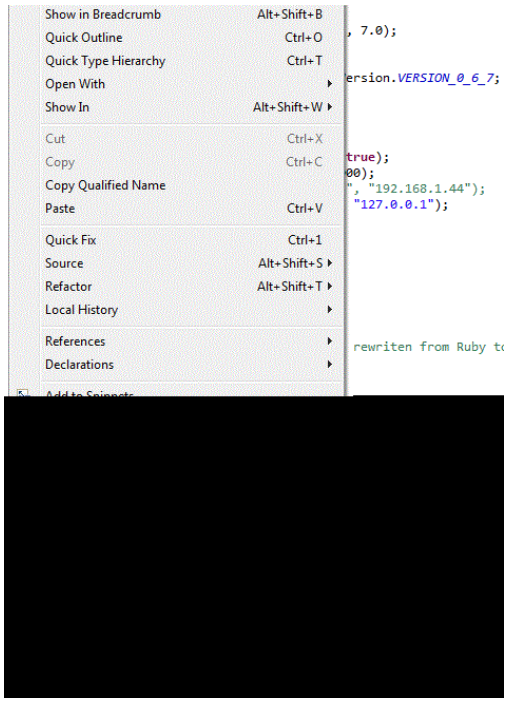
Výstup : *Program s implementovanou funkcionalitou.*

Zodpovedný: *Celý tím.*

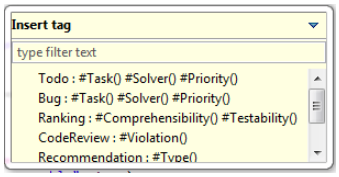
7.3.4.1 Identifikovanie neimplementovanie funkcionality.

Proces pri ktorom sa do zdrojovom kódu pridá značka TODO. Označíte časť kódu kde chceme implementovať funkcionalitu. Pravým kliknutím na vyznačenú oblasť sa nám zobrazí zoznam ktorý je vidno na obrázku číslo 15 zvolíme *PerConIK Tools → Insert Tag*. Zobrazí sa nám výber všetkých značiek, z ktorých zvolíme značku TODO. Jej Opis je v metodike: "Metodika na pridávanie značiek pomocou nástroja PerConIK".

Obrázok 15 - Pridanie požadovanej značky značky.



Obrázok 16 - Pridelenie solver-a.



Jedná sa o jednoduchý proces kedy člen tímu ktorý sa práve chystá doplniť funkcionalitu. Doplní do značky údaj #Solver(x) kde x bude jeho email. Dôraz sa kladie nato aby každý člen tímu si zobral sám funkcionalitu keďže tým je riadený pomocou agilnej metódy scrum.

7.3.4.2 Implementácia funkcionality.

V tomto procese sa snaží programátor implementovať danú úlohu. Dôležité je poznamenať, že nato aby mohol začať implementovať danú funkcionalitu musí pridať značku #Solver() proces 5.2, aby nedochádzalo k situáciám, kedy dvaja programátori riešia jednu úlohu. Za správne implementované sa považuje taká úloha ktorá je otestovaná a funkčná.

7.3.4.3 Posun funkcionality na review.

Po implementovaní a otestovaní členom tímu bude potrebné vymazať značku TODO a vytvoriť novú značku Recommendation z údajom #Type(other) a textom za značkou CodeReview aby iný člen tímu vedel o tom, že tejto časti kódu treba spraviť review. Taktiež je potrebné aby koniec párovej značky TODO upravil tak aby obsahovala všetky implementované časti.

Obrázok 17 - Príklad posunu funkcionality na review.

```
//Recommendation #Type(Other) CodeReview | sppred@gmail.com 2013-12-08T14:43:07.6920000Z
    void implementovaná_funkcionalita(){
    }
//@< | sppred@gmail.com 2013-12-08T14:43:07.6920000Z
```

7.3.5 Proces vyriešenia chyby v kóde.

Proces pri ktorom je identifikovaná, spracovaná a vyriešená chyba v zdrojovom kóde programu.

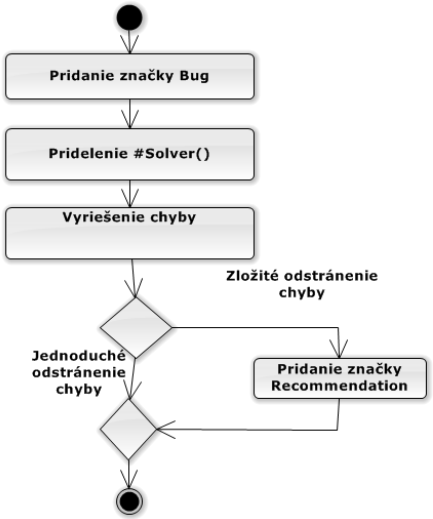
#	Krok	Kapitola
1	Identifikovanie Chyby.	7.3.5.1
2	Pridelenie solver-a.	7.3.5.2
3	Označenie chyby za vyriešenú.	7.3.5.3

Vstup : *Program s chybou .*

Výstup : *Program bez chyby.*

Zodpovedný: *Manažér podpory vývoja.*

Obrázok 18 - diagram Odstraňovania chýb v kóde



7.3.5.1 Identifikovanie Chyby.

Je potrebné aby chyba v programe bola označená a bola jej vhodne určená priorita. Pre tento účel sa používa značka z PerConIK-u Bug. Jej Opis je v metodike: "Metodika na pridávanie značiek pomocou nástroja PerConIK". Značka sa pridáva podobne ako značka TODO ktorá je opísaná v kapitole 5.1.

7.3.5.2 Označenie chyby za vyriešenú.

Ak sa jednalo o jednoduchú chybu ktorá nemenila veľkú časť kódu jednoducho vymaže značka bug.

7.3.6 Proces CodeReview.

Proces pri ktorom sa snažíme identifikovať porušenie konvencií pri vytváraní tohto kódu. Tento proces je podrobnejšie špecifikovaný v dokumente : "Horná metodika pre tvorbu, údržbu a dokumentáciu kódu" od Mateja Bádala. Z pohľadu životné procesu značiek CodeReview obsahuje nasledujúce pod-procesy:

#	Krok	Kapitola
1	Proces odporúčaného CodeReview	7.3.6.1
2	Proces pravidelného CodeReview	7.3.6.2

Vstup : *Program z nekvalitným kódom.*

Výstup : *Program z kvalitnejším kódom.*

Zodpovedný: *Manažér kontrola kvality.*

7.3.6.1 Proces odporúčaného CodeReview

Proces ktorý sa vykonáva vždy ak manažér kvality nájde v kóde značku Recommendation z textom CodeReview. V skratke sa dá povedať že vymaže túto značku a nahradí ju značkou CodeReview. Člen týmu ktorý danú časť kódu vyvíjal následne opraví nedostatky a odstráni značku CodeReview.

7.3.6.2 Proces pravidelného CodeReview

Podobný proces ako proces odporúčaného code review s tým že manažér sám prechádza kód a hľadá v ňom nedostatky.

7.4 Manažment kvality

7.4.1 Úvod

Nasledujúci dokument obsahuje metodiku manažérskej úrovne, pre tvorbu, údržbu a dokumentáciu zdrojových kódov s dôrazom na prehľadnosť a zrozumiteľnosť.

7.4.2 Dedikácia metodiky

Metodika je určená pre všetkých členov tímu. Taktiež je určená pre manažéra kvality a manažéra plánovania.

7.4.3 Obsah metodiky

Metodika postupne opisuje jednotlivé procesy z pohľadu manažérskej úrovne. Pri každom procese je uvedené, kto je zodpovedný za daný proces. Nasledujúca tabuľka obsahuje zoznam všetkých procesov, ktoré sú v metodike opísané.

Tabuľka 12 - procesy

#	Názov procesu
1	Príchod nového člena
2	Práca na projektoch
3	Code review
4	Dohodnutie meetingu
5	Meeting
6	Pridelenie úlohy prepracovania kódu
7	Prepracovanie kódu
8	Overenie zmien

7.4.4 Nadväzujúce dokumenty

S dokumentom súvisia ďalšie dokumenty, ktorých účel je buď doplniť túto metodiku, alebo ju ďalej rozviesť a bližšie opísať procesy, ktoré prechádzajú aj do iných oblastí.

- **[1] - Metodika pre tvorbu, údržbu a dokumentáciu kódu** – Dolná metodika, ktorá opisuje konkrétne postupy pri tvorbe údržbe a dokumentácii kódu. Dolná metodika by mala byť podkladom, z ktorého sa vychádza pri procesoch opísaných v nasledujúcej hornej metodike.

7.4.5 Slovník pojmov

Code review – Proces, pri ktorom sa skúma kvalita vytvoreného kódu z celkového pohľadu, ale aj z pohľadu zavedených interných konvencií.

Code review meeting – stretnutie programátorov a ľudí zodpovedných za kvalitu kódu, na toto stretnutí sú prebrané chyby a spôsoby ich odstránenia

7.4.6 Opis rôl

Manažér kvality – Osoba zodpovedná za kvalitu a finálny stav kódu, jeho úlohou je takisto code review, vedie meeting, kde sa rozoberajú jednotlivé chyby.

Manažér plánovania – Osoba zodpovedná za naplánovanie a rozdelenie jednotlivých úloh v tíme.

Člen tímu – Programátor, ktorý prispieva svojimi schopnosťami pri vývoji softvéru, musí sa riadiť konvenciami vymedzenými v dolnej metodike

7.4.7 Metodika

7.4.7.1 Príchod nového člena tímu

Pri príchode nového člena tímu je nutné, aby sa nový člen čo najskôr oboznámil s konvenciami, ktoré sa v danom tíme používajú pri tvorbe zdrojového kódu. Na tento účel slúži dolná metodika [1], ktorú tím musí mať vo vytlačenej forme. Nový člen po vybavení formálnych záležitostí dostane dolnú metodiku, ktorú si musí naštudovať. Obdržanie tejto metodiky má na starosti manažér kvality. Nový člen bude riešiť prípadne otázky, alebo nezrovnalosti takisto s manažérom kvality.

Vstup – nový člen tímu

Výstup – zaučený člen tímu

Zodpovedný – manažér kvality

7.4.7.2 Práca na projektoch

Člen tímu pracuje na pridelených úlohách. Pri vytváraní zdrojového kódu sa musí riadiť konvenciami zapísanými v dolnej metodike [1], ako aj svojím úsudkom, tak aby pri tejto tvorbe nevytváral anti-vzory. Vytvorený kód musí správne zdokumentovať v súlade so zavedenými konvenciami.

Vstup – požiadavka na vypracovanie úlohu

Výstup – vytvorený zdrojový kód

Zodpovedný – člen tímu

7.4.7.3 Code review

Manažér kvality musí v pravidelných intervaloch vykonávať code review. Dĺžka týchto intervalov závisí na internej dohode tímu. V prípade šprintu, ktorý trvá dva týždne, sa code review musí vykonávať minimálne raz počas behu tohto šprintu, a taktiež na konci šprintu.

Code review spočíva v tom, že manažér kvality preskúma vytvorený kód a snaží a hľadá v ňom časti, ktoré sú v rozpore s dolnou metodikou [1], ale aj anti-vzory vo všeobecnosti.

Vstup – vytvorený zdrojový kód

Výstup – overený zdrojový kód

Zodpovedný – manažér kvality

7.4.7.4 Dohodnutie meetingu

Ak je pri procese code review odhalených viacero chýb, prípadne chyba, ktorá sa viackrát opakuje, je potrebné stretnutie členov tímu, na ktorom sa tieto chyby rozoberú a ukážu sa aj možné riešenia. Dohodnutie tohto meetingu má na starosti manažér plánovania, ktorý určí termín, kedy sa takýto meeting bude konať. Na meetingu sa musí zúčastni väčšina členov tímu, ktorí vytváranú zdrojové kódy, tomu sa tento termín musí prispôsobiť.

Vstup – požiadavka na dohodnutie meetingu

Výstup – naplánovaný meeting

Zodpovedný – manažér plánovania

7.4.7.5 Meeting

Manažér kvality si pripraví prezentáciu, v ktorej poukáže na chyby nájdené počas code review. Ku každej chybe naznačí možné riešenie, alebo sa odkáže na dolnú metodiku [1]. Identifikované chyby spíše aj s ich autormi a poskytne ich manažérovi plánovania.

Vstup – požiadavka na uskutočnenie meetingu

Výstup – uskutočnený meeting

Zodpovedný – manažér kvality

7.4.7.6 Pridelenie úlohy prepracovania kódu

Manažér plánovania jednotlivým osobám zo zoznamu s chybami naplánuje úlohu na prerobenie týchto chýb.

Vstup – zoznam chýb s ich autormi

Výstup – pridelená úloha na prepracovanie kódu

Zodpovedný – manažér plánovania

7.4.7.7 Prepracovanie kódu

Člen tímu prepracuje kód, kde sa nachádza chyba. Ako podklad pri tom použije dokument s dolnou metodikou [1]. V prípade, že ide o väčšiu zmenu, konzultuje riešenie s manažérom kvality. Po oprave kódu informuje o tejto skutočnosti manažéra kvality.

Vstup – požiadavka na prepracovanie kódu

Výstup – prepracovaný kód

Zodpovedný – člen tímu

7.4.7.8 Overenie zmien

Manažér kvality dostane informáciu o tom, že kód s chybami bol prerobený. Opäť vykoná

code review, ale len na kóde, ktorý bol prepracovávaný. V prípade, že identifikuje ďalšie problémy, znovu spíše ich zoznam s autorom a poskytne ho manažérovi plánovania na prerobenie. Vysvetlenie chýb a návrh riešenia v tomto prípade rieši individuálne s členom tímu.

Vstup – notifikácia o prepracovaný kód

Výstup – overený kód

Zodpovedný – manažér kvality

7.5 Manažment riadenia

7.5.1 Úvod dokumentu

Tento dokument opisuje manažment riadenia vývoja softvéru pomocou agilnej metodológie scrum. Dokument je vhodný pre menšie tými pracujúce na nie veľmi rozsiahlych softvérových projektoch.

7.5.2 Slovník pojmov

Scrum

Inkrementálna a iteratívna metodika pre agilný vývoj softvérového produktu

Šprint

Základná časová jednotka metodiky scrum. Šprint je časovo ohraničené obdobie, vopred stanovenej dĺžky, v ktorom sú vypracovávané úlohy.

UserStory

Zachytáva požiadavky zákazníka na produkt vo forme Čo, Kto a Prečo – Čo je funkcionalitou, Kto s tým bude pracovať a Prečo je to treba implementovať

7.5.3 Roly

Jednotlivé role a popis účastníkov, ktorí vystupujú v manažmente riadenia metodiky Scrum sú opísané v tabuľke 8.

Tabuľka 13 - Role a popis účastníkov Scrum-u

Rola	Popis
ProductOwner	• Reprezentuje zákazníka • Zadáva a upravuje požiadavky • Udáva prioritu požiadavkám • Hodnotí splnenie/nesplnenie úloh
ScrumMaster	• Člen tímu • Zabezpečuje správny priebeh šprintu

	• Moderuje tímové stretnutia
Vývojový tím	• Je tvorený študentmi • Je zodpovedný za vývoj softvéru • Hodnotí náročnosť úloh
Člen týmu	• Študent • Vyberá si úlohu, na ktorej chce pracovať • Je zodpovedný za splnenie úlohy

7.5.4 Elementy

V Tabuľke 9 sú popísané základné elementy, ktoré sú využívané pri riadení projektu pomocou metodiky Scrum.

Tabuľka 14 - Elementy Scrum Metodiky

Element	**Popis**
ProductBacklog	• Zoznam požiadaviek zoradený podľa priority • Reprezentuje čo treba spraviť a v akom poradí • Správcom je ProductOwner • ProductOwner určuje priority
Požiadavka	• Predstavuje funkcionalitu alebo vlastnosť produktu, ktorú treba vytvoriť • Je písaná ako “ userstory“ • Vytvára a upravuje ProductOwner • Je pomerne abstraktná • Typicky sa delí na viacero úloh
SprintBacklog	• Zoznam úloh pre jeden šprint • Vzniká odoberaním požiadaviek z ProductBacklog-u a rozdelením na jednotlivé úlohy • Požiadavky si vyberá vývojový tím
Úloha	• Má konkrétne znenie • Úlohu si vyberá aj vypracováva člen týmu • Vzťahuje sa ku konkrétnej požiadavke
Tímové stretnutia	• Konajú sa raz za týždeň • 3 typy podľa toho v ktorej fáze šprintu sa uskutočňujú
ProductIncrement	• Predstavuje súbor všetkých dokončených požiadaviek v šprinte

7.5.5 Riadenie vývoja softvéru

Na sledujúce podkapitoly opisujú procesy vykonávané v rámci metodiky Scrum. Každý proces má svoj vstup a výstup a tiež roly účastníkov ktoré sa daného procesu účastina.

7.5.6 ProductBacklog

7.5.6.1 Pridanie požiadavky

Role:ProductOwner

Vstup: Prázdny alebo z časti zaplnený ProductBacklog

Výstup:ProductBacklog rozšírený o novú požiadavku

Na základe identifikovaných vlastností, ktoré by mal produkt spĺňať, pridá ProductOwner novú požiadavku do ProductBacklog-u. Pridávanie požiadaviek je možné v ktorejkoľvek fáze vývoja.

7.5.6.2 Definícia požiadavky

Role:ProductOwner

Vstup:ProductBacklog

Výstup: ProductBacklog s presne definovanými požiadavkami

Každá požiadavka musí mať jednoznačne určený názov, popis, prioritu a podmienky splnenia. Všetky vlastnosti požiadavky určuje ProductOwner. Názov jednoznačne určuje požiadavku. Popis má formu "userstory". Priorita je určená na základe dôležitosti pre výsledný produkt. Podmienky splnenia definujú kedy sa úloha považuje za splnenú (napr.: novovytvorený kód musí prejsť unit testami).

7.5.6.3 Zmena požiadavky

Role: ProductOwner

Vstup: ProductBacklog

Výstup: ProductBacklog s upravenou požiadavkou

Zmenu požiadavky má v kompetencii ProductOwner. Zmeniť môže všetky vlastnosti danej požiadavky (názov, popis, prioritu, podmienky splnenia), ak požiadavka ešte nie je vykonávaná.

7.5.6.4 Odstránenie požiadavky

Role:ProductOwner

Vstup:ProductBacklog

Výstup:ProductBacklog bez odstránenej požiadavky

ProductOwner môže odstrániť požiadavku z ProductBacklog-u ak bola upravená špecifikácia produktu a daná požiadavka už v novej špecifikácii nehrá rolu.

7.5.7 Šprint Backlog

7.5.7.1 Výber požiadaviek

Role:ProductOwner

Vstup:ProductBacklog

Výstup: Šprint Backlog

Na začiatku každého šprintu vyberá ProductOwner požiadavky z ProductBacklog-u a tým vytvorí Šprint Backlog. Požiadavky sú vyberané na základe priority.

7.5.7.2 Rozdelenie úloh

Role: Členovia tímu

Vstup. Šprint Backlog

Výstup: Rozdelené úlohy medzi členov tímu

Členovia tímu rozdelia každú požiadavku zo Šprint Backlog-u na menšie podúlohy. Potom si každý člen tímu vyberie úlohu na ktorej chce pracovať.

7.5.7.3 Vloženie úloh do plánovacieho systému

Role: Člen tímu – manažér plánovania

Vstup: Zoznam úloh s riešiteľmi

Výstup:Úlohy vložené do plánovacieho systému

Manažér plánovania tvorí zoznam úloh s riešiteľmi počas delenia úloh. Následne vkladá plánované úlohy do systému kde každý člen tímu vidí aké úlohy má pridelené a do kedy ich treba spraviť.

7.5.7.4 Reportovanie úloh

Role: Členovia tímu

Vstup: Úloha v plánovacom systéme

Výstup: Úloha v plánovacom systému s reportom

Každý člen tímu musí reportovať postup práce na úlohách v plánovacom systéme. Reporty je nutné podávať hneď po ukončení práce na úlohe aj keď ešte úloha nie je hotová. Reportovaním sa sleduje čas strávený na úlohe a percentuálny odhad dokončenia úlohy.

7.6 Manažment testovania

7.6.1 Úvod

Predmetom tejto metodiky je určenie procesov prebiehajúcich počas celej fázy testovania. Taktiež definuje zodpovedné osoby spolu so vstupným a výstupným výsledkom každého procesu. Táto metodika pokrýva manažment kvality a testovania.

7.6.2 Dedikácia metodiky

Metodika je primárne určená pre tím č. 4 na predmete Tímový Projekt I. a II. v akademickom roku 2013/2014, ale taktiež aj pre programátorov (testerov), využívajúcich programovací jazyk Java a vývojové prostredie Eclipse.

7.6.3 Použité pojmy

Tabuľka 15 - použité pojmy a ich vysvetlenie

#	Pojem	Vysvetlenie
1	jednotkový test	nízkoúrovňový test, zameraný na testovanie funkčných jednotiek (zväčša tried) zdrojového kódu
2	JUnit	framework pre testovanie zdrojových kódov písaných v jazyku Java
3	framework	programový balík nástrojov pre riešenie komplexných úloh
4	Eclipse	vývojové prostredie v ktorom využijeme framework Junit.
5	Testovacia trieda	trieda vykonávajúca testy nad testovanou triedou
6	Testovaná trieda	trieda, ktorá je testovaná testovacou triedou
7	TestCase	základná trieda z knižnice JUnit, od ktorej je odvodená jedna testovacia trieda vykonávajúca test(y) nad testovanou triedou
8	TestSuite	základná trieda z knižnice JUnit, od ktorej je odvodená jedna testovacia trieda zoskupujúca množinu testovacích tried

7.6.4 Manažment testovania

Software je nutné testovať v každej fáze vývoja. Táto kapitola obsahuje celý proces testovania v rámci vývojového prostredia Eclipse, jazyku java a jednotkového testovania za pomoci frameworku JUnit. Taktiež sú tu obsiahnuté roly a zodpovednosti jednotlivých účastníkov.

7.6.5 Roly zahrnuté do procesov testovania

Tabuľka 16 - roly obsiahnuté v procese testovania a ich zodpovednosti

#	Rola	Zodpovednosti
1	Vedúcitestovania	Koordinácia testovacieho tímu Vedenieceléhoprocesutestovania Rozdeľovanie a prideľovanie jednotlivých úloh Vyhodnocovanie celého procesu testovania
2	Vývojár	Vypracovanie testov pre triedy, ktoré boli určené na testovanie Navrhovanie riešení pri odhalení chýb pri procese testovania
3	Tester	Vypracovanie testov pre triedy, ktoré boli určené na testovanie Spúšťanie testov Identifikácia odhalených skutočností Reportovanie odhalených chýb Navrhovanie riešení pri odhalení chýb pri procese testovania

4	Zapisovateľ	Zapisovanie zistených skutočností pri celom procese testovania

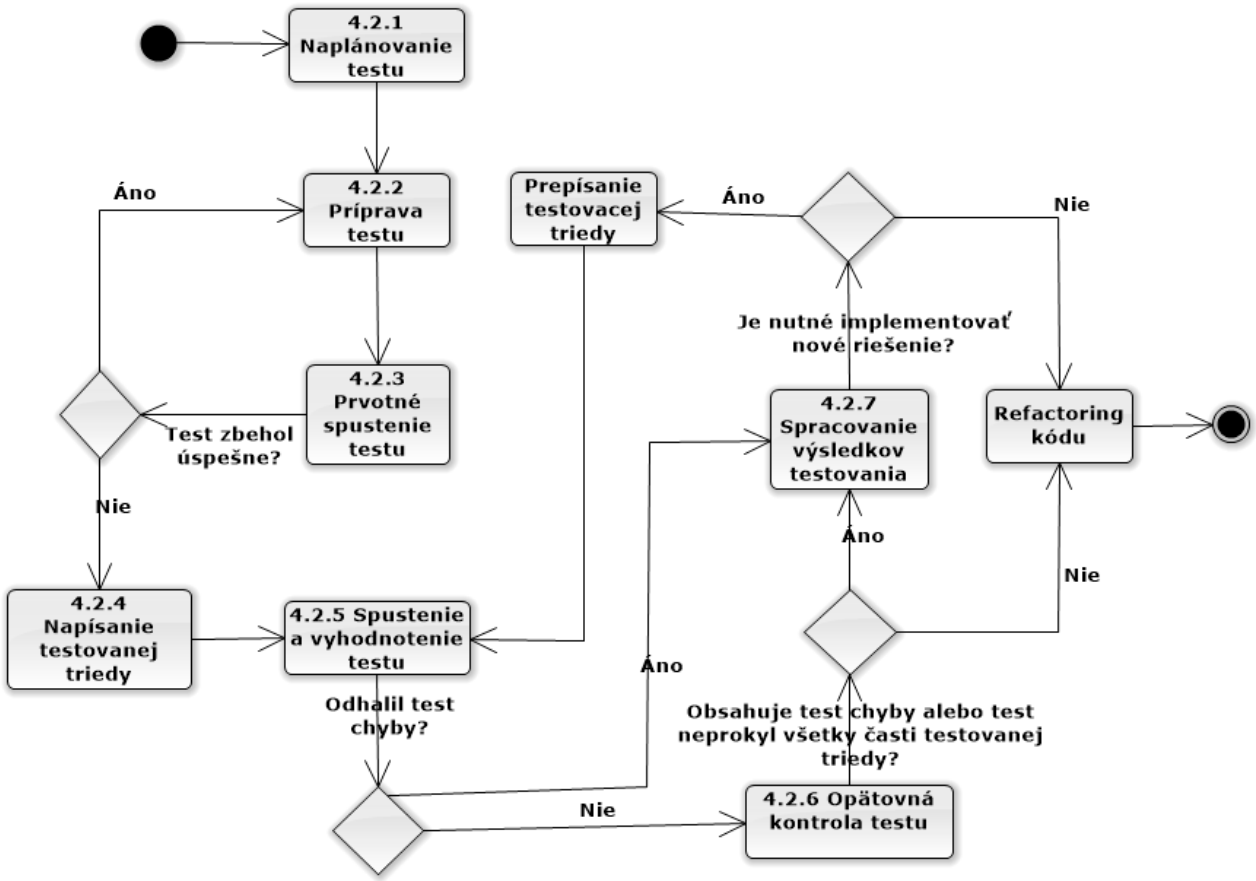
7.6.6 Procesy definované v oblasti manažmentu testovania

Tabuľka 17 - procesy definované v rámci tejto metodiky

#	Názov	Kapitola
1	Proces naplánovaniatestovania	4.2.1
2	Proces prípravy testu	4.2.2
3	Proces prvotného spustenia testu	4.2.3
4	Proces napísania testovanej triedy	4.2.4
5	Proces spustenia a vyhodnotenia testu	4.2.5
6	Proces opätovnej kontroly testu	4.2.6
7	Proces spracovania výsledkov testovania	4.2.7

Na obrázku č. 19 je znázornená postupnosť horeuvedených procesov.

Obrázok 19 - diagram aktivít v procese testovania



7.6.7 Proces naplánovania testovania

Vstup: požiadavka na otestovanie softvérového projektu
Výstup: pridelenie úloh, dokumentácia testovania
Zodpovedný: vedúci testovania

Vedúci testovania poverí rolou každého zo zainteresovaných ľudí v rámci procesu testovania. V prípade, že nie je vytvorená dokumentácia testovania, v ktorej sa zaznamenáva celý priebeh jeho procesu, vytvorí sa. Následne vedúci pridelí úlohy všetkým zainteresovaným, ktoré je potrebné vykonať ešte pred samotným začatím procesu testovania. Základom je spustenie IDE Eclipse spolu s konfiguráciou frameworku Junit (vo väčšine prípadov je framework Junit už predkonfigurovaný, ale môže nastať prípad, kedy je nutná konfigurácia).

7.6.8 Proces prípravy testu

Vstup: dokumentácia testovania s požiadavkami na testovanie
Výstup: zdrojový kód testovacej triedy
Zodpovedný: vývojár

Vývojár vytvorí v prostredí Eclipse prvotný test definujúci požiadavku na funkcionalitu kódu, ktorá sa od neho očakáva. Taktiež sa týmto uistí, že presne chápe funkcionalitu a požiadavky na testovaný komponent, čo zaručí, že samotný kód sa neodchýli od pôvodného zámeru a cieľa definovaného pri plánovaní testu.

