Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Dokumentácia k inžinierskemu dielu

Tím sixPack (č.7)

Bc. Jozef Blažíček

Bc. Ján Ďurica

Bc. Jakub Chalachán

Bc. Matúš Ivanoc

Bc. Maryna Kovalenko

Bc. Miloš Štefčák

Vedúci projektu: Ing. Ivan Kapustík v z. Ing. Marián Lekavý, PhD.

Predmet: Tímový projekt I

Ročník: 2016/2017

Obsah

1. Úvod

Bc. Maryna Kovalenko

Pedagógovia a študenti Fakulty informatiky a informačných technológií Slovenskej technickej univerzity sa už od roku 2000 venujú simulovanému robotickému futbalu v rámci projektu RoboCup[1]. Na vývoji sa podieľajú tými študentov v rámci predmetu Tímový projekt či študenti pracujúci na svojich bakalárskych alebo diplomových prácach.

Cieľom je vytvoriť robotický futbalový tým schopný poraziť ľudských majstrov sveta vo futbale. V rámci súťaže bolo vytvorených päť líg:

- Liga Humanoidných (Humanoid)
- Stredná liga (Middle size)
- Simulačná liga (Simulation)
    - 2D
    - 3D
- Malá liga (Small size)
- Štandardná liga (Standard platform)

V rokoch 2000 - 2008 tento vývoj prebiehal v rámci 2D ligy. Od roku 2006 prebieha vývoj v rámci 3D simulačnej ligy. V rozšírenej lige sú jednotliví simulovaní "hráči" modelmi reálnych robotov, ktorí sú vybavení rôznymi senzormi a pohybujú sa ovládaním jednotlivých kĺbov, ktorými robot disponuje.

Na našej škole sa každoročne koná turnaj v simulovanom robotickom futbale RoboCup at FIIT. Účelom je porovnať jednotlivých hráčov a dozvedieť sa viac o inovatívnych myšlienkach a použitých algoritmoch. Hlavným cieľom projektu je pokračovanie na vývoji hráča simulovaného robotického futbalu, za účelom zdokonalenia už existujúcich schopností hráča a vytvorením nových.

Dokumentácia k inžinierskemu dielu poskytuje „Big picture“ vyvíjaného projektu. Okrem globálnych cieľov dokumentácia obsahuje aj podrobný popis technickej časti k práci, ktorú náš tím na tomto projekte vykonal, detaily realizácie doplnenej funkcionality a vylepšenia už existujúcej.

[1] http://www.robocup.org/

2. Celkový pohľad na projekt

Bc. Maryna Kovalenko

Výsledkom spoločnej práce študentov a pedagógov FIIT STU nad Robocup je vývoj hráča robotického futbalu JIMa, ktorý aktuálne beží na serveri rcssserver3d-0.6.7 (viď kapitola 4.1 Server). Server má informácie o aktuálnom stave hry, ktorý sa dá graficky zobraziť pomocou aplikácie rcsmonitor3d.

Projekt sa skladá z 3 častí:

1. Jim
2. RoboCupLibrary
3. TestFramework

Zdrojový kód je napísaný v Jave s využitím XML.

Súčasťou projektu je aj Wiki stránka. Na doplnení jej štruktúry sa zameral minuloročný tím, avšak wiki stále potrebuje aktualizáciu a doplnenie dôležitých informácií, ktoré sú pomoc pre tímové, bakalárske a diplomové projekty.

2.1. Schopnosti agenta

Agent dokáže chodiť, kopať do lopty, otočiť sa o 60 stupňov, určiť polohu spoluhráčov, súperov, lopty a seba. Vie detegovať vlastný pád, postaviť sa maximálne za 3s., stabilizovať sa pri vstávaní a v prípade, že nemá naplánovaný žiaden LowSkill.

Agent má implementované základné vnímanie taktiky a to, či jeho tím útočí alebo bráni. Pri vedení lopty vie nielen udržať smer na súperovu bránu, ale aj vystreliť na ňu.

2.2. JIM

Java aplikácia agenta, ktorý sa pripojí k serveru, používa vybratú taktiku a hrá futbal. Spusteniu agenta predchádza spustenie RCSS servera. Agent má jednoduché GUI umožňujúce opätovné načítanie pohybov a preplánovanie. Vstupným bodom agenta je trieda sk.fiit.jim.init.Main. Na začiatku sa vykonajú nasledovné operácie:

- Načítavanie pohybov z XML súborov v adresári moves
- Načítanie anotácií
- Spracovanie parametrov príkazového riadku
- Spustenie TFTP servera použitého pre komunikáciu s TestFrameworkom
- Prípadné vytvorenie GUI, keďže ešte nefunguje a dané časti kódu sme zakomentovali
- Vstup do hlavného cyklu

2.2.1. Pohyby

Správanie agenta je riadené tzv. High Skills a Low Skills. Plánovač spravovaný triedou HighSkillPlanner vytvára inštancie high skillov (vyšší pohyb), ktoré počas svojho behu vyberajú, ktorý low skill (nižší pohyb) sa má vykonať. Low skill predstavuje len skupinu metadát, zapísaných pomocou XML, ktoré určujú jeho názov, ďalšie podobné informácie a počiatočnú fázu. Práve v týchto fázach sa nachádza jadro pohybu. Fázy nie sú pevne spojené s low skillmi, ale väčšina fáz patrí k jednému konkrétnemu Low Skillu, čo je vyjadrené ich menom.

Na načítanie pohybov z XML súborov slúži trieda SkillsFromXmlLoader. Objekty pohybov a fáz sú spravované triedami LowSkills a Phases (balík sk.fiit.jim.agent.moves). Keďže načítanie pohybov je značne pomalé, ich parsovaná podoba sa ukladá do súboru ./movecache, ktorý sa pri budúcom načítaní použije, pokiaľ od jeho vytvorenia nebol žiadny pohyb zmenený. V takom prípade sa súbor vytvorí nanovo.

2.3. RoboCupLibrary

Knižnica ktorá obsahuje triedy reprezentujúce rožné matematické výpočty, anotácie a geometrické objekty. Obsahuje triedy spoločné pre agenta a Test Framework. Malo sa používa.

Tím Infinity (akad. rok 2014/2015) vykonal kompletný refaktoring RobocupLibrary, Aktuálna dokumentácia k RobocupLibrary nie je ani na stránkach minuloročných tímov, ani na Wiki. Cieľom je zachovať, prípadne aj rozšíriť túto knižnicu, ktorá by mala odstrániť závislosť medzi Jim-om a TestFrameworkom, čo aktuálne nie je dodržané.

2.4. TestFramework

Slúži na získanie spätnej väzby od hráča. Refaktoring balíku ui a backendu spravil tím TeamBender. Hlavným zámerom je zostrojiť robotického futbalového trénera, ktorý by dokázal učiť hráčov novým taktikám a pohybom automaticky.

Interakcia medzi agentom a serverom prebieha prostredníctvom posielania správ, obsahom ktorých sú tzv. S-výrazy, pričom komunikácia je jednosmerná: agent len odosiela správy a TestFramework ich len prijíma. Server (TestFramework) je implementovaný triedou AgentMonitor v balíku sk.fiit.testframework.monitor, ktorá sa stará o prijímanie nových spojení od agentov. Každé jedno takéto spojenie je reprezentované triedou AgentMonitorThread v balíku sk.fiit.testframework.monitor, ktorá má na starosti spracovanie správ prichádzajúcich od agenta pomocou triedy AgentMonitorMessage v tom istom balíku. Na komunikáciu s Testframework-om slúži v agentovi trieda TestFrameworkCommunication v balíku sk.fiit.jim.agent.communication.testframework.

2.5. Identifikované nedostatky

Na základe dokumentácií predchádzajúcich tímov a testovania projektu boli nájdené nasledujúce nedostatky, odstránením ktorých sa budeme venovať :

- Hráč má problém s vyhodnocovaním svojej polohy na ihrisku, keď nevidí kontrolné body.
- WIKI potrebuje úpravy neaktuálnych stránok, najmä z čias, keď sa na RoboCupe používalo Ruby.
- GUI u JIMA nefunguje: zbytočne dve okná, ktoré by sa dali spojiť do jedného. Obsahuje časti, ktoré nič nerobia.
- Prepínanie medzi testovacími taktikami.
- Hráč stojaci na mieste začne samovoľne padať.
- Nefungujú anotácie v TestFramework-u.
- Nefunguje prepínanie testovacích scenárov v TestFramework-u.
- Meranie disciplín turnaja v TestFramework-u neobsahuje všetky disciplíny turnaja.
- Chôdza pomocou ZMP (Zero Moment Point) nie je implementovaná v žiadnej z taktík.
- Informácie o čiarach nie sú použité.
- Použité trigonometrické funkcie (sin a cos) patria medzi najpomalšie matematické operácie.

- Komunikácie Jima s TestFrameworkom cez pomalý TFTP protokol.
- Koordinácia a implementácia bp a dp - Každoročne niekoľko študentov pracuje na tejto téme vo svojich bakalárskych a diplomových prácach. Neexistuje efektívny mechanizmus koordinácie a aplikovania zmien.
- Brankár - V projekte sa zatiaľ nenachádza brankár, aj keď sa na jeho vylepšovaní stále pracuje v prácach študentov.

Pri vypracovaní celkového pohľadu na systém boli použité:

1. Dokumentácia tímu TeamBender (akad. rok 2015/2016).
2. Dokumentácia tímu Infinity (akad. rok 2014/2015).
3. Robocup Wiki.

3. Ciele

Bc. Jozef Blažíček

Na začiatku semestra nás vedúci projektu Ing. Ivan Kapustík a členovia minuloročného tímu oboznámili s nedostatkami projektu a možnými vylepšeniami.

3.1. Ciele pre zimný semester

1. **Vylepšenie vyhodnocovania polohy hráča použitím čiar**
   Agent dokáže rozpoznávať čiary na ihrisku, zatiaľ nie je implementovaný mechanizmus na ich rozpoznávanie (agent dostáva informácie, ale nepoužíva ich) a využitie. Medzi potenciálne možnosti využitia informácií o čiarach patrí vylepšenie lokalizácie hráča na ihrisku.

2. **Analýza a spracovávanie informácií o čiarach**
   Analyzovanie, aké informácie dostáva agent zo serveru so zameraním na informácie o čiarach. (viď kapitola 5.2 Analýza zahraničných tímov a kapitola 5.4 Správy zo servera)

3. **Zobrazenie čiar v TestFrameworku**
   TestFramework okrem iného zobrazuje aktuálny stav hry, ako je na serveri a ako ho vnímajú agenti. Nakoľko čiary sú pevne dané, je potrebné doimplementovať vnímanie čiar z pohľadu agenta (viď kapitola 5.5. Vykreslenie čiar do TestFramework-u).

4. **Aktualizácia wiki**
   Súčasťou projektu je wiki stránka[2], z ktorej čerpajú informácie študenti pri riešení bakalárskych a diplomových projektov, ako aj pre tímy pokračujúce v rámci tímového projektu (viď kapitola 6. wiki).

[2] http://labss2.fiit.stuba.sk/TeamProject/2016/team07is-si/wiki/index.php/Hlavn%C3%A1_str%C3%A1nka

4. Používané technológie

Bc. Jozef Blažíček, Bc. Ján Ďurica, Bc. Matúš Ivanoc

4.1. Server

Bc. Jozef Blažíček

Agent posiela serveru informácie o zmene nastavenia jednotlivých kĺboch (viď kapitola 5.3 Komunikácia so serverom) a zo serveru prijíma informácie o aktuálnom stave hry (viď kapitolu 5.4 Správy zo srvera).

Minuloročný tým (tím Bender) používal rcsserver3d vo verzii 0.6.7 avšak už niekoľko rokov sú dostupné zdrojové kódy k niekoľkým novým verziám serverov, táto podkapitola zahŕňa porovnanie jednotlivých verzií.

Po niekoľkých hodinách pokusov o spojazdnenie najnovšej verzie servera (0.6.10) sa ho stále nepodarilo úspešne spustiť, preto zatiaľ pokračujeme prácu so staršou verziou (0.6.7).

Porovnanie serverov vychádza z „realise note“ k danej verzii serveru. V každej verzii serverov boli odstránené bugy. Pri každej verzii sú spomenuté hlavné zmeny.

