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

Predmet: Tímový projekt I, Tímový projekt II

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. (viď kapitola 5. Čiary – prvá etapa)

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 8. wiki).

3.2. Ciele pre letný semester

1. **Dokončiť vylepšenie polohy hráča s použitím informácií o čiarach**
   Pokračovať v práci začatej v zimnom semestri (viď kapitola 6 Čiary – druhá etapa). Na základe informácií o projekte nadobudnutých v zimnom semestri sme vytvorili návrh vyhodnocovania polohy agenta, ktorý pokryje všetky prípady, kedy agent nevedel vyhodnotiť svoju polohu – t.j. agent by mal byť schopný vždy po prijatí informácií o videných objektoch aproximovať svoju polohu na ihrisku.

2. **Aktualizácia wiki**
   Pokračovať v aktualizácii informácii na wiki stránke (viď kapitola 8. wiki).

3. **TDD**
   Vytvorenie unit testov pre stanovené úlohy a ciele. Na základe videných objektov (čiar a kontrolných bodov – t.j. vlajok a bránok), existuje niekoľko spôsobov, akými môže prebehnúť vyhodnotenie polohy agenta na ihrisku (viď kapitola 7. TDD).

[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

Táto kapitola obsahuje opis technológií, ktoré sme používali počas vypracovávania projektu.

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

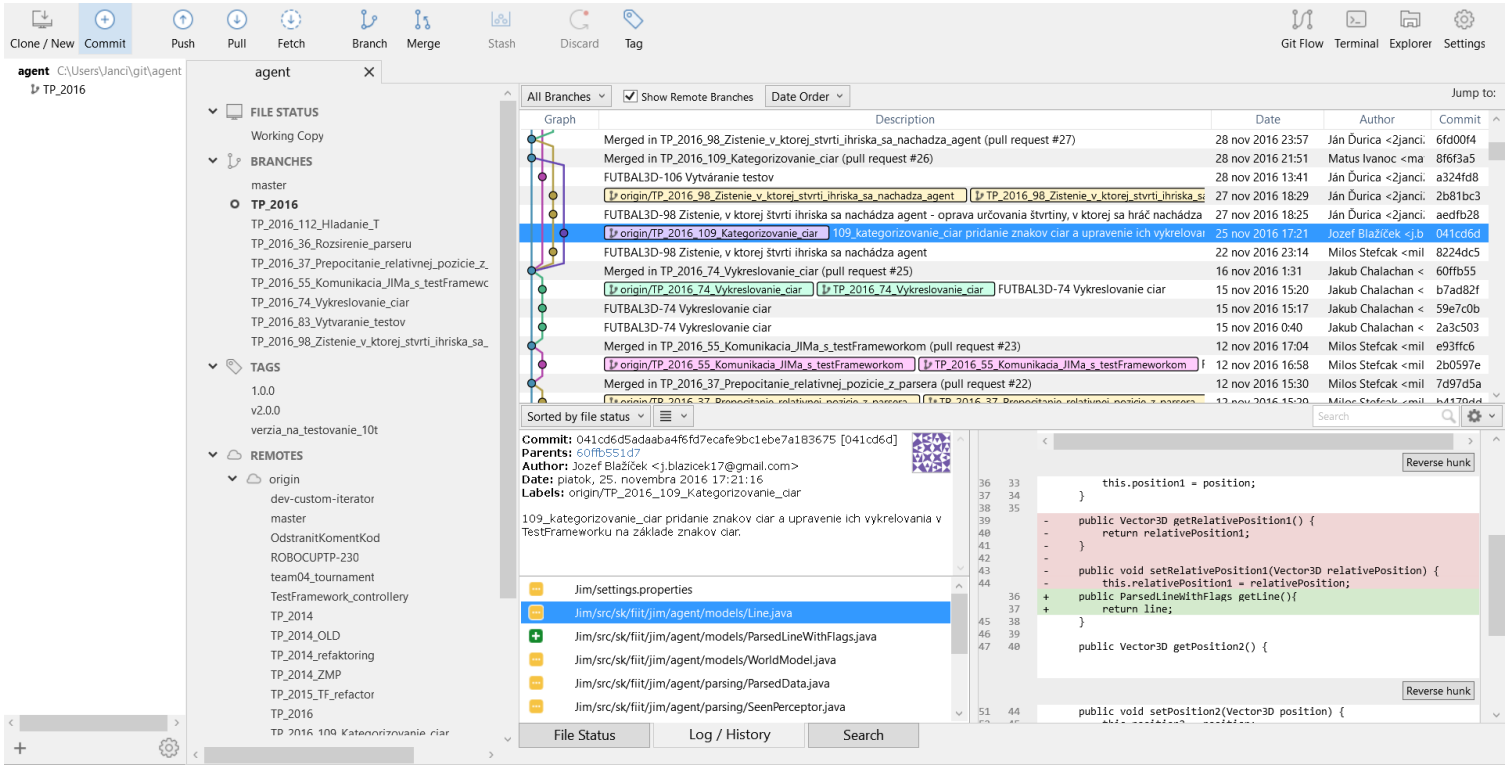
SourceTree je voľne dostupný nástroj a zároveň klient pre systém na správu verzií Git. Keďže náš tím používa tento systém, ako hlavný projektový manažér a človek so skúsenosťami s používaním tohto nástroja, nakoľko je to veľmi užitočný nástroj, hlavne pre ľudí, čo pred tým s Gitom veľa nepracovali. SourceTree 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 – prvá etapa

Bc. Jozef Blažíček, Bc. Maryna Kovalenko, Bc. Matúš Ivanoc

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**

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.

Výhody použitého algoritmu:

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

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.

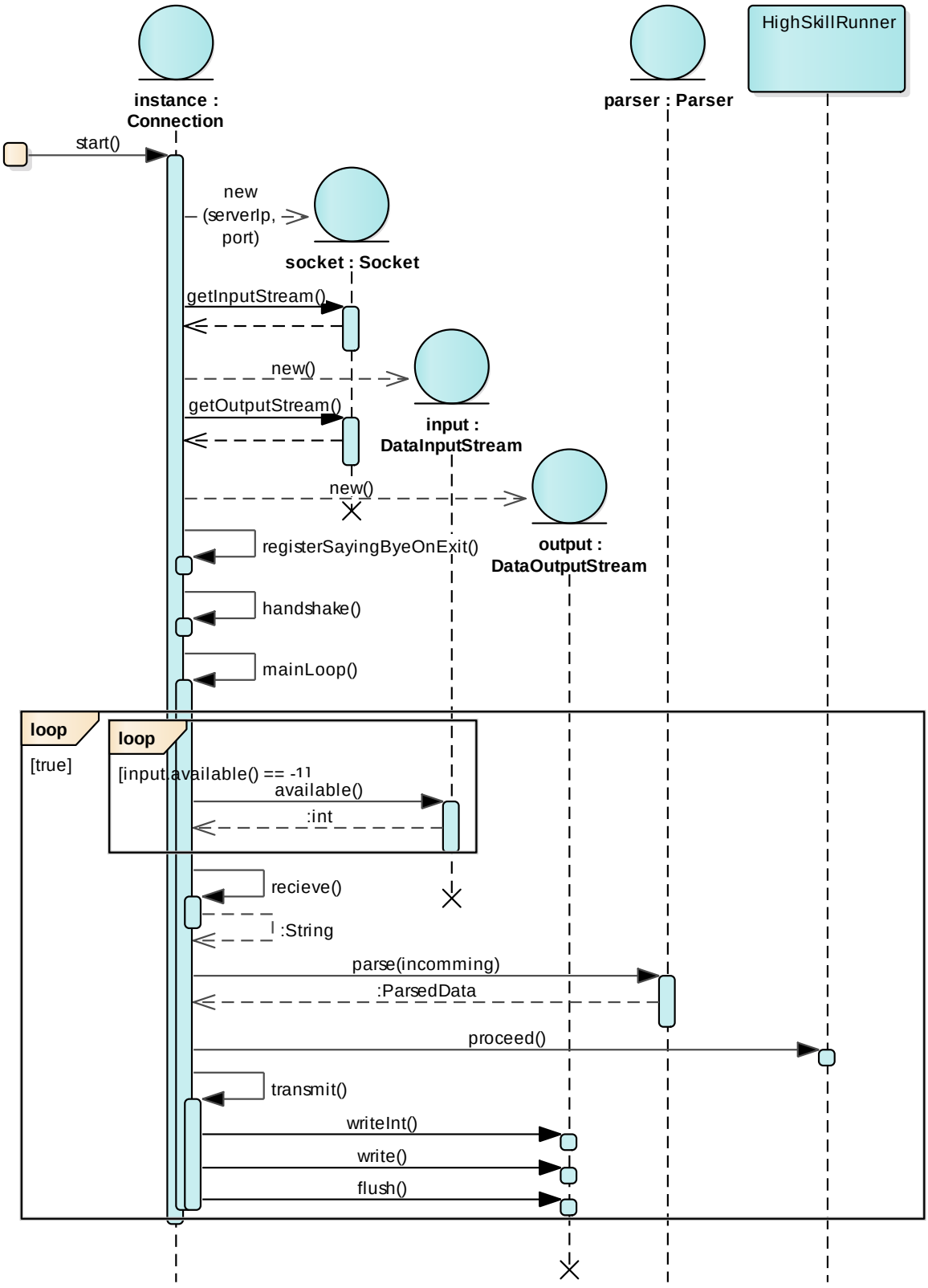
5.2.4. Odkazy

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



Obrázok 2: UML diagram sekvencií - Priebeh komunikácie agenta so serverom

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 ...)“).