7.6.9 Proces prvotného spustenia testu

Vstup: zdrojový kód testovacej triedy
Výstup: funkčný test
Zodpovedný: vývojár

Vývojár musí všetky novonapísané testy spustiť ešte pred napísaním samotného kódu a je nutné, aby skončili neúspechom, pretože pokiaľ neexistuje kód, ktorý by umožnil úspešné ukončenie, nemôže ani test skončiť úspechom. V prípade, že by nejaký test v tomto kroku skončil úspešne , čo značí chybu, je nutné, aby vývojár navrhol a implementoval vhodné riešenie tohto problému.

7.6.10 Proces napísania testovanej triedy

Vstup: funkčný test
Výstup: zdrojový kód testovanej triedy
Zodpovedný: vývojár

Pokiaľ budú pripravené testovacie triedy je potrebné aby vývojár napísal zdrojový kód testovanej triedy, ktorý má test pokryť. Tento proces sa priamo netýka tejto metodiky, ale je dôležitou súčasťou pri procese testovania.

7.6.11 Proces spustenia a vyhodnotenia testu

Vstup: zdrojový kód testovacej triedy spolu s testovanými triedami, dokumentácia testovania
Výstup: aktualizovaná dokumentácia testovania
Zodpovedný: tester, zapisovateľ

Tester spustí jednotkový test a zapisovateľ zaznamenáva výsledky testu. V prípade, že sa reálny výstup testu líši od očakávaného, autor zdrojového kódu, prípadne iný vývojár alebo tester, navrhne vhodné riešenie. Všetky návrhy zapisovateľ zaznamená do dokumentácie testovania.

7.6.12 Proces opätovnej kontroly testu

Vstup: dokumentácia testovania
Výstup: aktualizovaná dokumentácia testovania
Zodpovedný: tester, zapisovateľ

V prípade, že test neodhalil žiadne chyby, tester prekontroluje testovaciu triedu, či je správne implementovaná. Pokiaľ sa nájdu chyby v implementácii, je nutné, aby navrhol vhodné riešenie, ktoré zapisovateľ zaznamená do dokumentácie testovania.

7.6.13 Proces spracovania výsledkov testovania

Vstup: dokumentácia testovania
Výstup: aktualizovaná dokumentácia testovania
Zodpovedný: vedúci testovania

Vedúci testovania po obdržaní dokumentácie testovania vyhodnotí testovanie z hľadiska chýb a návrhov riešení a určí ako sa bude postupovať. V prípade, že sú v dokumentácií testovania uvedené návrhy riešení testovacích tried, poverí vývojára, aby uskutočnil ich implementáciu. V opačnom prípade vykoná refaktoring kódu.

7.6.14 Zoznam nadväzujúcich metodík a dokumentov

Konvencia zápisu zdrojového kódu

Konvencia komentovania zdrojového kódu

Konvencia refaktoringu zdrojového kódu

7.7 Manažment dokumentovania

7.7.1 Pojmy

Používateľský príbeh – úloha, vykonávaná v jednom šprinte

Šprint – dvojtýždňový cyklus

Centrálna dokumentácia – aktuálna celková dokumentácia

7.7.2 Roly

Manažér dokumentácie – človek, ktorý je poverený správou dokumentácie

Zákazník – zadávateľ projektu

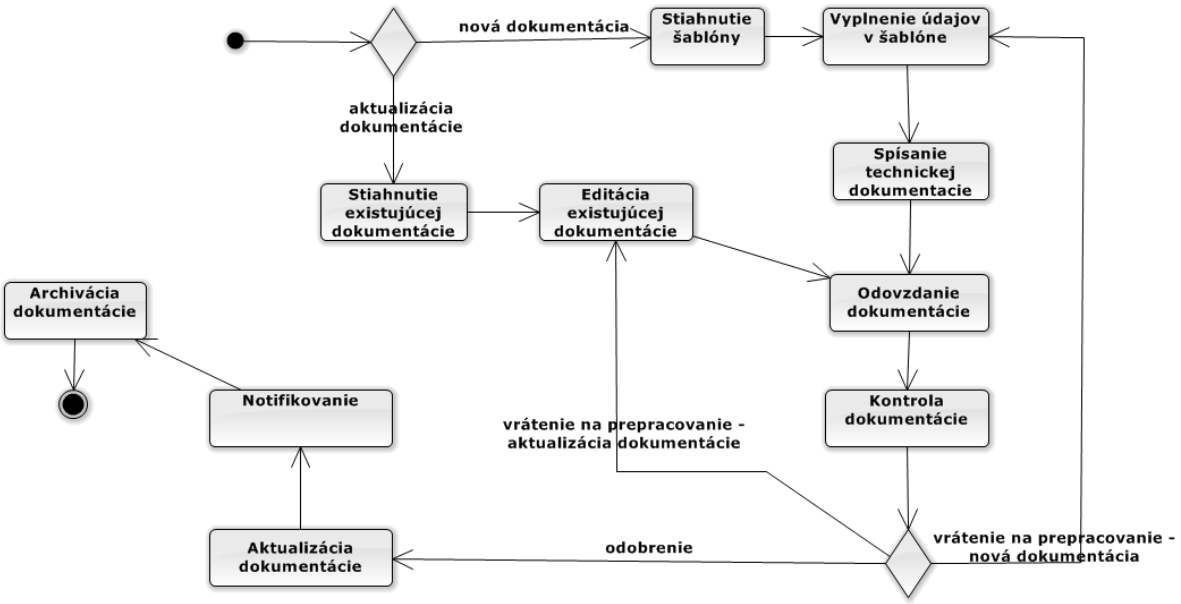
Vývojár – človek, ktorý sa podieľa na realizácii používateľského príbehu

Manažér kvality – poverený kontrolou dokumentácie

7.7.3 Popis metodiky

Metodika popisuje spôsob tvorby technickej dokumentácie. Proces tvorby je znázornený na obrázku č. 20. Proces tvorby začína rozhodnutím, či ide o tvorbu novej dokumentácie alebo aktualizáciu už existujúcej. Ak ide o tvorbu novej dokumentácie, nasleduje stiahnutie a vyplnenie údajov v šablóne, spísanie technickej dokumentácie. Ak ide o aktualizáciu dokumentácie, je potrebné dokumentáciu stiahnuť, editovať ju. Po vykonaní týchto procesov nasleduje proces odovzdania dokumentácie, jej kontrola. Po kontrole je potrebné rozhodnúť o tom či dokumentácia spĺňa požiadavky. Ak nespĺňa, podľa druhu sa dokumentácia vracia na prepracovanie. Ak požiadavky spĺňa, nasleduje proces aktualizácie dokumentácie, proces notifikácie a pred ukončením proces archivácie dokumentácie.

Obrázok 20- aktivity diagram



7.7.4 Popis procesov

7.7.4.1 Proces 1 – Stiahnutie šablóny

Vstup: Odkaz na šablónu pre dokumentáciu

Výstup: Stiahnutá šablóna dokumentácie

Zodpovedný: Vývojár

Opis procesu: Vývojár sťahuje šablónu z úložiska na svoj pracovný stroj.

7.7.4.2 Proces 2 – Vyplnenie údajov v šablóne

Vstup: Stiahnutá šablóna dokumentácie

Výstup: Vyplnená šablóna dokumentácie (základné údaje, názov, autor, čas)

Zodpovedný: Vývojár

Opis procesu: Vývojár vypĺňa základné údaje ako názov, meno autora, dátum a čas, krátky popis, nadväznosť na iné časti.

7.7.4.3 Proces 3 – Spísanie technickej dokumentácie

Vstup: Používateľský príbeh

Výstup: Dokument s opísaným používateľským príbehom, návrhom, implementáciou, testovaním

Zodpovedný: Vývojár

Opis procesu: Vývojár dokumentuje používateľský príbeh, jeho návrh, implementáciu a testovanie.

7.7.4.4 Proces 4 – Odovzdanie dokumentácie

Vstup: Dokument s opísaným používateľským príbehom

Výstup: Dokument uložený v úložisku dát

Zodpovedný: Vývojár

Opis procesu: Vývojár odovzdáva vytvorený dokument do spoločného úložiska dát.

7.7.4.5 Proces 5 – Kontrola dokumentácie

Vstup: Dokument uložený v úložisku

Výstup: Rozhodnutie o odobrení alebo vrátení dokumentácie

Zodpovedný: Manažér dokumentácie, Manažér kvality

Opis procesu: Manažér dokumentácie a manažér kvality kontroluje stav odovzdanej dokumentácie. Ak je stav vhodný, dokumentácia je odobrená a posunutá ďalej. Ak je stav nevyhovujúci, dokumentácia sa vracia na prepracovanie.

7.7.4.6 Proces 6 – Stiahnutie existujúcej dokumentácie

Vstup: Odkaz na existujúcu dokumentáciu

Výstup: Stiahnutá existujúca dokumentácia

Zodpovedný: Vývojár, Manažér dokumentácie

Opis procesu: Vývojár sťahuje na svoj pracovný stroj existujúcu dokumentáciu podľa odkazu, ktorý mu poskytne manažér dokumentácie.

7.7.4.7 Proces 7 – Editácie existujúcej dokumentácie

Vstup: Stiahnutá existujúca dokumentácia

Výstup: Upravená existujúca dokumentácia

Zodpovedný: Vývojár

Opis procesu: Vývojár upravuje už existujúcu dokumentáciu, aktualizuje potrebné časti.

7.7.4.8 Proces 8 – Aktualizácie dokumentácie

Vstup: Odovzdaná dokumentácia

Výstup: Aktualizovaná centrálna dokumentácia

Zodpovedný: Manažér dokumentácie

Opis procesu: Manažér dokumentácia aktualizuje centrálnu dokumentáciu o nové časti z odovzdanej dokumentácie.

7.7.4.9 Proces 9 – Notifikovanie

Vstup: Aktualizovaná centrálna dokumentácia

Výstup: Notifikovanie všetkých zainteresovaných strán

Zodpovedný: Manažér dokumentácie

Opis procesu: Manažér dokumentácie rozosiela notifikácie o aktualizovaní centrálne dokumentácie všetkým zainteresovaným stranám – zákazníkovi, vývojárom, manažérovi kvality, prípadne ďalším.

7.7.4.10 Proces 10 – Archivácia dokumentácie

Vstup: Centrálna dokumentácia

Výstup: Archívna dokumentácia

Zodpovedný: Manažér dokumentácie

Opis procesu: Manažér dokumentácie archivuje do úložiska dát centrálnu dokumentáciu

8 Zápisnice zo stretnutí

V tejto kapitole sú uvedené všetky zápisy zo stretnutí počas semestra.

8.1 Zápis 1. Stretnutia tímu č.4

Dátum: 2.10.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík, Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Samuel Benkovič

Michal Petráš

Martin Adámik

Vladimír Bošiak

Michal Čerešňák

Igor Homola

Téma:

Vypracoval: Matej Bádal

Opis stretnutia:

Úvod do predmetu

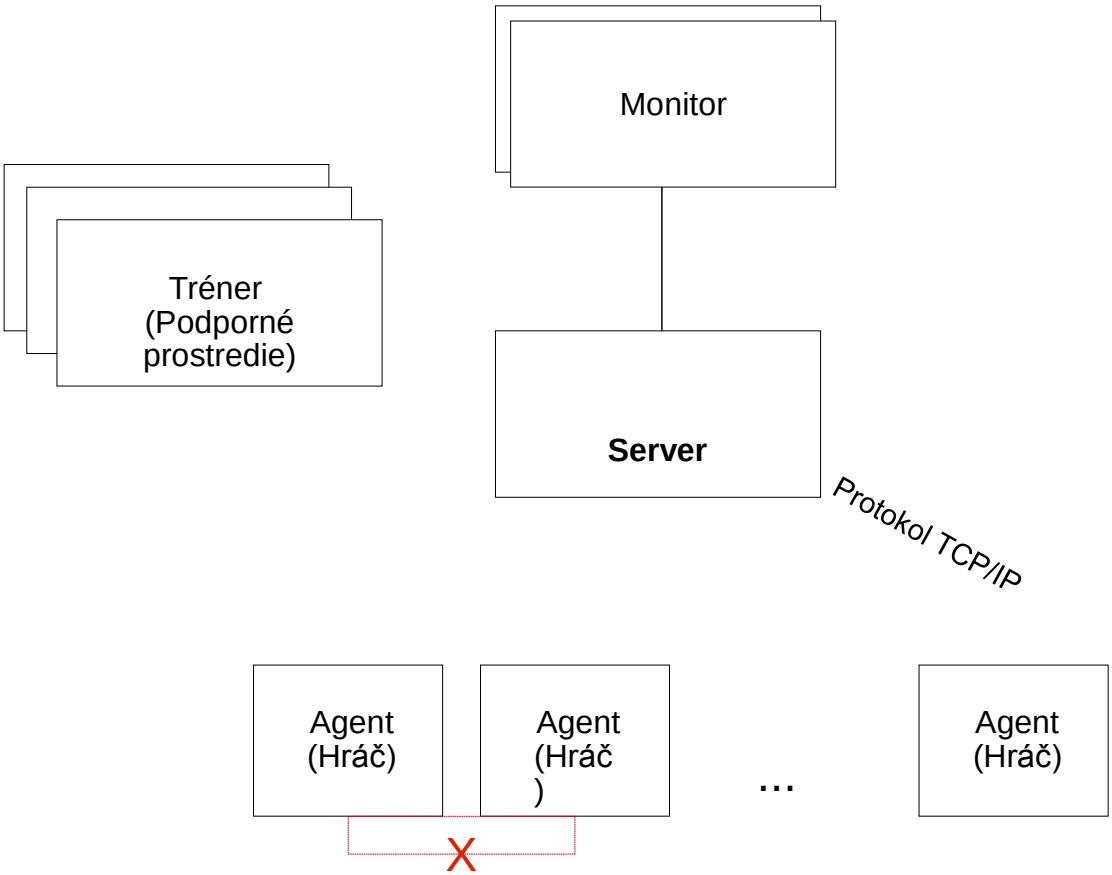
- rozprava o dokumentácii – finálne musí byť dokumentácia odovzdaná
- pozor na dodržiavanie termínov
- stretnutia primárne vedú tímy

Úvod do problematiky

- opis fungovania pohybov robota
- opis SERVERA
  - SERVER simuluje väčšinu vecí (ihrisko, loptu)
  - SERVER ododiela informácie o stave sveta a jednotlivých robotov
  - SERVER pracuje v krokoch, každý krok ma 20 ms
- robot Nao (Aldebaran Robotics)
  - low skill opisy pohybov
  - high skill fázy pohybov
  - riadenie kĺbov
  - pohyby kĺbov v časovom úseku (stabilizované a

nestabilizované)
- skladanie vyšších pohybov (dostaň sa k lopte, postav sa...)
- rozhodovací strom

- dôležitý komponent – monitor -> pripája sa na server a zobrazuje simuláciu
- podporné prostredie slúži (hlavne) na testovanie
  - niekedy je napojené priamo na hráča
- komunikácia medzi agentami navzájom je **ZAKÁZANÁ**
- bližší opis hráča
  - pozícia hráča je vždy 0,0 pre neho samého
  - hráč si ráta svoju pozíciu na ihrisku sám
  - každý agent má zorný uhol – 120 stupňov
- návrhy na zlepšenia
  - highskilly sú spracované dosť nepríjemne
  - bolo by treba zlepši rozhodovanie vrámci tímu
- priblíženie SCRUM-u a predmetu
  - buď je úloha hotová, alebo nie
  - kto za šprint nemá nič hotové, nemal by dostať zápočet



Úlohy na ďalší týždeň:

ID	Člen tímu	Popis úlohy
1	-nezadané	Rozdelenie úloh v tíme
2	-nezadané	Skúsiť si spustenie nejakého hráča
3	-nezadané	Určenie SCRUM mastera
4	-nezadané	Založenie dokumentácie
5	-nezadané	Premyslieť, čo by sme chceli (konkrétne) robiť
6	-nezadané	Dohodnúť si technológie pre komunikáciu a dokumentáciu

8.2 Zápis 2. Stretnutia tímu č.4

Dátum: 9.10.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma: Začiatok prvého šprintu

Vypracoval: Michal Čerešňák

Opis stretnutia:

- Prebehla kontrola úloh z minulého týždňa
- Kontrola ako sa komu podarilo nainštalovať hráča

- Martin Adámik – Linux 12.04: server, monitor, agent – OK, Jim – chyby
- Michal Čerešňák – Windows XP: server, monitor, agent – OK, hráč sa nepohybuje
- Matej Bádal – Linux Mint: server, monitor – OK, Jim a RoboCupLibrary – chyby
- Michal Petráš – Windows 7: server, monitor, agent – OK
- Igor Homola – Windows 8: server – chyba
  Linux: server, monitor – OK, Eclipse – chyba

- Do budúceho týždňa odstrániť problémy s inštaláciou hráča
- K inštalácii hráča musí každý člen týmu vypracovať dokument obsahujúci
  - Problémy pri inštalácii
  - Riešenie problému
  - Použité nástroje a ich verzie (Java, Eclipse, ...) + typ operačného systému

- Ing. Marián Lekavý nám dodal nové zdrojové kódy
- Zápisy zo stretnutí treba dodávať na web do 24 hodín od stretnutia
- Pre ohodnocovanie náročnosti úlohy sme použili Planning poker cards
- Žiaden člen týmu nesmie mať na konci šprintu 0 bodov
- Úloha pre SCRUM Mastera
  - Na stretnutia mať vytlačený zoznam úloh s percentom splnenia + burndownchart graf

- Úloha 1. Šprintu
  - Dôkladne sa oboznámiť s hráčom
  - Naštudovať dokumentáciu a kód
  - Rozdeliť analýzu medzi členov týmu
  - Zistiť kde sa nachádzajú nastaviteľné časti hráča
  - Každý člen týmu musí vedieť zmeniť nejakú triviálnu vec v správaní hráča (napr. mávanie rukou)
  - Každý člen týmu musí napísať unittest na nejakú vybranú metódu
  - Usporiadať tímový brainstorming – vymyslieť nápady na druhý šprint a usporiadať ich do backlogu
  - Každá úloha musí byť zdokumentovaná

- Doplnková úloha 1. Šprintu
  - Oboznámiť sa s architektúrou svetových tímov
  - Analyzovať minimálne 7 tímov

- Výsledok 1. Šprintu
  - Vedieť čo máme k dispozícii a na čom ideme ďalej stavať

Predošlé úlohy:

ID	Člen tímu	Popis úlohy	Stav
1	-nezadané Dokončené	Rozdelenie úloh v tíme	
2	-nezadané	Skúsiť si spustenie hráča	Prebieha
3	-nezadané	Určenie SCRUM Mastera	Dokončené
4	-nezadané	Založenie dokumentácie	Prebieha
5	-nezadané	Premyslieť čo by sme chceli (konkrétne) robiť	Prebieha

6	-nezadané	Určiť technológie pre komunikáciu a dokumentáciu	Dokončené

Úlohy na ďalší týždeň:

ID	Člen tímu	Popis úlohy	Stav
7	Bádal, Homola	Spojazdniť tímový web	
8	všetci	Zaradenie úloh do backlogu	
9	všetci	Spojazdnenie hráča	
10	Benkovič	Vedenie analýzy kódu	
11	Bádal, Bošiak	Analýza kódu – Jim	
12	Čerešňák, Homola	Analýza kódu – RoboCupLibrary	
13	Petráš, Adámik	Analýza kódu – TestFramework	
14	všetci	Unit Test	
15	všetci	Zmena správania hráča	
16	všetci	Analýza svetových tímov	

8.3 Zápis 3. Stretnutia tímu č.4

Dátum: 16.10.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma: Priebeh prvého šprintu

Vypracoval: Igor Homola

Opis stretnutia:

- Prebehla kontrola backlogu, burndown chartu a úloh z minulého týždňa
- Brainstorming ohľadne úloh do backlogu
- Informácia k unit testom
    - Unit test treba vytvoriť k časti kodu ktora sa nebude menit
    - Možnostiunit testu - pohľad na svet, parser správ zo servera, knižnice
- Návrh úloh do backlogu
- Kontrola ako sa komu podarilo nainštalovať hráča
    - všetci členovia tímu spustili hráča a testframework

- Úloha 1. Šprintu
    - Dôkladne sa oboznámiť s hráčom
    - Naštudovať dokumentáciu a kód
    - Analýza prostredia - Samuel Benkovič
        - Jim
            - Matej Bádal
            - Vladimír Bošiak
        - TestFramework
            - Martin Adámik
            - Michal Petráš
        - RoboCupLibrary
            - Michal Čerešňák
            - Igor Homola
    - Zistiť kde sa nachádzajú nastaviteľné časti hráča
    - Každý člen týmu musí vedieť zmeniť nejakú triviálnu vec v správaní hráča (napr. mávanie rukou)
    - Každý člen týmu musí napísať unittest na nejakú vybranú metódu
    - Každá úloha musí byť zdokumentovaná

- Výsledok 1. Šprintu
    - Vedieť čo máme k dispozícii a na čom ideme ďalej stavať
    - Vytvorenie backlogu

Predošlé úlohy:

ID	Člen tímu	Popis úlohy	Stav
1	Bádal, Homola	Spojazdniť tímový web	Dokončené
2	všetci	Zaradenie úloh do backlogu	Prebieha
3	všetci	Spojazdnenie hráča	Dokončené

Úlohy na ďalší týždeň:

ID	Člen tímu	Popis úlohy	Stav
4	všetci	Zaradenie úloh do backlogu	Prebieha
5	Benkovič	Vedenie analýzy kódu	
6	Bádal, Bošiak	Analýza kódu – Jim	
7	Čerešňák, Homola	Analýza kódu – RoboCupLibrary	

8	Petráš, Adámik	Analýza kódu – TestFramework
9	všetci	Unit Test
10	všetci	Zmena správania hráča
11	všetci	Analýza svetových tímov

8.4 Zápis 4. Stretnutia tímu č.4

Dátum: 23.10.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma: Priebeh štvrtého stretnutia

Vypracoval: Samuel Benkovič

Opis stretnutia:

- Vedúcemu projektu boli prezentované výsledky úloh s prvého šprintu.
  - Unit Test
    - Samuel Benkovič dostal prerobiť unit test, problém riešenia bol ten, že testované metódy boli závislé od hry.
  - Správanie hráča
    - Všetci členovia tými splnili túto úlohu.
  - Analýza projektu
    - pri analýze sa mali dorábať aj komentáre.
    - mali sme spisovať veci ktoré sa dajú odstrániť.
    - Jim
      - potrebný refaktoring.
      - Stav kódu : Zlý.

        - TestFramework
            - prijateľná štruktúra.
            - Stav kódu : Primeraný.
        - RoboCupLibrary
            - nevyužívané metódy.
            - Stav kódu : Primeraný.
    - BackLog
        - definovať presnejšie úlohy - napríklad pri kope úlohe stabilizovať kop do lopty určiť kedy je táto úloha splnená napríklad z 10 pokusov spadne len 1.
        - máme spojiť backlog s druhým tímom.

- Boli identifikované nasledovné stories :
    - 9.1 Verzia servera (critical)
    - 9.2 Hĺbková analýza zahraničných tímov (critical)
    - 9.3 Zjednodušiť časť ruby (critical)
        - Odhadovaná redundancia 20-30%
    - 4.2 Odstrániť ruby (critical)

    Najprv zjednodušiť (9.3) potom odstrániť (4.2)

    - 9.4.1 Spojazdniť automatický anotátor (minor)
        - Dorobiť anotácie (minor)
    - 9.4.2 Vytvoriť architektúru highskills (major)
    - 9.4.3 Návrh strategickej vrstvy (major)
    - Implementácia 9.4 (major-)
    - 9.5.1,2 Odstránenie nepoužívaného kódu (major)
    - 9.5.3 Presun sekcií do RoboCupLibrary (minor)
    - 9.5.4 Premenovať Lowskill (trivial)
    - 9.6 Analýza DP - dorobiť do 9.4.2 a 9.4.3 (critical)
    - 9.6.2 Integrácia DP - implementácia (viac stories) (minor)
    - 9.7.1 Test framework - doplniť UI (minor+)
    - 9.7.2 upraviť framework podľa zmien v architektúre (minor+)
    - 9.8 Zmena taktiky na základe výkrikov (minor)
    - 4.1 Vylepšenie stability hráča po kopnutí do lopty (minor)
        - Vylepšiť evolučný prístup, alebo diplomový projekt
        - dlhý kop je jedno či hráč spadne pri krátkom nesmie spadnúť.
        - kopov je veľký počet za zápas
        - doplniť do stories.
    - Parametrický kop do lopty
      vzdialenosť a uhol kam chceme loptu kopnúť (minor)
    - 4.3 Rozbehanie GIT-u (!blocker)
    - Automatický build-systém
      minimálne 1x denne (integrácia s GIT) (critical)

- 4.4 Aktualizovať návody na inštaláciu na wiki (critical)
- 4.5 Editor pohybov => export do XML (trivial)
- 4.6 Vylepšiť príchod k lopte (minor)
  - Tak aby bol dostatočne rýchli a dostatočne presný
- 4.9.1 Nastavenie sa k lopte (minor)
- 4.9.2 Rozhodnutie čo s loptou (minor)
- 4.12. Vytvorenie rozhodovanie pri strele hráčov (trivial)
- 4.13. Hľadanie lopty (minor)
  - Otáčať viac hlavou
- 4.14. Zlepšiť driblovanie (minor)
- 4.15 Auto-reštart hry (trivial)

- Vždy je treba pretestovať viac krát. overenie ako finálne 100 pokusov.

- V architektúre treba zohľadniť :
  - Strategická vrstva
    - zadá, že hráči majú vykonať určitú stratégiu
    - Príklad : útok po pravom krídle
  - Taktická vrstva
    - posúdi situáciu a rozhodne čo ktorý hráč ma vykonávať
    - Príklad : Jeden ide po loptu, druhý mu nabieha na prihrávku.
  - Highskill
    - Highskill danú akciu ktorú ma vykonať a rozkúskuje ju na menšie časti lowskilly
    - Príklad : Choď po loptu rozkúskuje na natoč sa, kráčaj , natoč sa ...
  - Lowskill
    - realizuje pohyb hráča po ihrisku.
    - Príklad: Kráčaj (hráč ohýba svoje kĺby a kráča)
- Určili sme si čo sa bude vykonávať v tomto šprinte :
  - 9.1 Verzia servera.
  - 9.3 , 4.2 Zjednodušiť a odstrániť ruby
    - rozdeliť triedy, highskilly,
    - rozdeliť ruby do balíčkov.
    - preprogramovať do javy.
      - otestovať a až potom zmeniť ruby referencie na tie java.
  - 9.5.1,2 Odstránenie nepoužitého kódu.
    - Opatrne aby ste nevymazali len nedokončenú ale stále potrenú funkcionalitu.

8.5 Zápis 5. Stretnutia tímu č.4

Dátum: 30.10.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samule Benkovič

Téma: Šprint 2 – progres

Vypracoval: Michal Petráš

Opis stretnutia:

- Začiatok stretnutia
  Martin začal stretnutie, informoval Mariána o jednotlivých úlohách a ich stave.
  Marián sa pýtal na kritickú úlohu "spoločný backlog".
  Spoločný backlog bol založený v Jire.

- Informovanie o stave úloh jednotlivých členov
  - Mišo Čerešnák spojazdnil GIT, začal analyzovať zahraničný tím a začal analyzovať diplomovku
  - Martin Adámik má hotovú analýzu zahraničného tímu, začal analyzovať diplomovku, taktiež spojazdnil GIT
  - Matej Bádal spojazdnil GIT, vybral si tím na analýzu. Snažil sa spojazdniť SSH na tímovom serveri.
  - Vlado Bošiak spojazdnil GIT, vybral si tím na analýzu, taktiež pripravuje návody na analýzu pre inštaláciu na MACu a Linuxe.
    - Martin Adámik sa pýtal na zdokumentovanie inštalácie, s ohľadom na zmeny v ruby kóde. Marián odpovedal, že návod má byť písaný s ohľadom na túto skutočnosť.