4.1.1. rcsserver3d-0.6.8 a rcsserver3d-0.6.8.1

- Označovanie tímov / správ (identifikácia, z ktorého tímu prichádza správa)
- Score reporting (eg. GS(unum 8)(team left)(sl 1)(sr 2)(t0.0))
- max. 11 typov robotov

4.1.2. rcsserver3d-0.6.9

- Nové pravidlo – lopta sa musí dotýkať protihráča alebo spoluhráča mimo stredového kruhu
- pred tím ako môže tím skórovať
- Penaltové výkopy sú priame
- Pridanie šumu (asi nejaký nový výpočet)

4.1.3. rcsserver3d-0.6.10

- Pridané modeli pre vizuálne rozlišovanie rôznych typov robotov

4.2. Používané pluginy

Bc. Matúš Ivanoc

V rámci vývojového procesu je vhodné použiť niektoré pluginy do nástroja Eclipse, aby sa zlepšila buď kvalita kódu, alebo uľahčila práca v našom tíme počas vývoja. Uvedené sú len aktuálne použiteľné pluginy. Medzi nimi sa nachádzajú také, ktoré sa zaoberajú vyhľadávaním nedostatkov v kóde a tiež nástroj na udržiavanie programátorských štandardov.

- Checkstyle
- Eclipse PMD
- FindBugs

4.2.1. Checkstyle

Plugin, ktorý dohliada na dodržiavanie štandardov v kóde a vo výsledku zlepšuje čitateľnosť kódu. V tímovom prostredí, kde každý má svoju škálu zvykov ako písať kód, je dobré mať 1 štandard, ktorý sa dodržiava.

4.2.2. Eclipse PMD

Eclipse PMD plugin integruje známy analyzátor zdrojového kódu PMD do vývojového prostredia Eclipse. PMD slúži na vyhľadávanie zlých vzorov v kóde.

Pri každom uložení zdrojového kódu, Eclipse PMD vykoná skenovanie kódu a vyhľadá potenciálne problémy ako bugy, duplicitné, neoptimálne alebo zbytočne komplikované časti kódu.

Kde je to možné, plugin poskytuje možnosť automatickej opravy. Tento plugin tak pomáha k udržiavaniu kvality kódu aj v prípade, že na projekte pracujú viacerí programátori.

4.2.3. FindBugs

Je to plugin na detekciu chýb v jazyku Java, ktorý využíva statickú analýzu približne 200 vzorov, ktoré poukazujú na zlý kód. Jedná sa napríklad o nekonečné rekurzívy, zlé využitie Java knižníc alebo uviaznutia v kóde. Je vhodný na rozsiahle projekty, kde sa predpokladá 1 porucha na zhruba 1000 – 2000 riadkov kódu. Skenovanie v kóde možno spúšťať manuálne alebo v inkrementálnych krokoch.

4.3. Enterprise Architect 12.1

Bc. Jozef Blažíček

Na vytvorenie diagramov použitých v dokumentácii bol použitý Enterprise Architect 12.1 od Sparx Systems s licenciou, ktorou disponuje FIIT a ktorá bola na tieto účely poskytnutá na obdobie oboch semestrov v akademickom roku 2016/2017.

4.4. SourceTree

Bc. Ján Ďurica

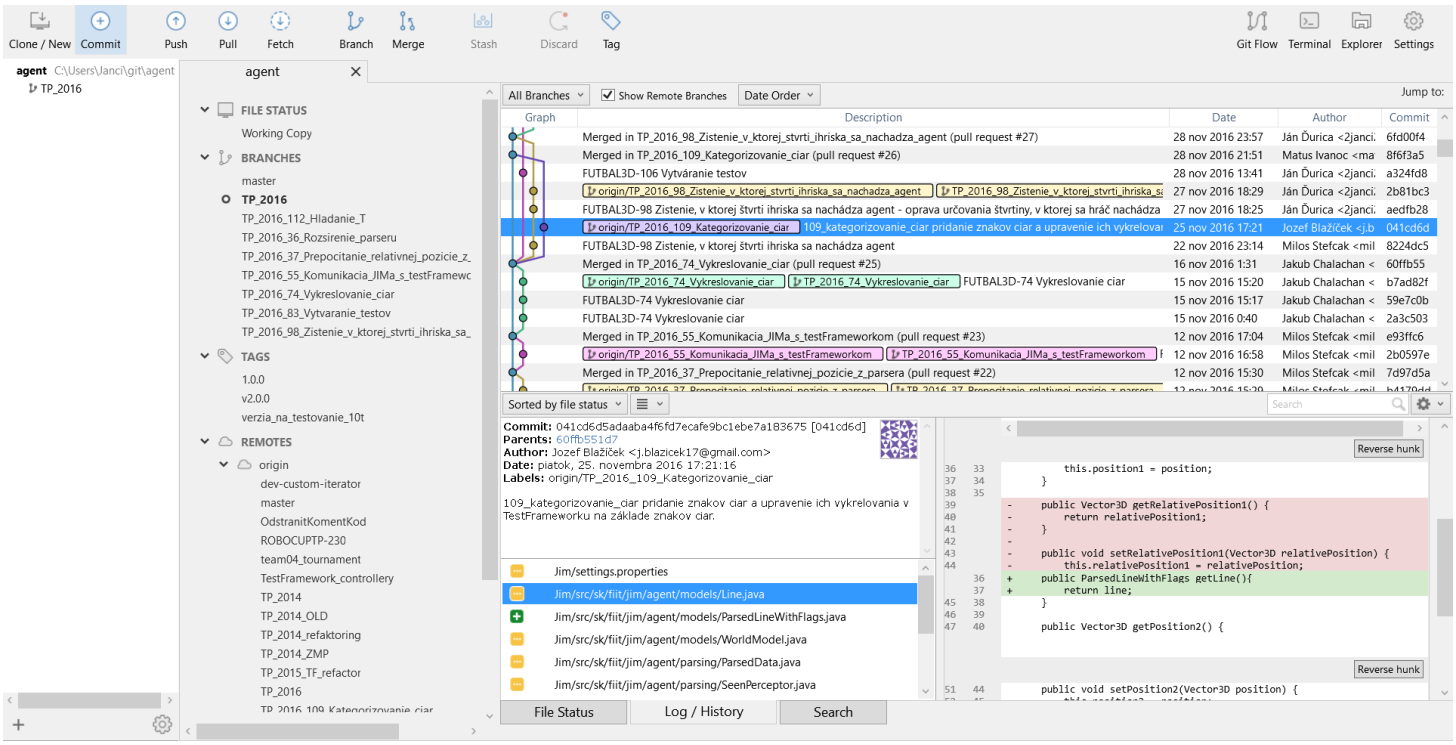
SourceTree je voľne dostupný nástroj a zároveň klient pre systém na správu verzií Git. Keďže náš tím používa tento systém, ako hlavný projektový manažér a človek so skúsenosťami s používaním tohto nástroja, nakoľko je to veľmi užitočný nástroj, hlavne pre ľudí, čo pred tým s Gitom veľa nepracovali. SourceTree ponúka jednoduché grafické rozhranie, ktoré používateľovi poskytuje prehľad o aktuálnej vetve, commitoch a mnohých ďalších užitočných vecí, o ktoré by bol používateľ pri používaní príkazov v príkazovom riadku ukrátený.

4.4.1. Správa vetiev

Spravovanie vetiev v SourceTree je jednoduché. Novú vetvu si môžeme vytvoriť z ľubovoľného commitu kliknutím pravého tlačidla myši zvolením možnosti „Branch...“. Taktiež prepnutie vetiev prebehne po dvojkliku ľavým tlačidlom na vybranú vetvu. Nástroj rozlišuje lokálne a vzdialené vetvy, takže používateľ má stále prehľad o všetkých vetvách.

4.4.2. Prehľad o vykonaných zmenách

Nástroj zobrazuje zmeny a rozdiely medzi aktuálnou verziou súborov u používateľa a poslednou aktuálnou verziou, ktorá je nahraná na repozitári. Taktiež si môže u jednotlivých commitoch okamžite zobraziť, ktoré súbory boli zmenené/pridané/odstránené, a samozrejme, môže si pozrieť konkrétne zmeny. Ako je vidieť na obrázku nižšie, v zozname commitov je modrým vyznačený zvolený commit, o ktorom hneď vieme základné informácie ako v ktorej vetve bol vytvorený, kedy, kým. V ľavej dolnej časti je vidieť zoznam zmenených súborov (žltý štvorček) a pridaného súboru (biele + v zelenom štvorčeku). Po vybraní ľubovoľného súboru sú v pravej časti zobrazené zmeny súboru, a teda odstránené a pridané riadky. SourceTree totižto zobrazuje zmeny po riadkoch, čiže zmena jedného slova na riadku sa prejaví ako jedno odstránenie a jedno pridanie riadku. Červenou farbou so znamienkom - na ľavej strane značí, že riadok bol odstránený, vedľa neho vidíme aj jeho číslo. Zelenou farbou sú, naopak, vyznačené riadky, ktoré boli pridané.



Obrázok 1: Ukážka nástroja SourceTree

4.4.3. Všetko na jednom mieste

Ako som už spomínal vyššie, všetky potrebné veci, ktoré potrebujeme pre správu verzií, sú v nástroji SourceTree dostupné na hlavnej obrazovke. Ide teda o:

- commitovanie,
- pushovanie,
- pullovanie,
- vytvorenie vetvy,
- vytvorenie stashu,
- zobrazenie vetiev
    - lokálnych,
    - na serveri,
- zobrazenie tagov,
- zobrazenie všetkých informácií o commite.

4.4.4. Dostupnosť

Nástroj je voľne dostupný a je možné si ho nainštalovať na operačných systémoch Windows aj Mac OS X. Je vytvorený spoločnosťou Atlassian, od ktorej používame aj iné produkty ako JIRA, HipChat či BitBucket, čo prináša výhodu ich vzájomnej spolupráce.

5.Čiary

Bc. Jozef Blažíček, Bc. Maryna Kovalenko

Cieľom s najvyššou prioritou je vylepšenie lokalizácie hráča pomocou rozpoznávania čiar. Pre dosiahnutie tohto cieľa je potrebné vykonať niekoľko krokov. Táto kapitola dokumentuje kroky, ktoré vykonal náš tým pri dosahovaní tohto cieľa.

1. **Analyzovanie zahraničných tímov** (viď kapitola 5.2. Analýza zahraničných tímov)
   Zistenie ako a či využívajú nejaké tímy informácie o čiarach pri lokalizácií hráča.
2. **Analyzovanie komunikácie agenta so serverom** (viď kapitola 5.3. Komunikácia so serverom)
   Ako prebieha komunikácia agenta so serverom, kde prijíma, parsuje a spracováva údaje.
3. **Analyzovanie správ zo servera** (viď kapitola 5.4. Správy zo servera)
   Aké informácie dostáva agent zo servera.
4. **Rozšírenie parseru o získavanie údajov o čiarach** (viď kapitola 5.5. Parsovanie údajov)
5. **Vykreslenie čiar do TestFramework-u** (viď kapitola 5.6. Vykresľovanie čiar do TestFramework-u)
   Vykreslenie čiar z pohľadu agenta do Test-Frameworku pre analýzu, ako sú čiary reprezentované na serveri a možnosti uplatnenia informácií o nich.
6. **Kategorizovanie čiar**
   Určenie či koncové body čiary, ktoré vidí agent, sú koncovými bodmi čiary.
7. **Analyzovanie histórie polohy agenta**
   V projekte je implementovaná história polohy agenta, je potrebné analyzovať ako sa to používa.
8. **Určenie časti ihriska, kde sa agent nachádza / nachádzal**
9. **Rozlišovanie priesečníkov**

Ďalšie kroky ešte neboli presne definované, nakoľko sme v stave plnenia vyššie uvedených a na základe ich výsledkov sa stanoví ďalší postup.

5.1. Lokalizácia agenta v prostredí RoboCup

Bc. Maryna Kovalenko

V prostredí RoboCup sa predpokladajú nasledovné charakteristiky, ktoré musia byť braté do úvahy pre zvolenie vhodnej metódy lokalizácie:

1. Geometria stien ohraničujúcich pole a čiar nakreslených na ihrisku sú známe,
2. Prostredie je dynamické (na ihrisku sa pohybujú hráči aj lopta),
3. Úloha musí byť vykonaná bez prerušenia pre dlhšiu dobu,
4. Prostredie nemôže byť modifikované,
5. Agenti sa môžu naraziť jeden na druhého.