Agent zo servera dostáva rôzne informácie (viď kapitola 5.4. Správy zo servera a kapitola 7.1.2. Správy zo servera – Stav hry) 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	Čo agent počuje

Tabuľka 1: S-výrazy, ktoré sa môžu vyskytnúť v správe zo servera

Obrázok 3: Vizualizovaný stav hry, v akom boli odchytené správy (rcsmonitor3d), agent, od ktorého pochádzajú správy je vyznačený v červenom krúžku.

```
(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
(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)

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)

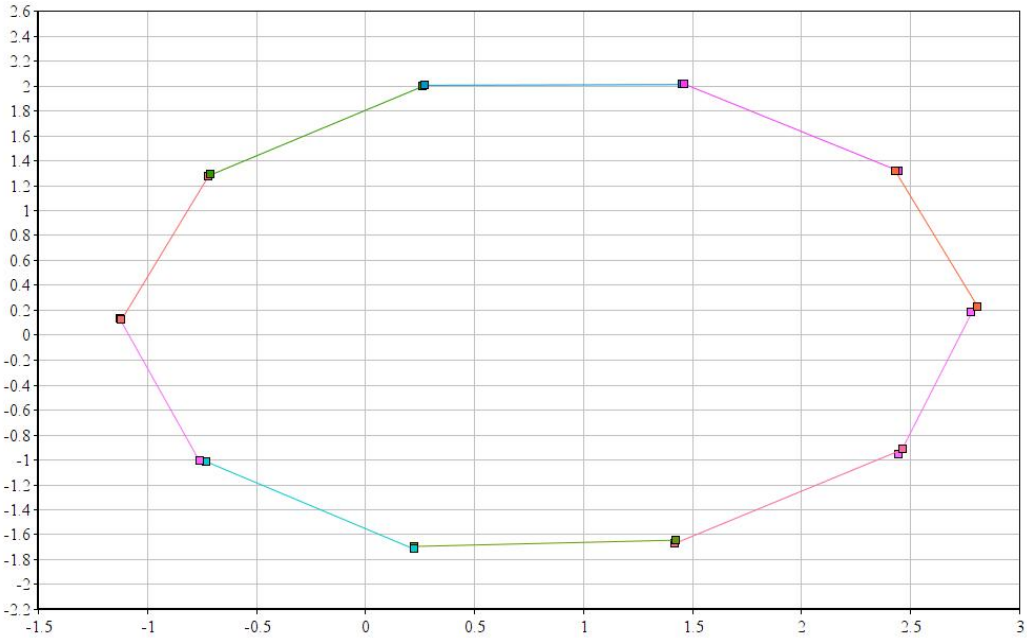
Rohové vlajky, stĺpy bránok a loptu vidí agent ako jeden bod (obsahuje s-výraz pol), čiary vidí ako dva koncové body (koncové body časti čiary, ktorú vidí, nie nevyhnutne celú čiaru).

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



Obrázok 4: Ako vidí agent stredový kruh[3]

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

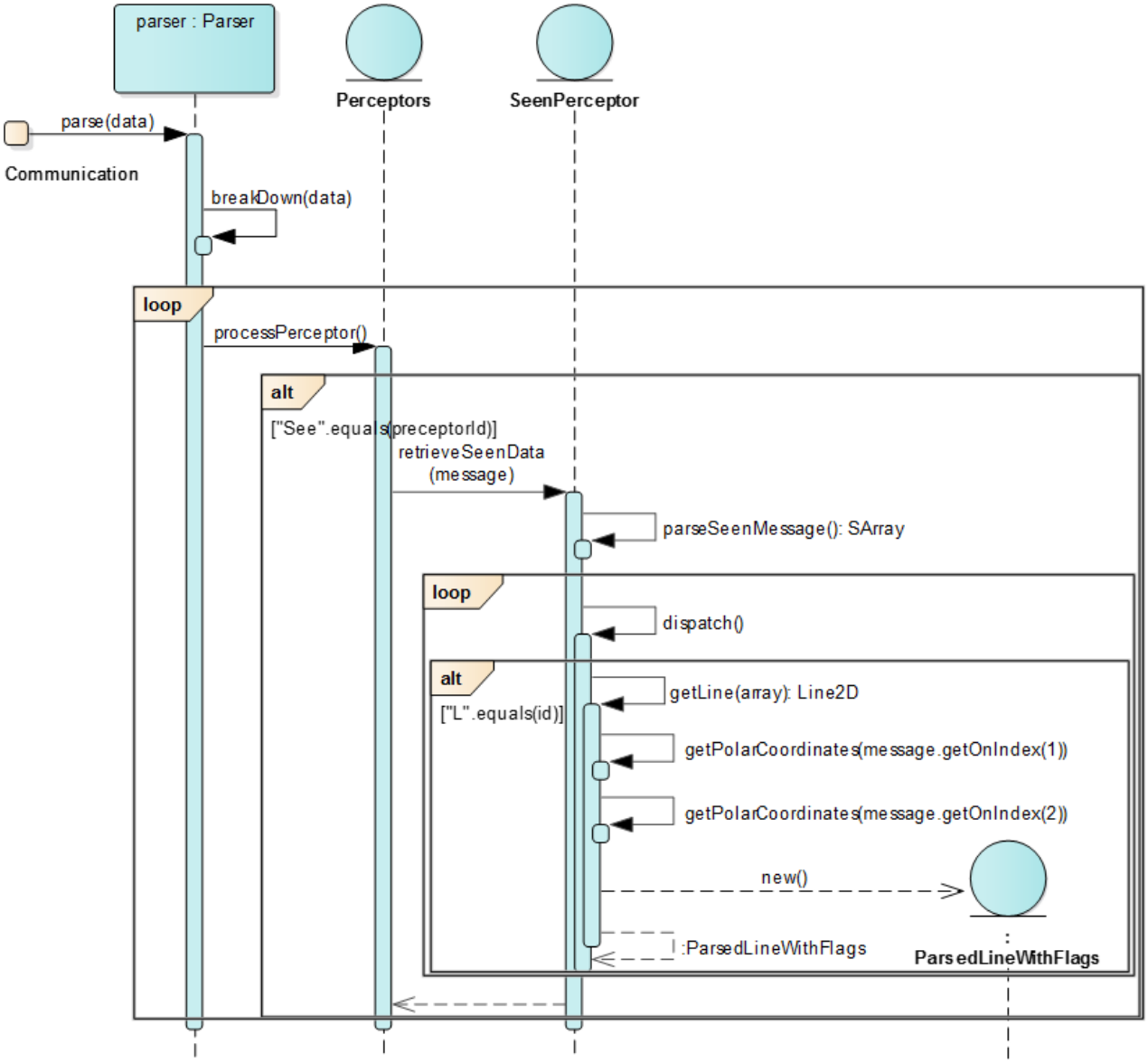
Pri implementácii boli vytvorené 4 triedy z projektu Jim a pridaná jedna trieda na uchovávanie informácií o čiarach s príznakmi pre jednotlivé koncové body (tj. či daný bod je koncovým bodom čiary alebo nie).:

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[4].
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).

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

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



Obrázok 5: UML diagram sekvencií - priebeh doimplementovanej 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*.

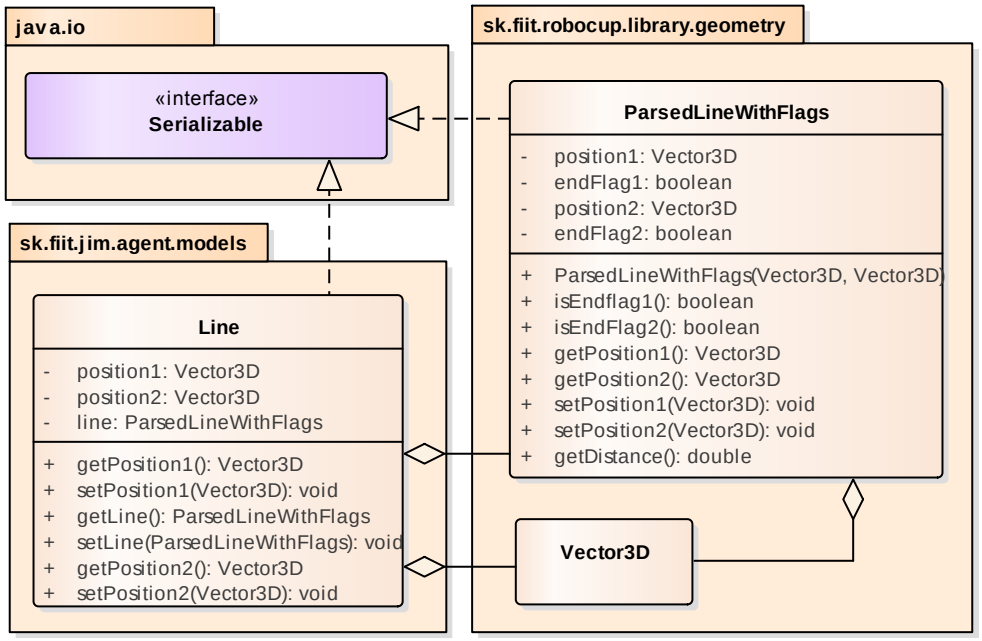
Funkcie *retrieveSeenData()* triedy *SeenPerceptor* prebieha podobne ako funkcia *parse()* triedy *Parser*. Správa sa rozdelí na jednotlivé časti, tentokrát uložené v podobe objektu typu *SArray*. Následne je vytvorený objekt *SeenPerceptorData*, kde sa získané dáta uložia. V cykle je volaná funkcia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



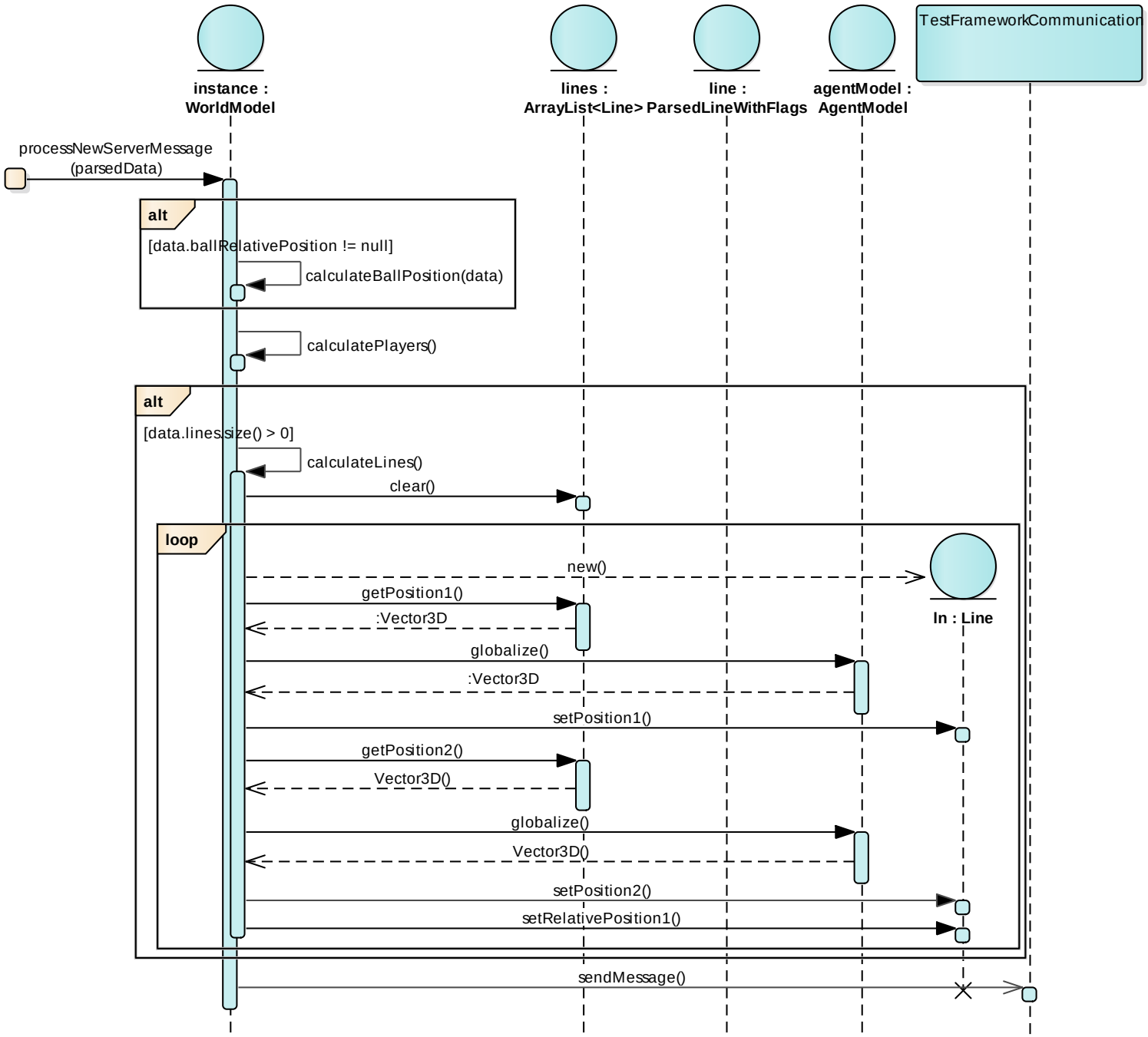
Obrázok 6: *UML diagram sekvencií – Spracovanie novej správy pre server.*

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.

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



Obrázok 7: UML diagram tried - trieda Line

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

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

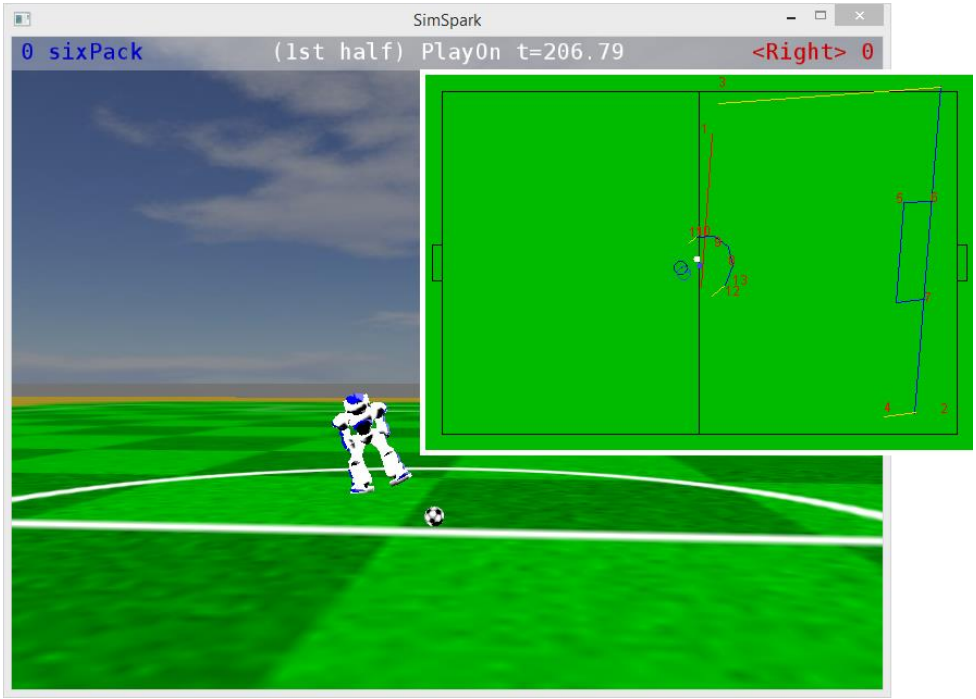
Pracovalo sa v triede „GameView“ a metóde *paintComponent()*. Táto metóda vykresľuje komponenty na 2D mape vrátane pozície lopty. Doplnil sa for cyklus, kde sa postupne vykresľujú čiary, ktoré posiela JIM agent. Skúšali sme vykresľovať čiary podľa absolútnych a aj relatívnych súradníc. Výstup bol dosť 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:

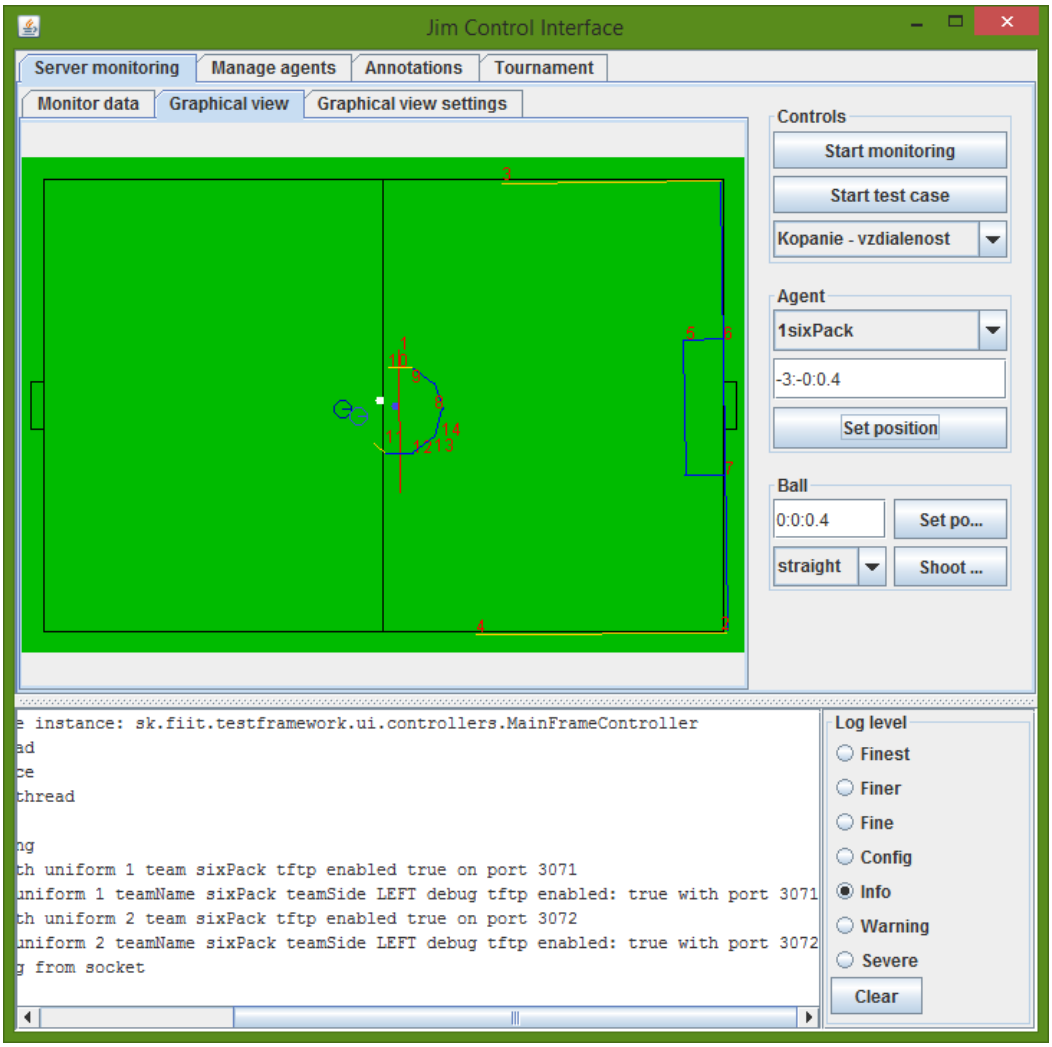