- Michal Petráš spojazdnil GIT, analyzoval zahraničný tím a zároveň začal analyzovať diplomovku. Taktiež spisuje dokumentáciu z prvého šprintu.
    - Mal otázku ohľadne dokumentácie – ako spisovať zmenu správania a podobné malé podúlohy, špecificky uvádzať autorov ? Marián odpovedal, že je vhodné spisovať to do odstavcov, prípadne uviesť autorov.
- Samo Benkovič spojazdnil GIT, poskytol analýzu zahraničného tímu z jeho bakalárske práce. Plánuje sa pustiť do ruby kódu.
    - Marián upozorňuje na skutočnosť, že to nie je vec, ktorú dokončím a hneď pôjde.
- Igor Homola spojazdnil GIT, analyzoval zahraničný tím a začal analyzovať diplomovku

- Diskusia
    - Ruby – Marián upozorňuje na ruby kód a pýta sa ako sme si to podelili. Náš tím si zobral na starosti plánovač.
    - Interface – Marián odporúča vytvoriť a dohodnúť si interface tak aby sme neboli od seba závislí a mohli pracovať súčasne
    - Upozornenie – Marián špecificky upozorňuje na to, že ruby kód je vec , ktorá ma potenciál byť nestihnutá!
    - Burn down chart – graf nie je veľmi dobrý – dôsledok veľkého počtu zadaní odovzdávaných počas víkendu
    - Igor Homola oznamuje , že bola podaná prihláška na TP CUP

8.6 Zápis 6. Stretnutia tímu č.4

Dátum: 6.11.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samule Benkovič

Téma: Zhodnotenie práce po druhom šprinte a definovanie cieľov pre tretí šprint

Vypracoval: Vladimír Bošiak

Opis stretnutia:

- Prezentácia výsledkov šprintu
  - Prezentácia analýzi zahraničných tímov
    - Hamburg Bit-Bots
    - Nao Team HTWK
    - FC Portugal
    - Austin Villa
    - Upennalize
    - MagmaOffenburg
  - Prezentácia analýzi diplomových projektov
    - Hudec J. – rýchla chôdza
    - Ďurčák – rozhodovacia logika hráča a riešenie kolizí
  - Prezentácia vytovrenej inštalačnej príručky pre MacOS a Linux
  - Prezentácia prepísaného kódu z Ruby do Java
    - Konzultácia zmien
    - Problémi s komunikáciou s druhým tímom a chyba integrácie
- Zlepšenia
  - pomenovanie taskov
  - úplne odstránenie Ruby
- Retrospektíva
  - Zlepšenie komunikácie s druhým tímom

- Zavedenie hlavičiek do súborov (autor, tím, rok)
- Riešenie znalosti GIT-u
- Riešenie konfliktov s druhým tímom
- Vytvorené tasky dávať vedúcemu na pridelenie priorít
- Keď chceme niečo vymazať, treba to označiť ako *TODO vymazať* ak to zostane aj po mesiaci v kóde môžeme vymazať danú časť
- Určenie vrstiev – strategická, taktická

Predošlé úlohy:

ID	Člen tímu	Popis úlohy	Stav
RFCMTROLL-76	Adámik, Bošiak	Vytvorenie inštalačných príručiek pre MacOS a Linux	splnené
RFCMTROLL-3	Bádal, Bošiak, Homola, Adámik, Petráš, Čerešňák	Vytvorenie analýzi zahraničných tímov	splnené
RFCMTROLL-70	všetci	Prepísať plánovanie do jazyku Java	splnené

Úlohy na ďalší týždeň:

ID	Člen tímu	Popis úlohy
ROBOCUPTP-30	Tím 9	V projekte Jim je potrebne prepisat vsetky konstanty zavisle od verzie servera
ROBOCUPTP-62	všetci	Určenie, či zakomentovaný kód funguje
ROBOCUPTP-61	všetci	Odstránenie nepoužívaných funkcií a premenných
ROBOCUPTP-39	všetci	Návrh a implementícia vrstiev (strategická, taktická vrstva)
ROBOCUPTP-38	všetci	Vytvoriť architektúru highskills

8.7 Zápis 7. Stretnutia tímu č.4

Dátum: 13.11.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma: Šprint 3 - progress

Vypracoval: Martin Adámik

Opis stretnutia:

- Začiatok stretnutia
  Marián začal stretnutie s požiadavkou o informovaní o stave úloh.
- Informovanie o stave úloh jednotlivých členov:
  - Samuel Benkovič odstránil ruby a riešil návrh architektúry.
  - Igor Homola vložil jednotlivé úlohy do systému Jira.
  - Michal Petráš analyzoval zakomentované časti kódu a spisoval veľkú dokumentáciu.
    - Michal Petráš sa taktiež spýtal na zdokumentovanie celkového pohľadu. Marián ozrejmil ako sa má zdokumentovať celkový pohľad na produkt.
  - Vladimír Bošiak navhroval architektúru a refaktoroval triedu agent model v balíku Jim.
  - Martin Adámik analyzoval zakomentované časti kódu.
  - Matej Bádal odstránil ruby a riešil návrh architektúry.

        - Matej Bádal taktiež ozrejmil Mariánovi, že ruby sa v doterajšej fáze nemôže ešte úplne odstrániť, pretože komunikuje s testframework-om, ale čo sa týka hráča bolo ruby už úplne odstránené.
    - Michal Čerešňák analyzoval zakomentované časti kódu.
- Diskusia
    - Mariánovi a Ivanovi bola predvedená doposiaľ navrhnutá architektúra. Následne sa začala diskusia, do ktorej sa zapájal každý člen tímu.
    - Marián s Ivanom navrhli, aby si taktiky vyhodnocovali svoju fitness, ktorá bude rozhodným prvkom pri výbere najvhodnejšej taktiky.
    - Ivan navrhol, aby bol vytvorený default high skill, čo znamená, že hráč bude stáť, ale bude mať celkový prehľad o tom, kde sa lopta nachádza.
    - Treba vytvoriť príkladové taktiky a následne 5 až 10 testovacích scenárov na testovanie jednotlivých taktík.

Predošlé úlohy:

ID	Člen tímu	Popis úlohy	Stav
9.5.2	Adámik, Petráš, Čerešňák	Určiť, či zakomentovaný kód funguje alebo nie.	Prebieha
	Všetci	Návrh a implementácia taktickej vrstvy	Prebieha
9.4.3	Všetci	Navrh a Implementacia strategickej vrstvy	Prebieha
9.4.2	Všetci	Vytvoriť architektúru Highskills	Prebieha
	Petráš	Vytvoriť dokumentáciu	Prebieha
	Benkovič	Úplne odstrániť RUBY z kódu	Dokončené

Úlohy na ďalší týždeň:

Pokračovať v rozpracovaných úlohách.

8.8 Zápis č.8 stretnutia tímu č.4

Dátum: 20.11.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samul Benkovič

Téma: Ukončenie šprintu, návrh nových úloh na ďalší šprint

Vypracoval: Matej Bádal

Opis stretnutia:

- preriešenie webovej stránky
- prezentáciu práce vykonanej za posledný šprint
- diskusia ohľadom zakomentovaných častí kódu
- riešenie vytvorenej architektúry
  - vysvetlenie a opis jednotlivých aspektov architektúry vedúcim
  - navrhnutie možnosti konfigurácie pomocou XML
  - vysvetlenie podstaty šablónovej situácie
  - prediskutovanie aspektu architektúry – situácií
  - vedúcim sa príliš nepozdáva myšlienka so šablónovými situáciami
  - návrh, že situácia by mohla mať ohodnotené atribúty
- vedúci poukazujú na to, že niekto v budúcnosti bude možno chcieť iné delenie ihriska, ako na 4. kvadranty
- porovnávanie situácii bude prebiehať pomocou porovnávania stringov
- vyhodnocovanie situácii bude prebiehať pri výbere taktík
- stratégia bude mať zoznam taktík, ktoré sa dajú vykonať
- Ivan navrhuje, že jednotlivé taktiky musia ma istú zotrvačnosť – nesmú sa prerušiť ak dôjde k chvíľkovej zmene situácie
- vedúcich zaujíma stanovenie defaultnej situácie

- navrhnutá taktika „ZORIENTUJ SA“
- diskusia ohľadom tejto taktiky

- Taktika sa sama musí vyhodnotiť, či je hotová
- vyhodnocovanie fitness by malo by rozdielne pri výbere taktiky a iné pri zisťovaní, či taktika už skončila
- rozoberá sa pokračovanie taktiky pri zmene situácie
- padla otázka, či viac situácii sa môže naraz vyskytovať
  dohodli sme sa, že ihrisko bude rozdelené na 4 kvadranty
  rieši sa identifikácia jednotlivých situácii
  každá TAKTIKA bude mať inicializačnú a progresovú podmienku
  konkrétne situácie – taktiky majú vlastnú podmienku

Predošlé úlohy:

ID	Člen tímu	Popis úlohy	Stav

Úlohy na ďalší týždeň:

ID	Člen tímu	Popis úlohy
1	Tomáš Nemeček	Unit testy
2	TP04	Stratégia
3	TP09 - Moravčík	Taktika
4	Benkovič	Situácie - FrameWork
5	Bošiak	Observer + Selector
6	Bádal	Parser XML
7	Benkovič/Linner	Papierový prototyp

8.9 Zápis č.9 stretnutia tímu č.4

Dátum: 27.11.2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

SamuelBenkovič

Téma: Priebeh štvrtého šprintu, prezentácia papierových prototipov

Vypracoval: Igor Homola

Opis stretnutia:

- Prezentácia papierového prototypu - taktiky (detailný popis papierového prototypu sa nachádza na stránke timu9)
    - Prototyp útočníka v sektore 4L (je to oblasť napravo od protivníkovho brankára). Potreba do implementovať:
        - Počet voľných spoluhráčov (počítajú sa tí hráči, ktorých agent vidí)
        - Rýchlosť protihráča a čas, za ktorý je schopný pristúpiť k danému hráčovi
        - Či je daný hráč voľný. Tento parameter sa určuje na základe času, za ktorý k nemu môže prísť protihráč
        - Diskusia k prototypu :spoluhráči by mali komunikovať a spoločne dosahovať ciele. Taktika by nemala mať len situačné rozhodovanie.
        - Je potrebné riadiť sa pravidlami robotického futbalu - kontrola či hráč neprihráva do offside.
    - Prototyp útok s prihrávkou
        - Útok zľava s prihrávkou – hráč z pravej strany ihriska prihrá predsunutému ľavému spoluhráčovi a ten pokračuje v útoku.
        - Útok sprava s prihrávkou – zrkadlová situácia ako v útok zľava s prihrávkou
        - Diskusia: navrhnuté prototypy by mali problém realizovať zložitejšie správanie. Napr. pri preberaní prihrávky by hráč čakal dokým lopta nepríde na požadované miesto a až následne by pokračoval v pohybe.
        - Vytvárať názvy taktík ktoré by reprezentovali taktiku
    - prototyp brankára
        - Diskusia: potreba uvažovať s rôznymi druhmi ukončenia taktiky brankára
            - protihráč je príliš blízko na to, aby brankár stihol zareagovať
            - spoluhráč príde brankárovi na pomoc – získa loptu
            - protivník kopol loptu a brankár stihne zareagovať

- Priebeh 4. šprintu

Problémy na riešenie
- pomenovanie taktik
- príliš krátke taktiky (mali by byt dlhšie)
- problém nadväznosti taktik

Vyriešené
- framework pre taktiky
- Situácie - riešenie je schválené
- vyber situácii
- funkcionalita selektora

To Do
- implementovať (prerušeniehighskillu po zmenení taktiky) taktika musípodaťinformáciu o svojom dokončení
- Identifikovať taktiky
- doplniť do komentára (hlavičky) taktiky na čo je a čorobíkonkrétna taktika
- rozdelenie taktik do balíčkov na útočné a obranne
- Riešenie rol hráča bude riešené pomocou situácii (brankár)
- implementovať do selektora seter ktorý bude vedieť meniť aj samotná taktika
- implementovať factory - automatizovaný spôsob inicializácie objektov
- Observer

Predošlé úlohy:

ID	**Člen tímu**	**Popis úlohy**
1	TomášNemeček	Unit testy
2	TP04	Stratégia
3	TP09 - Moravčík	Taktika
4	Benkovič	Situácie - FrameWork
5	Bošiak	Observer + Selector
6	Bádal	Parser XML
7	Benkovič/Linner	Papierový prototyp

Úlohy na ďalší týždeň:

ID	**Člen tímu**	**Popis úlohy**
1	TomášNemeček	Unit testy
2	TP04	Stratégia
3	TP09 - Moravčík	Taktika
4	Benkovič	Situácie - FrameWork
5	Bošiak	Observer + Selector
6	Bádal	Parser XML

8.10Zápis č.10 stretnutia tímu č.4

Dátum: 4.12. 2013

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Vladimír Bošiak

Igor Homola

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma: Koniec štvrtého šprintu

Vypracoval: Michal Čerešňák

Opis stretnutia:

- **Kontrola úloh**
  - Parser XML – implementovaný
  - Stratégie a taktiky – implementované
  - Observer – sleduje taktiky a stratégie
  - Selektor – bez zmeny oproti minulému týždňu

- **Úlohy do budúceho týždňa**
  - Vyriešiť podmienky v taktikách
  - Dokončiť integráciu stratégií taktík a situácií
  - Otestovať funkčnosť integrovaných častí
  - Doplniť dokumentáciu
  - Aktualizovať webovú stránku
  - Odovzdať dokumentáciu v elektronickej podobe
  - Odovzdať dokumentáciu na stretnutí vo vytlačenej podobe
  - Odovzdať zoznam častí, ktoré sme robili my a ktoré druhý tím
    - Zoznam je spoločný pre oba tímy

8.11Zápis 11. Stretnutia tímu č.4

Dátum: 27.2.2014

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma: Zhodnotenie práce v prvom týždni a vytýčenie cieľov na ďalší týždeň.

Vypracoval: Samuel Benkovič

Opis stretnutia:

- stránka - nefungujú zápisnice treba opraviť linky
- reflexia - netreba používať v súčasnej hierarchií agenta potrebné od komunikovať z druhým tímom.
- unit testy - vytvoriť balíčky pre všetky triedy a dodržať rovnakú hierarchiu.
- musíme lepšie definovať úlohy ,lepšie zadefinovať cieľ úlohy, čo má byť hotové.
- dohodli sme sa na úlohe implementovať správanie hráča tak aby dokázal útočiť z ľavej strany, z pravej strany alebo stredom.
- dohodli sme sa ,že musíme aj zadefinovať konkrétne taktiky nielen metódu run ale aj initcondion, progrescondition poprípade implementovať konkrétne situácie.

Predošlé úlohy:

ID	Člen tímu	Popis úlohy	Stav
110	Celý tým	Unit Testy pre architektúru	Dokončené.
135	Matej Bádal, Michal Petráš, Vladimír Bošiak	Prezentácia práce za prvý semester.	Dokončené.

Úlohy na ďalší týždeň:

ID	Člen tímu	Popis úlohy
141	Celý tým	Implementácia troch taktík spolu s príslušnými situáciami.
110	Samuel Benkovič	Vytvoriť balíčky na testy.

8.12 Zápis č. 12 stretnutia tímu č.4

Zápis 12. Stretnutia tímu č.4
Dátum: 6.3.2014
Miestnosť: Jobsovo softvérové štúdio
Prítomní: Vedúci: Ing. Ivan Kapustík
Ing. Marián Lekavý, PhD.
Študenti: Matej Bádal
Vladimír Bošiak
Igor Homola
Martin Adámik
Michal Petráš
Michal Čerešňák
Samule Benkovič
Téma: Zhodnotenie práce po prvom šprinte a definovanie cieľov pre druhý šprint
Vypracoval: Michal Petráš
Opis stretnutia:

- Ústna dohoda o nahadzovaní taskov do jiry človekom, ktorý píše zápisnicu
- Potrebné nahodiť plán pretože stále chýba na stránke

- Samo začal stretnutie. Hovorí o unit testoch a ich vyhodnotení. Unit testy dopadli z nášho pohľadu dobre.
- Marián sa pýta koľko máme otestovaného kódu ? Odporúča nám pustiť nástroj
- Samo opisuje vytvorenie taktík a situácií – nový balík octant, taktiež opisuje ako funguje AttacLeft
- Marián sa pýta teda ako to je otestované a čo to robí ?
  - Odpoveď – používa sa defaultne taktika attackleft
- Maťo opisuje AttackMid a považuje ho za implementovaný
- Mišo opisuje base deffend – taktiež považuje za implementovaný
- Samo opisuje ako fungujú octanty
  - Situácie sú odvodené zo starého kódu a teda by mali byť funkčné
- Marián upozorňuje, že by bolo lepšie počítať časové vzdialenosti , nakoľko teraz sa počítajú len vzdialenosť
- Maťo upozorňuje na to, že stred určených kvadrantov sa bude prekrývať
- Marián navrhuje, že by bolo dobré urobiť unit testy na konfliktné situácie
- Vlado poukazuje na vytváranie nových inštancií v taktikách pre každú situáciu
- Marián odpovedá , že je potrebné venovať sa Factory a nie singletonu, ktorý je výslovne anti vzorom.
- Marián sa pýta či situácia trvá dlhšie alebo len v danom okamihu a poukazuje na to, že je potrebné na to dávať pozor
- Marián žiada novú feature – namiesto new situation sa bude volať factory
- Vlado navrhuje currentlist do metódy run
- Marián – statické metódy, mali by byť všetky rovnaké
- Maťo navrhuje obmedziť volanie každých 20ms pri taktikách na kontrolu napr. každú sekundu alebo dve
- Marián sa pýta, či počas šprintu nastali nejaké problémy – nenastali
- Maťo ale namieta, že druhý tím urobil svoju prácu trocha neskôr ako sme čakali a tak sme boli obmedzení
- Marián poukazuje na dohodu medzi tímami
- Dohoda o CODE REVIEW
- Marián upozorňuje na to, že je potrebné aby na konci šprintu všetko fungovalo
- Vlado poukazuje na prerobenie situácií
- Zhodnotenie úloh
  - Unit tesy – splnené
  - Prezentácia a dokumentácia – splnené

8.13Zápis 13. Stretnutia tímu č.4

Dátum:13.3.2014

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma:Zápis zo stretnutia (status meeting)

Vypracoval: Vladimír Bošiak

Opis stretnutia:

- Zhodnotenie prvého týždňa šprintu
  - Splnené:
    - Implementácia metódy pre kontrolu taktiky *AttackLeft* (kvôli overenie správnosti riešenia => prototyp riešenia)
    - Zlepšenie výberu taktiky v triede *SelectorController*
    - Unit testy pre hraničné situácie na ihrisku
  - Návrhy
    - Vykonať code review po dvojiciach v rámci domény
    - Vykonať celkový code review (code review by mal vykonávať ten, ktorý o danej časti nemá znalosti)

8.13.1 [ROBOCUPTP-148] Implementacia Run Created: 06/Mar/14 Updated: 11/Mar/14

Status:	Open
Project:	Robocup_tp09
Component/s:	AllTeam_04_09, Team_04
Affects Version/s:	None
Fix Version/s:	None

Type:	Task	**Priority:**	Critical
Reporter:	Michal Petras	**Assignee:**	Unassigned
Resolution:	Unresolved	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	4 hours, 40 minutes	**Remaining Estimate:**	4 hours, 40 minutes
Σ Time Spent:	1 hour, 50 minutes	**Time Spent:**	20 minutes
Σ Original Estimate:	5 hours	**Original Estimate:**	5 hours

Sub-Tasks:	Key	Summary	Type	Status	Assignee
	ROBOCUPTP-166	Implementacia Run - Attack Left	Sub-task	Resolved	Samuel Benkovic
	ROBOCUPTP-167	Implementacia Run - Attack Mid	Sub-task	Open	
	ROBOCUPTP-168	Implementacia Run - Attack Right	Sub-task	Open	
	ROBOCUPTP-169	Implementacia Run - Defend Agressive	Sub-task	Resolved	
	ROBOCUPTP-170	Implementacia	Sub-task	Open	

	ROBOCUPTP-171	Run - Defend Implementacia Run - Defend Position	Sub-taskResolved

Description

Potrebne zaimplementovat novy run, ktory bude pocitat aj s Check a abort. Ide o problem s ukoncenimhighskillov, ktoremusimevedietukoncit a vymazathighskill queue

8.13.2 [ROBOCUPTP-149] Implementacia Check (+ abort) Created: 06/Mar/14 Updated: 11/Mar/14

Status:	Open
Project:	Robocup_tp09
Component/s:	AllTeam_04_09, Team_04
Affects Version/s:	None
Fix Version/s:	None

Type:	Task	**Priority:**	Critical
Reporter:	Michal Petras	**Assignee:**	Unassigned
Resolution:	Unresolved	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	1 day, 4 hours	**Remaining Estimate:**	1 day, 4 hours
Σ Time Spent:	2 hours, 30 minutes	**Time Spent:**	1 hour
Σ Original Estimate:	1 day, 5 hours	**Original Estimate:**	1 day, 5 hours

Sub-Tasks:	Key	Summary	Type	Status	Assignee
	ROBOCUPTP-164	Implementacia Check (+ abort) - Attac...	Sub-task	Resolved	Samuel Benkovic
	ROBOCUPTP-165	Implementacia Check (+ abort) - Attac...	Sub-task	Open	
	ROBOCUPTP-172	Implementacia Check (+ abort) - Attac...	Sub-task	Open	
	ROBOCUPTP-173	Implementacia Check (+ abort) -	Sub-task	Open	

		Deffend	
	ROBOCUPTP-174	Implementacia Check (+ abort) - Defen...	Sub-taskResolved
	ROBOCUPTP-175	Implementacia Check (+ abort) - Defen...	Sub-taskResolved
Units:	13		

Description

Vytvorenie check - pre kontrolu, ci moze byt highskillukonceny, resp. vymazanyhighskill queue. Potrebne zaimplementovat i STATE pre jednotlive taktiky. Na zaklade parametrov a STATE pisat i check

Poznámka: v taktikách sa táto metóda volá isNewState.

8.13.3 [ROBOCUPTP-177] MetodacheckSituation

Created: 13/Mar/14 Updated: 13/Mar/14 Due: 20/Mar/14

Status:	Open
Project:	Robocup_tp09
Component/s:	Team_04
Affects Version/s:	None
Fix Version/s:	None

Type:	Task	**Priority:**	Minor
Reporter:	Vladimir Bosiak	**Assignee:**	Unassigned
Resolution:	Unresolved	**Votes:**	0
Labels:	None		
Remaining Estimate:	4 days		
Time Spent:	Not Specified		
Original Estimate:	4 days		

Description

Prerobitsituacie na singleton / vytvorenie factory / checkSituation bude static.
Dovod je aby sme nevytvarali nove instanciesituacii.

8.13.4 [ROBOCUPTP-176] Celkove testovanie Taktiky-Selector

Created: 13/Mar/14 Updated: 16/Mar/14 Due: 20/Mar/14

Status:	Open
Project:	Robocup_tp09
Component/s:	Team_04
Affects Version/s:	None
Fix Version/s:	None

Type:	Task	**Priority:**	Major
Reporter:	Vladimir Bosiak	**Assignee:**	Unassigned
Resolution:	Unresolved	**Votes:**	0
Labels:	None		
Remaining Estimate:	4 days		
Time Spent:	Not Specified		
Original Estimate:	4 days		

Description

Dorobenie unit testov pre taktikyskontrolovat code coverage.

8.14Zápis 14. Stretnutia tímu č.4

Dátum: **26.3.2014**

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma: Dokončenie implementácie tak,aby boli roboti schopní hrať futbal

Vypracoval: Matej Bádal

Opis stretnutia:

- Na začiatku stretnutia sa rieši BURNDOWN chart
  - Prečo vyzerá tak, ako vyzerá
- Vedúci nám pripomína, že dokumentáciu by sme mali vypracovávať priebežne
- Rieši sa zoznam úloh
  - BEAM bol zrušený a bola na neho vytvorená samostatná taktika
  - Samo na tabuľu maluje aktuálnu implementáciu taktík (ako fungujú)
  - roboti musia o sebe vedieť
  - nesmú sa všetci natlačiť na seba, lebo spadnú
- Bolo nám oznámené, že pôjdeme na IIT.SRC, takže musíme pripraviť plagát
- Na IIT.SRC budeme prezentovať ukážky zo správania robotov

8.15Zápis 15. Stretnutia tímu č.4

Dátum: **3.4.2014**

Miestnosť: Jobsovo softvérové štúdio

Prítomní: Vedúci: Ing. Ivan Kapustík

Ing. Marián Lekavý, PhD.

Študenti: Matej Bádal

Vladimír Bošiak

Martin Adámik

Michal Petráš

Michal Čerešňák

Samuel Benkovič

Téma: Dokončenie implementácie tak, aby boli roboti schopní hrať futbal

Vypracoval: Michal Čerešňák

Opis stretnutia:

- Na začiatku stretnutia sa rieši prezentácia funkčnosti riešenia
  - nefunguje highskillkick
- Treba vytvoriť plagát na IIT.SRC formátu A1
- Diskusia k Jim-ovi
  - Čísla dresov a Id určuje server v nastavení settings sa dajú čísla dresov nastaviť explicitne
  - Samo hlási, že taktiky ešte nefungujú ideálne – problémom je initCondition. Navrhuje odstrániť pozíciu lopty v initCondition.
  - Treba zmeniť beamovanie na rôzne pozície bližšie k lopte
  - Vlado vysvetľuje prečo ešte nechal otvorený taskSelector
  - V taktikách bude nutné lepšie zadefinovať suitabilitu
- Marián chce, aby sme mu poslali mail o tom, aký je problém s kopacímhighskillom
- Treba sa dohodnúť s druhým tímom na spojazdnení kopu
- Do budúcna by mal by sme mali roboti vedieť hrať futbal, aby bolo vidieť aj taktiky nie že všetci idú po lopte

8.16Zápis 17. Stretnutia tímu č.4

Dátum: **8.4.2014**
Miestnosť: Jobsovo softvérové štúdio
Prítomní: Vedúci: Ing. Ivan Kapustík
Ing. Marián Lekavý, PhD.
Študenti: Matej Bádal
Vladimír Bošiak
Martin Adámik
Michal Petráš
Michal Čerešňák
Samuel Benkovič
Téma: Dokončenie implementácie tak, aby boli roboti schopní hrať futbal
Vypracoval: Samuel Benkovič
Opis stretnutia:
☐ Diskusia oplagáte a zistilo sa že bude pravdepodobne na šírku.
☐ Beam bol úspešne opravený, dohodli sme si e by sme mali rieši aj výkop na za iatku hry.
☐ úspešne implementovaná init condition, bola presunutá podmienka ktorá udávala polohu lopty do prescribedSituations.
☐ null v move ak hrá ide za loptou.
☐ Problém z jirou bol identifikovaný v rovnakom pomenovaní šprintov.
☐ Upravili sme prioritu Lokalizácie na blocker aby sme kládli dôraz na túto úlohu.

Preberací protokol
Tímový projekt I

Projekt: Robocup3D

Odovzdávajúci tím: RFC Megatroll

Preberajúci: Ing. Marián Lekavý, PhD.

Dátum odovzdania: 22.05.2014

Odovzdané dokumenty: 1.tlačená dokumentácia k inžinierskemu dielu (počet strán 101)
2.tlačená dokumentácia k riadeniu (počet strán 119)

Vedúci tímu Ing. Marián Lekavý, PhD.