Všetky tieto faktory sú dôvodom zložitého scenáru pre spôsoby lokalizácie.

Pri určení polohy je potrebne brať do úvahy vysoký šum pri získavaní informácií z prostredia vzhľadom k meniacim sa podmienkam v priebehu získavania údajov.

5.2. Analýza zahraničných tímov

Bc. Maryna Kovalenko

5.2.1. Tím: NeverMost (Čína)

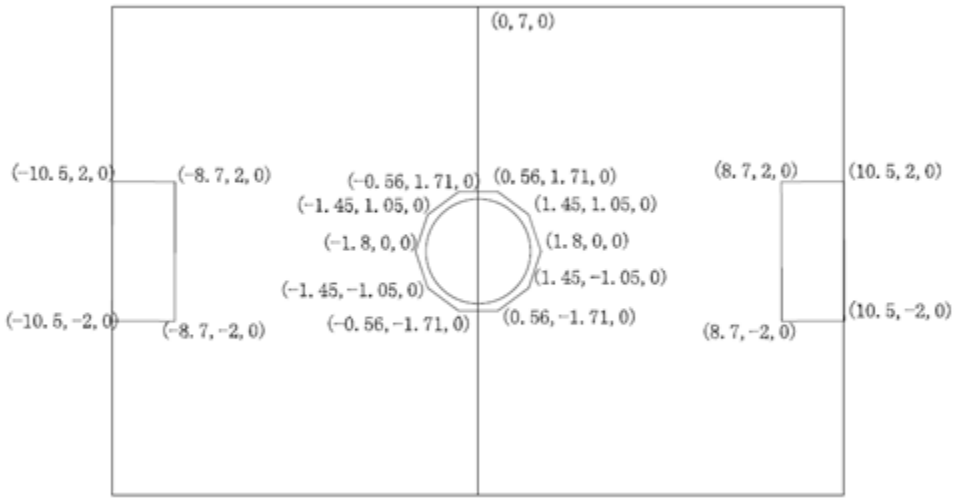
V novej verzii rcssserver3D sú pridané informácie o čiarach. Agent môže vidieť rad bodov vrátane koncové body čiar a priesečníky spôsobené vizuálnym limitom. Ak koncové body čiar môžu byť získané, vzniká rovnaká situácia ako keby existovalo viac fixných vlajok na ihrisku, takým spôsobom sa znižuje obtiažnosť výpočtu lokalizácií agenta.

Nasledujú kroky algoritmu extrakcie koncových bodov čiar:

- Lokalizácia koncových bodov čiar pomocou tradičnej metódy lokalizácie bez uvažovania šumu a vizuálneho limitu. Výsledok tohto kroku je znázornený na Obrázku 1.
- Použitie vzdialenosti medzi koncovými bodmi a vlajkami (F1L, F1R, atď.) pre extrahovanie koncových bodov čiar. Je možne namapovať koncové body do pozície bodov na Obrázku 1.

Výhody použitého algoritmu:

- Je nezávislý od výsledku metódy tradičnej lokalizácie
- Získavame viac bodov z globálnej vízie
- Získavame viac globálnych vlajok



Obrázok 2: Ihrisko, ktoré používa tím NeverMost (staršia verzia)

Výsledok ukazuje, že použitie informácií o čiarach viditeľne vylepší lokalizáciu agenta.

Smer rozvoja tímu: Používanie priesečníku medzi víziou oblasti a čiarami na ihrisku pri lokalizácii.

5.2.2. Tím Karachi Koalas 3D Simulation Soccer Team (Nemecko)

Pri lokalizácii používajú nie len informácie o čiarach, ale aj informácie o orientačných bodoch. Pre zabezpečenie fungovania sa vyžaduje aspoň jeden orientačný bod a čiara. Aby bolo možné identifikovať správnu čiaru medzi ostatnými, karteziánsky systém svetového modelu sa prevedie na karteziánsky systém agenta. Dĺžka čiary sa používa pre zistenie polohy koncových bodov v systéme svetového modelu, následne sa karteziánsky systém prepne. Poloha agenta sa vypočíta s použitím

koncových bodov čiary a orientačného bodu. Pre vylepšenie presnosti výpočtov a zníženia zašumenia používajú Kalman filter, ktorý umožni získať lepši odhad ballPolar poskytnutý serverom.

Mechanizmus preposielania správ bol tiež použitý na získanie informácií o prostredí. Hráč, ktorý vidí loptu a je tiež presvedčený o svojej pozícii na ihrisku, pošle informáciu o pozícii lopty ostatným členom tímu, ktorí nevidia loptu, ale budú vedieť sa rozhodovať na základe prijatej správy. Napríklad agent, ktorý zašiel príliš ďaleko a nevidí loptu priamo dostane správu od brankára, že lopta je za ním, teda nevidiaci agent môže použiť spätnú chôdzu, aby sa dostal bližšie k lopte.

5.2.3. Výsledky analýzy prác zahraničných tímov

Lokalizácia na základe čiar je stále v experimentálnom stave, aj keď niektoré výsledky sú veľmi optimistické. V túto chvíľu sa pokusy uskutočnili len v simulátore. Táto metóda je vždy schopná nájsť pozíciu agenta, ale výpočet trvá významnejšie dlhšie, než v prípade klasickej lokalizácie. Neriešeným problémom stále zostava zrkadlová symetria ihriska.

5.2.4. Odkazy

The NeverMost 3D Soccer Simulation Team Desctiption 2012
http://chaosscripting.net/files/competitions/RoboCup/WorldCup/2012/3DSim/tdps/NeverMost_TDP.pdf

Team Description Paper for World RoboCup 2014
http://fei.edu.br/rcs/2014/TeamDescriptionPapers/SoccerSimulation/Soccer3D/karachikoalas_TDP.pdf

Utilizing the Structure of Field Lines for Efficient Soccer Robot Localization
https://www.ais.uni-bonn.de/papers/advrob2012_schulz_behnke.pdf

Line Structure-based Localization for Soccer Robots
https://www.ais.uni-bonn.de/papers/HSR09_Schulz_Localization.pdf

Cooperative Object Localization Using Line-Based Percept Communication
http://dl.acm.org/citation.cfm?id=1423557

5.3. Komunikácia so serverom

Bc. Jozef Blažíček

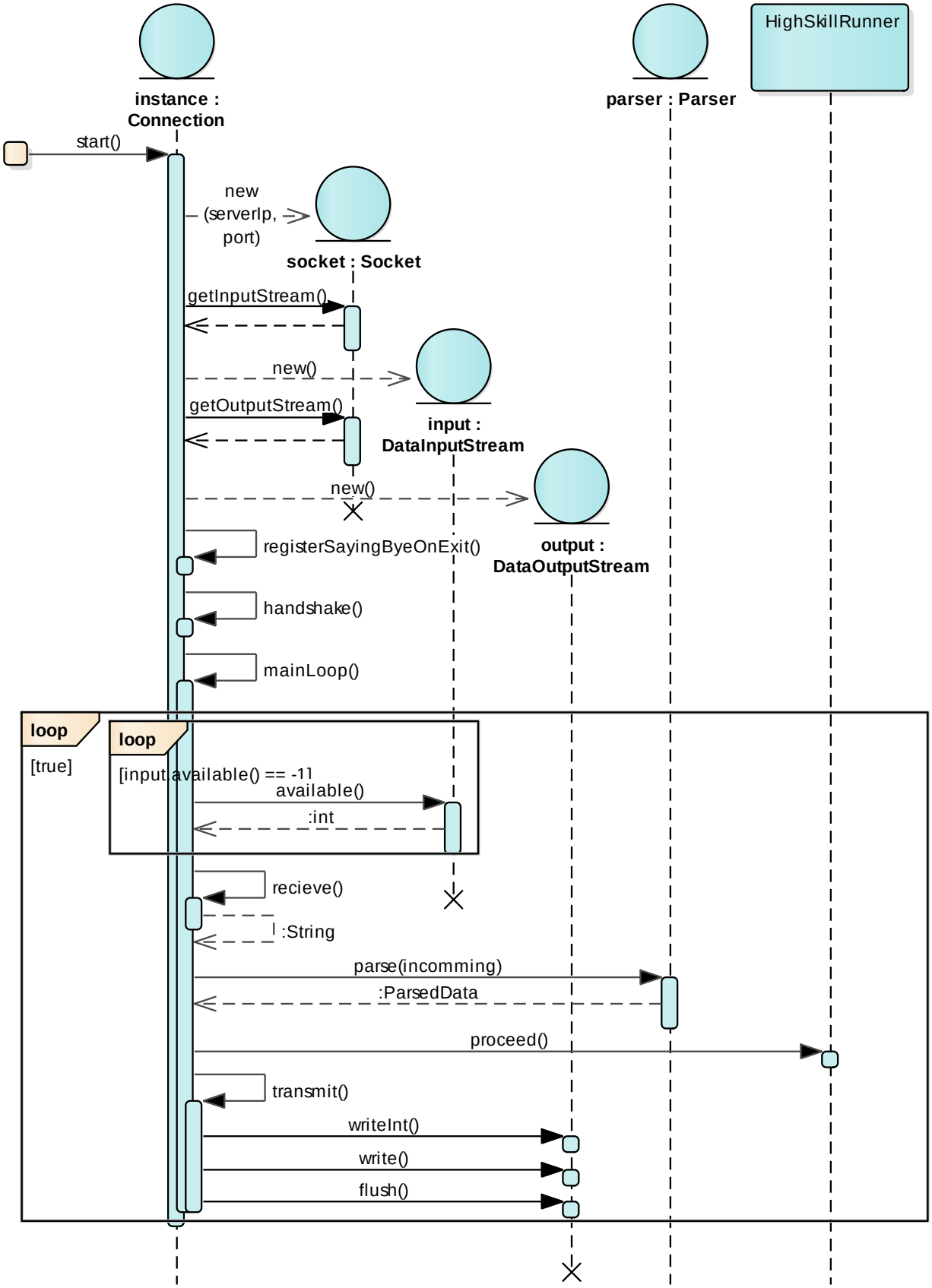
Celá hra prebieha ako interakcia medzi agentom a serverom (viď kapitola 4.1 Server). Server má informácie o aktuálnom stave hry, tj. Rozloženie hráčov, čas, poloha lopty, ... Tento stav sa dá graficky zobraziť pomocou aplikácie rcssmonitor3d.

Server poskytuje minimálne dva typy robotov – „Nao“ a „Socerbot“. Nao predstavuje virtuálnu reprezentáciu rovnomenného robota a je používaný aj v projekte Robocup na FIIT.

Nao sa pohybuje pomocou dvadsiatich dvoch otočných kĺbov.

Agent so serverom komunikuje prostredníctvom TFTP (angl. trivial file transfer protocol) protokolu nad TCP (protokol riadenia prenosu). Obsahom posielaných správ sú tzv. S-výrazy.

S-výraz je formátovaný znakový reťazec (čitateľný človekom aj strojom). Začína („(“) a končí („)“) znakom obyčajnej zátvorky. Ako prvý zankový reťazec obsahuje identifikátor typu z preddefinovanej množiny ([time,see,pol,player,...] ...). S-výraz môže v sebe obsahovať ďalšie s-výrazy (ak to definícia daného výrazu povoľuje, napr. „(L (pol ...)(pol ...)“).



Obrázok 3: UML diagram sekvencií - Priebeh komunikácie agenta so serverom

Agent zo servera dostáva rôzne informácie (viď kapitola 5.4. Správy zo servera) a odosiela informácie o tom, ako by sa malo zmeniť aktuálne nastavenie kĺbov. Tieto informácie sú výsledkom

výpočtov, ktoré berú do úvahy polohu agenta, aktuálnu stratégiu, low & high skills, čas a.i.. Tieto výpočty prebiehajú v niekoľkých triedach.

Agent je pripojený na server cez port 3100. Komunikácia agenta so serverom prebieha v triede *sk.fiit.jim.agent.communication.Communication*, trieda implementuje návrhový vzor singleton.