- prekrývanie čísla čiar s rovnakými súradnicami začiatočných bodov,
- nevieme triviálne určiť, ktorý bod čiary je začiatočný.

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



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

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

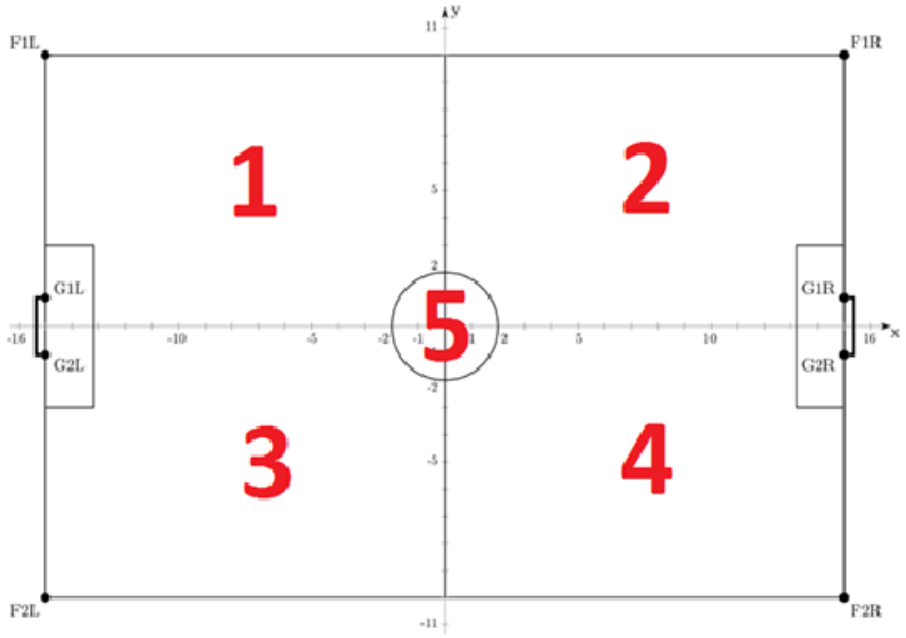
Bc. Miloš Štefčák

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. $x^2 + y^2 \Leftarrow (2 = \text{polomer stredového kruhu})^2$



Obrázok 10: Rozdelenie ihriska na 5 častí

6. Čiary – druhá etapa

Jozef Blažíček

(1) Koľko kontrolných bodov vidí agent?
Pre dostatočne presné určenie polohy potrebuje agent mať informácie o minimálne dvoch bodoch.

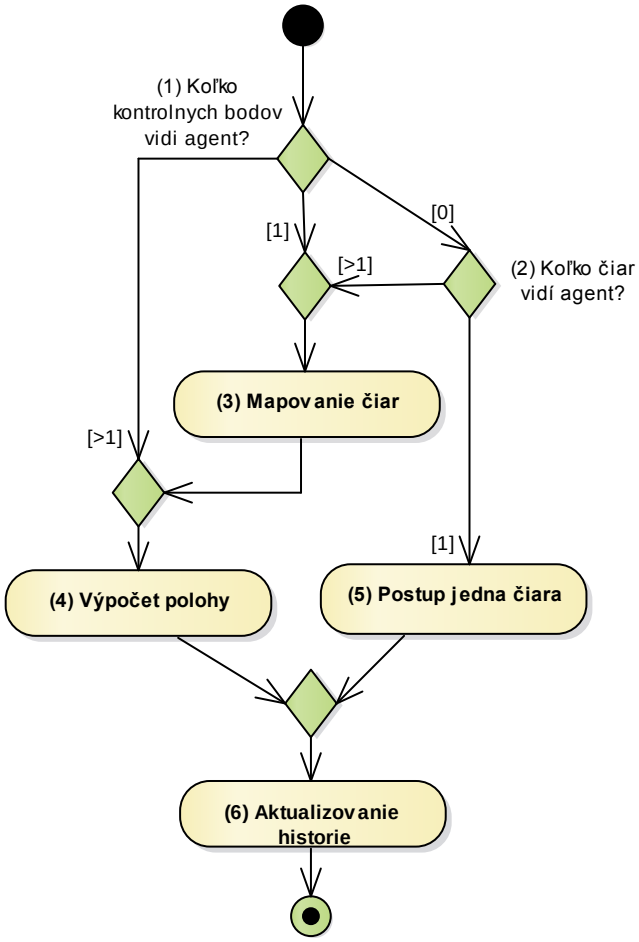
(2) Koľko čiar vidí agent?
Pre mapovanie čiar je potrebné, aby agent videl aspoň dve čiary (z dvoch čiar sa dajú odvodiť vzťahy)

(3) Mapovanie čiar
Ak agent vidí nejaký kontrolný bod alebo aspoň dve čiary, je možné namapovať videné čiary na čiary ihriska. Ak nevidí kontrolný bod, mapovanie prebehne s použitím histórie.

(4) Výpočet polohy
Poloha sa vypočíta z namapovaných bodov, t.j. bodov o ktorých vieme absolútnu polohu (napr. L1R má súradnice (15,10,0)) a relatívnu (agent vidí daný bod ako (10,5,0)).

(5) Postup jedna čiara
Špeciálnym prípadom aproximácie polohy agenta je, keď vidí iba jednu čiaru.

(6) Aktualizovanie histórie
Aktualizovanie histórie polohy agenta.



Obrázok 11: Návrh postupu vyhodnocovania polohy s použitím čiar

Mapovanie čiar

Mapovanie čiar sa bude vykonávať iba dovtedy, pokým sa nepodarí namapovať požadovaný počet bodov na absolútne súradnice ihriska.

Na základe videných objektov môže mapovanie čiar prebehnúť dvoma spôsobmi:

1) Ak agent vidí aj kontrolný bod – primárne sa nemapujú čiary, ktorých súčasťou je daný kontrolný bod, ak nie je dostatočný počet bodov, vykoná sa dodatočne rozpoznávanie vzorov (spôsob 2).
2) Ak agent vidí iba čiary – vykoná sa rozpoznávanie vzorov – t.j. hľadajú sa závislosti medzi čiarami (ako napríklad: stredová čiara je kolmá na okrajovú, ...). Pri mapovaní čiar bude pomáhať tzv. rozpoznávanie vzorov.

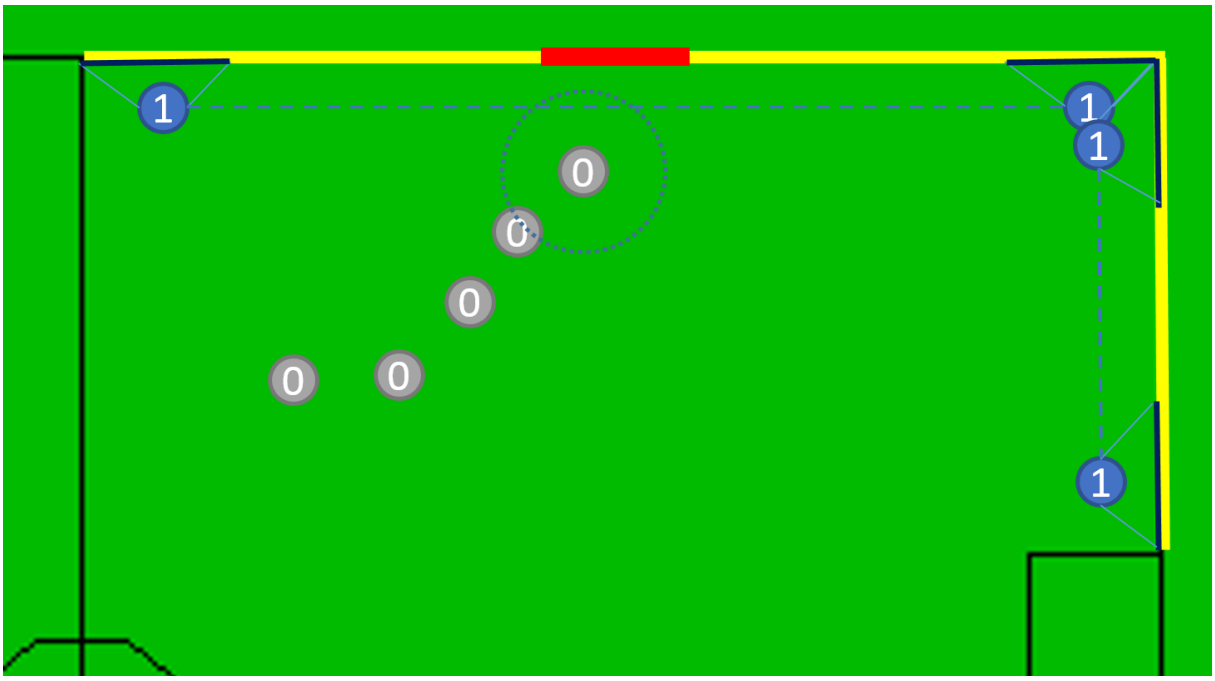
Postup jedna čiara

Ak agent vidí iba jednu čiaru (viď obrázok 12 – červená čiara), môže sa jednať iba o okrajové čiary ihriska.

1. Na základe histórie (viď Obrázok 12 - šedé kruhy s číslom 0) sa identifikuje kde sa naposledy agent nachádzal – v ktorej časti ihriska (viď kapitola 5.8. Určenie časti ihriska,

v ktorej sa agent nachádza), tým sa eliminuje počet čiar, ktoré mohol agent vidieť na ihrisku na jednu štvrtinu (viď Obrázok 12 – žlté čiary).

2. Z relatívnych súradníc agenta vieme, v akej vzdialenosti od čiary sa nachádza a vzdialenosť ku koncovým bodom časti čiary, ktoré vidí. Pre obe čiary vieme určiť koncové/hraničné body, kde by sa mohol agent nachádzať (o x metrov doľava/doprava od stredovej čiary a y metrov od čiary k stredu) (viď Obrázok 12 - modré kruhy s číslom 1).