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

Tímový projekt – RoboCup 3D

Dokumentácia k inžinierskemu dielu

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

Obsah

1 Úvod

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

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

1.1 Ciele

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

1.2 Plán

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

1.3 Plán na letný semester

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

2 Šprinty

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

2.1 Šprint č. 1

Tabuľka 1 - Backlog šprintu 1

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

Tabuľka 2 - rozdelenie úloh v šprinte

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

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

2.1.1 Analýza zdrojových kódov

2.1.1.1 Jim

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

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

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

2.1.1.2 TestFramework

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

2.1.1.2.1 Moduly (Balíky)

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

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

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

3. MONITOR

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

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

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

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

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

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

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

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

DESTROY - posiela agent pri odpojení

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

WORLDMODEL - posiela model sveta agenta do test frameworku

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

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

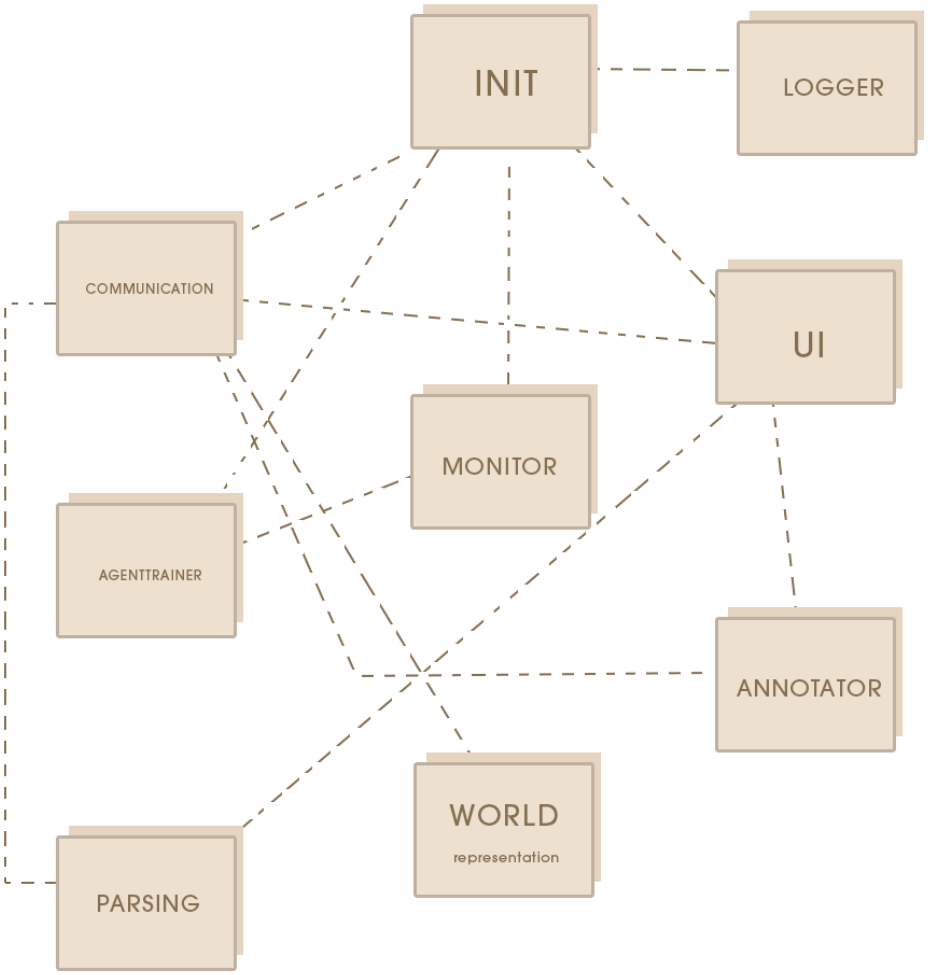
2.1.1.2.2 Nastavenia TESTFRAMEWORKU

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

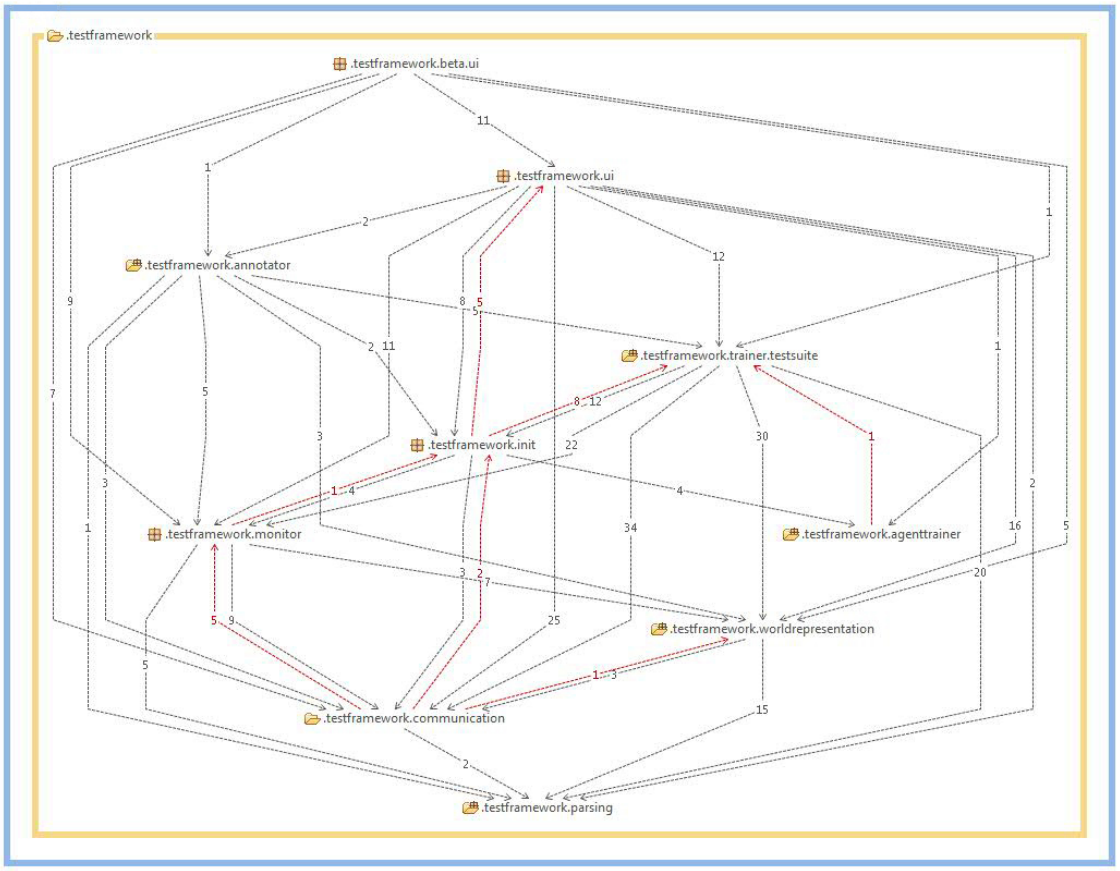
robocup.server.port.player – port na pripojenie hráča k robocup servera
robocup.player.dir – zložka, v ktorej sa nachádza hráč
robocup.player.command – príkaz na spustenie hráča
testframework.monitorAgent.ip – IP adresa, na ktorej počúva testframework na spatnú väzbu
testframework.monitorAgent.port -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

Konfiguračný súbor je prečítaný pri spustení aplikácie.

Obrázok 1- identifikované prepojenia



Obrázok 2 - architektúra testframework-u



2.1.1.3 RobocupLibrary

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

2.1.1.3.1 Balíky

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

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

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

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

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

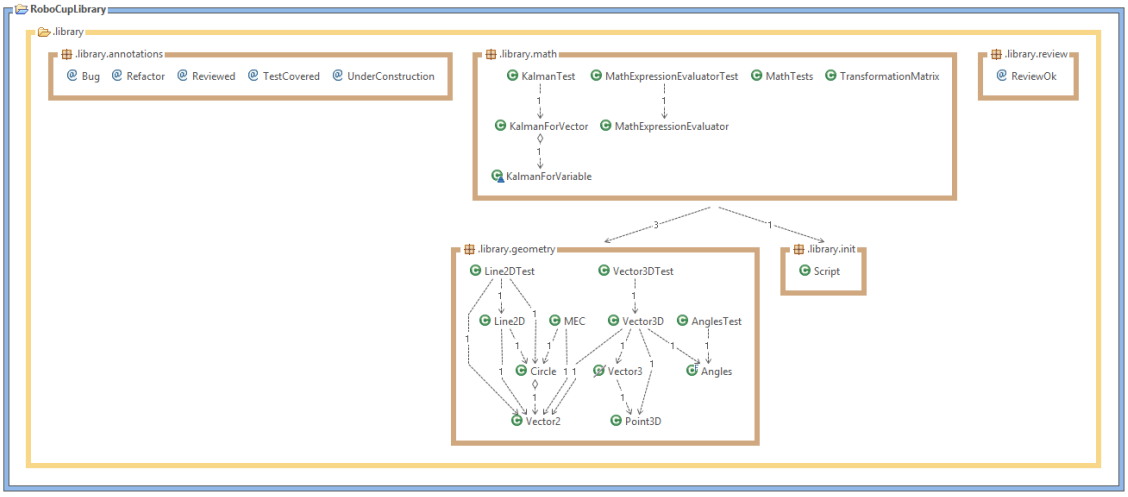
sk.fiit.robocup.library.math
Tento balík obsahuje sedem súborov, ktoré implementujú Kalmanove filtre, prevody matematických výrazov a transformácie matíc.

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

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

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

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



Obrázok 3

2.1.2 Spustenie hráča

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

2.1.2.1 Windows XP Professional 32b Service Pack 3

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

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

2. Inštaláciasimspark verzia 0.2.4-win32:

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

3. Inštalácia rcssserver3d verzia 0.6.7-win32:

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

4. Inštalácia Ruby:

   http://rubyinstaller.org/downloads/

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

Inštalácia hráča:

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

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

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

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

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

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

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

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

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

2.1.3 Zmena správania

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

2.1.4 Unit testy

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

2.1.5 Web

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

2.1.6 Prehľad o tímoch

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

Team DARwIn
NimbRo TeenSize
JoiTech
HuroEvolution AD
Tsinghua Hephaestus

2.1.7 Backlog

Vylepšenie stability hráča po kopnutí do lopty

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

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

Odstránenie ruby kódu

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

Zlepšenie inštalácie

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

Pozn.: Filesystem, export

Editor pohybov – export do XML

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

Pozn.: Iba v prípade, že to naozaj nefunguje

Vylepšenie príchodu k lopte

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

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

Zlepšenie kopnutia do lopty

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

Zlepšenie rozhodovania pri kopaní

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

Analýza minuloročných a zahraničných tímov

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

Pozn.: Napr. kvôli architektúre.

Analýza diplomových prác

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

Zlepšenie rozhodovanie pri situácií hráč na hráča

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

Pozn.: Nie je čo zlepšovať – musíme vytvoriť.

Zlepšiť hľadanie pozície lopty

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

Zlepšiť predkopávanie lopty pred hráčom

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

Pozn.: Dribbling

Implementovať príkaz na reštart hry po skončení

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

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

Refactoring kódu

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

2.2 Šprint č. 2

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

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

Tabuľka 3

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

2.2.1 Analýza zahraničných tímov

2.2.1.1 Hamburg Bit-Bots

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

2.2.1.1.1 Výskum

Bakalárske práce:

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

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

Iný výskum:

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

2.2.1.1.2 Úspechy

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

2.2.1.2 FC Portugal 3D

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

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

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

2.2.1.2.1 Ciele výskumu

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

2.2.1.2.2 Architektúra agenta

Architektúru má tím rozdelenú do viacerých balíkov nasledovne nasledovne:

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

2.2.1.2.3 Optimalizácia

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

2.2.1.3 Analýza svetového tímu magmaOffenburg

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

2.2.1.3.1 Architektúra

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

Obrázok 4 - architektúra Magma-AF

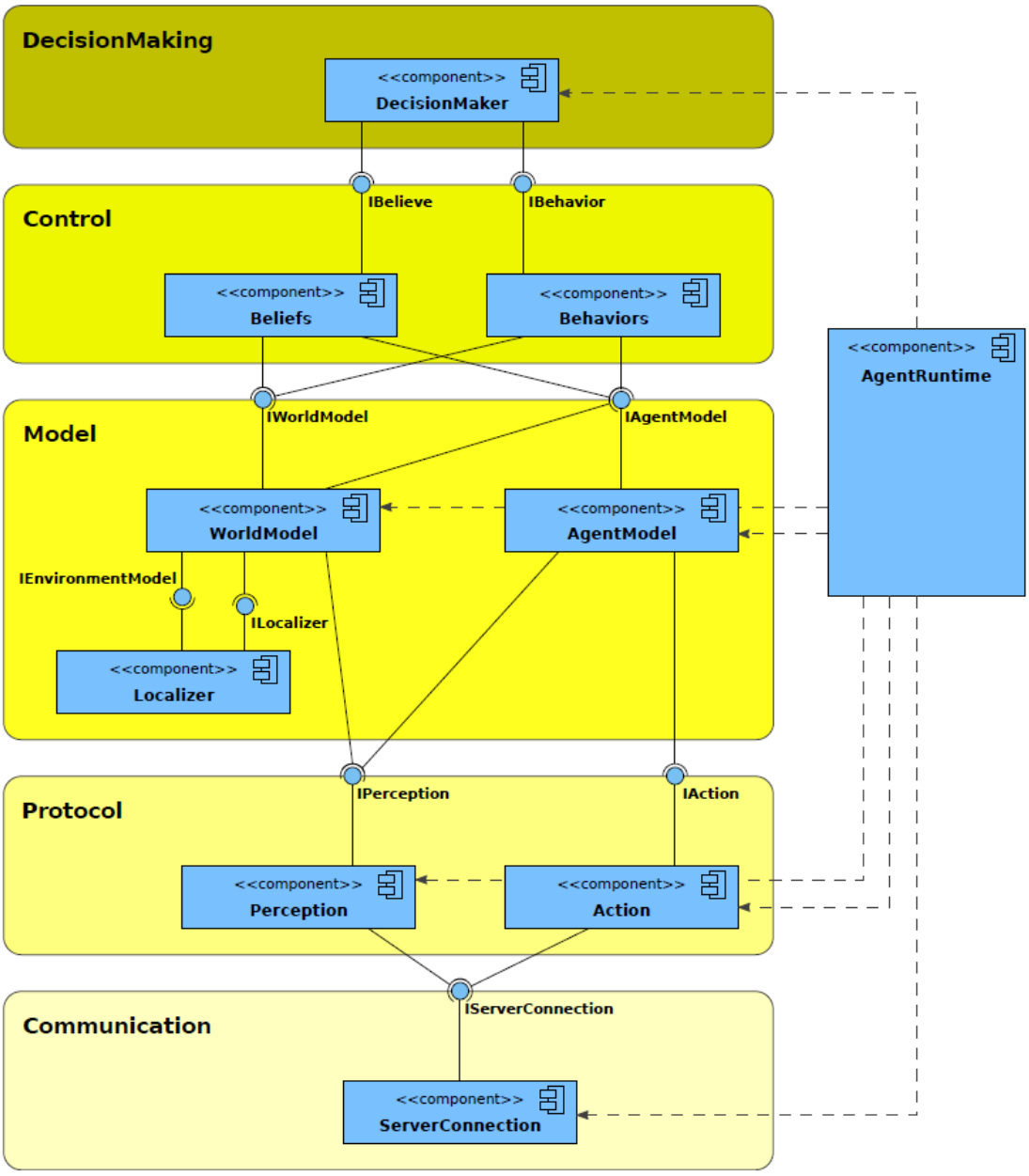


Figure 1: Layered, component-based architecture of the magma-AF.

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

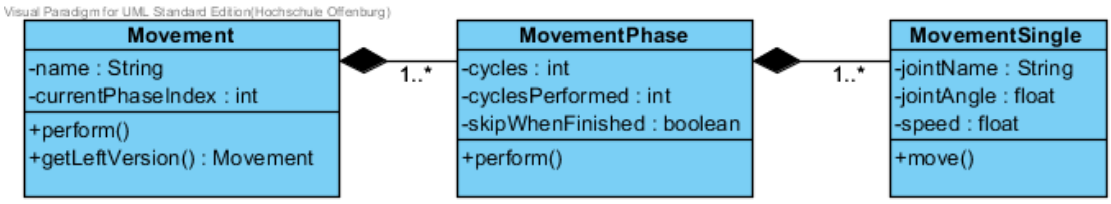
1. Analýza novej správy(Perception)
2. Aktualizácia vnútorného stavu(AgentModel)
3. Aktualizácia stavu prostredia (WorldModel)
4. Rozhodnutie o ďalšom vykonávanom pohybe (DecisionMaker)
5. Priradiť konkrétne pohyby efektorom ktoré sú uložené v AgentModel (AgentModel)

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

2.2.1.3.2 Pohyby

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

Obrázok 5 - pohyby



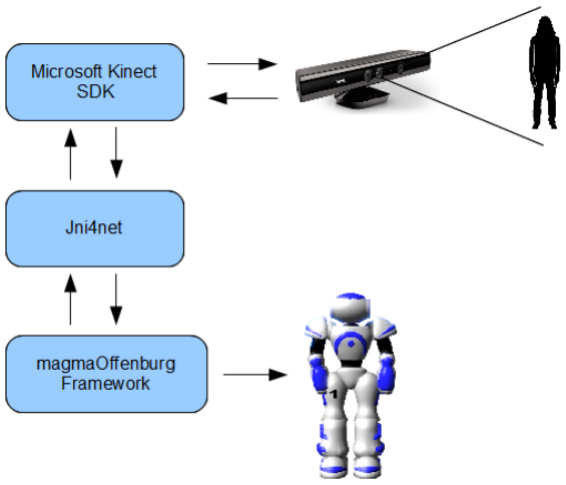
Obrázok 2 - Model "Movementframework" tímu MagmaOffenburg

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

2.2.1.3.3 KinectControl

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

Obrázok 6 - kinect



2.2.1.4 Analýza zahraničného tímu UT Austin Villa

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

2.2.1.4.1 Optimalizácia vstávania

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

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

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

2.2.1.4.2 Optimalizácia kopania

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

Fixed pose keyframe

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

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

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

2.2.1.4.3 Inverse kinematics

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

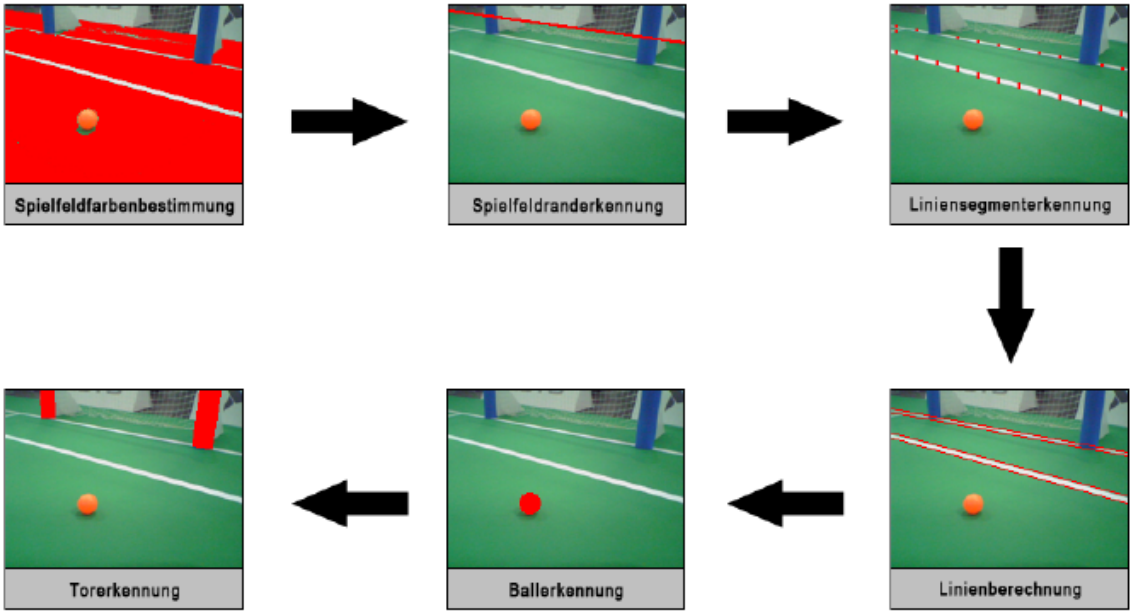
2.2.1.5 Analýza zahraničného týmu Nao-Team HTWK Leipzig

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

2.2.1.5.1 Počítačové videnie

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

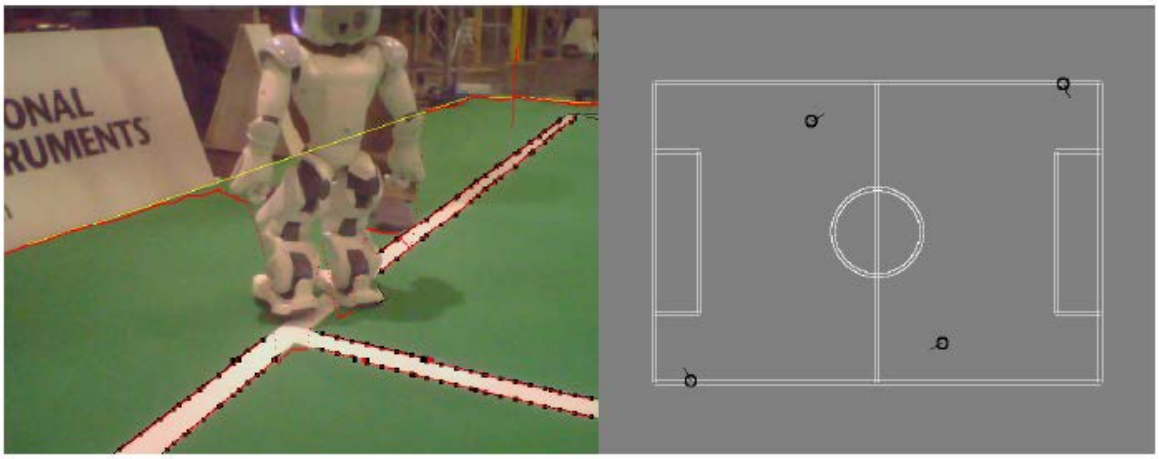
Obrázok 7 - Detekcia a rozpoznávanie objektov



2.2.1.5.2 Lokalizácia

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

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



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

Obrázok 9 - Pozícia rúk robota pri chôdzi

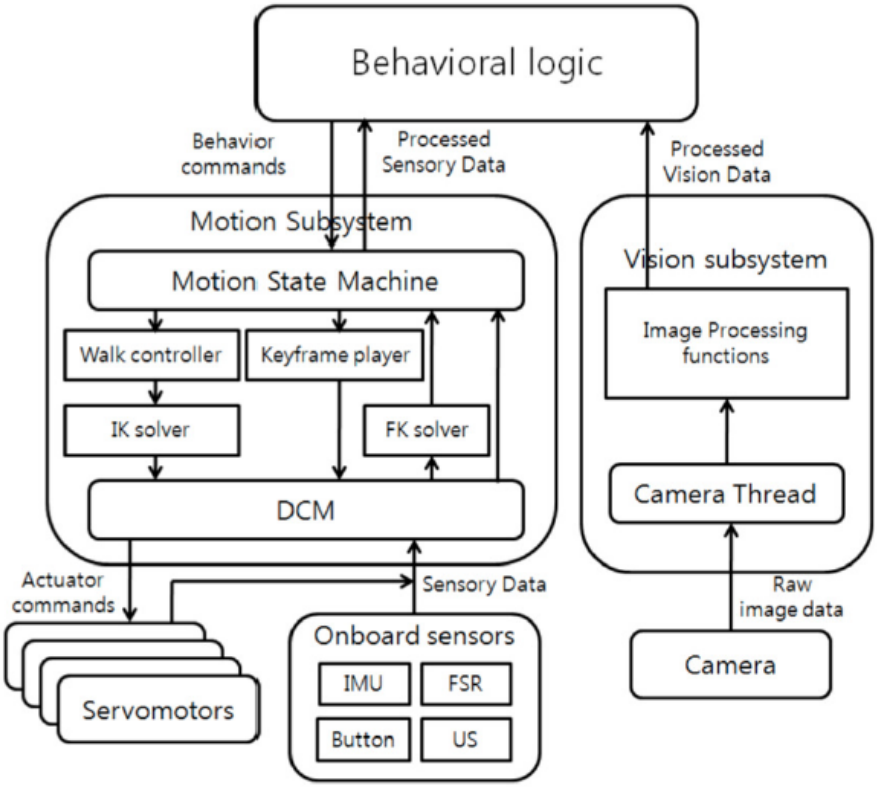
2.2.1.6 Tím Upennalizers

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

2.2.1.6.1 Súčasná architektúra

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

Obrázok 10 - logika správania



Na obrázku je zachytená kompletná architektúra, ktorá pozostáva z :

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

5. Systém videnia
6. Chovanie – logika

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

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

2.2.1.6.2 Softvérové moduly a ich rozdelenie

Detailnejšie boli rozpísané len moduly relevantné pri našom vývoji.

1.Kamera

2.Videnie

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

3.Svet

Modeluje robotov stav na ihrisku, vrátané pózy robota.

4.Telo

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

5.Pohyb

Diktuje hlavné pohyby hráčovi ako postavenie, sadnutie a pod.

6.Chôdza

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

7.Kop

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

8.Keyframes (kľúčové snímky)

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

9.Stav hry

Získava a spracúva stav hry.