Komunikácia so serverom začína zavolaním funkcie start(), kde sa na začiatku vytvoria objekty potrebné na komunikáciu. V metóde *registerSayngByeOnExit()* sa nastaví ukončenie spojenia pri skončení vykonávania programu či už pri normálnom ukončení (dosiahnutím koncového bodu vykonávania), alebo prerušením vykonávania. O ukončenie spojenia sa postará JRE. V metóde *handshake()* sa následne inicializuje spojenie so serverom, kde na konci metódy agent odošle inicializačný s-výraz so svojim číslom a názvom tímu **(init (num <cislo>) (teamName <názov_tímu>))**.

V metóde *mainLoop()* prebieha nekonečný cyklus. Na začiatku cyklu sa čaká, pokým nepríde nejaká správa zo servera. Po prijatí správy metódou *recieve()* sa prijaté údaje spracujú v parseri (viď kapitola 5.4 Parsovanie údajov).

Prvým volaním metódy *proceed()* triedy *sk.fiit.jim.agenthighskill.runner.HighSkillRunner* sa vytvorí nové vlákno, na ktorom bude prebiehať plánovanie akcií. Plánovanie akcií je realizované pomocou triedy sk.fiit.jim.agent.highskill.runner.HighSkillPlanner.

Na konci nekonečného cyklu sa odošle správa s akciami na server.

5.4. Správy zo servera

Bc. Jozef Blažíček

Typ S-výrazu	**Význam**
time	Čas
GS	Stav hry (game state)
GYR	Gyroskop (gyro rate)
ACC	Akcelerometer (accelerometer)
HJ	Otočný kĺb (hinge joint)
See	Čo agent vidí
FRP	(Force Resistance)
AgentState	Stav agenta (teplota a batéria)
Hear[3]	Čo agent počuje

Tabuľka 1: S-výrazy, ktoré sa môžu vyskytnúť v správe zo servera (prvých 6 sa nachádza aj v odchytených správach, zvyšné pochádzajú zo simspark wiki[4])

```
(time (now 90.00))(GS (t 0.00) (pm BeforeKickOff))(GYR (n torso) (rt 0.00 0.00
0.00))(ACC (n torso) (a 0.00 0.00 9.81))(HJ (n hj1) (ax 0.00))(HJ (n hj2) (ax -
0.00))(HJ (n raj1) (ax -0.00))(HJ (n raj2) (ax -0.00))(HJ (n raj3) (ax 0.00))(HJ
(n raj4) (ax -0.00))(HJ (n laj1) (ax -0.00))(HJ (n laj2) (ax -0.00))(HJ (n laj3)
(ax 0.00))(HJ (n laj4) (ax -0.00))(HJ (n rlj1) (ax -0.00))(HJ (n rlj2) (ax 0.00))(HJ
(n rlj3) (ax 0.00))(HJ (n rlj4) (ax 0.00))(HJ (n rlj5) (ax 0.00))(HJ (n rlj6) (ax
0.00))(HJ (n llj1) (ax -0.00))(HJ (n llj2) (ax -0.00))(HJ (n llj3) (ax 0.00))(HJ
(n llj4) (ax 0.00))(HJ (n llj5) (ax 0.00))(HJ (n llj6) (ax -0.00))

(time (now 90.02))(GS (t 0.00) (pm BeforeKickOff))(GYR (n torso) (rt 0.00 0.00
0.00))(ACC (n torso) (a 0.00 0.00 9.81))(HJ (n hj1) (ax 0.00))(HJ (n hj2) (ax -
0.00))(HJ (n raj1) (ax -0.00))(HJ (n raj2) (ax -0.00))(HJ (n raj3) (ax 0.00))(HJ
(n raj4) (ax -0.00))(HJ (n laj1) (ax -0.00))(HJ (n laj2) (ax -0.00))(HJ (n laj3)
(ax 0.00))(HJ (n laj4) (ax -0.00))(HJ (n rlj1) (ax -0.00))(HJ (n rlj2) (ax 0.00))(HJ
```

[3] Agenti môžu medzi sebou komunikovať iba prostredníctvom servera (procesy agentov nesmú komunikovať medzi sebou priamo).

[4] http://simspark.sourceforge.net/wiki/index.php/Perceptors

(n rlj3) (ax 0.00))(HJ (n rlj4) (ax 0.00))(HJ (n rlj5) (ax 0.00))(HJ (n rlj6) (ax 0.00))(HJ (n llj1) (ax -0.00))(HJ (n llj2) (ax -0.00))(HJ (n llj3) (ax 0.00))(HJ (n llj4) (ax 0.00))(HJ (n llj5) (ax 0.00))(HJ (n llj6) (ax -0.00))

(time (now 90.04))(GS (t 0.00) (pm BeforeKickOff))(GYR (n torso) (rt 0.00 0.00 0.00))(ACC (n torso) (a 0.00 0.00 9.81))(HJ (n hj1) (ax 0.00))(HJ (n hj2) (ax -0.00))**(See (G2R (pol 20.66 -22.62 0.51)) (G1R (pol 19.87 -17.46 0.64)) (F1R (pol 19.28 8.92 -1.40)) (F2R (pol 25.51 -41.64 -1.19)) (P (team Infinity) (id 1) (rlowerarm (pol 0.19 -35.37 -21.18)) (llowerarm (pol 0.19 33.23 -21.35))) (L (pol 8.03 -60.05 -4.12) (pol 5.03 36.89 -6.41)) (L (pol 25.47 -41.58 -1.26) (pol 19.25 8.67 -1.58)) (L (pol 19.26 9.08 -1.50) (pol 3.50 59.91 -9.03)) (L (pol 25.51 -41.60 -1.20) (pol 19.66 -60.12 -1.58)) (L (pol 17.65 -13.18 -1.79) (pol 19.89 -29.93 -1.62)) (L (pol 17.65 -13.23 -2.04) (pol 19.42 -11.73 -1.48)) (L (pol 19.91 -29.99 -1.38) (pol 21.50 -27.82 -1.46)) (L (pol 9.24 -49.27 -3.51) (pol 8.11 -46.06 -3.89)) (L (pol 8.11 -46.11 -4.03) (pol 6.90 -48.02 -4.45)) (L (pol 6.91 -47.89 -4.53) (pol 6.14 -56.50 -5.24)) (L (pol 6.15 -56.40 -5.08) (pol 6.17 -59.87 -5.15)) (L (pol 9.98 -59.92 -3.22) (pol 9.93 -55.64 -3.38)) (L (pol 9.95 -55.63 -3.32) (pol 9.25 -49.66 -3.26)))**(HJ (n raj1) (ax -0.00))(HJ (n raj2) (ax -0.00))(HJ (n raj3) (ax 0.00))(HJ (n raj4) (ax -0.00))(HJ (n laj1) (ax -0.00))(HJ (n laj2) (ax -0.00))(HJ (n laj3) (ax 0.00))(HJ (n laj4) (ax -0.00))(HJ (n rlj1) (ax -0.00))(HJ (n rlj2) (ax 0.00))(HJ (n rlj3) (ax 0.00))(HJ (n rlj4) (ax 0.00))(HJ (n rlj5) (ax 0.00))(HJ (n rlj6) (ax 0.00))(HJ (n llj1) (ax -0.00))(HJ (n llj2) (ax -0.00))(HJ (n llj3) (ax 0.00))(HJ (n llj4) (ax 0.00))(HJ (n llj5) (ax 0.00))(HJ (n llj6) (ax -0.00))

Tabuľka 2: Odchytené správy zo servera (zvýraznené s-výrazy s informáciami o tom čo agent vidí)

Obrázok 4: Vizualizovaný stav hry, v akom boli odchytené správy (rcsmonitor3d), agent, od ktorého pochádzajú správy je vyznačený v červenom krúžku.

Správu zo servera agent dostáva približne každých 0.02s, pričom každá tretia takáto správa obsahuje informácie o tom, čo agent vidí (podľa simspark wiki by mala každá druhá správa obsahovať

informácie o tom, čo agent počuje, keďže agenti ešte medzi sebou nekomunikujú, nemôžeme túto skutočnosť overiť).

5.4.1. Čo agent vidí

S-výraz	Objekt
F1L, F1R, F2L, F2R	Vlajky v rohoch ihriska
G1L, G1R, G2L, G2R	Stĺpy bránky
P	Hráč
B	Lopta
L	Čiara

Tabuľka 3: S-výrazy a objekty, ktoré agent vidí

(pol <vzdialenosť> <uhol1> <uhol2>)
<vzdialenosť> - vzdialenosť k bodu
<uhol1> - horizontálny uhol, pod ktorým vidí bod, v stupňoch, s odchýlkou 2 stupne
<uhol2> - vertikálny uhol, pod ktorým vidí bod, v stupňoch, s odchýlkou 2 stupne

(P (team <tím>) (id <id>) (<časť_robota> (pol ...) (...)))
<tím> - názov tímu
<Id> - číslo hráča v danom tíme
<časť_tela> - ktorú časť robota vidí agent: (head, rlowerarm, llowerarm, rfoot, lfoot)

Rohové vlajky, stĺpy bránok a loptu vidí agent ako jeden bod (obsahuje s-výraz pol), čiary vidí ako dva koncové body (koncové body časti čiary, ktorú vidí, nie nevyhnutne celú čiaru).

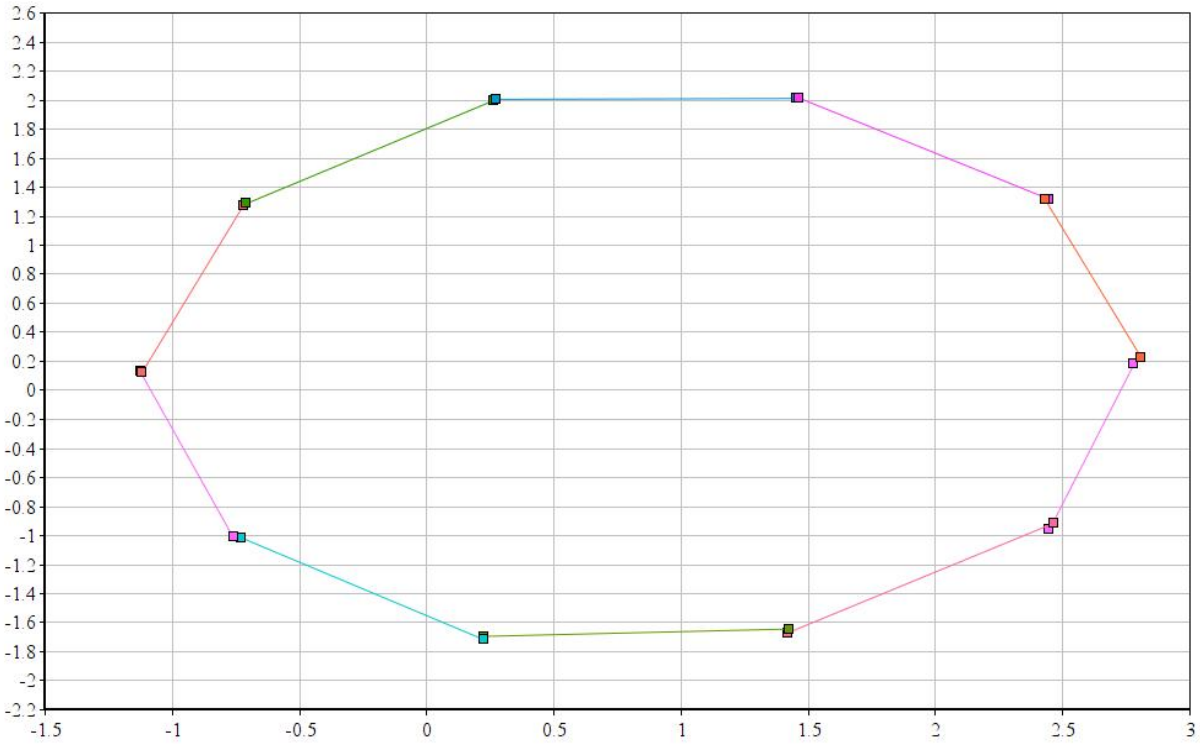
S-výraz informujúci o hráčovi obsahuje informácie, do ktorého tímu patrí hráč, aké má číslo a ktoré časti robota vidí (ako body (pol)).

Agent vidí okrajové čiary ihriska (4x), stredovú čiaru (1x), čiary bránkoviska (3x) a stredový kruh ako 10 úsečiek.