3. Spojením hraničných bodov vzniknú dve úsečky, kde by sa mohol nachádzať agent, aby videl danú časť čiary (viď Obrázok 12 – modré prerušované čiary).
4. Na základe poslednej polohy, času, (taktiky, akcelerometra,...) ,... sa dá odhadnúť, akú asi vzdialenosť prešiel agent (priesečníkom úsečiek a vzdialenosti získame body, kde by sa mohol nachádzať) (viď Obrázok 12 - modrý prerušovaný kruh)
5. Na základe histórie sa pre oba priesečníky vypočíta pravdepodobnosť, s akou by sa mohol nachádzať na danom bode. Bod s väčšou pravdepodobnosťou je bod, kde sa nachádza agent.



Obrázok 12: Vizualizácia postupu aproximácie polohy agenta, ak vidí jednu čiaru

6.1. Vytvorenie histórie polohy agenta

Jozef Blažíček, Maryna Kovalenko

Pri implementácii boli upravené dve triedy, jedna vytvorená a upravený jeden súbor z projektu Jim.

Vytvorenie triedy PositionHistory.java, ktorá uchováva historické informácie o polohe agenta a metadáta.

Do súboru settings.properties bola pridaná hodnota POSITION_HISTORY_SIZE = 1000 reprezentujúca maximálnu veľkosť histórie.

Trieda sk.fiit.jim.Settings bola rozšírená o načítanie tohto nastavenia.

Z triedy sk.fiit.jim.agent.models.AgentModel bola upravená metóda processNewServerMessage(). Do metódy bolo pridané vymazanie histórie a jej rozšírenie.

Vymazanie histórie sa vykoná na začiatku metódy, ak stav/mód hry (prijatý zo servera) nie je:

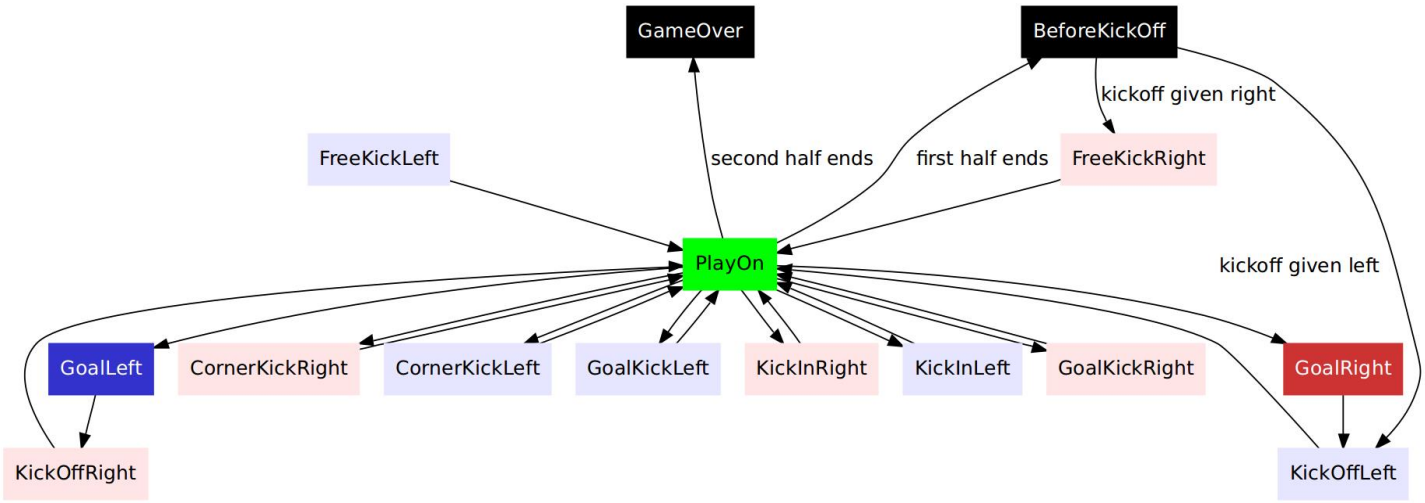
- PLAY_ON
- KICK_OFF_RIGHT

- KICK_OFF_LEFT
- FREE_KICK_LEFT
- FREE_KICK_RIGHT

Aktualizácia (rozšírenie) histórie sa vykoná, ak agent dostal informácie o vlajkách a/alebo čiarach zo servera. Ak veľkosť histórie presiahne POSITION_HISTORY_SIZE, odstráni sa najstarší zo záznamov.

Počas jedného polčasu (300s) veľkosť histórie dosahuje 14 927 (po prvom polčase) – 15 001 (po druhom polčase) položiek.

6.1.1. Správy zo servera – Stav hry



Obrázok 13: Jednotlivé stavy hry a vzťahy medzi nimi[5]

(GS (t <čas_hry>) (pm <playmode>))

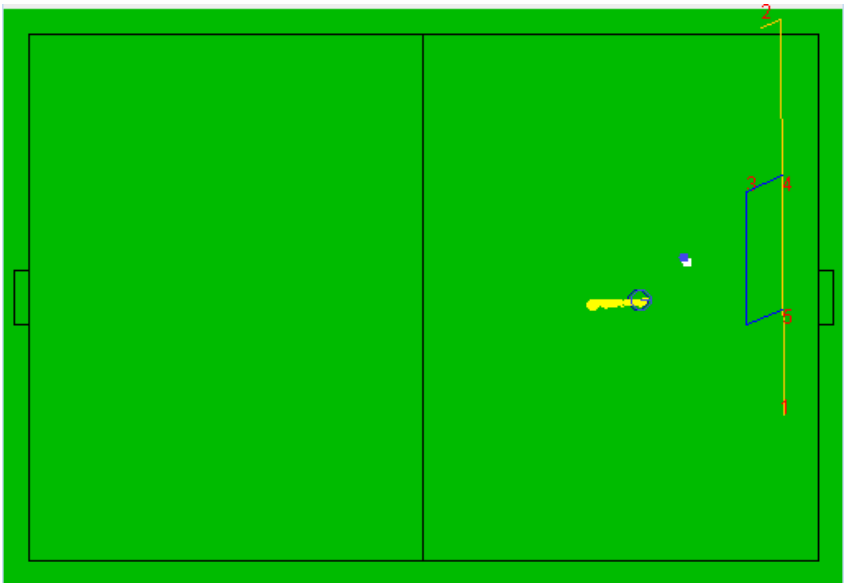
S-výraz	PlayMode	Význam / kedy nastane
BeforeKickOff	BEFORE_KICK_OFF	Pred začiatkom polčasu
PlayOn	PLAY_ON	Počas hry (ak nenastala iná z udalostí)
KickOffLeft	KICK_OFF_LEFT	
KickOffRight	KICK_OFF_RIGHT	
KickInLeft	KICK_IN_LEFT	
KickInRight	KICK_IN_RIGHT	
CornetKickLeft	CORNER_KICK_LEFT	
CornerKickRight	CORNER_KICK_RIGHT	
GoalKickLeft	GOAL_KICK_LEFT	
GoalKickRight	GOAL_KICK_RIGHT	
GameOver	GAME_OVER	Po 600s (koniec hry)
GoalLeft	GOAL_LEFT	
GoalRight	GOAL_RIGHT	
FreeKickLeft	FREE_KICK_LEFT	
FreeKickRight	FREE_KICK_RIGHT	

Tabuľka 5: Možné stavy hry

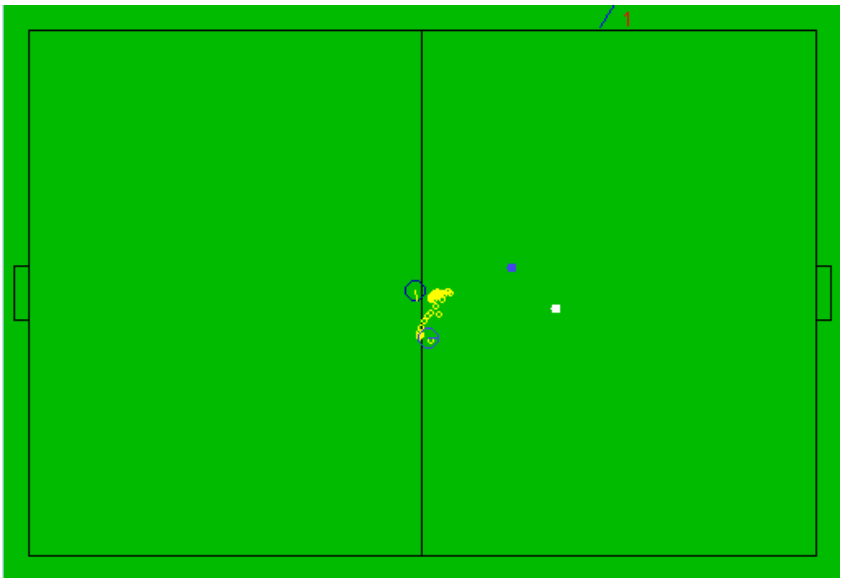
[5] http://simspark.sourceforge.net/wiki/index.php/Play_Modes