10.Stav hlavy

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

11.Stav tela

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

2.2.1.6.3 Videnie

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

2.2.1.6.4 Lokalizácia

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

2.2.1.6.5 Pohyb

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

Obrázok 11 - parametre chôdze

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

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

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

2.2.1.6.6 Kopanie

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

2.2.1.6.7 Keyframing

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

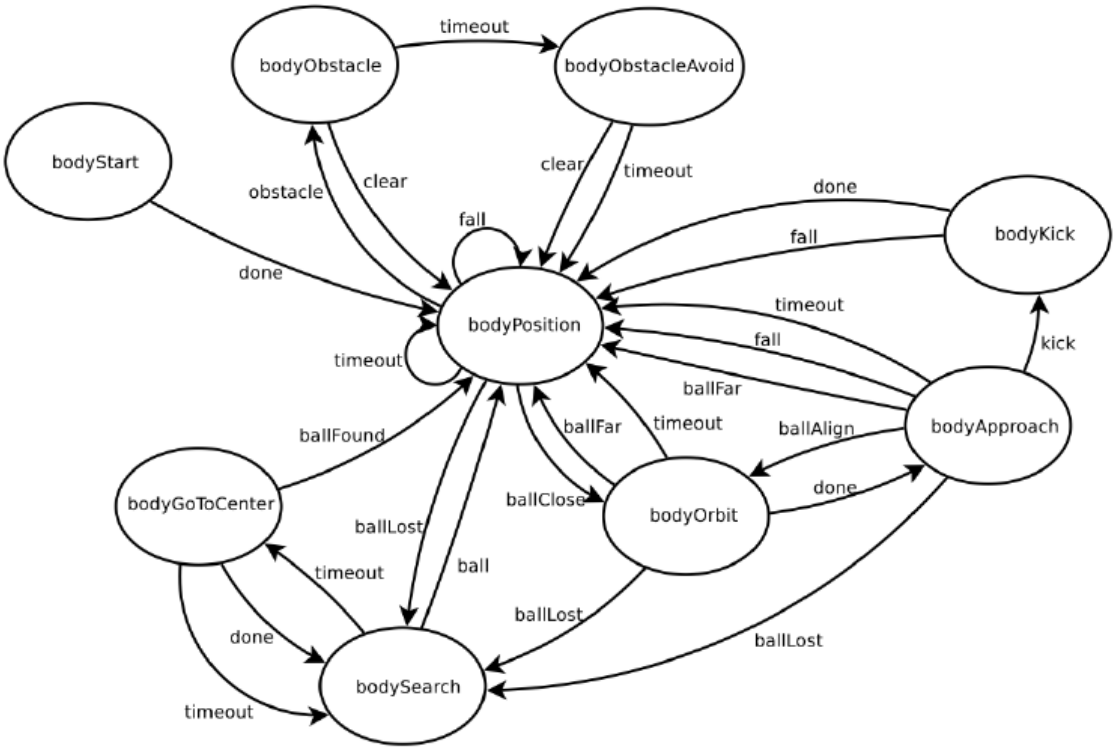
2.2.1.6.8 Kĺby

Robot tohto tímu taktiež obsahuje 22 kĺbov, ktoré plne využíva

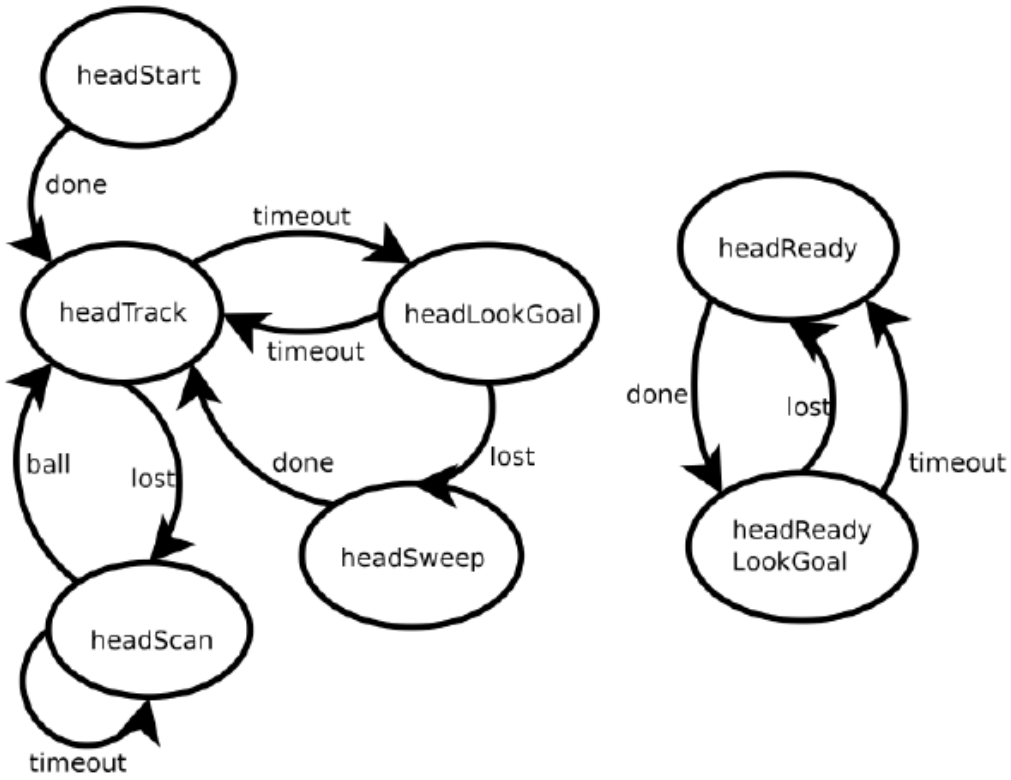
2.2.1.6.9 Správanie

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

Obrázok 12 - deterministický automat pre rozhodovanie



Obrázok 13 - deterministický automat



2.2.1.6.10 Zmena správania

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

2.2.2 Web na školskom serveri

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

2.2.3 Balíky pre ruby

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

2.2.4 Prepísanie plánovania

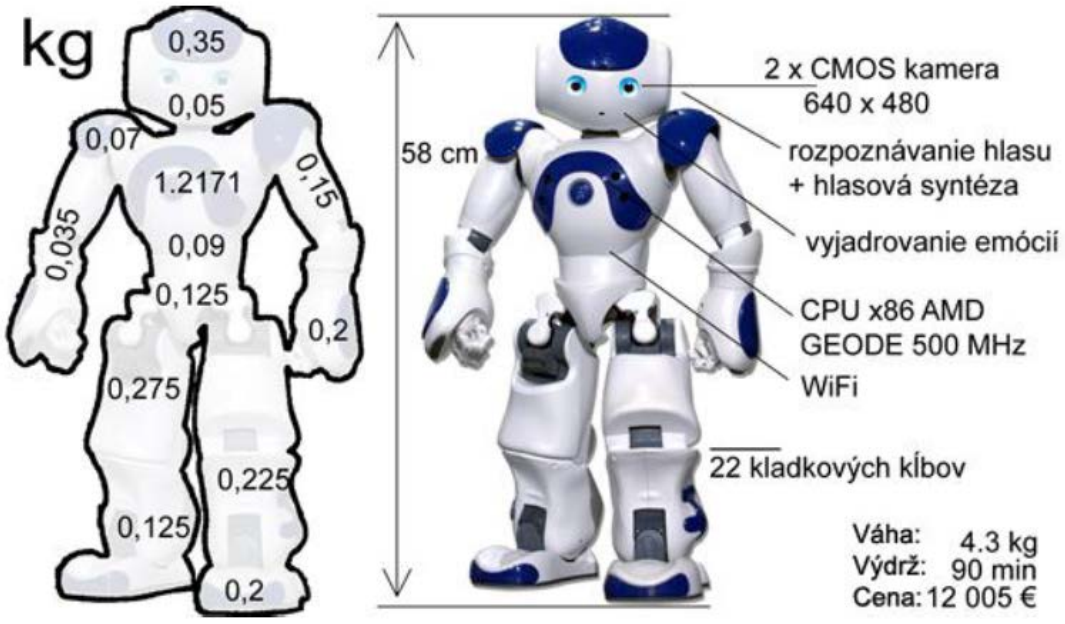
sk.fiit.jim.plan.Plan.java
Je implementovaná trieda ktorá má totožnú funkciu v Jimovy ako mal súbor plan.rb. Súčasťou tejto triedy je veľmi dôležitý rad highskillov s ktorým pracuje metóda control, ktorá buď zabezpečí vykonanie prvého z highskillov alebo v prípade ak máme rad prázdny vykoná metódu replan.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

2.2.5 Analýza diplomových prác

2.2.5.1 Motorika hráča simulovaného robotického futbalu

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

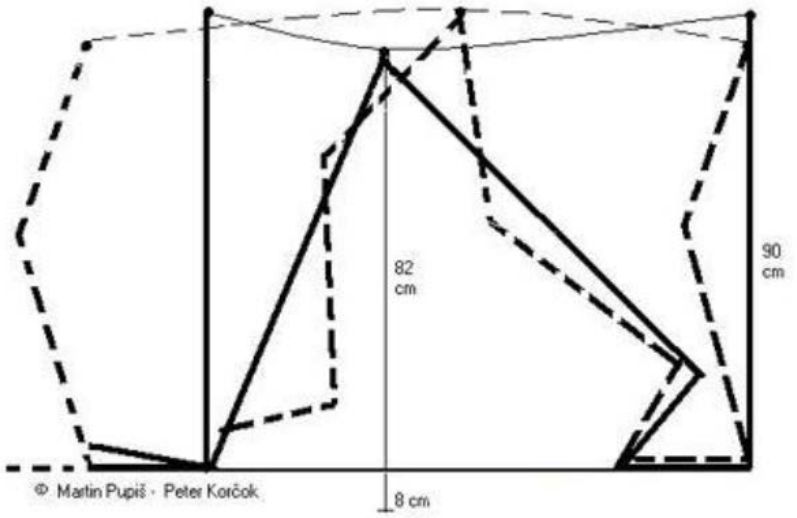
Obrázok 14 - NAO robot



2.2.5.1.1 Biomechanika chôdze a behu

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

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



2.2.5.1.2 Možnosti Nao robota

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

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

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

2.2.5.1.3 Algoritmy pre vývoj dynamickej chôdze

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

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

- **Fuzzy logika**

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

  - **Fuzzy generátor kroku**

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

  - **Fuzzy generátor sklonu tela**

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

  - **Generátor primitívnej chôdze**

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

- **Neurónové siete**

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

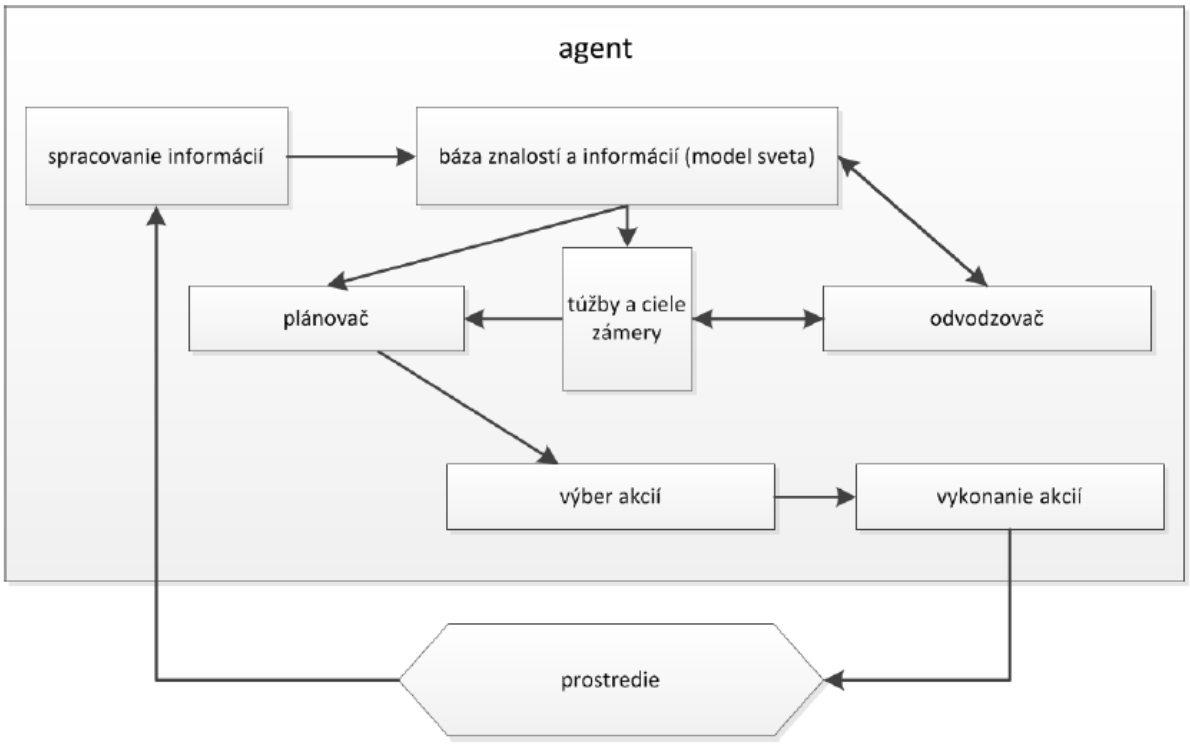
Riešené funkcie pre skvalitnenie chôdze

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

2.2.5.1.4 Model agenta

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

Obrázok 16 - architektúra BDI



2.2.5.1.5 Implementované riešenie

Zero-Moment Point

K výpočtu ZMP boli použité nasledujúce rovnice:

Obrázok 17 - vzorec

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

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

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

2.2.5.1.6 Upravované triedy

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

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

2.2.5.1.7 Vytvorené plány

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

2.2.5.1.8 Odporúčania do budúcna

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

2.2.5.1.9 Záver

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

2.2.5.2 Rozhodovacia logika

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

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

Komunikácia s ostatnými hráčmi

Tímové formácie

Vytvorenie nástroja na testovanie

2.2.5.2.1 Rozhodovanie hráča

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

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

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

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

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

2.2.5.2.2 Kolízny model hráča

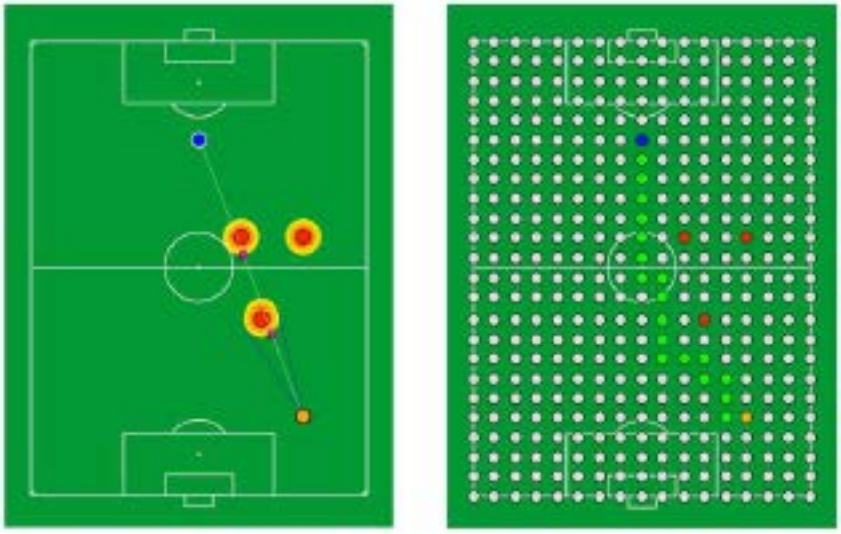
Slúži na zhodnotenie aktuálnej situácie na ihrisku z pohľadu hráča a jeho zámerov, či mu pri ich Vykonávaní hrozí kolízia s iným hráčom. Ak dôjde k odhaleniu možnej kolízie, v rámci kolízneho modelu sa navrhnú všetky potrebne opatrenia, aby sa tomu včas zabránilo.

2.2.5.2.3 Rozhodovanie pomocou grafov

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

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

Obrázok 18 - situácia z diplomovky



2.2.5.2.4 Predikcia pohybu hráča

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

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

2.2.5.2.5 Pozícia hráča vo formácii

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

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

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

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

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

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

2.2.5.2.6 Komunikácia

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

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

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

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

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

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

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

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

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

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

2.2.5.2.7 Nástroj na testovanie

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

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

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

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

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

Požadované funkcie

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

2.2.5.2.8 Implementácia rozhodovania

2.2.5.2.8.1 Kolízny model

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

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

2.2.5.2.8.2 Výber alternatívneho cieľa

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

2.2.5.2.8.3 Využitie grafov pri rozhodovaní

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

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

2.2.5.2.8.4 Predikcia pohybu hráča

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

2.2.5.2.8.5 Strategické pozície

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

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

2.2.5.2.8.6 Komunikácia

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

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

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

2.2.5.2.8.7 Nástroj na testovanie

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

2.2.5.2.9 Testovanie

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

2.2.5.2.9.1 Testovanie riešenia kolízií

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

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

2.2.5.2.9.2 Testovanie predikcie smeru pohybu hráča

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

2.2.5.2.9.3 Testovanie formácií

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

2.2.5.2.9.4 Testovanie tímového správania

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

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

2.2.5.2.9.5 Testovanie komunikácie

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

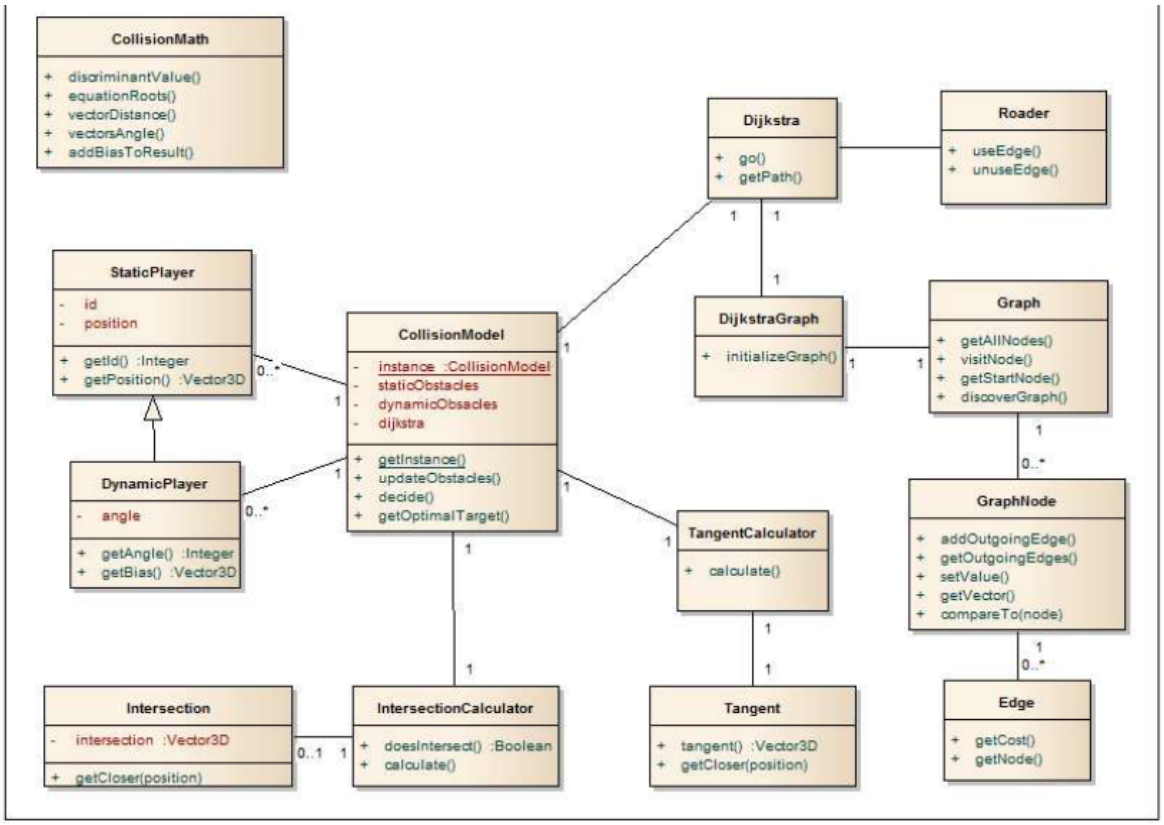
2.2.5.2.10 Vylepšenia

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

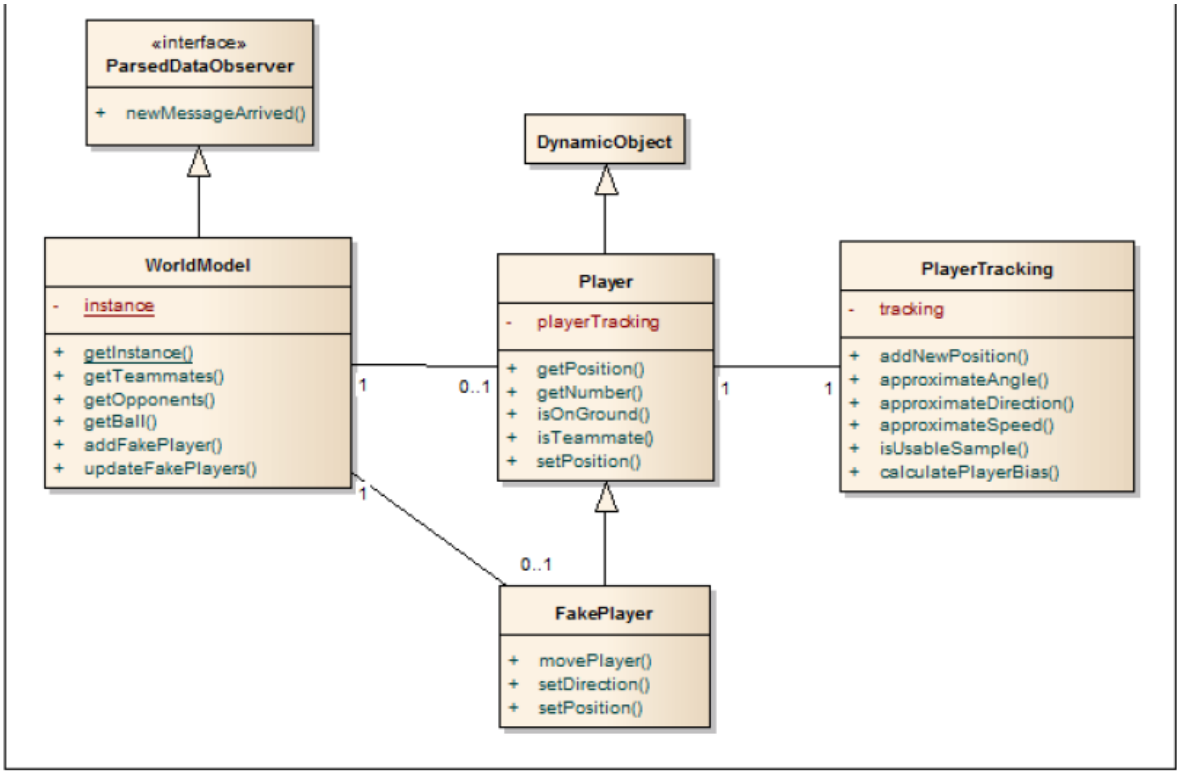
1. Zlepšenie presnosti získaných údajov
2. Predikcia pohybu hráča, tak aby sa zohľadňovala vzdialenosť hráča od prekážky
3. Vymeniť elipsu za dve kružnice, jedna kružnica by bola okolo aktuálnej polohy hráča a druha okolo odhadovanej budúcej polohy
4. Dynamické menenie rozmerov elipsy na základne pohybu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

2.2.5.2.11 Diagramy

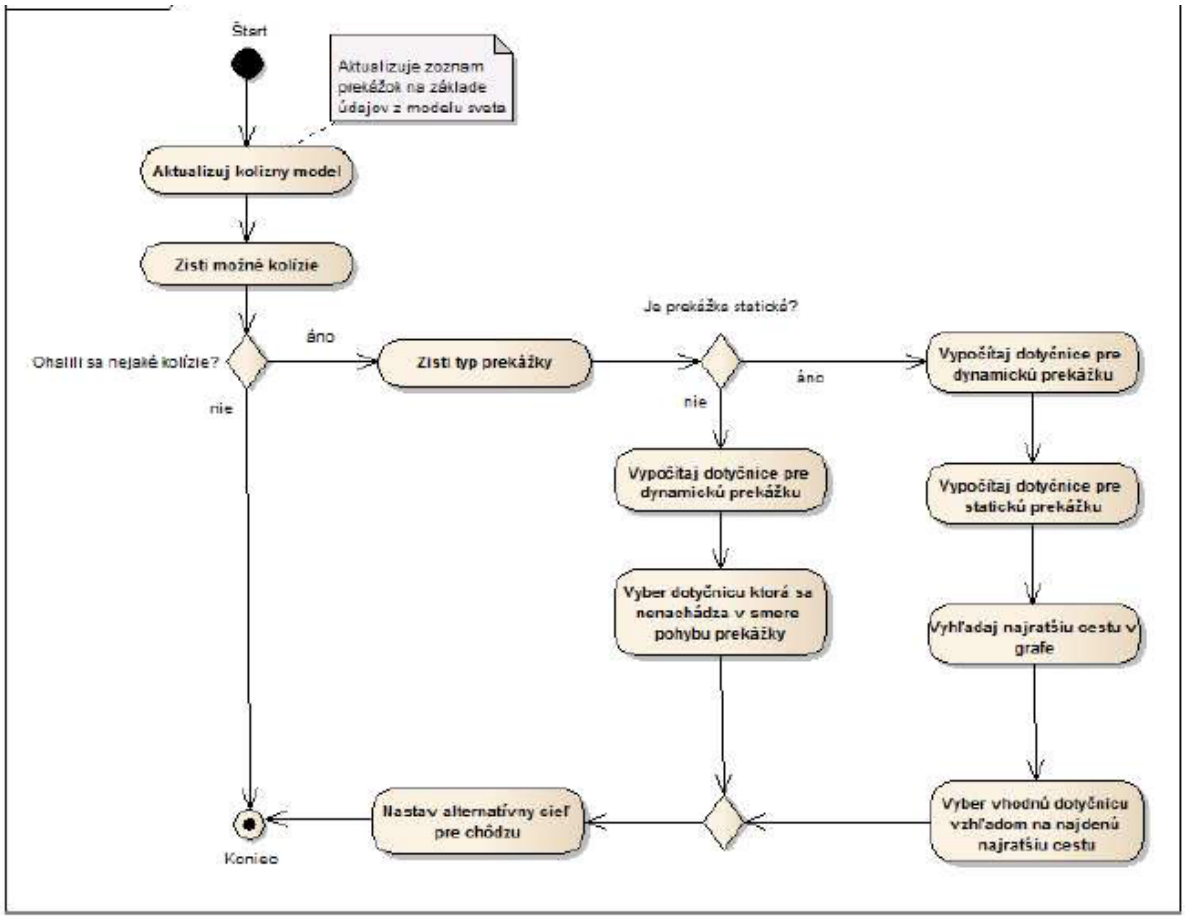
Obrázok 19 - diagram tried pre kolízny model



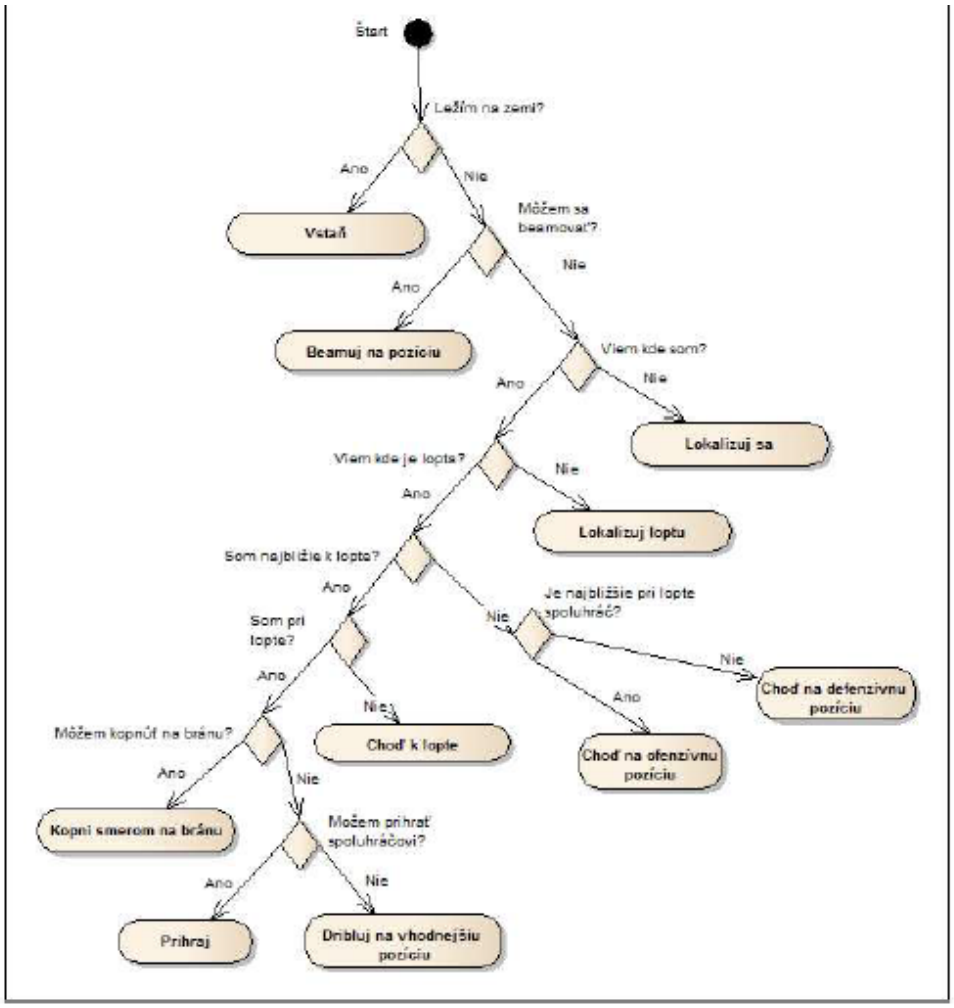
Obrázok 20 - diagram tried pre kolízny model



Obrázok 21 - algoritmus pre rozhodovanie kolízií



Obrázok 22 - Rozhodovanie hráča



2.2.6 GIT

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

2.2.7 Návody na inštaláciu

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

2.2.8 Spoločný backlog

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

2.2.9 Dokumentácia

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

2.3 Šprint č. 3

Tabuľka 4 - backlog šprint

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

Tabuľka 5 - rozdelenie úloh

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

2.3.1 Funkčnosť zakomentovaného kódu

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

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

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

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

TacticalInfo.java – nedoriešené

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

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

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

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

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

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

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

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

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

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

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

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

2.3.2 Odstránenie zakomentovaného kódu, nepoužívaných funkcií a premenných

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

2.3.3 Architektúra – situácie,taktiky, highskilly

2.3.3.1 Situácie

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

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

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

```
public enum Quadrant {
        1,2,3,4
}

class Situation {
```

```
  private Integer scoreLeft = 0;
  private Integer scoreRight = 0;
  private boolean iHaveBall = true;
  private boolean rivalHasBall = false;
  private boolean iAmNearBall = false;
  private boolean nobodyHasBall = false;
  private Quadrant iAmInQuadrant = 2;
  private Quadrant  ballIsInQuadrant = 1;
  private integer howManyPlayersAreNearBall = 2;
}
```

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

2.3.3.2 Stratégie

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

Kedy sa mení stratégia?