X1	Y1	X2	Y2
2.808061017604897	0.228093015333443	2.429068557019643	1.31805105409419
2.447423566519683	1.316588951660239	1.462303144586890	2.01477178041361
1.452027921257389	2.010688175627931	0.273394317285386	2.00513189291299
0.261025905275772	1.995020067448341	-0.70911778030806	1.28661644695298
-0.72043615375596	1.274804902838681	-1.11855788533631	0.12446626650613
-1.12611265222600	0.129162114400951	-0.76040195151666	-1.0127452297397
-0.73051029472440	-1.01404892136326	0.222938915615861	-1.7155806631713
0.227118404532778	-1.69409656383515	1.426799920671391	-1.6435122989606
1.421154510834064	-1.66827338697430	2.463108734358931	-0.9160303083489
2.447304126557575	-0.96012922577852	2.779778860871767	0.17970883909117

Tabuľka 4: Údaje použité na vizualizáciu stredového kruhu[5]

[5] Údaje boli odchytené po prepočte z relatívnej polohy na koordináty ihriska (viď kapitola 5.6.1 Prepočet súradníc čiar)



Obrázok 5: Ako vidí agent stredový kruh[6]

5.5. Parsovanie údajov

Bc. Jozef Blažíček

V projekte je implementovaná trieda sk.fiit.jim.*Communication.parsing.Parser.* Jej metóda *parse()* je volaná vždy po prijatí správy zo servera (viď kapitola 5.3 Komunikácia so serverom).

5.5.1. Vykonané zmeny

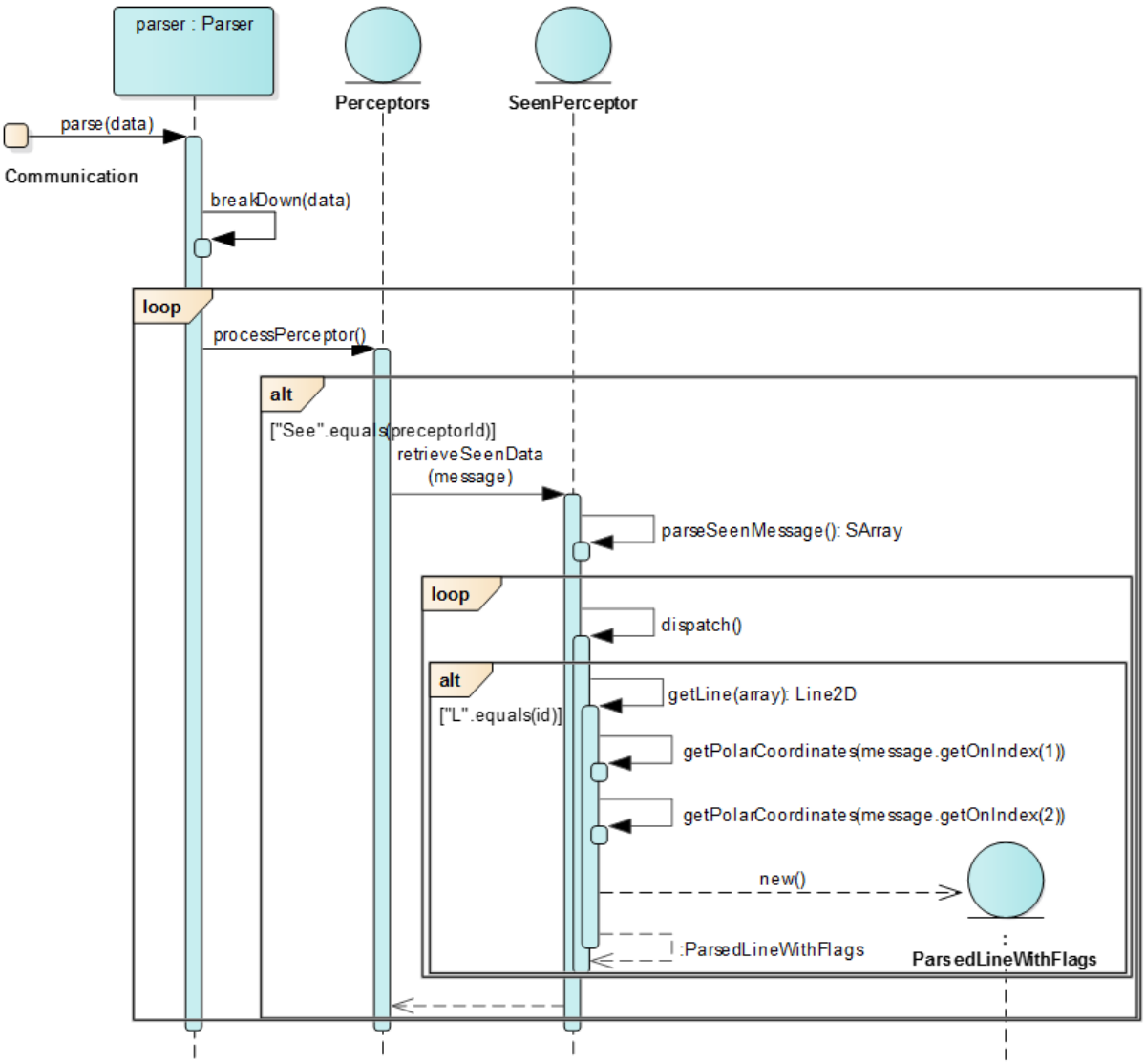
Pri implementácii boli vytvorené 4 triedy z projektu Jim a pridaná jedna trieda na uchovávanie informácií o čiarach s príznakmi pre jednotlivé koncové body (tj. či daný bod je koncovým bodom čiary alebo nie).:

1. Doplnenie zoznam čiar do dvoch tried balíka *sk.fiit.jim.agent.parsing* a to *ParsedData SeenPerceptorData*. Zoznam obsahuje objekty typu Line2D, ktorých cieľom je reprezentácia čiar v relatívnych koordinátoch[7].
2. Doplnenie parsera – funkcie retrieveSeenData v triede sk.fiit.jim.agent.parsing.Perceptors o kopírovanie dát o čiarach zo seenData (SeenPerceptorData) do data (ParsedData).
3. Doplnenie parsera – triedy sk.fiit.jim.agent.parsing.SeenPerceptor.
   - Doplnenie funkcie *dispatch()* o rozpoznávanie čiar a pridanie medzi dáta.
   - Vytvorenie funkcie na extrakciu dát o čiare *getLine().*

Priebeh doimplementovanej časti parseru je znázornený na obrázku 5 v podobe UML diagramu sekvencií (diagram neznázorňuje celý priebeh parsovania, iba časť týkajúcu sa získavania informácií o čiarach).

[6] Na vykreslenie kruhu bolo použité: http://www.onlinecharttool.com/graph?selected_graph=xy

[7] Súradnicová sústava začínajúca v mieste, kde sa nachádza robot – bod (0,0,0) začiatok osí je robot, konkrétne jeho spodná časť.



Obrázok 6: UML diagram sekvencií - priebeh doimplementovaniej časti parsera.

Priebeh doimplementovanej časti parseru je znázornený na obrázku 1 v podobe UML diagramu sekvencií (diagram neznázorňuje celý priebeh parsovania, iba časť týkajúcu sa získavania informácií o čiarach).

Objekt typu *Parser* sa nachádza v triede *Communication* a jeho metóda parse() je volaná vždy po prijatí správy zo servera.

Na začiatku sa vytvorí objekt typu *ParsedData*, ktorý v sebe uchováva informácie získané zo serveru a vracia funkcia po ukončení. Následne sa prijatý S-výraz rozdelí na jednotlivé časti vo funkcii *breakDown()*. V cykle je volaná *processPerceptor()* triedy *Perceptors* pre každú časť prijatého S-výrazu.

Funkcia *processPerceptor()* identifikuje časť správy a zavolá príslušnú funkciu na získanie dát (napr. pre informácie z gyroskopu, o čase, stave hry, čo agent vidí, …).

Ak sa jedná o informácie, čo agent vidí (S-výraz „see“), zavolá sa funkcia retrieveSeenData() triedy *SeenPerceptor*, ktorá vráti objekt obsahujúci dané dáta (SeenPerceptorData). Po návrate s tejto funkcie sa skopírujú dáta do objektu typu *ParsedData*.

Funkcie *retrieveSeenData()* triedy *SeenPerceptor* prebieha podobne ako funkcia *parse()* triedy *Parser*. Správa sa rozdelí na jednotlivé časti, tentokrát uložené v podobe objektu typu *SArray*. Následne je vytvorený objekt *SeenPerceptorData*, kde sa získané dáta uložia. V cykle je

volaná funkcia *dispatch()*, ktorá na základe typu získava dáta o polohe lopty, ostatných hráčov, čiar alebo fixných objektov (trieda *FixedObject*).

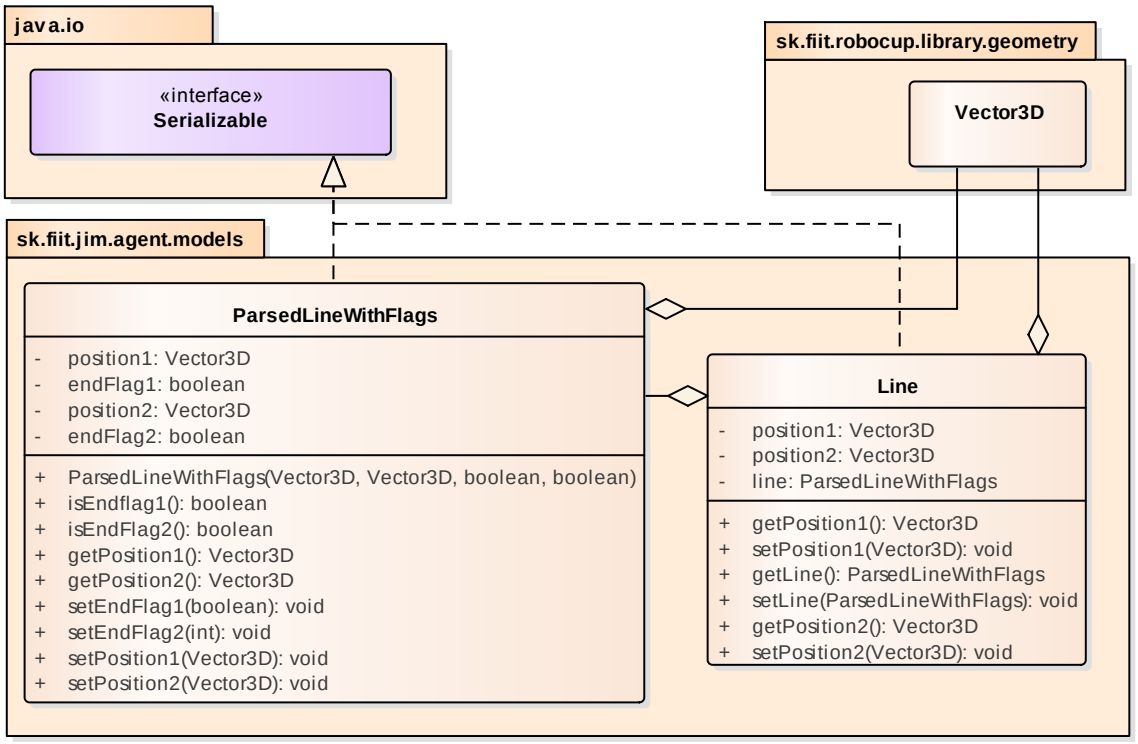
V prípade čiar je volaná funkcia *getLine()*. Funkcia najskôr získa dva objekty typu *Vector3D* z objektu typu *SArray* volaním funkcie *getPolarCoordinates()*. Následne na ak uhol Phi je z intervalu -58° až 58°, nastaví že daný bod je koncovým bodom čiary, inak že nie je. Po vytvorení týchto dvoch objektov a získani príznakov sa vytvorí a vráti nový objekt typu PrasedLineWithFlags, v opačnom prípade null. Vrátený objekt sa vo funkcii *dispatch()* pridá do zoznamu čiar.

5.6. Vykreslenie čiar do testFramework-u

5.6.1. Prepočet súradníc čiar

Bc. Jozef Blažíček

Funkcia *globalize()* v triede *sk.fiit.jim.agent.models.AgentModel* vykonáva prepočet relatívnej polohy na súradnice ihriska (s nulovým bodom v strede ihriska). Tento prepočet vychádza z informácii kde si agent „myslí“, že sa nachádza.