6.1.2. Vykresľovanie histórie polohy agenta

Matúš Ivanoc



Obrázok 14: Agent sa pohybuje smerom k bráne



Obrázok 15: Vykresľovanie 250 položiek so silným zašumením

Pri implementácii vykresľovania histórie boli doplnené triedy *GameView* o cyklus vykresľovania a do tried *AgentModel* a *PositionHistory* (viď kapitola 6.1. Vytvorenie historie polohy agenta) boli doplnené metódy get, set pre prístup k atribútom.

sk.fiit.testframework.ui.GameView

V metóde *paintComponent()* sa doplnil get na históriu pozície agenta *wholeHistory* ktorá je reprezentovaná Listom. Následne sa vytvorí sub list posledných 250 pozícií agenta – *subHistory*. Ten sa cez for cyklus vykresľuje na mapu testFrameworku.

sk.fiit.jim.agent.models.AgentModel

Do triedy sa doplnili metódy get a set pre parameter *positionHistory.*

sk.fiit.jim.agent.models.PositionHistory

Doplnené metódy get a set pre parametre triedy.

6.2. Identifikácia vzorov čiar na ihrisku

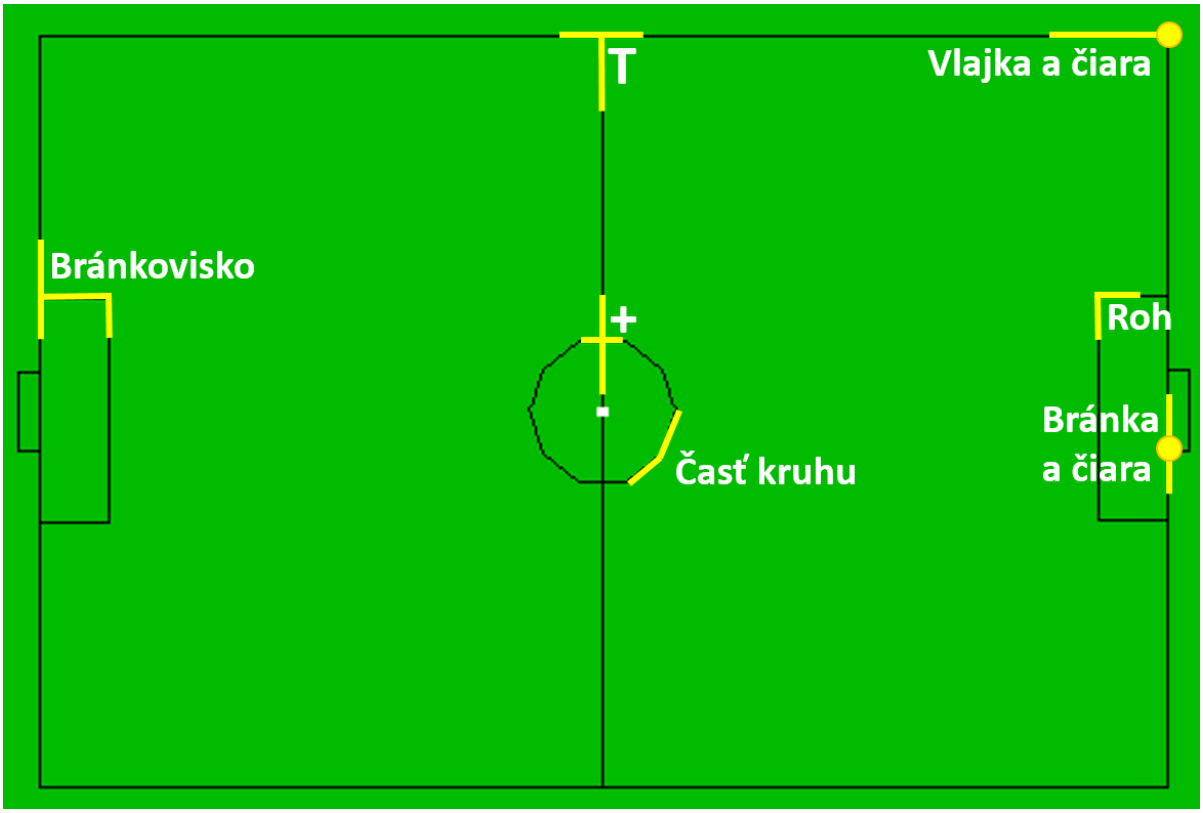
Jozef Blažíček

Na ihrisku sme identifikovali niekoľko vzorov:

- „T“
- „+“
- „Bránkovisko“
- „Roh“
- „Časť kruhu“
- „Vlajka a čiara“
- „Bránka a čiara“

Trieda sk.fiit.jim.agent.models.LinePatternRecognition implementuje funkcie na rozpoznávanie týchto vzorov. Vzory budú použité na mapovanie prijatých informácií o čiarach na čiary ihriska.

Rozdiel medzi „vlajkou a čiarou“ a „bránkou a čiarou“ je v tom, že v prvom prípade sa jedná o zistenie, či dva body (jeden bod z dvoma rôznymi zašumeniami) sú tým istým bodom a pri bránke sa jedná o bod nad čiarou.



Obrázok 16: Vzory čiar na ihrisku

6.2.1. Spoločný bod

Funkcia zistí, či čiara a bod by mohli byť tým istým bodom (jeden bod z dvoma rôznymi zašumeniami). Najskôr sa vypočíta maximálna vzdialenosť, v ktorej by sa mohol bod nachádzať. Maximálna vzdialenosť je vzdialenosť medzi bodom a tým istým bodom otočeným o 4 stupne v oboch smeroch (horizontálne aj vertikálne), t.j. medzi bodom a = (r, phi, theta) a‘ = (r, phi + 4°, theta + 4°). 4°

je maximálny možný rozdiel medzi bodmi (šum je z intervalu $\langle -2°, 2° \rangle$). Ak oba body sa nachádzajú v maximálnej vzdialenosti, funkcia vráti bod čiary, ktorý je bližšie k danému bodu.

Na rovnakom princípe funguje funkcia pre zistenie, či dve čiary majú spoločný bod. Ak je viac možností, vrátia sa ako body tie, ktoré sú k sebe najbližšie.

6.2.2. Bránkovisko

Funkcia *isGoalBox()* využíva dve funkcie – isT() a isCorner(). Na základe ich výsledkov zostaví zoradenie bodov. Funkcia celkovo vracia 24 kombinácií. (funkcia počíta aj s orientáciou čiar (t.j. či spoločným bodom je prvý alebo druhý bod čiary.))

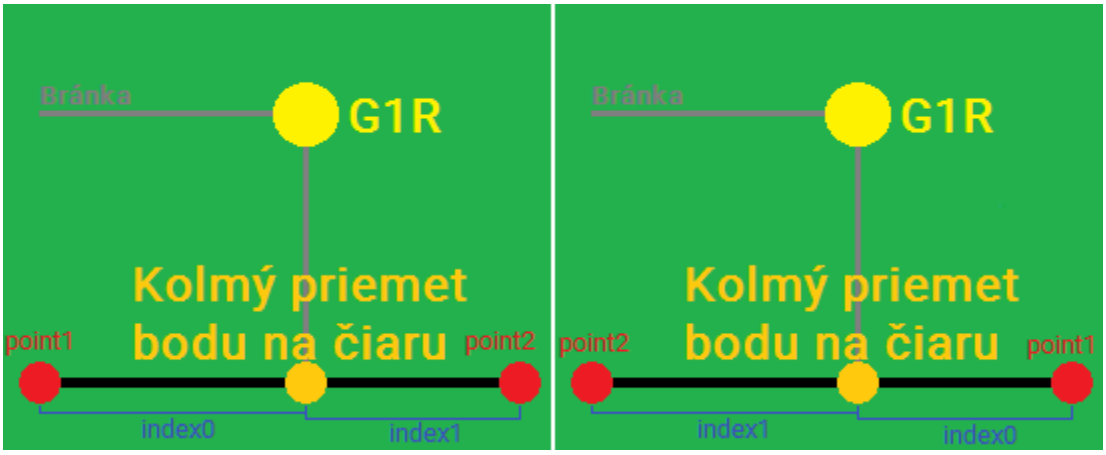
6.2.3. Roh a Časť Kruhu

Funkcie *isCorner()* pre roh a *areCircleLines()* pre kruhové čiary najskôr zistia, či a ktoré body sú spoločné body čiar. Následne sa vypočíta uhol medzi čiarami.