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

Implementácia učenia?

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

2.3.3.3 Taktiky

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

Kedy sa (ne)mení taktika?

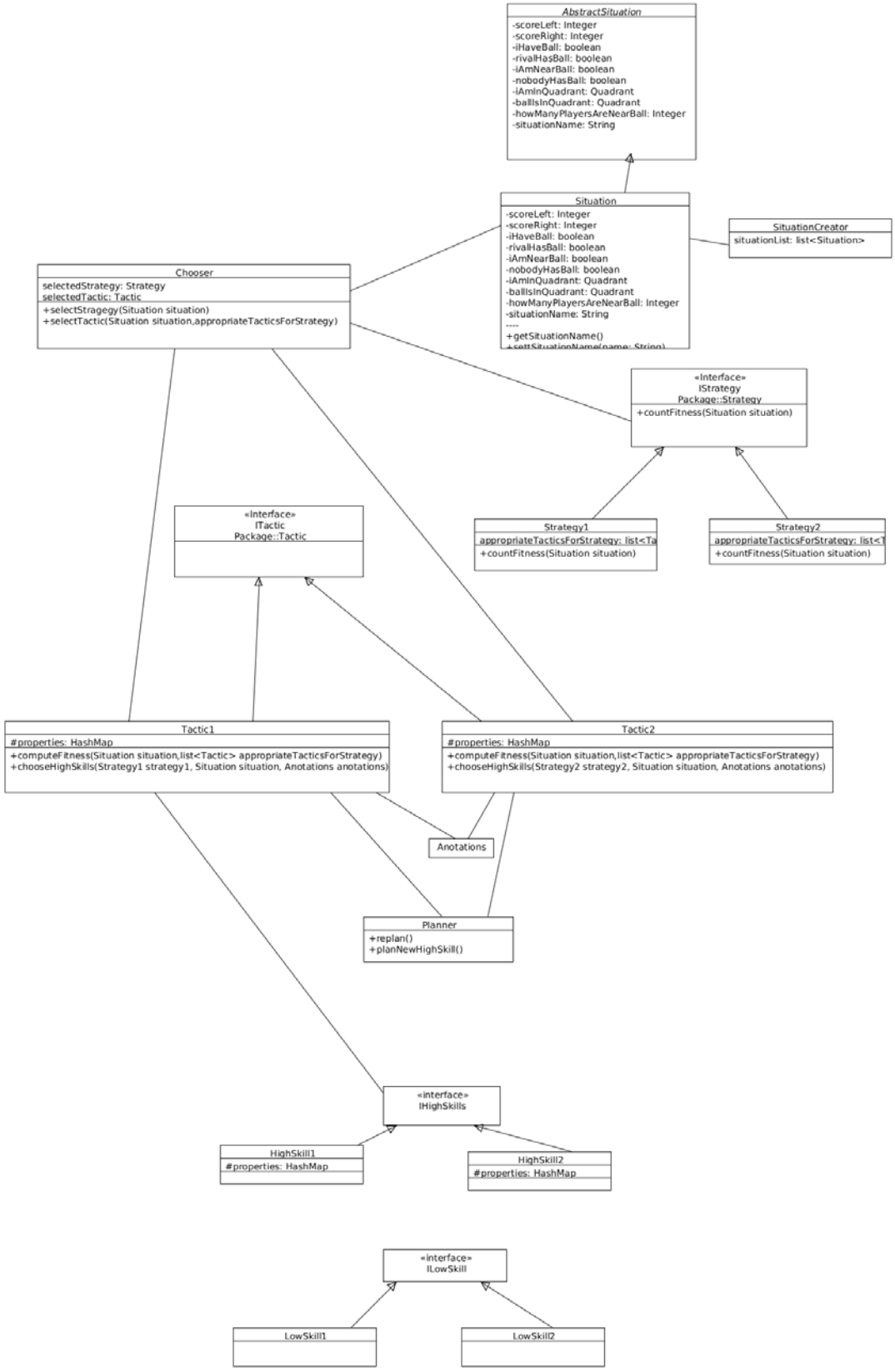
- Pokiaľ nemám oveľa lepšiu taktiku, tak ju nemením
- pokiaľ prejdem do iného kvadranta,

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

2.3.3.4 HighSkilly

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

Obrázok 23 - Návrh architektúry



2.3.4 Úprava dokumentácie

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

2.4 Šprint č. 4

Tabuľka 6 - šprint č.4

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

2.4.1 Framework – situácie

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

2.4.2 Framework – stratégie

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

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

2.4.3 Framework – taktiky

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

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

2.4.4 Dokumentácia

Dokončená dokumentácia a sprehľadnenie niektorých diagramov.

2.4.5 Selector a Observer

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

2.5 Šprint č. 5

Tabuľka 7 - šprint 5

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

2.5.1 Finalizovať dokumentáciu

Finalizácia dokumentácie prebehla spájaním všetkých ostávajúcich častí do centrálnej dokumentácie

2.5.2 Vylepšiť selector

Selector bol aktualizovaný. Chybné časti boli odstránené.

2.5.3 Otestovanie architektúry

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

2.6 Šprint č. 6

ID	AKO	CHCEM	ABY
6.1	Člen tímu	Doimplementovanie taktík	Dokončiť taktiky
6.2	Člen tímu	Vytvorenie potrebných situácií	Situácie potrebné pre rozhodovanie v taktikách
6.3	Člen tímu	Vytvorenie logiky pre taktiky	Vymyslieť správanie a rozhodovanie
6.4	Člen tímu	Prepojenie s nižšou vrstvou	Spojiť taktické rozhodovanie s vykonávaním highskillov

2.6.1 Doimplementovanie taktík

Potrebné doimplentovanie reálnych taktík. Doposiaľ bola v projekte jedna ukážková trieda pre taktiky, ktorá bola akýmsi frameworkom.

Nové zadefinované taktiky:

1. AttackMid
2. AttackLeft
3. AttackRight
4. DefendAgresive
5. Defend
6. DefendPosition

Povaha taktík vyplýva priamo z ich názvov.

2.6.2 Vytvorenie potrebných situácií

Vytvorené viaceré balíky pre situácie. Hlavnými balíkmi potrebnými pre realizovanie taktík sú : sk.fiit.jim.decision.situation.octan, sk.fiit.jim.decision.situation.

2.6.2.1 sk.fiit.jim.decision.situation.octan

Obsahuje všetky octantové situácie, pomocou ktorých vie taktika zistiť rôzne informácie o hráčovi, protihráčoch ale i lopte.

Konkrétne ide o nasledovné situácie:

1. BallInQuadrant
2. BallInMidQuadrant
3. EnemyInQuadrant
4. MeInQuadrant
5. TeamMateInQuadrant

Pre všetky vyššie uvedené situácie bolo vytvorené rovnaké množstvo podsituácií, ktoré kontrolujú konkrétny kvadrant. Na obrázku č. 24 je možné vidieť konkrétne riešenie situácie MeIn2L.

```
public class MeIn2L extends Situation {

    public static final double POSITION_MAX_X = 0;
    public static final double POSITION_MIN_X = -8;
    public static final double POSITION_MIN_Y = 0;

    private AgentModel agentModel = AgentModel.getInstance();

    public boolean checkSituation() {
        if ((this.agentModel.getPosition().getX() <= MeIn2L.POSITION_MAX_X)
                && (this.agentModel.getPosition().getX() >= MeIn2L.POSITION_MIN_X)
                && (this.agentModel.getPosition().getY() >= MeIn2L.POSITION_MIN_Y)) {
            return true;
        }
        return false;
    }
}
```

Obrázok 24 - MeIn2L situácia

2.6.2.2 sk.fiit.jim.decision.situation

V tomto balíku boli vytvorené situácie, ktoré sú špecifické pre rôzne taktiky. Konkrétne sme vytvorili nasledovné situácie:

1. BallIsOurs
2. BallIsTheir
3. BallOnGoal
4. EnemyInFrontOfMe
5. FarFromEnemy
6. FightForBall
7. NoOneHasBall

Situácie sú pomenované jednoznačne a z ich názvu vyplýva, ktorú časť pokrývajú. Na obrázku č. 25 je možné vidieť situáciu NoOneHasBall.

```
public class NoOneHasBall extends Situation {

    private FightForBall fight = new FightForBall();

    private AgentModel agentModel = AgentModel.getInstance();

    @Override
    public boolean checkSituation() {
        if (this.agentModel.getDistanceNereastToBall(WorldModel.getInstance()
                .getTeamPlayers()) >= FightForBall.EDGE_DISTANCE_FROM_BALL
                && this.agentModel.getDistanceNereastToBall(WorldModel
                .getInstance().getOpponentPlayers()) >= FightForBall.EDGE_DISTANCE_FROM_BALL) {
            return true;
        }
        return false;
    }
}
```

Obrázok 25 - NoOneHasBall situácia

2.6.3 Vytvorenie logiky pre taktiky

Pre každú taktiku bola vytvorená logika a rozhodovanie na základe situácií. Pre príklad uvedieme taktiku AttackLeft.

1. možnosť
> máme loptu && sme v kvadrante naľavo
> protihráči sú v opačnom kvadrante (pred nami)

2. možnosť

> máme loptu v kvadrante napravo

> väčšina protihráčov je v kvadrante pred nami
> potenciálne voľný hráč v ľavom kvadrante nasej časti

3. možnosť
> mame loptu v kvadrante napravo
> protihráči sú v kvadrante pred nami
> potenciálne voľný hráč v našom ľavom kvadrante

4. možnosť
> protihráčov kvadrant naľavo
> protihráči kvadrant napravo

5. možnosť
> protihráčov kvadrant napravo
> protihráči v rovnakom kvadrante

6. možnosť
> lopta nikoho
> lopta v kvadrante protihráča naľavo
> voľný hráč v kvadrante naľavo našom

7. možnosť
> lopta nikoho
> lopta v kvadrante protihráča naľavo
> voľný hráč v protihráčovom kvadrante naľavo

8. možnosť
> lopta nikoho
> lopta napravo v protihráčovom kvadrante
> voľný hráč naľavo
> náš hráč bližšie k lopte ako super

9. možnosť
> lopta nikoho
> lopta v ľavom kvadrante
> vzdialenosť oponenta a náš rovnaká

10. možnosť
> mame loptu
> sme v našom kvadrante naľavo

> lávy kvadrant súpera obsahuje menej hráčov ako súperov pravý (šanca preraziť)

11. možnosť
> nemám loptu
> kvadrant pred nami jeden super s loptou

Attack left - progress condition

1. možnosť
> spadol som
> lopta už nie je naša
> oponent je blízko (zadefinujeme blízko ? .. napi meter , dva ?)

2. možnosť
> nahral som
> lopta nie je naša && oponent ma loptu

3. možnosť
> lopta nie je nikoho
> spadol som
> oponent je bližšie ako my

4. možnosť
> lopta je naša
> útočíme
> v kvadrante pred nami je pomerná väčšina oponentov (teda náš blokujú)

5. možnosť
> mame loptu
> nemám spoluhráča ktorému by som mohol nahrať
> nemám priamy výhľad na branú (stoji predo mnou niekto)

6. možnosť
> loptu získal oponent

2.6.4 Prepojenie s nižšou vrstvou

Taktiky boli prepojené s nižšou vrstvou vyvíjanou druhým tímom. Prepojenie bolo vykonané na základe volania príkazov nižšej vrstvy tak ako je možné vidieť na obrázku č.26. Toto prepojenie bolo vykonané na všetkých potrebných miestach.

```
this.moveExec.move(Vector3D.cartesian(ballX, ballY, 0),
        MovementSpeedEnum.WALK_SLOW);
```

Obrázok 26 - Volanie nižšej vrstvy

2.7 Šprint č. 7

ID	AKO	CHCEM	ABY
7.1	Člen tímu	Unit testy pre situácie Octant	Otestovať hraničné situácie
7.2	Člen tímu	Prerobiť taktiky na Base	Zlepšenie prehľadnosti balíkov
7.3	Člen tímu	Konštanty v situáciach	Ľahšia manažovateľnosť situácií
7.4	Člen tímu	Metóda checkSituation	Možnosť zastaviť vykonávanie správania
7.5	Člen tímu	Implementácia taktík	Prerobenie taktík podľa novej dohody aby sme vedeli posielať highskilly do queue
7.6	Člen tímu	Implementácia check+abort	Možnosť zrušiť vykonávanie highskillov, kontrola v check – odstránenie neustáleho volania metódy run
7.7	Člen tímu	Selector	Prerobiť selector podľa novej dohody na check a abort
7.8	Člen tímu	Celkové testovanie taktík	Otestovať správanie robota podľa taktík

2.7.1 Unit testy pre situácie Octant

Vytvorené unit testy pre všetky hraničné situácie. Upravené veľkosti ihriska z 7.5 na 8. Vytvorené nové balíky pre testovanie:

1. sk.fiit.jim.tests.decision.situation.octan
2. sk.fiit.jim.tests.decision.situation

Vytvorené testy, ktoré odhaľujú hraničné situácie pri rozhodovaní či daná situácia platí alebo nie. Na obrázku č. 27 je možné vidieť ukážku jedného takéhoto unit testu.

Obrázok 27- Unit test pre kvadrant R1

```
@Test
public void R1Test() throws Exception {
    BallIn1R R1 = new BallIn1R();
    // wrong x
    setup(Vector3D.cartesian(-7, -2, 0));
    Assert.assertFalse(R1.checkSituation());

    // wrong y
    setup(Vector3D.cartesian(-9, 2, 0));
    Assert.assertFalse(R1.checkSituation());

    // wrong x,y
    setup(Vector3D.cartesian(-7, 2, 0));
    Assert.assertFalse(R1.checkSituation());

    // both ok.
    setup(Vector3D.cartesian(-9, -2, 0));
    Assert.assertTrue(R1.checkSituation());
    }
```

2.7.2 Prerobenie taktík na Base

Po dohode bolo potrebné prerobiť taktiky tak, aby mali svoju Base triedu, ktorá rozširovala triedu Tactic. Tieto Base triedy boli pridané do všetkých balíkov, ktoré sa týkajú taktík.

2.7.3 Konštanty v situáciách

Pridané konštanty v situáciách umožňujú lepšie manažovať situácie a ich nastavenie. Na obrázku č.28 je možné vidieť využitie týchto konštánt v kóde.

Obrázok 28 - Konštanty v situáciách

```
public class EnemyIn2L extends Situation {

    public static final double POSITION_MIN_X = -8;
    public static final double POSITION_MAX_X = 0;
    public static final double POSITION_MIN_Y = 0;

    private List<Player> opponents = WorldModel.getInstance()
            .getOpponentPlayers();

    public boolean checkSituation() {
        for (Player p : opponents) {
            if (p.getPosition().getX() > EnemyIn2L.POSITION_MIN_X
                    && p.getPosition().getX() < EnemyIn2L.POSITION_MAX_X
                    && p.getPosition().getY() > EnemyIn2L.POSITION_MIN_Y) {
                return true;
            }
        }
        return false;
    }
}
```

2.7.4 Metóda checkSituation

Reimplementácia attackRight, vytvorenie metódy checkState, respektíve presunutie existujúcich podmienok z metódy run, do metódy checkState, tak aby sa mohla volať viackrát.

2.7.5 Implementácia taktík

Upravené taktiky: AttackLeft, AttackRight, AttackMid, Defend

2.7.6 Implementácia check+aborty

Pre zlepšenie ovládania taktík bola metóda run rozdelená na dve nové metódy *checkState* a *run*, kvôli tímto zmenám bolo nutné pozmeniť aj triedu *SelectorController*, ktorá po novom volá metódu *startTactic()*.

Metóda *checkState* zisťuje aktuálny stav taktiky, podľa ktorého vieme zvoliť správny high skill. Ak sa taktiky nachádza v rovnakom stave nový high skill sa nevykonáva.

Metóda *run* už nekontroluje stav a zistené stiuácie hry ale iba vykonáva high skill na základe stavu taktiky.

Taktiež bola doimplementovaná možnosť náheho ukončenia taktiky a high skillov taktiky pomocou metódy *abort*.

2.7.7 Selector

V triede *SelectorController* vznikla podmienka pre zisťovanie rovnakej situácie. V prípade, že sa situácie nezmenili selector nebude zisťovať vhodnosť iných taktík (je to zbytočné).

V tiedach taktík bolo odstránené výtváranie nových inštancií situacií pre zisťovanie stavu taktiky. Stav taktiky sa overuje pomocou zoznamu situácií, ktoré sú posielané z triedy *SelectorController* (tak ako je to pri výbere stratégií).

2.7.8 Celkové testovanie taktík

Vytváranie jednoduchých unit testov pre situácie a testovanie samotného správania agenta na servri.

2.8 Šprint č.8

ID	AKO	CHCEM	ABY
8.1	Člen tímu	AttackMid config	Konfigurovať taktiku
8.2	Člen tímu	AttackLeft config	Konfigurovať taktiku
8.3	Člen tímu	Defense tactic config	Konfigurovať taktiku
8.4	Člen tímu	Testovanie	Správne rozhodovanie
8.5	Člen tímu	Beam	Umiestnenie hráčov
8.6	Člen tímu	AttackRight config	Konfigurovať taktiku

2.8.1 AttackMid config

Počiatočná podmienka pre attackLeft, attackRight, AttackMid bola upravená tak aby hráč bral ohľad na to kde sa nachádzajú súperi a lopta. Bolo pridané rozhodovanie a hra viacerých hráčov, teda hra počíta s id hráčmi. Najbližší hráč ide za loptou a ostatní hráči sa ho snažia podporovať.

2.8.2 AttackLeft config

Bolo pridané rozhodovanie a hra viacerých hráčov, teda hra počíta s id hráčmi. Najbližší hráč ide za loptou a ostatní hráči sa ho snažia podporovať.

2.8.3 AttackRight config

Bolo pridané rozhodovanie a hra viacerých hráčov, teda hra počíta s id hráčmi. Najbližší hráč ide za loptou a ostatní hráči sa ho snažia podporovať.

2.8.4 Defense tactic config

Bolo pridané rozhodovanie a hra viacerých hráčov, teda hra počíta s id hráčmi. Najbližší hráč ide za loptou a ostatní hráči sa ho snažia podporovať. Ide o obrannú časť hry.

2.8.5 Testovanie

Bol opravený stav kedy sme nevolali nový highskill ani v prípade, že v queue nie je žiaden iný ďalší highskill a aktuálny highskill sa už skončil.

2.8.6 Beam

V parent triede taktík z názvom tactic som vytvoril metódu beam ktorá na začiatku zápasu presunie daného hráča podľa jeho čísla na určitú pozíciu. Riešenie pracuje z číslami hráčov 1 až 4. V prípade ak nejaká taktika si chce sama definovať beam nastane overwrite tejto metódy.

2.9 Šprint č.9

ID	AKO	CHCEM	ABY
9.1	Člen tímu	Plagát	IIT SRC
9.2	Člen tímu	BeamFix	Nerovnomerné rozloženie hráčov
9.3	Člen tímu	Init Condition fix	Fixnúť init condition aby sa nevyberala len jedna taktika
9.4	Člen tímu	Player ID	Rozdelenie hráčov podľa ID
9.5	Člen tímu	Testovanie	Otestovanie taktík s dodaným kopom a pohybom
9.6	Člen tímu	Lokalizácia	Hráč vedel kde sa nachádza.
9.7	Člen tímu	Selector	Prispôsobenie selectoru pre potreby hry

2.9.1 Plagát

Bol vytvorený plagát pre potreby IIT SRC. Plagát bol tvorený na 2 krát nakoľko bolo zistené, že orientácia sa tohto roka zmenila.

2.9.2 BeamFix

Fixnutý beam hráča, nerovnomerné rozloženie. Napr. dvaja hráči bližšie k lopte a dvaja vzadu ako obrana.

2.9.3 InitCondition Fix

Počiatočná podmienka pre attackLeft, attackRight, AttackMid bola upravená tak aby hráč bral ohľad na to kde sa nachádzajú súperi a lopta.

2.9.4 PlayerID

Do všetkých taktík bolo zaimplementované rozhodovanie na základe id hráča. Toto nám umožnilo lepšie podporovanie hráčov v útoku alebo obrane.

2.9.5 Testovanie

Testovaním sa zistilo, že hráč ako taký ma problémy s lokalizáciou lopty hneď na začiatku hry. V tom prípade sa taktika nevykonáva presne podľa požiadaviek.

2.9.6 Lokalizácia

Lokalizácia hráča bola upravená nasledovne:
Oprava zahadzovania veľkých výkyvov pozície
-- pôvodne zahadzovanie (ktoré tam bolo od začiatku) sa mohlo ľahko dostať do stavu, kedy zahadzovalo každú vypočítanú pozíciu
- čekovanie lastTimeFlagSeen in Walk highskill
-- ak nie je žiadna vlajka dlho videná, agent sa otoci o 45 stupňov
- odstránené nepotrebné localizovania vo Walk highskill,
- pridaná metóda MovementHighSkill.move(speed), ktorá umožňuje volanie highskillu Walk ku lopte (nie na určitú pozíciu)

2.9.7 Selector

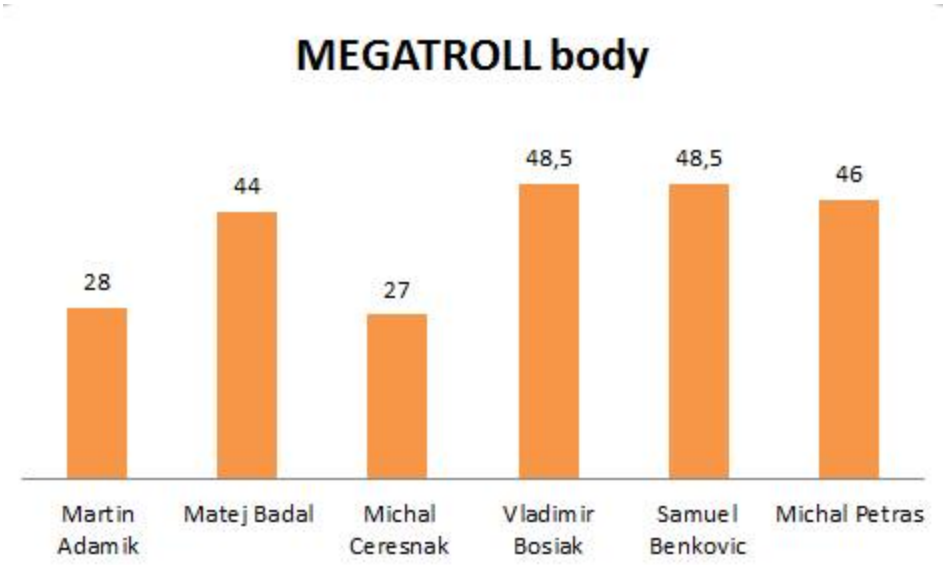
Selektor bol uzatvorenýa prehlásený za funkčný.

2.10 Šprint č. 10

ID	AKO	CHCEM	ABY
10.1	Člen tímu	Rozdelenie bodov	Výkon práce
10.2	Člen tímu	Debug panel	Vytvoriť prehľadný debug panel
10.3	Člen tímu	Debug, Fix, testing	Opraviť novo vzniknuté chyby, odstrániť novo objavené bugy
10.4	Člen tímu	IIT SRC	Zúčastniť sa prezentácie
10.5	Člen tímu	Dokumentácia	Pripraviť na odovzdanie, doplniť potrebné časti
10.6	Člen tímu	Doplniť Wiki	Aktualizovať a udržiavať pre budúcnosť

2.10.1 Rozdelenie bodov

Na základe práce počas semestra bolo vytvorené rozdelenie bodov pre všetkých členov tímu. Rozdelenie je zobrazené na nasledujúcom obrázku:



2.10.2 Debug panel

Pri debugovaní sme zistili, že by bol nápomocný debug panel, ktorý by zobrazoval potrebné údaje, kam robot smeruje a prípadne čo vidí. Následne bol vytvorený panel s podobnou funkcionalitou.

2.10.3 Debug, fix, testing

V tejto úlohe sme naďalej debugovali a testovali hru viacerých hráčov, stále sa nám nedarilo doviesť hru k úplnej funkčnosti, problémy vznikali pri začiatku hry, kedy sa absolútna pozícia lopty niekoľkonásobne líšila od tej skutočnej. Tento problém neustále pretrvával. Taktiež bol doladený beam hráčov.

2.10.4 IIT SRC

Všetci členovia tímu sa zúčastnili vedeckej konferencie.

2.10.5 Dokumentácia

Dokumentovali sme taktiky, spisovali najnovšie zmeny.

2.10.6 WIKI

Vyššie uvedené dokumentované časti sme aktualizovali aj na WIKI.

2.11 Šprint č. 11

Tento šprint slúžil na vylaďovanie posledných veľkých nedostatkov, ktoré samotný hráč má. Podarilo sa odstrániť problém chodenia za loptou. Následne sa spisoval celkový pohľad na produkt , vytvárali sa jeho diagramy a opisy.