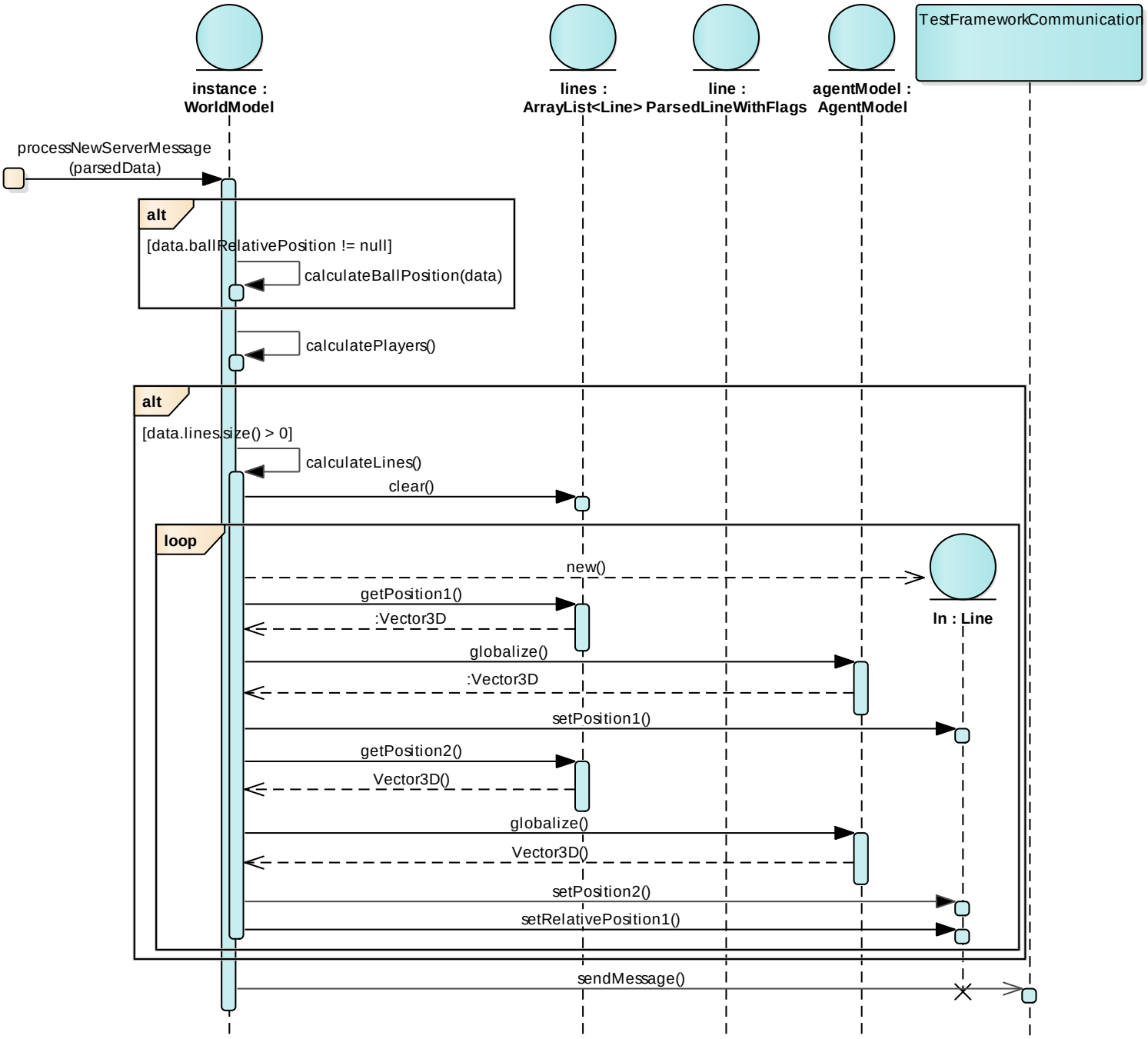
Obrázok 7: UML diagram sekvencií – Spracovanie novej správy pre server.

Novovytvorená trieda *sk.fiit.jim.agent.models.Line* slúži ako kontajner na uchovanie informácií o relatívnej polohe čiar a prepočítanej polohe na koordináty ihriska.

Obsahuje štyri premenné typu *Vector3D* (*sk.fiit.robocup.library.geometry.Vector3D*), inicializované na nulový vektor:

- position1 koordináty prvého bodu čiary v súradniciach ihriska
- position2 koordináty druhého bodu čiary v súradniciach ihriska
- line relatívna pozícia čiary s príznakmi jednotlivých bodov

Do triedy *sk.fiit.jim.models.WorldModel* bol pridaný zoznam čiar (*ArrayList* objektov typu *Line*) a funkcia na aktualizáciu tohto zoznamu – *calculateLines()*.



Obrázok 8: UML diagram tried - trieda Line

Spracovanie novej správy sa v podstate skladá zo štyroch krokov:

1. Vypočítanie polohy lopty.
2. Vypočítanie polohy hráčov.
3. Vypočítanie čiar.
   Ak zoznam čiar, získaný spracovaním z parsera (viď kapitola 5.5.1. Vykonané zmeny) nie je prázdny, vyprázdni sa zoznam čiar (typu Line v triede WorldModel) a následne sa vo for-cykle pre všetky čiary z parsera najskôr vytvorí nový objekt typu *Line*, kde sa postupne nastavia informácie o bodoch.
4. Odoslanie správy do TestFramework-u.

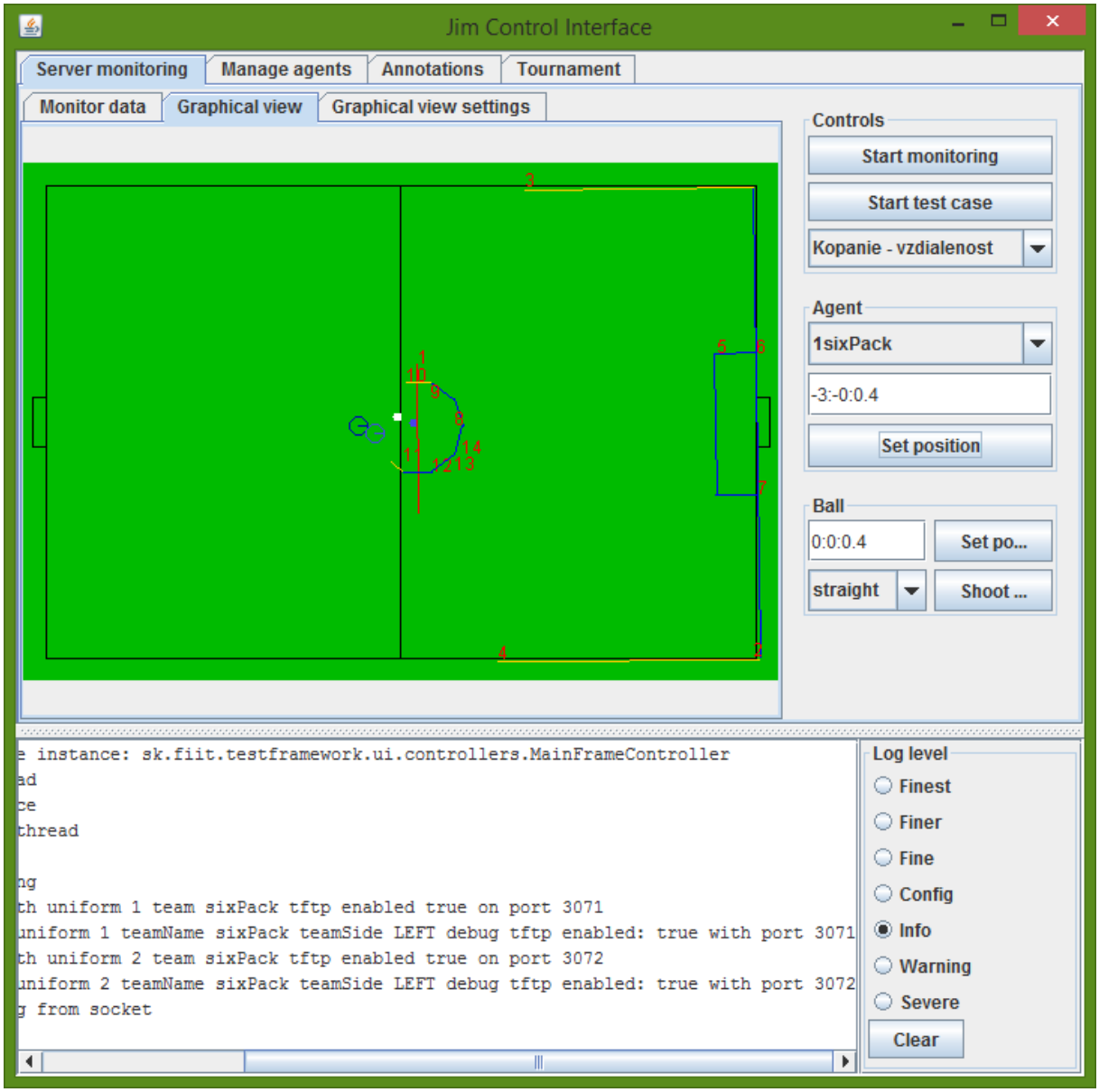
5.6.2. Vykreslenie čiar

Bc. Matúš Ivanoc

Pracovalo sa v triede „GameView“ a metóde *paintComponent()*. Táto metóda vykresľuje komponenty na 2D mape vrátane pozície lopty. Doplnil sa for cyklus, kde sa postupne vykresľujú čiary, ktoré posiela JIM agent. Skúšali sme vykresľovať čiary podľa absolútnych a aj relatívnych súradníc. Výstup bol dosť podobný s rozdielom, že relatívne súradnice mali posunutú orientáciu o 90 stupňový uhol.

Bol pokus očíslovať jednotlivé čiary, ale je problém s číslovaním z dvoch dôvodov:

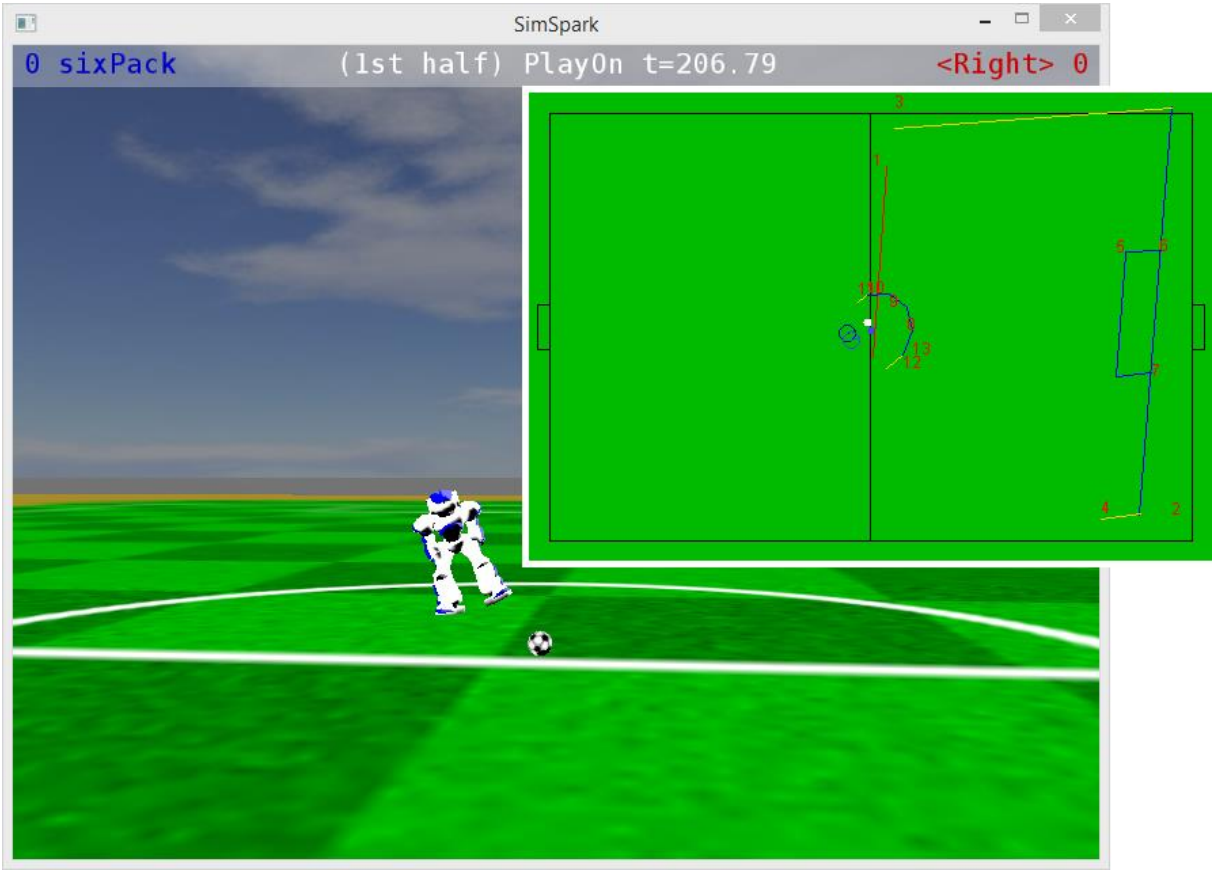
- prekrývanie čísla čiar s rovnakými súradnicami začiatočných bodov,
- nevieme triviálne určiť, ktorý bod čiary je začiatočný.