Ak dve čiary majú spoločný koncový bod, môže sa jednať iba o roh alebo o časť kruhu. (T sa nachádza v strede čiary, nemajú spoločný koncový bod).

Roh má 90° a časť kruhu 144°. Hranica akceptovateľnosti (spôsobenej zašumením je 117° (v polovici medzi 90 a 144)). Ak je uhol menší ako 117 -> považuje sa za roh, ak je väčší -> považuje sa za časť kruhu.

6.2.4. Bránka a čiara



Obrázok 17: Návratové hodnoty funkcie, zisťujúcej vzťah medzi bránkovým bodom a čiarou

Funkcia *goalAndLine()* Získa výšku, v akej by sa mal vrch bránky nachádzať od čiary (z FixedObject pre bránku). Vypočíta maximálnu odchýlku bodu. Vzdialenosť bodu od čiary a ak táto vzdialenosť je z intervalu (Výška - chyba) až (výška + chyba). Vracia null, ak nie je daná čiara koncová čiara, inak vráti pole dvoch hodnôt – vzdialenosti od kolmého bránkového bodu k videným koncom čiary (Viď Obrázok 17).

6.3. Využívanie Akcelerometra

Agent dostáva v každej správe informácie o zrýchlení v každom smere. Údaje sú nezašumené, ale nepresnosť je v zaokrúhlení na dve desatinné miesta. Pre rovnomerne zrýchlený pohyb platia vzťahy:

$$s = v_0 t + \frac{1}{2} a t^2$$

$$v = \frac{s}{t}$$

$$v = v_0 t + at$$

Zrýchlenie na z-ovej osi obsahuje gravitačné zrýchlenie ($g = 9{,}81 \frac{m}{s^2}$)

6.3.1. Aktualizovanie polohy s použitím zrýchlenia

Vzorec pre výpočet vzdialenosti s použitím rýchlosti, vypočítanej v predchádzajúcej iterácii produkuje veľké hodnoty.

Nie je jasné, či osi akcelerometra sú natočené spolu s agentom, alebo sú totožné s osami ihriska.

Tabuľky obsahujú akumulovanú absolútnu chybu vo vyhodnocovaní polohy troma spôsobmi: iba na základe videných kontrolných bodov, s úpravou polohy na základe zrýchlenia (v_0 je nulové) a na základe „natočeného" zrýchlenia (v závislosti od natočenia agenta). Čas, počas ktorého sa hodnoty zaznamenávali je 300s (jeden polčas hry).

Všetky tri typy sa počítali v rámci toho istého spustenia agenta (poloha sa v každom časovom okamihu vypočítavala všetkými troma spôsobmi).

So zrýchlením: $x = x_{i-1} - \frac{1}{2} a t^2$

S natočením:

1. Odpočítanie gravitácie od z-tovej osi.
2. Otočenie osí na základe natočenia agenta.
3. Výpočet vzdialenosti na základe vzorca „So zrýchlením".

	X	**Y**	**Z**
Videné	5 699,149251	**3 597,605437**	**17 744,609731**
So zrýchlením	6 151,088104	4 294,820116	18 336,837249
S natočeným zrýchlením	**5 669,217741**	3 774,310988	17 784,961617
Videné	**12 787,652371**	14 909,751625	20 652,335692
So zrýchlením	13 365,718871	**14 490,82049**	**20 620,826774**
S natočeným zrýchlením	24 117,199825	22 517,594629	25 436,483558
Videné	15 120,956122	11 271,359534	**14 404,390739**
So zrýchlením	14 636,982574	**7 719,684383**	16 392,732415
S natočeným zrýchlením	**14 571,318556**	7 874,908774	16 473,008237