3 Celkový pohľad

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

3.1 RoboCup

3.1.1 Náplň projektu

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

3.1.2 Úlohy projektu

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

3.1.3 Ciele projektu

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

3.2 Technológie

3.2.1 Java

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

3.2.2 Ruby

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

3.2.3 Xml

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

3.3 Architektúra

3.3.1 Jim

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

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

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

3.3.2 TestFramework

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

3.3.2.1 Moduly (Balíky)

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

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

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

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

- UI

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

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

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

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

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

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

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

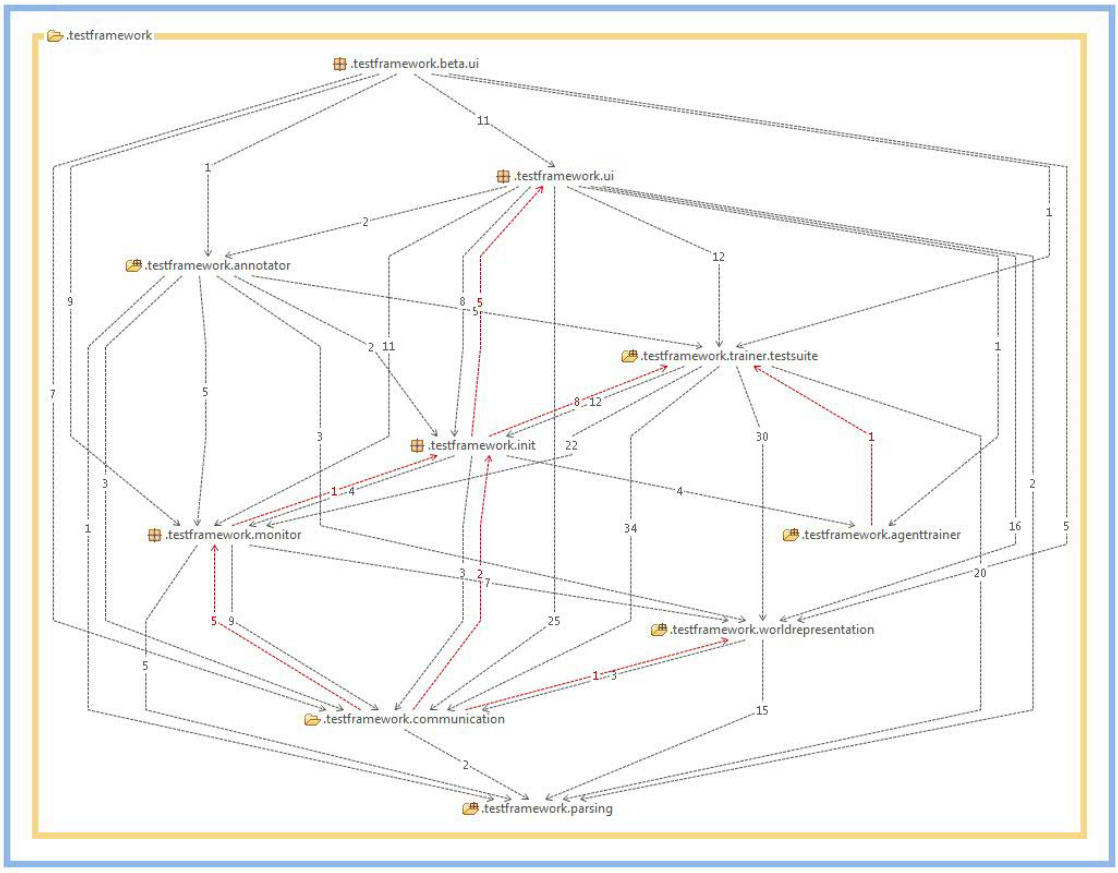
  Aktuálne existujú typ správ:
  INIT - posiela agent pri pripojení, obsahuje jeho číslo, názov tímu a stranu na ktorej hrá, takisto či má hráč zapnutý TFTP server a na ktorom porte

  DESTROY - posiela agent pri odpojení

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

WORLDMODEL - posiela model sveta agenta do test frameworku

Obrázok 29 - TestFramework



3.3.3 RoboCupLibrary

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

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

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

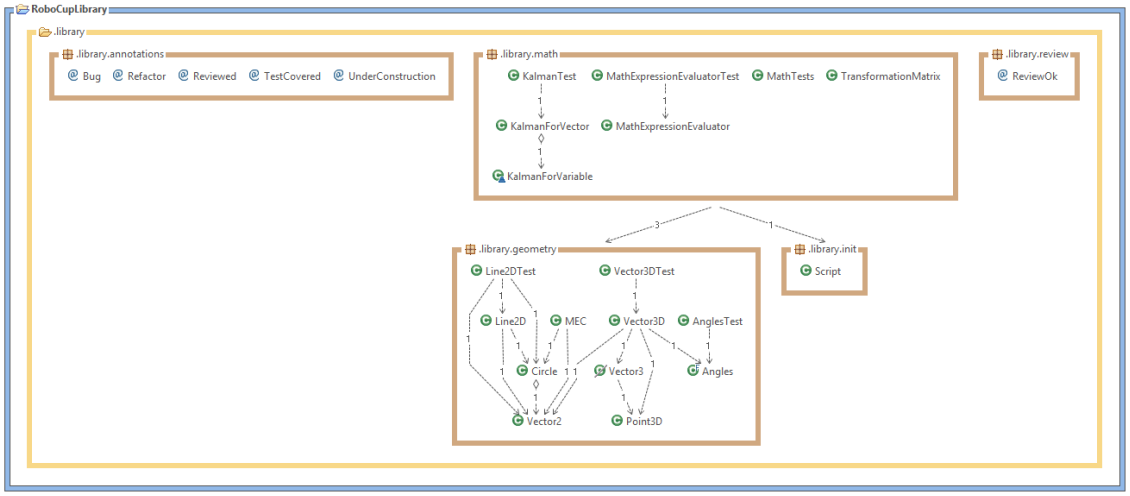
createScriptFrom(Stringfilepath): určí, ktorý skript sa má načítať podľa cesty k súboru
execute(): spustí načítaný script
registerBean(Stringname, Objectvalue): slúži iba na testovanie triedy Script
fetchBeans(String...names): nie je v programe vôbec využitá