Obrázok 9: Ukážka vykreslených čiar v TestFramework-u - hráč stojí na mieste

V závislosti od typu čiary (príznakov jej koncových bodov) sa vykreslia čiary do TestFrameworku:

- červená - agent nevidí ani jeden koncový bod čiary,
- žltá - agent vidí jeden z koncových bodov čiary,
- modrá - agent vidí celú čiaru.



Obrázok 10: Ukážka vykreslených čiar v TestFramework-u – hráč vstáva zo zeme

5.7. História polohy agenta

Agent uchováva históriu pozícii v triede *AgentModel* v premennej *Queue<Vector3D> listPoints*, ktorá je implementovaná ako *LinkedList*. Uchováva sa posledných 21 pozícií. Premenná sa však využíva len pri regresnom počítaní pozície hráča, ktoré je, zdá sa byť, vypnuté.

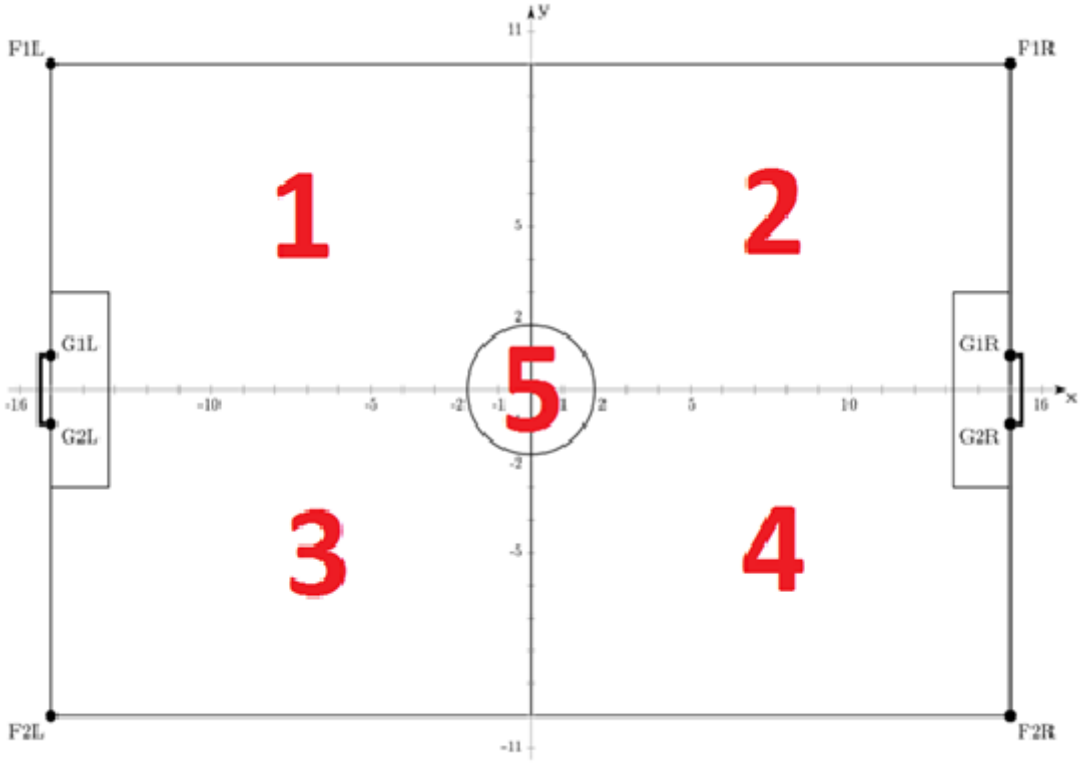
5.8. Určenie časti ihriska, v ktorej sa agent nachádza

Bc. Miloš Štefčák

Pri implementácii bola vytvorená 1 nová metóda v projekte Jim, triede „*sk.fiit.jim.agent.models.AgentModel*“.
Vytvorili sme metódu „*getQuarter*“ , ktorá ako parametre dostáva x-ovú a y-ovú súradnicu hráča a následne vracia štvrtinu ihriska, v ktorej sa hráč nachádza.

Určenie časti ihriska, v ktorej sa hráč nachádza, sa určuje nasledovne:

1. x = záporné, y = kladné
2. x = kladné, y = kladné
3. x = záporné, y = záporné
4. x = kladné, y = záporné
5. $x^2 + y^2 \Leftarrow (2 = \text{polomer stredového kruhu})^2$



5.9. Určenie priesečníkov

Bc. Matúš Ivanoc

Počítanie T priesečníkov čiar, sa aktuálne vykonáva v triede GameView v balíku *sk.fiit.testframework.ui.*Boli vypracované metódy isVertical() a isHorizonta(), ktorých vstupným parametrom je čiara z triedy Line. V týchto metódach sa zisťuje orientácia podľa súradníc koncových bodov x a y. Ak je čiara horizontálna, je predpoklad, že súradnice oboch koncových bodov čiary, ktoré agent vidí sú v osi Y rozdielne len do miery zašumenia. To isté platí pre isVertical(), kde sa porovnávajú x súradnice.
Následne sa ešte pridali metódy pointDeviation() a inRange(). Prvá metóda zisťuje, či je vzdialenosť dvoch bodov v jednej súradnici v medzi zašumenia, ktorú sme určili ako 0,2.

Následne výpočet priesečníkov prebieha v metóde findT(), ktorý tvorí sústava vetvení. Využíva sa hash mapa keySet, do ktorej sa naplnia všetky viditeľné čiary agentom. Následne sa porovnáva so sub mapou subKeySet, ktorá má o jednu čiaru v sebe menej. For cyklom sa vyberá z keySet prvá čiara, s ktorou sa porovnajú ostatné, či má niekde priesečník. Sub hash mapa má vždy o čiaru menej z prvého for cyklu. Zisťovanie prebieha porovnávaním vzdialeností koncových bodov podľa vyššie opísaných metód a nájdené priesečníky sa ukladajú do zoznamu typu *<Point2D>.*

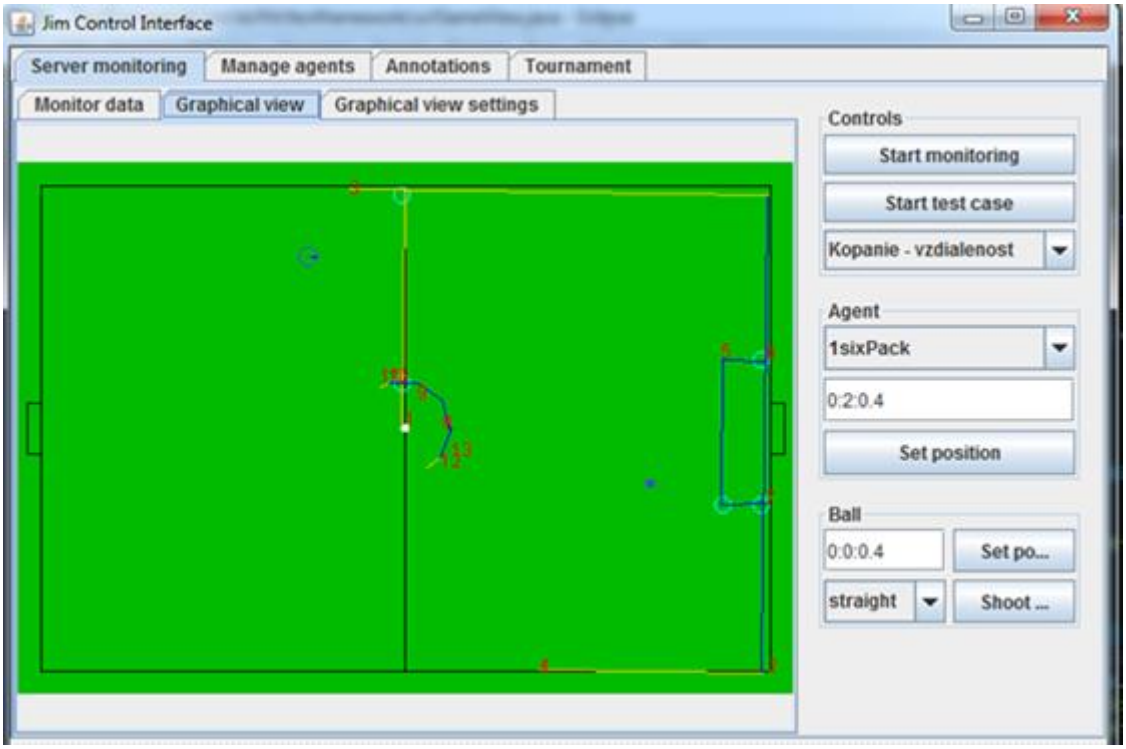
Zistené priesečníky sa potom vykresľujú na mape TestFrameworku v metóde paintComponent().

5.9.1. Výstup na 2D mape

Aktuálne je určovanie T priesečníkov len ako zobrazovacia funkcionalita v TestFrameworku a nemá žiadne využitie pre orientáciu. Slúži to pre pochopenie ako agent vníma čiary na ihrisku a ako sa s nimi dá pracovať. Výstup tejto úlohy nebol veľmi povzbudzujúci. Silné zašumenie z RC3D servera

spôsobuje značné odchýlky a aj po ladení tolerancií, s ktorými je potrebné počítať, sme nedokázali 100% určiť každý T priesečník ani rohový bod pri dvoch čiarach okolo bránkoviska. Je potrebné ďalej zapracovať na kategorizácii priesečníkov a zistiť, či je to T na stredovej čiare, alebo pri bránkach.

Do budúcna je potrebné tiež zvážiť a skúsiť sa zamerať na stabilizáciu zašumenia, ktoré posiela server spolu s čiarami. V aktuálnej forme je daný výstup dosť nestabilný a pre orientáciu agenta veľmi ťažko použiteľný.