Tabuľka 6: Nameraná chyba použitím troch spôsobov aktualizácie polohy agenta

S využitím zrýchlenia pri výpočte polohy sa nezlepšila presnosť určovania polohy (v niektorých prípadoch sa zhoršila).

Nakoľko vyhodnocovanie polohy veľmi závisí od vyhodnotenia na základe videných objektov, akcelerometer (zrýchlenie) nedokáže veľmi túto hodnotu zlepšiť.

6.3.2. História

Do histórie bola pridaná položka uchovávajúca vzdialenosť vypočítanú z akcelerometra. Táto hodnota sa aktualizuje v prípade, ak sa nebude do histórie pridávať nová položka.

6.4. Refaktorovanie vyhodnocovania polohy agenta

Jozef Blažíček

Pri vyhodnocovaní polohy agenta sa používa trieda FixedObject a jej metódy na získanie absolútnej polohy kontrolného bodu. Tento objekt sa používa ako kľúč pri mapovaní relatívnych súradníc bodov. Za účelom zachovania čo najväčšieho množstva kódu sme toto vyhodnocovanie refaktorovali. Z mapy FixedObject a relatívna poloha sme spravili zoznam objektov uchovávajúcich relatívnu aj absolútnu polohu bodu (*MappedPoint*). Tento zoznam sa bude rozširovať o namapované body čiar.

6.5. Načítavanie súradníc zo súboru

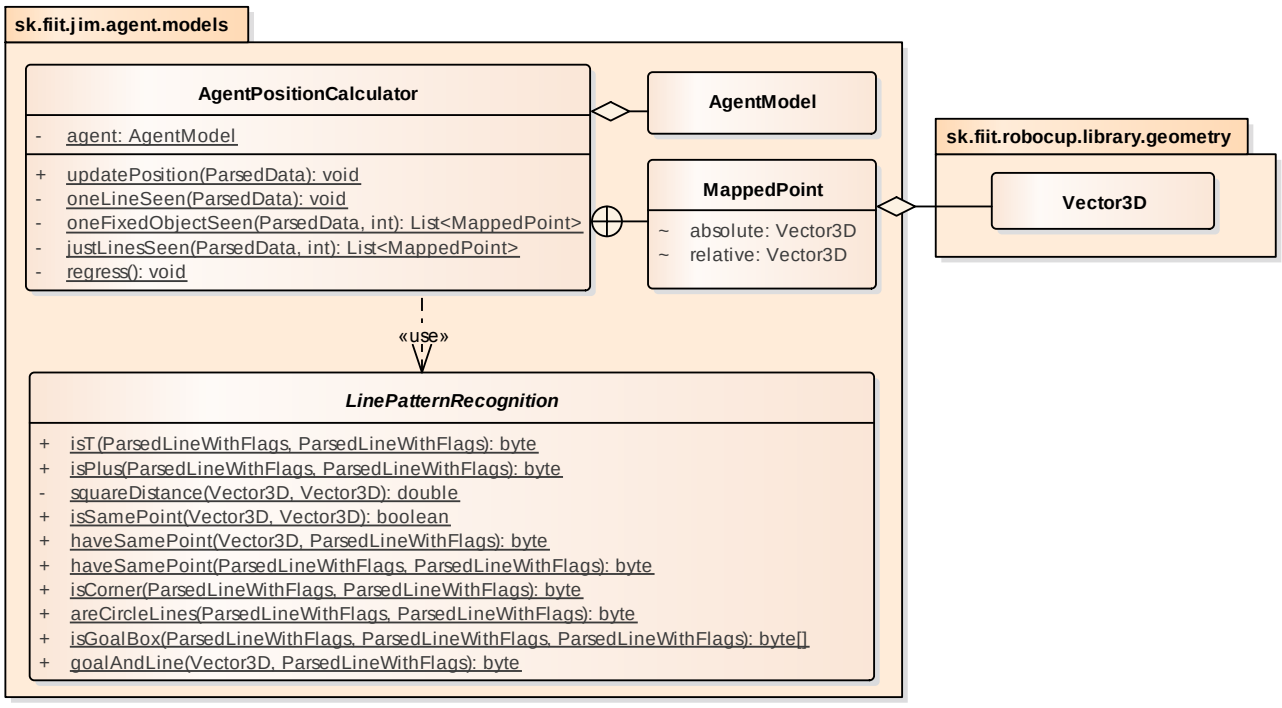
Veľkosť ihriska, stredového kruhu a bránkoviska sa môže zmeniť/líšiť v závislosti od verzie servera (zmena bola vo verziách 0.6.2, 0.6.5 a 0.6.7). Aby neboli potrebné implementačné zmeny a zásahy do kódu kvôli týmto zmenám, tieto informácie sme presunuli a budú sa načítavať zo súboru s nastaveniami (settings.properties).

6.6. Mapovanie bodov čiar

oneLineSeen(ParsedData) – funkcia určí polohu agenta, ak vidí iba jednu čiaru,

oneFixedObjectSeen(ParsedData,int) – funkcia vráti n – namapovaných bodov z čiar, ak agent vidí jeden kontrolný bod a čiary,

onlyLinesSeen*(ParsedData,int)* - funkcia vráti n – namapovaných bodov z čiar, ak agent vidí niekoľko čiar.



Obrázok 18: UML diagram tried - triedy vystupujúce pri vyhodnocovaní polohy agenta

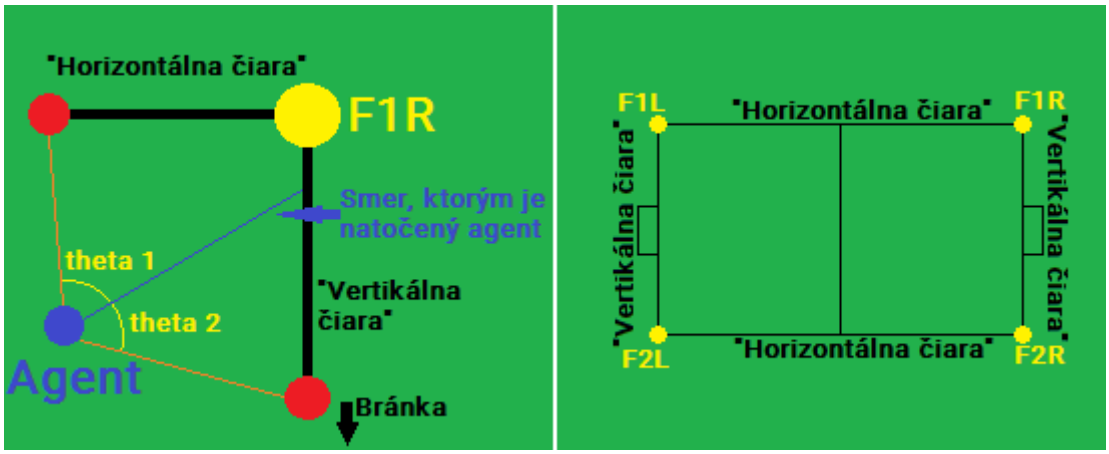
6.6.1. Ak agent vidí vlajku

Ak agent vidí vlajku, pravdepodobne vidí aj ďalšie (minimálne) dve čiary.

1. Zistenie, či čiara, ktorú agent vidí, má spoločný bod s vlajkou (vzor čiar „čiara a bod“)

Informácia o tom, že daná čiara má spoločný bod, nepostačuje na určenie, o ktorú čiaru sa jedná (môžu byť dve).

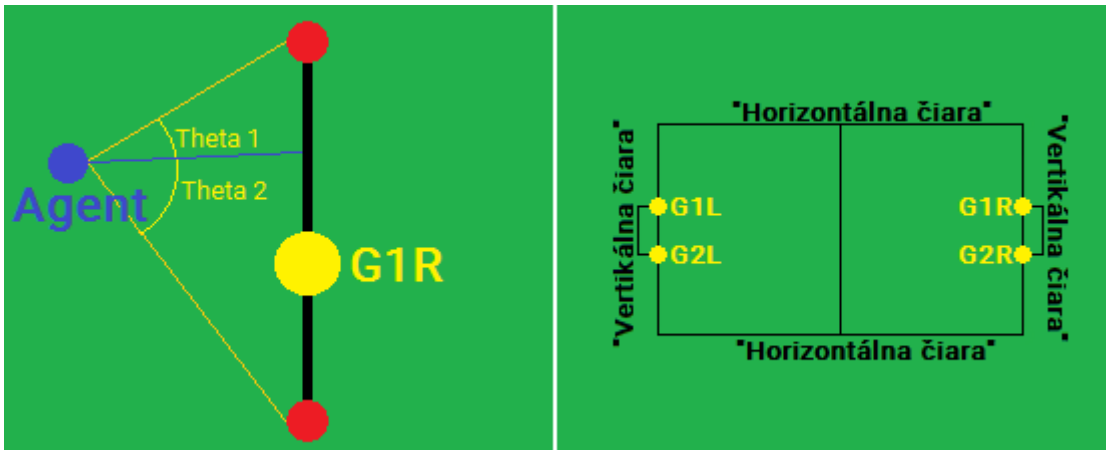
2. Ak agent vidí F1R – „horizontálnu čiaru“, vidí druhý bod čiary pod väčším uhlom theta (theta 1) ako „vertikálnu čiaru“ (theta 2).
   Uhly theta1 a theta2 sú theta uhly ku koncovým bodom čiary.
   - Ak agent vidí F1R a theta1 > theta2, potom bod, ktorý agent vidí pod uhlom theta1, je bodom z „horizontálnej čiary“ a theta2 je bodom z vertikálnej čiary.
   - Ak agent vidí F2R a theta1 > theta2, potom bod, ktorý agent vidí pod uhlom theta1, je bodom z „vertikálnej čiary“ a theta2 je bodom z „horizontálnej čiary“.