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

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

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

Obrázok 30 - RobocupLibrary



3.4 Zmeny

3.4.1 Odstránenie ruby

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

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

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

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

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

3.4.2 Zmena architektúry

3.4.2.1 Situácie

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

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

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

```
public enum Quadrant {
        1,2,3,4
}

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

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

3.4.2.2 Stratégie

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

Kedy sa mení stratégia?

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

Implementácia učenia?

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

3.4.2.3 Taktiky

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

Kedy sa (ne)mení taktika?

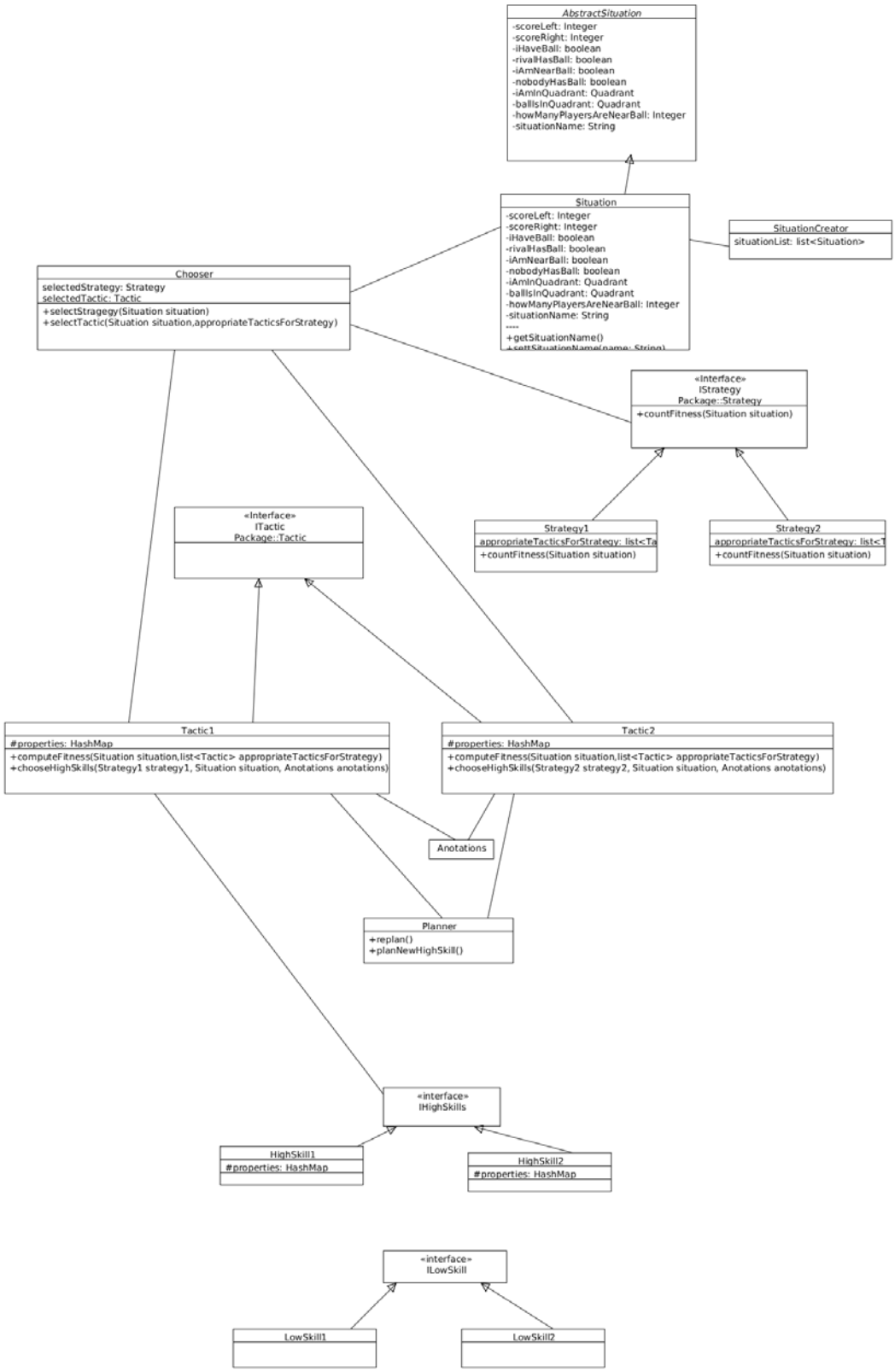
- Pokiaľ nemám oveľa lepšiu taktiku, tak ju nemením
- pokiaľ prejdem do iného kvadranta,

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

3.4.2.4 HighSkilly

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

Obrázok 31 - Návrh architektúry



3.4.3 Revízia architektúry

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

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

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

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

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

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

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

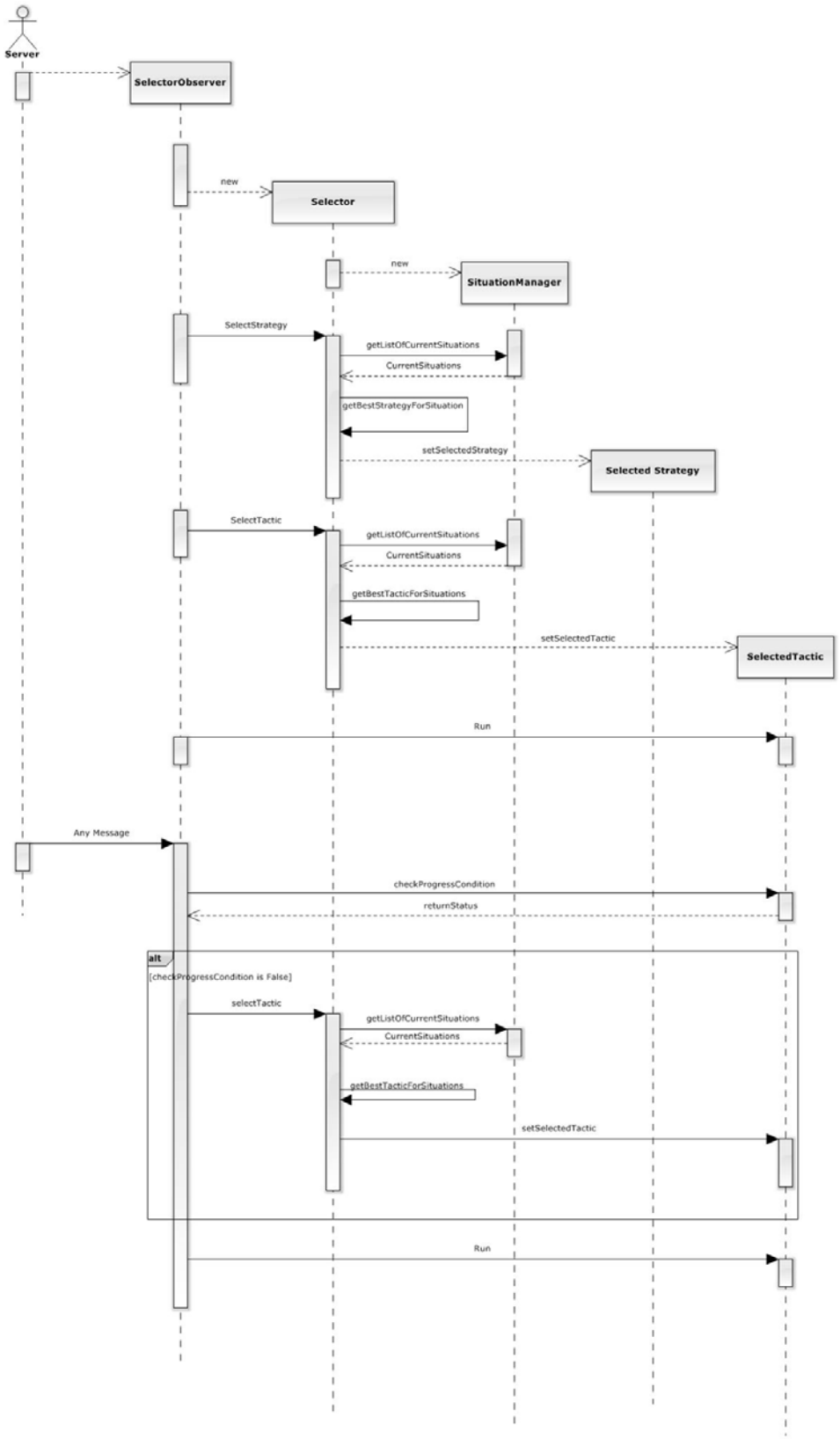
- *getInitCondition* - Podmienka ktorá musí byť splnená nato aby sa mala šancu vybrať daná taktika. Volá ju *selector*.
- *getSuitability* - V prípade ak viac taktík ma splnenú *InitCondition* dochádza k problému kedy treba vybrať jednu konkrétnu. Zložitejším algoritmom vypočítame číslo ku každej spornej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

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

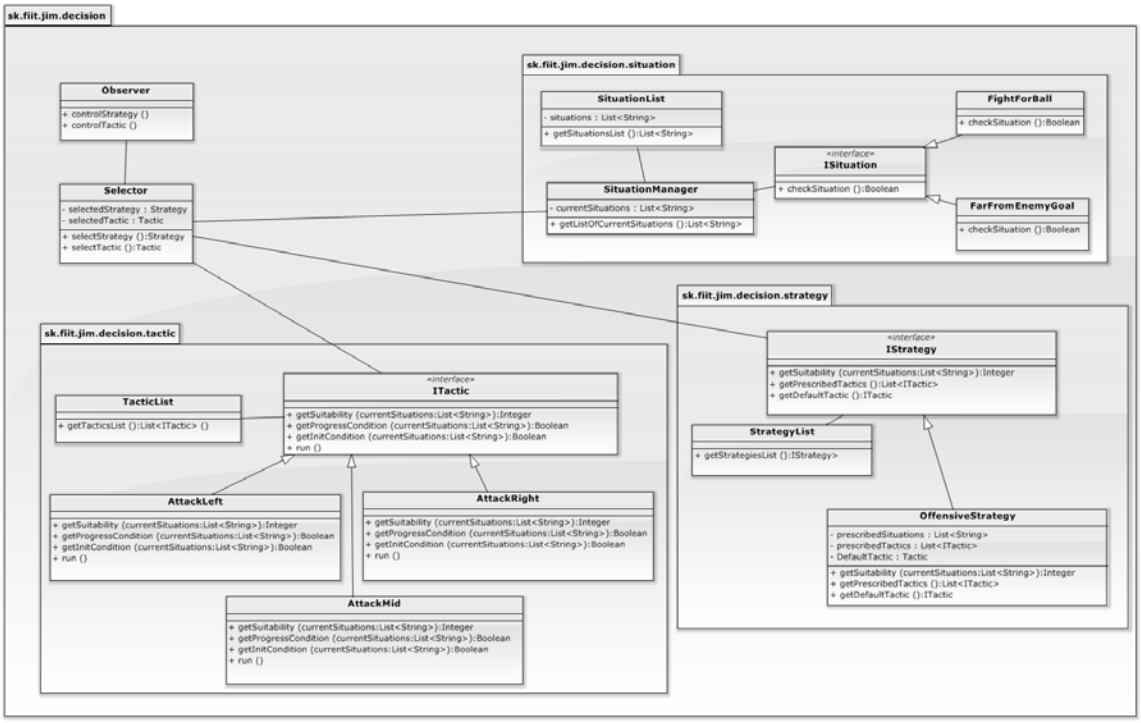
Celý balík obsahuje dve dôležité triedy. Jednou z nich je *SelectorObserver. Je to* trieda ktorá je volaná vždy keď hráč dostane zo servera akúkoľvek správu. Volá metódy *controlTactics* a *controlStrategy*. V metóde *controlTactics* sa kontroluje *ProgressCondition* aktuálnej taktiky. Pri nesplnení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

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

Obrázok 32 - sekvenčný diagram novej architektúry



Obrázok 33 - Diagram tried a Balíkov



3.5 Logika rozhodovania

Logika rozhodovania hráčov musí byť pomerne dobre premyslená. Nemalo by dochádzať ku kolíziám ani nezmyselným rozhodnutiam. Všetky tieto problémy je potrebné riešiť na úrovni taktík, kde na základe určitých podmienok prebehne vykonávanie nižších pohybov.

3.5.1 Spolupráca hráčov

Po dlhej diskusii sme dospeli k záveru, že samotní hráči musia poznať aspoň nejaké informácie o pozíciách ostatných spoluhráčov. Hlavný problém spočíval v tom, že ak by vykonávanie taktiky nasmerovalo všetkých spoluhráčov napríklad k lopte, mohlo by sa stať , že sa zrazia.

Je potrebné aby hráči spolupracovali v tzv. formáciách, ktoré budú viesť tím súbežne , logicky s loptou.

Pre ilustračný príklad môžeme uviesť možnosť paralelného útoku dopredu, kedy najbližší hráč prevezme loptu a jeho spoluhráči pokračujú spolu súbežne sním dopredu.

3.6 Taktiky

Časť hráča, ktorá ma na starosti taktické rozhodovanie v rámci hry. Dochádza k výberu najlepšej taktiky na základe preddefinovaných situácií pre každú taktiku. Po výbere konkrétnej taktiky dochádza k vykonávaniu a volaniu nižších pohybov.

3.6.1 Rozdelenie taktík

Taktiky sú rozdelené v rámci balíkov na útočné a obranné. V každom balíku je zahrnutých viacero taktík, ktoré zahŕňajú rôzne situácie. V našom projekte sme sa snažili pokryť základné rozdelenie tak aby sme ukázali možnosti nami vytvoreného frameworku.

3.6.2 Útočné taktiky

Útočné taktiky sú rozdelené na :

AttackLeft
AttackRight
AttackMid

3.6.2.1 AttackLeft

- taktika útočenia vľavo
- Musí spĺňať podmienky ako lopta vľavo ale aj najmenej hráčov v ľavo

3.6.2.2 AttackRight

- Taktika útočenia vpravo
- Musí spĺňať podmienky ako lopta vpravo ale aj najmenej hráčov vpravo

3.6.2.3 AttackMid

- Taktika útočenia stredom
- Ihrisko bolo rozdelené aj na stredné kvadranty, kontrolujú sa presné pozície

3.6.3 Obranné taktiky

Obranné taktiky sú rozdelené na :
Defend
DefendBase
DefendAgressive

3.6.3.1 Defend

- Základná taktika pre obranu

3.6.3.2 DefendBase

- Bránenie našej základne
- Skôr pasívne bránenie

3.6.3.3 DefendAgressive

- Bránenie útoku od nepriateľa
- Skôr agresívne bránenie

3.7 Situácie

3.7.1 Rozdelenie situácií

Situácie sú rozdelené podľa toho čoho sa týkajú. Rozdelené sú podľa balíkov na situácie týkajúce sa octantov a ostatné situácie.

3.7.2 Octant situácie

3.7.2.1 BallIn

Situácie ,ktoré riešia pozíciu lopty v konkrétnom kvadrante. Vytvorené sú jednotlivé situácie pre každý a jeden kvadrant samostatne.

3.7.2.2 MeIn

Situácie, ktoré riešia pozíciu mňa ako hráča v kvadrante. Vytvorené sú jednotlivé situácie pre všetky kvadranty.

3.7.2.3 EnemyIn

Situácie, ktoré riešia pozíciu nepriateľa v jednotlivých kvadrantoch. Vytvorené sú jednotlivé pre všetky kvadranty.

3.7.2.4 TeamMateIn

Situácie, ktoré riešia kontrolu pozície v jednotlivých kvadrantoch. Situácie sú taktiež vytvorené pre všetky existujúce kvadranty.

3.7.3 Ostatné situácie

3.7.3.1 BallAt

Situácie, ktoré riešia pozíciu lopty vľavo, vpravo alebo v strede.

3.7.3.2 BallCloseToMe

Situácia , ktorá vyhodnocuje či je lopta bližšie kumne ako k nepriateľovi.

3.7.3.3 BallFarFromMe

Situácia, ktorá vyhodnocuje či je lopta ďalej odo mňa ako od nepriateľa.

3.7.3.4 BallIsOurs

Situácia, ktorá vyhodnocuje či je lopta naša.

3.7.3.5 BallIsTheirs

Situácia, ktorá vyhodnocuje či je lopta nepriateľova.

3.7.3.6 MostEnemyIn

Situácie, ktoré vyhodnocujú najvyšší počet nepriateľov vľavo, vpravo alebo v strede.

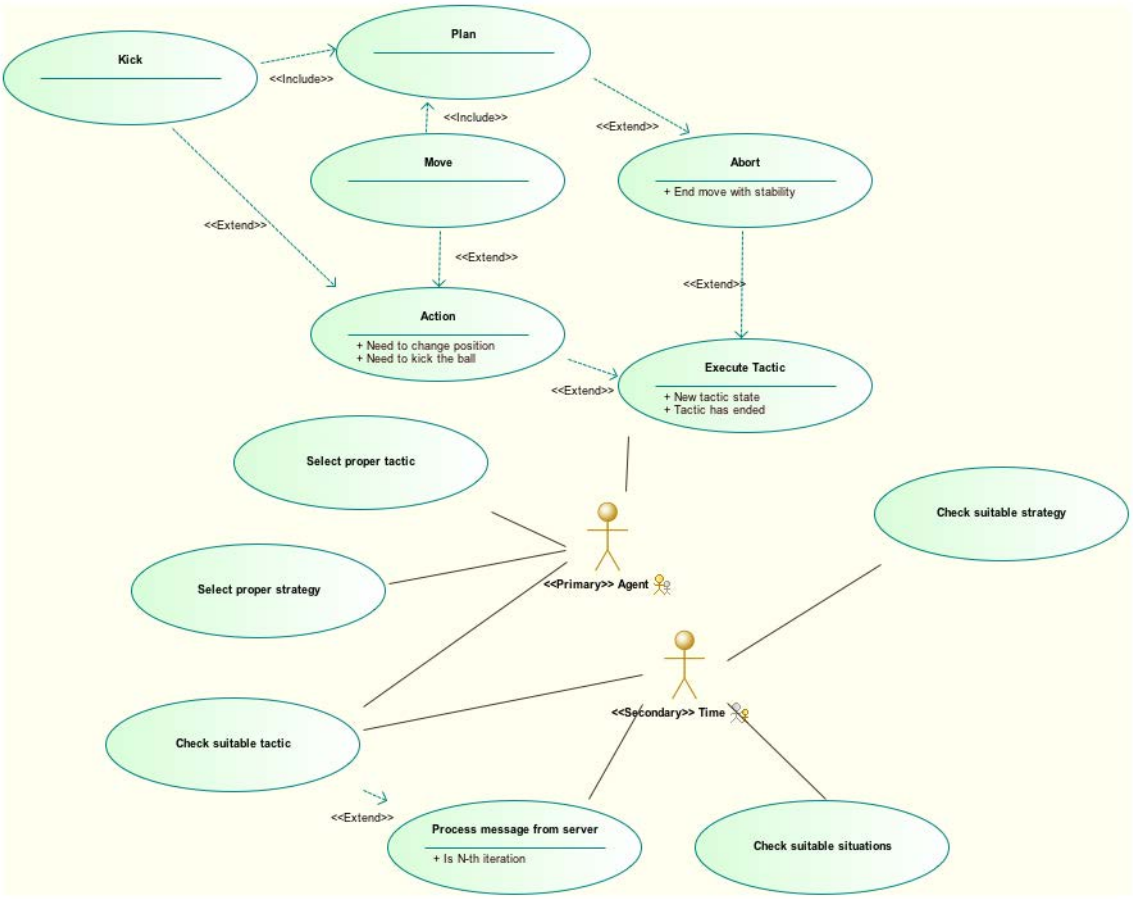
3.7.3.7 NoOneHasBall

Situácia, ktorá zisťuje stav – nikto nemá loptu.

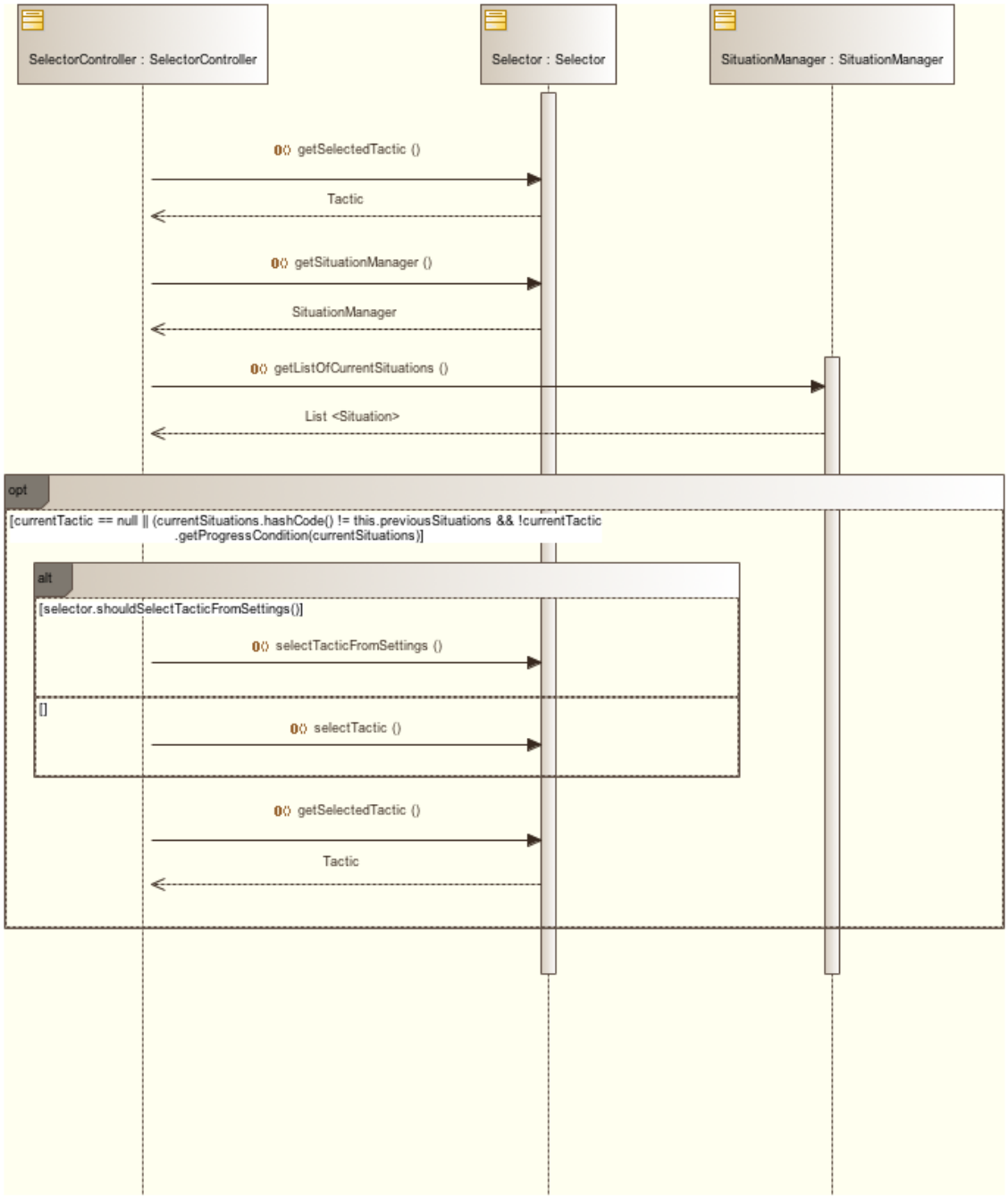
3.7.3.8 FightForBall

Situácia, ktorá je vyhodnotená len vtedy, ak prebieha boj o loptu.

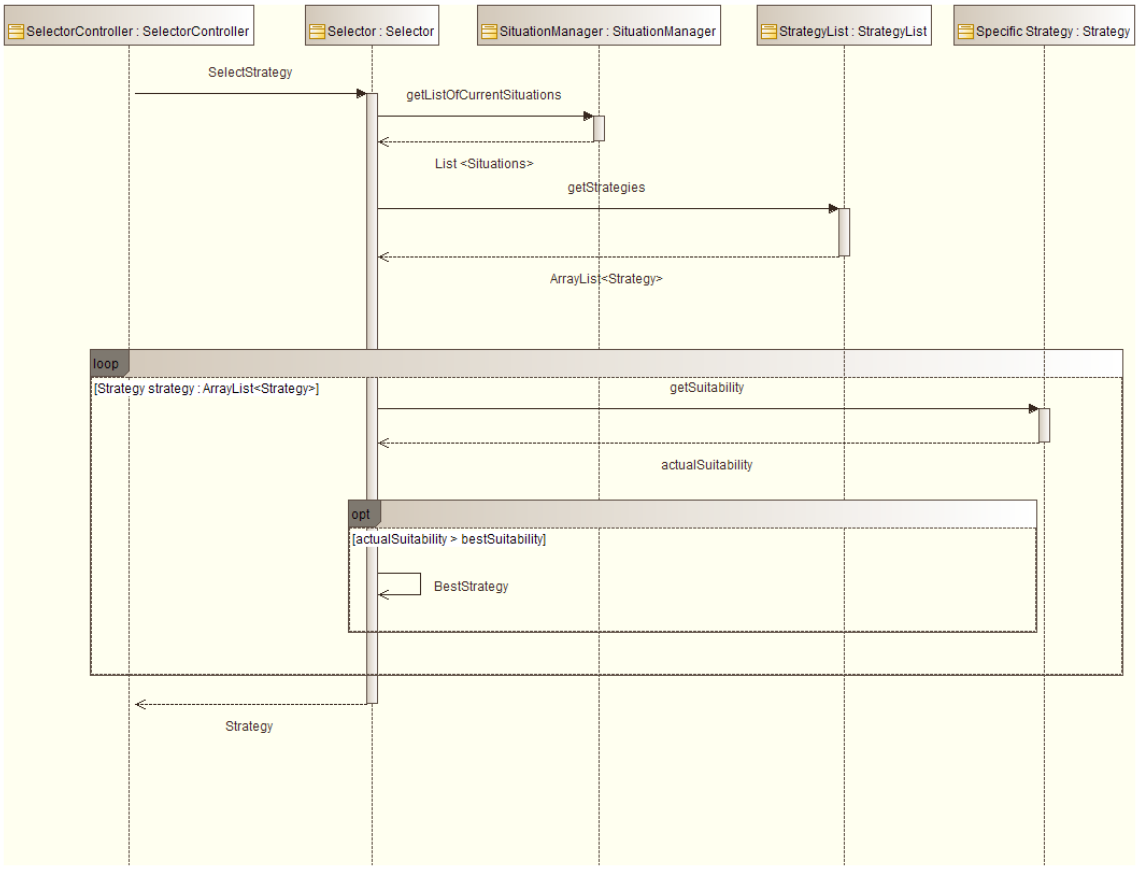
3.8 Diagramy



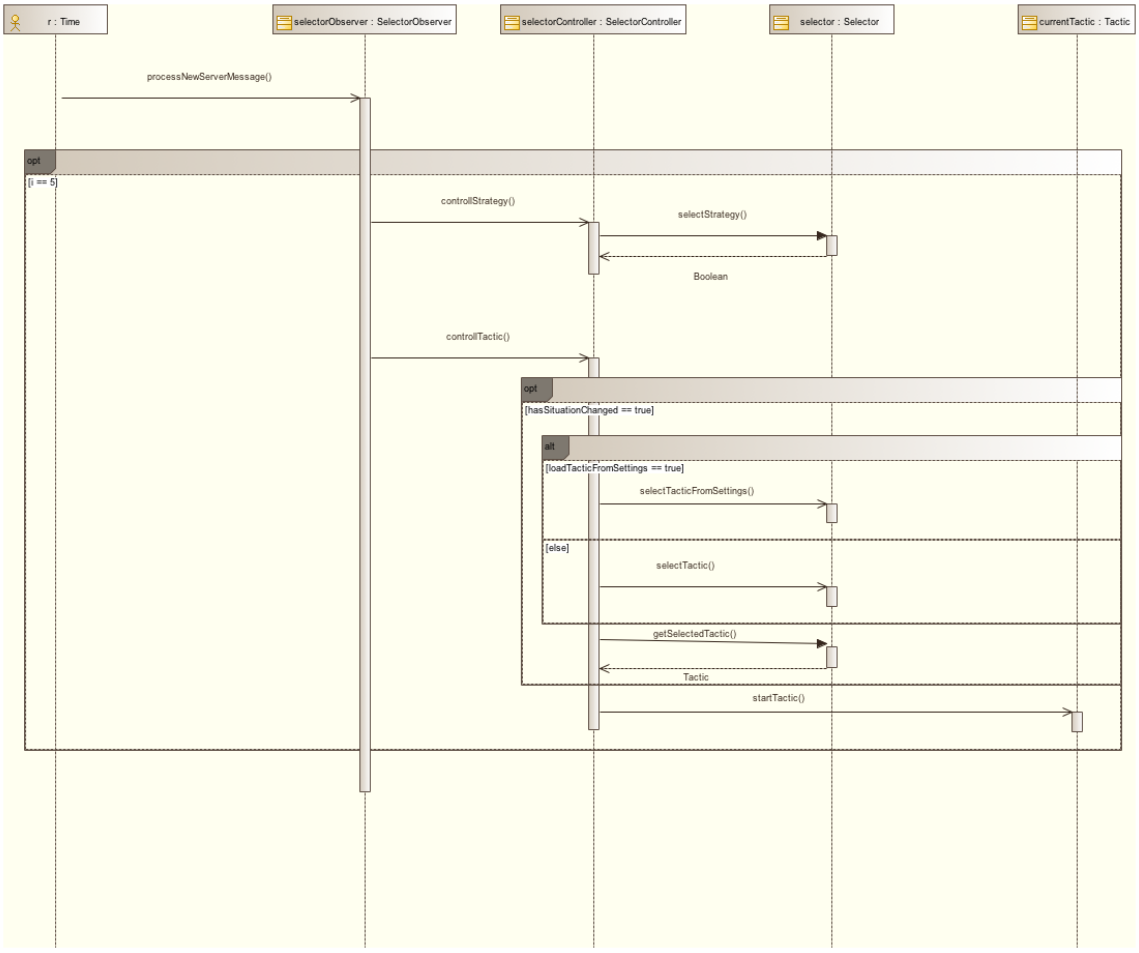
Obrázok 34 - Use Case diagram



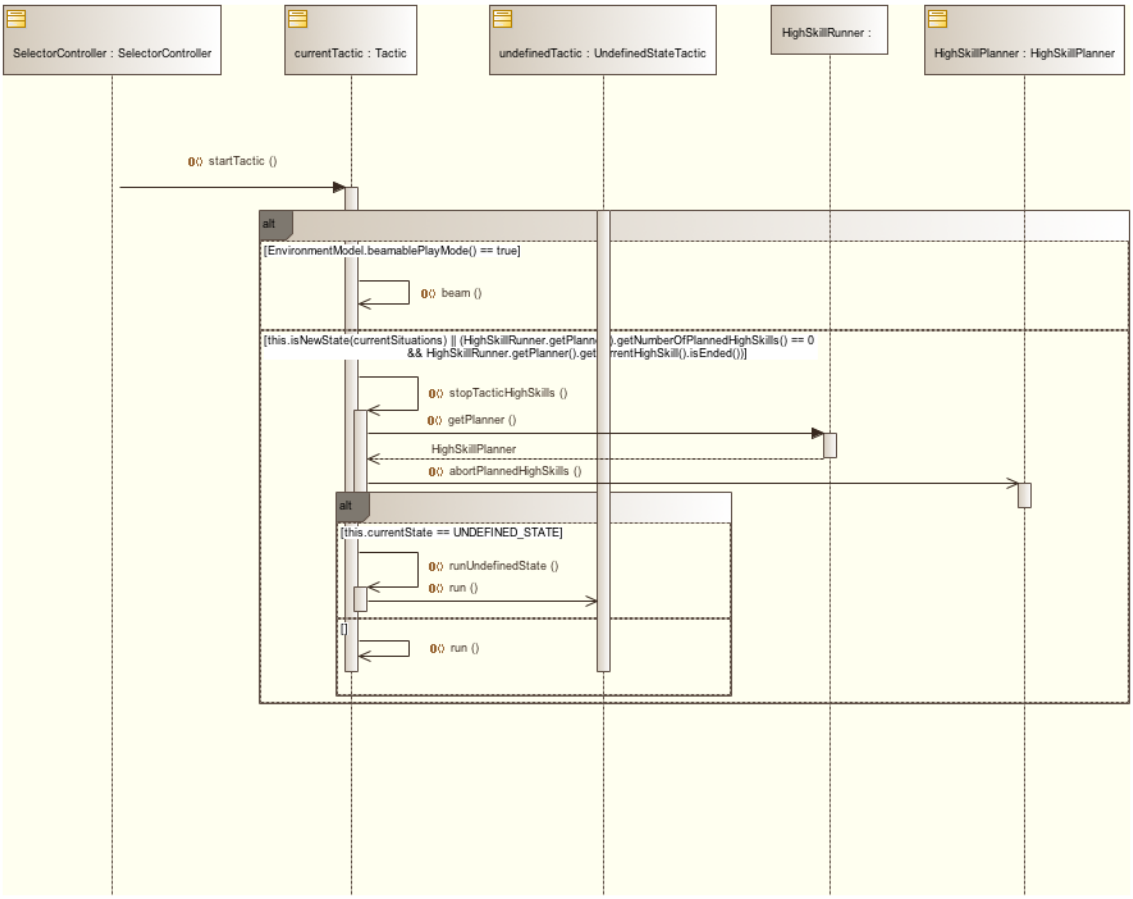
Obrázok 35 - Select Proper Tactic



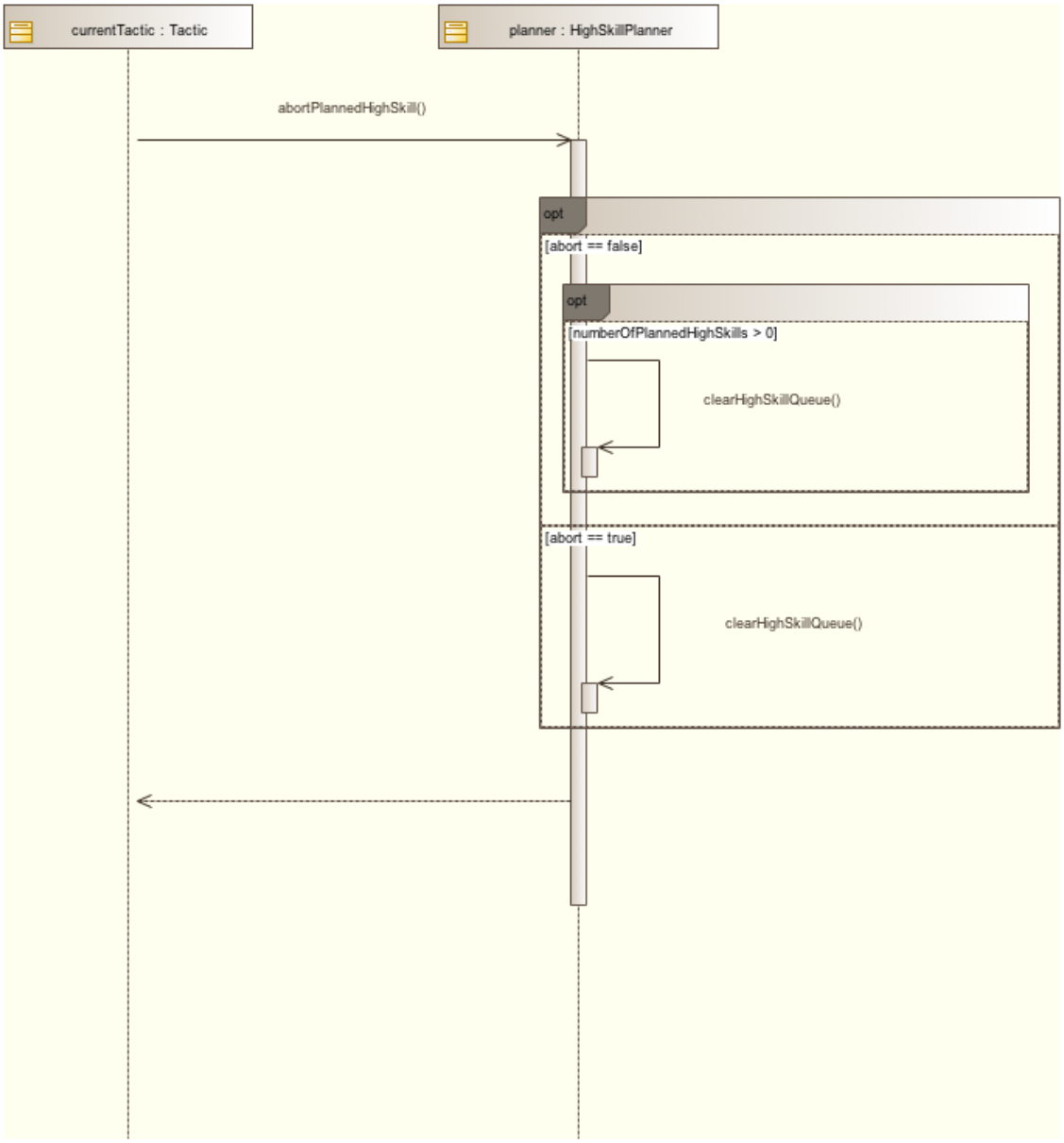
Obrázok 36 - Select Proper Strategy



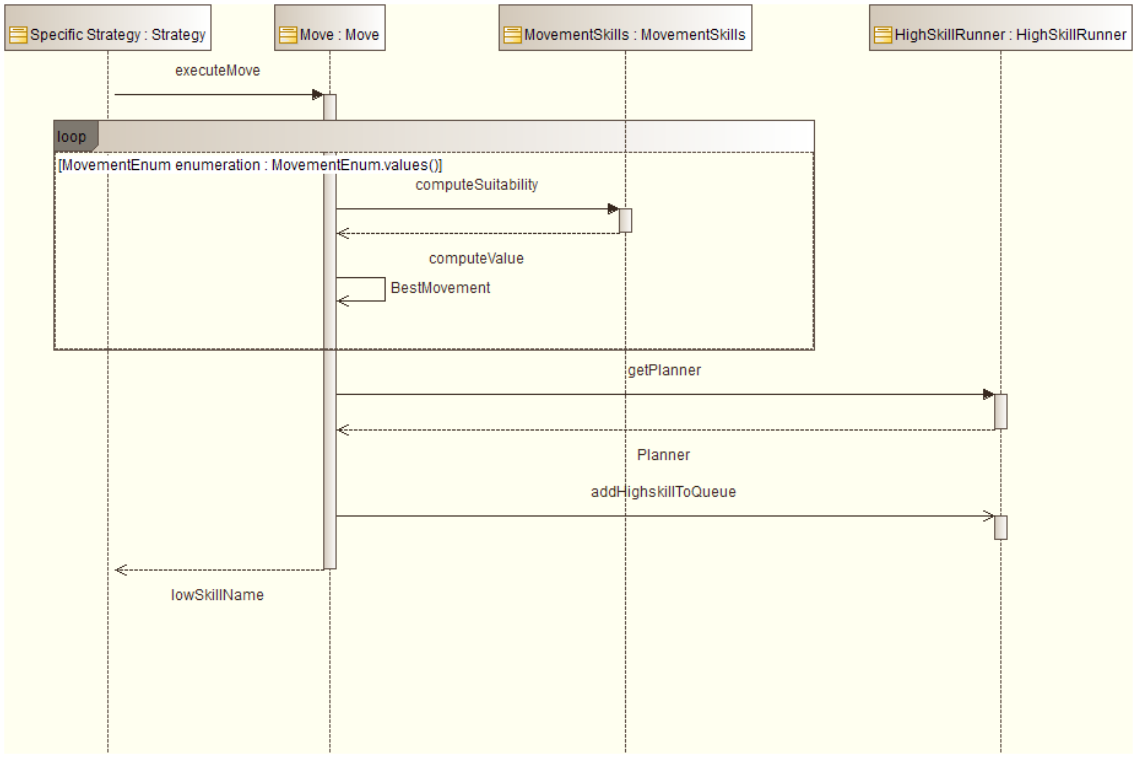
Obrázok 37 - Process new message from server



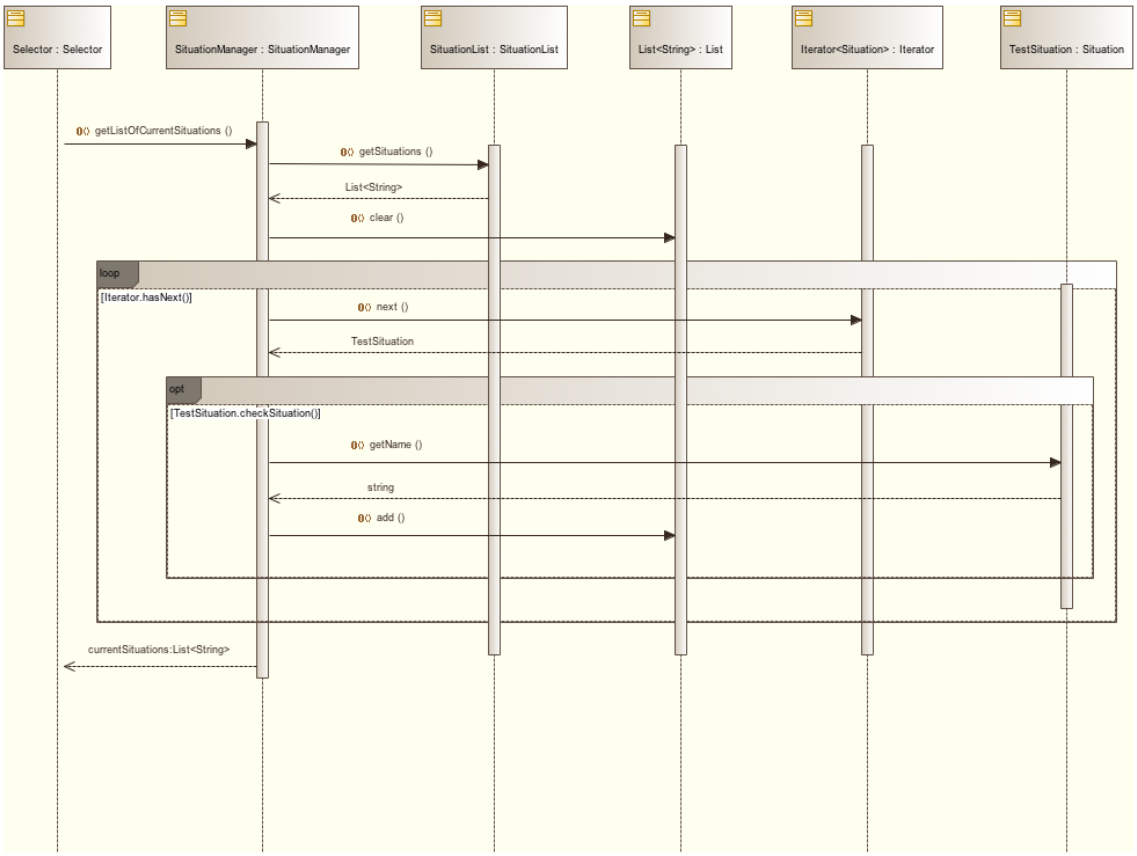
Obrázok 38 - Execute tactic



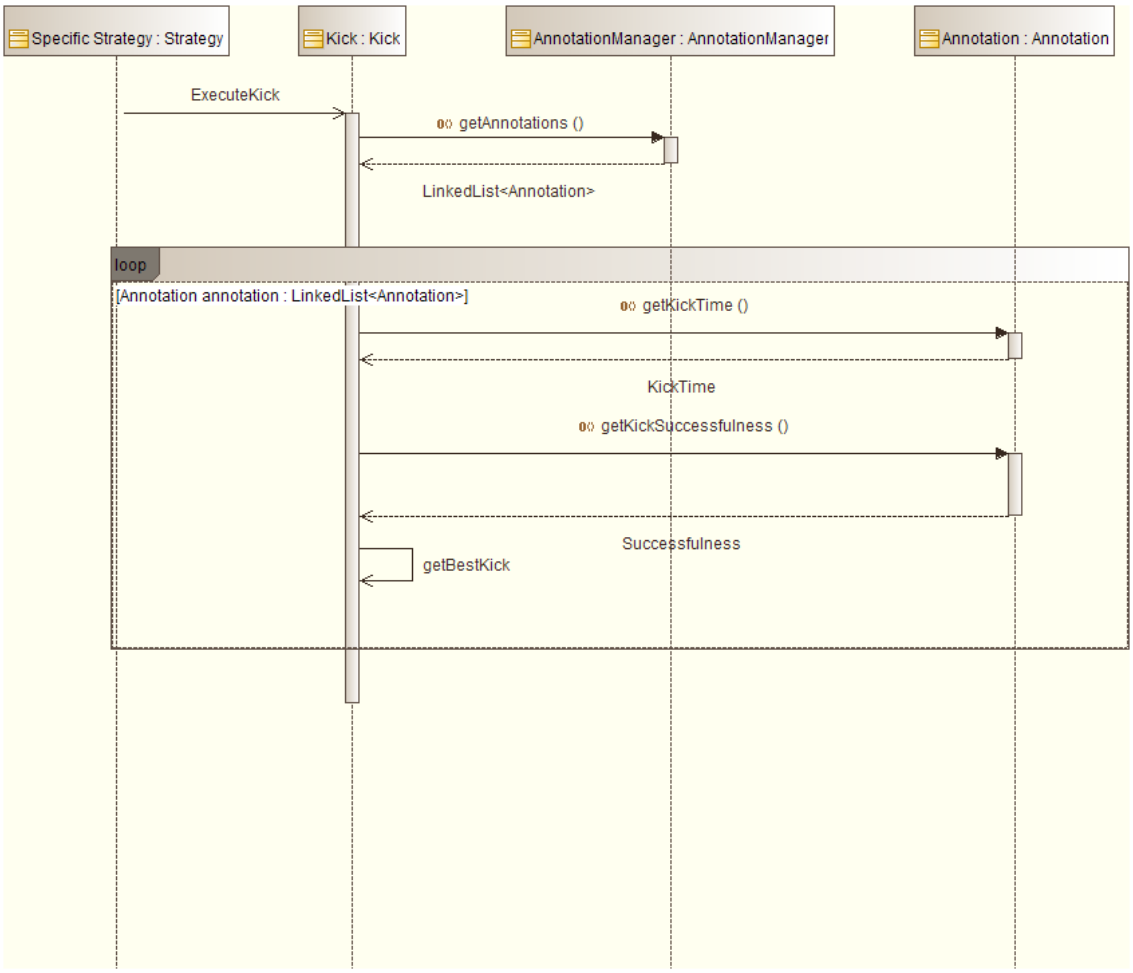
Obrázok 39 – Abort



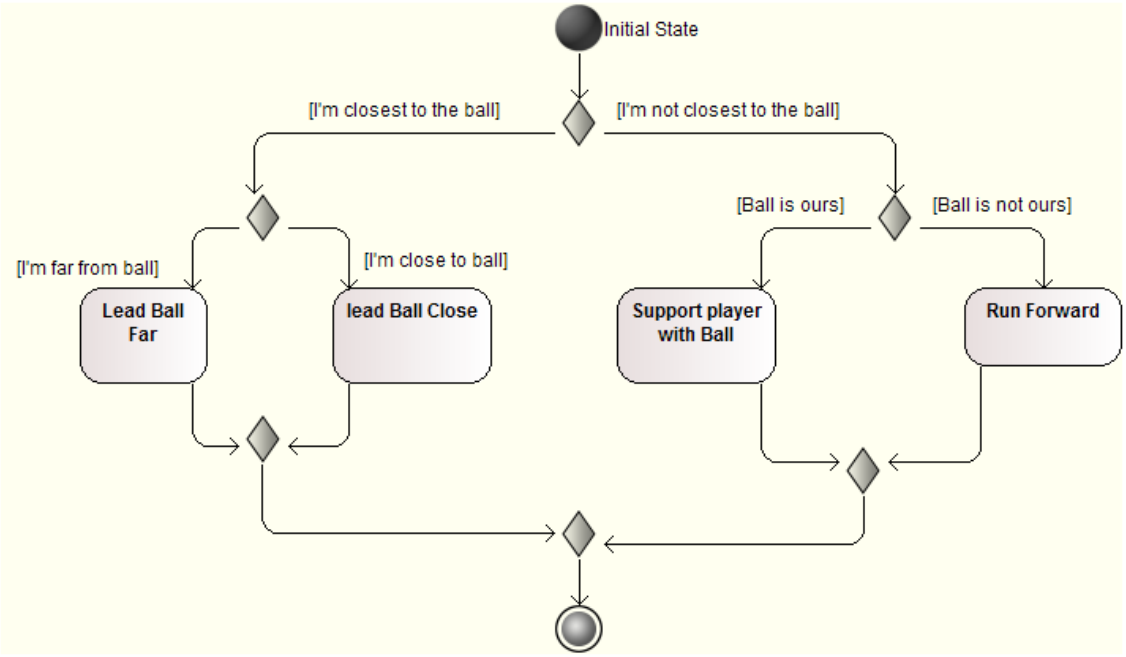
Obrázok 40 – Move



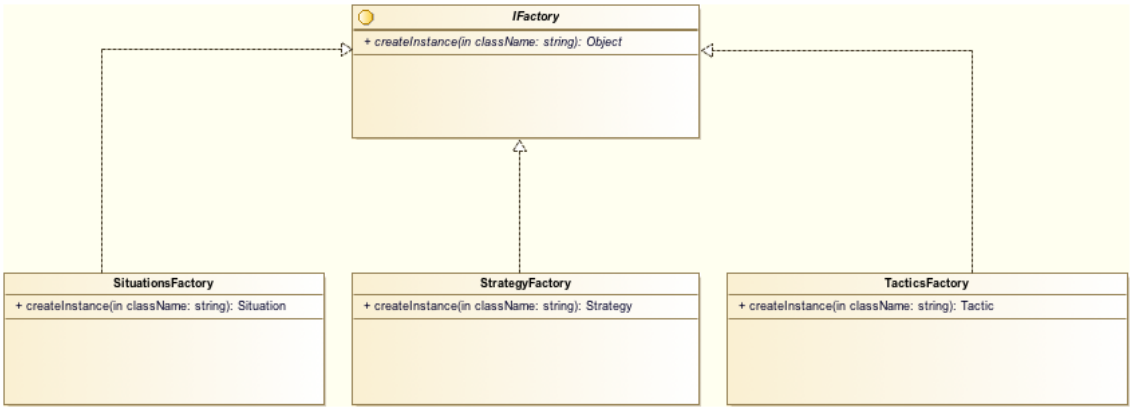
Obrázok 41 - UC check situations



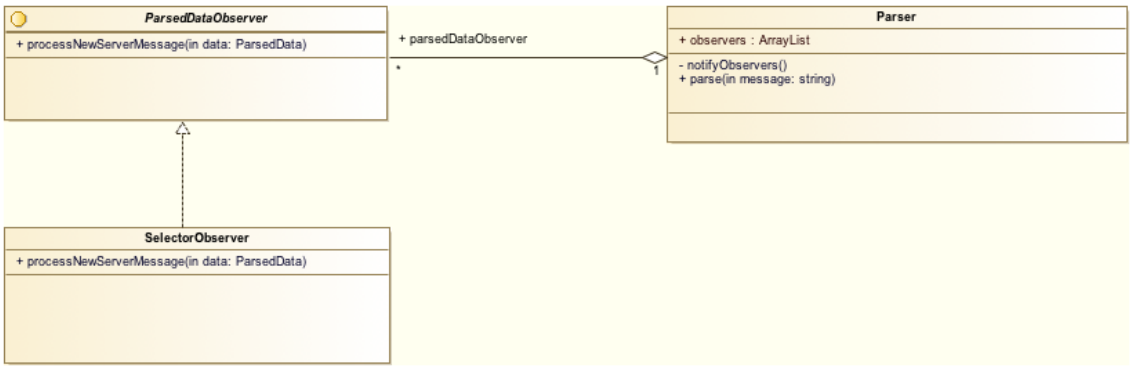
Obrázok 42 – Kick



Obrázok 43 - Attack Mid



Obrázok 44 - Factory Method



Obrázok 45 – Observer