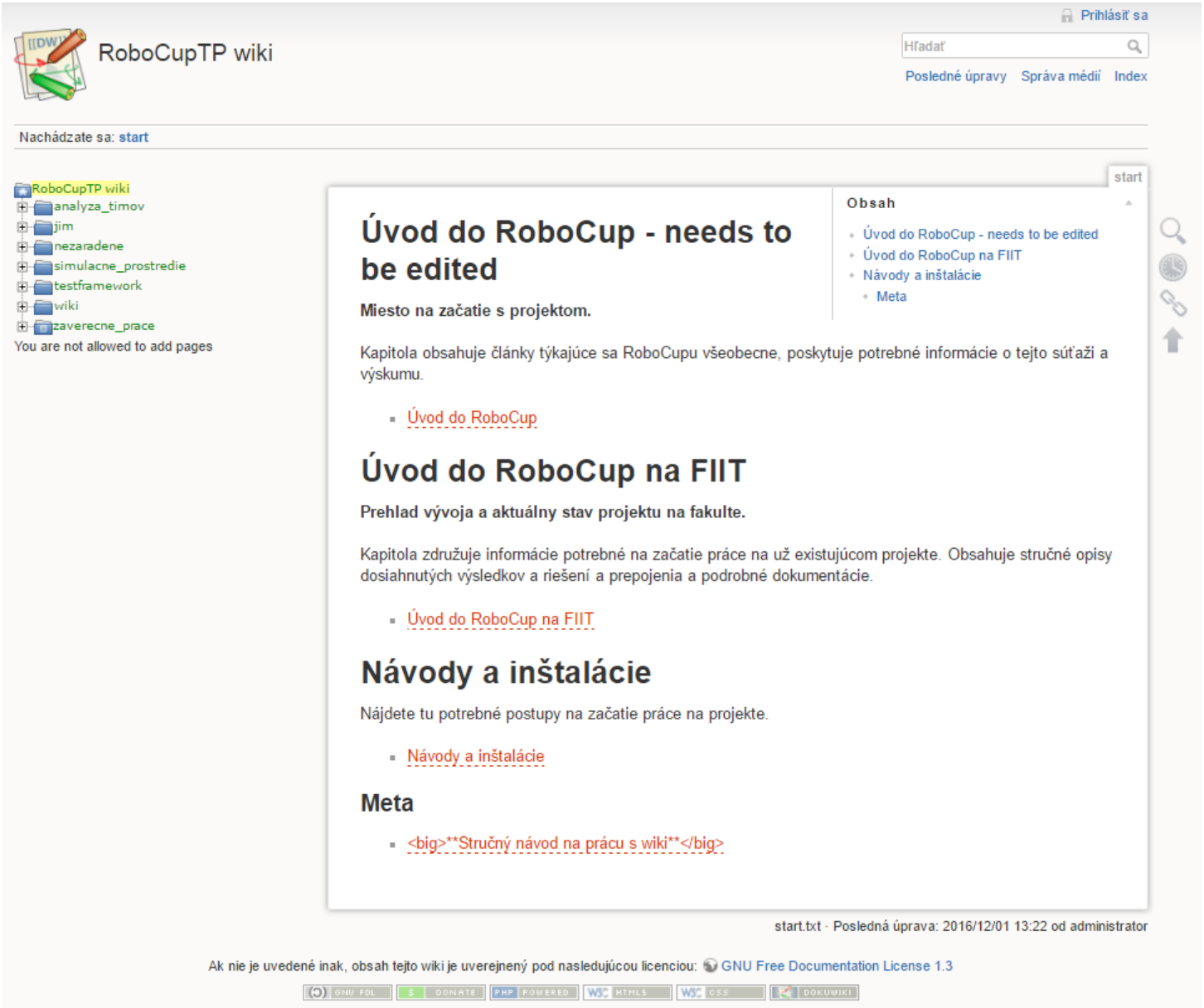


6.wiki

Bc. Jozef Blažíček, Bc. Matúš Ivanoc

6.1. DokuWiki

Bc. Matúš Ivanoc



Obrázok 11: Úvodná stránka wiki

Náš tím získal na oboznámenie s projektom aj Wiki tímového projektu, kde boli uchovávané znalosti nadobudnuté po minulé roky. Táto Wiki má slúžiť ako báza znalostí pre všetkých, ktorí na danej téme robia buď ako bakalári, diplomanti, alebo v rámci TP. Na začiatku boli problémy s inštaláciou Wiki, kde chýbala databáza od posledného tímu a nebolo jednoduché získať minuloročnú verziu. Po vyriešení počiatočných problémov, sme sa na porade tímu zhodli, že Wiki potrebuje prepracovať, ak sa má začať používať poriadne.

Problémy v Mediawiki:

- chýbajúca štruktúra stránok v akejkoľvek forme,
- chýbajúce navigačné menu,
- škaredý dizajn prostredia,
- neaktuálne informácie.

Dohodlo sa na potrebe vypracovania štruktúry s kategóriami pre jednotlivé stránky a pridaní jednoduchého navigačného poľa v menu paneli. Pri tejto diskusii padol návrh prejsť na lepší engine Dokuwiki, ktorý natívne podporuje túto funkcionalitu a stromovú štruktúrou kategórií.

Dokuwiki takisto podporuje množstvo rozšírení, ktoré pridávajú užitočné funkcionality do základnej verzie:

- stromová štruktúra menu (IndexMenu Plugin),
- zálohovací plugin (Backup Tool),
- WYSIWYG písanie textu (Ckgedit),
- prístupové práva (ACL Manager),
- vytváranie používateľov (User Manager),
- vytváranie stránok do menu štruktúry (Add New Page).

Ku konečnému rozhodnutiu prejsť na Dokuwiki sme sa rozhodli po otestovaní, že nebude problém presunúť stránky zo starej mediawiki. Zároveň sa pri tom vypracuje štruktúra stránok a obsah sprístupní pre upravovanie obsahu.

6.1. Štruktúra Wiki

Bc. Jozef Blažíček

Štruktúra wiki bola rozdelená do siedmych hlavných častí:

- **analyza_tímov**
  Obsahuje články týkajúce sa analýzy tímov (či už predchádzajúcich, čo pracovali na projekte na FIIT alebo zahraničných)
- **jim**
  Obsahuje články týkajúce sa projektu Jim a agenta
- **nezaradene**
  Obsahuje články, ktorým zatiaľ nebola pridelená žiadna z kategórií
- **simulacne prostredie**
  Simulacne prostredie by malo obsahovat informacie ohlaodm sveta
- **testframework**
  Obsahuje články týkajúce sa projektu TestFramework
- **wiki**
  Obsahuje články, týkajúce sa wiki
- **zaverecne prace**
  Obsahuje prehľad bakalárskych a diplomových prác vytvorených na našej fakulte.

6.2. Aktualizovanie informácií

Bc. Jozef Blažíček

Ďalším z dlhodobých cieľov je udržiavanie informácií na Wiki čo možno v najaktuálnejšom stave. Wiki slúži ako zdroj informácií pre nasledujúce tímy, autorov bakalárskych a diplomových prác.

6.2.1. Vytvorené články

Analýza využitia informácii o čiarach v zahraničných tímoch *(analyza_timov)*[8]
Informácie o využívaní informácií o čiarach zahraničnými tímami

Komunikácia agenta so serverom *(jim)*[9]
Opisuje priebeh komunikácie agenta so serverom.

Správy zo servera *(jim)*[10]
Opisuje správy prichádzajúce zo servera.

Zistenie, v ktorej štvrti sa nachádza agent *(jim)*[11]
Opisuje rozdelenie ihriska na časti a spôsob identifikácia.

Parsovanie čiar
(jim / vylepsenie_zistovania_polohy_hraca_na_zaklade_videnia_ciar)[12]
Opisuje princíp parsovania správ zo servera a bližšie približuje získavanie informácií o čiarach, používané a vytvorené triedy.

Prepočet relatívnej polohy z parsera
(jim / vylepsenie_zistovania_polohy_hraca_na_zaklade_videnia_ciar)[13]
Opisuje princíp prepočtu relatívnej pozície na globálne súradnice, taktiež opisuje vytvorenú triedu.

Zisťovanie T priesečníkov čiar *(testframework / ciary)*[14]
Opisuje hľadanie priesečníkov čiar.

Vykresľovanie čiar *(testframework / ciary)*[15]
Opisuje vykresľovanie čiar v TestFrameworku.

6.2.2. Aktualizované články

Analýza fyzikálneho modelu (jim)[16]
Aktualizovanie informácií o ihrisku
Aktualizovanie informácií (tabuľky) nastavenia kĺbov (maximálnych a minimálnych hodnôt) agenta.

[8] http://team07-16.studenti.fiit.stuba.sk/dokuwiki/doku.php?id=analyza_timov:analyza_vyuzitia_informacii_o_ciarach_v_zahranicnych_timoch
[9] http://team07-16.studenti.fiit.stuba.sk/dokuwiki/doku.php?id=jim:komunikacia_agenta_so_serverom
[10] http://team07-16.studenti.fiit.stuba.sk/dokuwiki/doku.php?id=jim:spravy_zo_servera
[11] http://team07-16.studenti.fiit.stuba.sk/dokuwiki/doku.php?id=jim:zistenie_v_ktorej_stvrti_ihriska_sa_nachadza_agent
[12] http://team07-16.studenti.fiit.stuba.sk/dokuwiki/doku.php?id=jim:vylepsenie_zistovania_polohy_hraca_na_zaklade_videnia_ciar:parsovanie
[13] http://team07-16.studenti.fiit.stuba.sk/dokuwiki/doku.php?id=jim:vylepsenie_zistovania_polohy_hraca_na_zaklade_videnia_ciar:prepocitanie_relativnej_pozicie
[14] http://team07-16.studenti.fiit.stuba.sk/dokuwiki/doku.php?id=testframework:ciary:t_priesecniky_ciar
[15] http://team07-16.studenti.fiit.stuba.sk/dokuwiki/doku.php?id=testframework:ciary:vykreslovanie_ciar
[16] http://team07-16.studenti.fiit.stuba.sk/dokuwiki/doku.php?id=jim:analyza_fyzikalneho_modelu

7.Záver

Bc. Jozef Blažíček

7.1. Prvá etapa (Zimný semester)

V prvej etape sme sa oboznámili s projektom a do istej miery analyzovali aktuálny stav projektu.

Pre túto etapu sme mali zvolené dva hlavné ciele:

1. Čiary (viď kapitola 5. čiary)
2. Wiki (viď kapitola 6. wiki)

7.1.1. Čiary

Analyzovali sme správy prichádzajúce zo servera (viď kapitola 5.4. Správy zo servera) so zameraním na informácie o čiarach. Následne sme do existujúceho parsera doplnili ich rozpoznávanie (viď kapitola 5..5. Parsovanie údajov). Prepočítali informácie na súradnice ihriska a rozšírili sme vizualizáciu v TestFrameworku o zobrazovanie čiar (viď kapitola 5.6. Vykreslenie čiar do testFramework-u), ako ich agent vidí. Táto vizualizácia mala za účel lepšie pochopiť, aké informácie dostáva agent a ako vníma svet.

Výsledkom vykonanej analýzy zahraničných tímov, ktoré používajú informácie o čiarach (viď kapitola 5.2. Analýza zahraničných tímov) bolo, že síce niektoré (dva) tímy využívajú informácie o čiarach, ale tieto akcie sú výpočtovo veľmi náročné a nedosahujú výrazne lepšie výsledky.

Nakoľko agent nevie o akú čiaru sa jedná, je najskôr potrebné ich mapovanie na čiary ihriska. Agent má implementovanú históriu posledných polôh, ktorá by mohla proces identifikácie čiar či hľadania nových kontrolných bodov urýchliť. Analýzou sme zistili (viď kapitola 5.7 História polohy agenta), že aj keď má agent informácie o predchádzajúcich polohách, tieto informácie sa v aktuálnej verzii projektu nevyužívajú.

Doimplementovali sme identifikáciu koncových bodov čiary (agent vidí dva body z čiary, nemusia to byť nutne koncové body danej čiary, identifikácia nesie informáciu, či daný bod je skutočným koncovým bodom čiary alebo nie). Cieľom je urýchlenie hľadania priesečníkov čiar a následnej identifikácie čiar. Implementovaná identifikácia priesečníkov čiar (viď kapitola 5.9. určovanie priesečníkov) zatiaľ nepoužíva tieto informácie o čiarach.

Okrem spomenutých rozšírení sme aj implementovali JUnit testy na dané úlohy.

Agent dostáva iba informácie o čiarach, konkrétne dva body z čiary, ktoré vidí. Nedostáva nijaký identifikátor, o ktorú čiaru sa konkrétne jedná, preto je najskôr potrebná identifikácia čiar na základe kontrolných bodov, alebo ak je možné, na základe rozloženia čiar, ktoré agent vidí.

Čiary môžu pridať viac bodov, na základe ktorých bude agent vyhodnocovať svoju polohu.

Algoritmus mapovania čiar a následného vyhodnotenia polohy by výrazne predĺžil celkový čas na vyhodnotenie polohy a výsledky nie sú o veľa presnejšie ako bez použitia čiar. Tento fakt uvádzajú aj zahraničné tímy vo svojich prácach.

7.2. wiki

Prešli sme z MediaWIki na DokuWiki, vytvorili štruktúru článkov (ktorá doteraz chýbala), aktualizovali informácie a doplnili naše výsledky (viď kapitola 6.wiki).