   - Ak agent vidí F1L a theta1 > theta2, potom bod, ktorý agent vidí pod uhlom theta1, je bodom z „vertikálnej čiary“ a theta2 je bodom z „horizontálnej čiary“.
   - Ak agent vidí F2L a theta1 > theta2, potom bod, ktorý agent vidí pod uhlom theta1, je bodom z „horizontálnej čiary“ a theta2 je bodom z „vertikálnej čiary“.
3. Výpočet dĺžky videnej časti čiary.
4. Odpočítanie/pripočítanie dĺžky čiary k súradnici kontrolného bodu v závislosti od čiary a vlajky.



Obrázok 19: Vzťah medzi uhlami, pod akými agent vidí koncové body čiary (vľavo) a horizontálne / vertikálne čiary ihriska (vpravo)

6.6.2. Ak agent vidí bránku

1. Zistenie, či čiara, ktorú agent vidí, je koncová „vertikálna čiara“ ihriska.
2. Výpočet vzdialeností od priemetu bodu ku koncom čiary, ktoré vidí agent.
3. Výpočet absolútnych súradníc bodov – pripočítanie / odpočítanie vzdialenosti od y-ovej súradnice bránkového bodu v závislosti od uhla, pod akým agent vidí daný bod.



Obrázok 20: Vzťah medzi uhlami, pod akými agent vidí koncové body čiary.

7. TDD

Jozef Blažíček

Testy boli vytvorené na otestovanie troch oblastí:

1. Vyhodnocovanie polohy Agenta – v závislosti od prijatých informácii, agent vyhodnocuje svoju polohu rôznymi spôsobmi:
    a. Ak agent vidí 3 kontrolné body z jednej strany ihriska – vie si vypočítať akým smerom je otočený a aj svoju polohu na ihrisku.
    b. Ak agent vidí aspoň dva kontrolné body na ihrisku – vie si celkom presne vypočítať svoju polohu.
    c. Ak agent vidí iba jeden kontrolný bod (v takom prípade vidí aspoň jednu ďalšiu čiaru) – prebehne mapovanie čiar a z nich sa získa ďalší bod, na základe ktorého sa určí poloha agenta.
    d. Ak agent vidí niekoľko čiar (aspoň dve) – na základe histórie prebehne mapovanie čiar a výpočet polohy.
    e. Ak agent vidí iba jednu čiaru – špecifický prípad aproximácie polohy.
2. Mapovanie vzorov
3. Prepočet relatívnej polohy objektov

7.1. Generátor videných objektov

Nakoľko kvôli šumu nie je možné, aby agent úplne presne určil svoju polohu, pre vstupy testovaní bol vytvorený generátor videných objektov, ktorý produkuje nezašumené informácie o videných objektoch. S týmito informáciami by mal agent presne určiť svoju polohu (keďže výpočty prebiehajú v radiánoch a hodnota konštanty PI tiež nie je presná, pri výsledku sa akceptuje tolerancia 0.01).

Generátor sa skladá z dvoch tried:

- sk.fiit.robocup.library.generator.SeenGenerator.java
- sk.fiit.robocup.library.generator.Geometry.java

Postup generovania

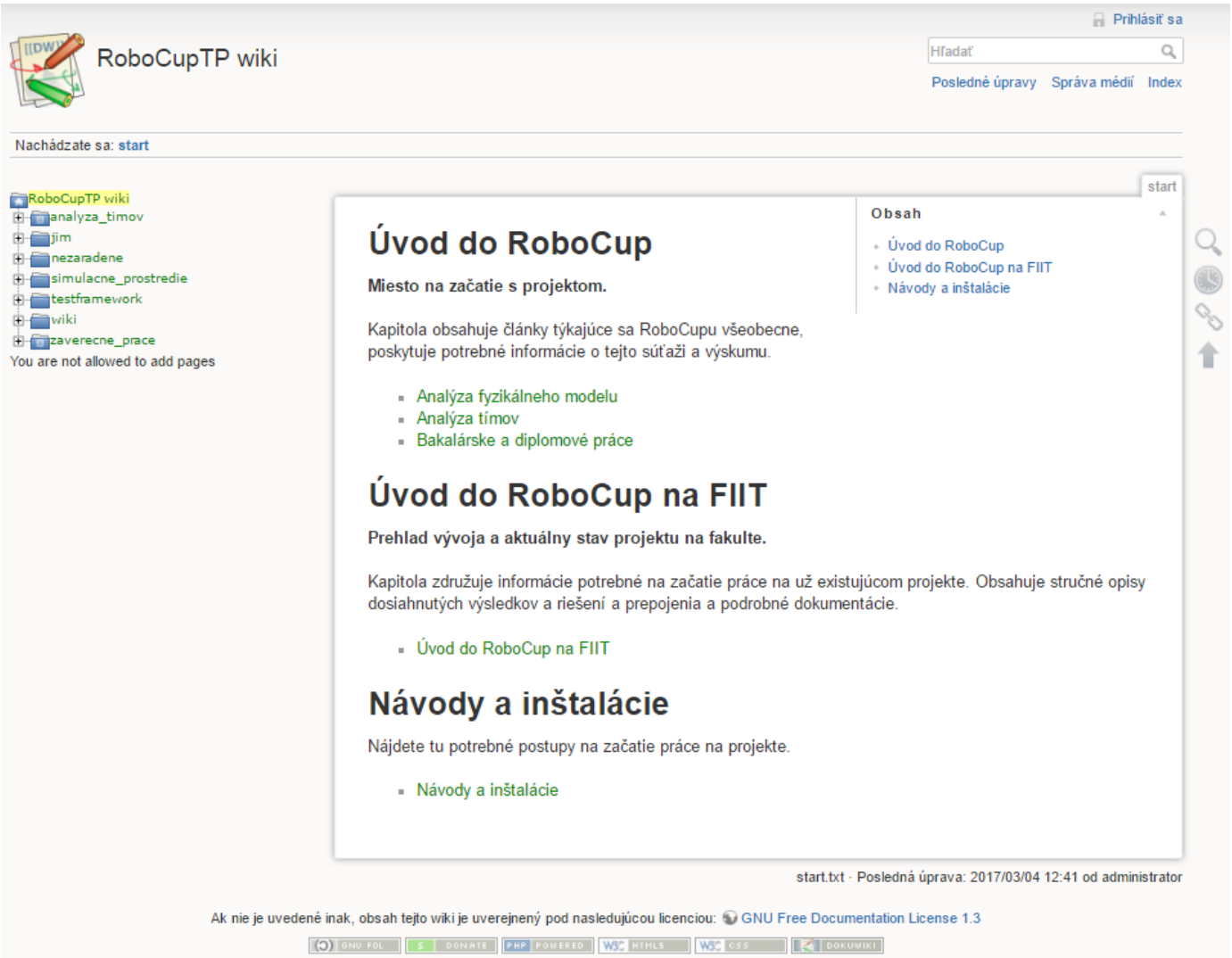
1. Inicializácia súradníc čiar a kontrolných bodov (vlajok a bránok).
2. Vytvorenie 4 rovín reprezentujúcich hranice viditeľnosti agenta (roviny sú zostavené tak, že ich normálové vektory smerujú smerom do viditeľnej oblasti agenta).
3. Pre každú plochu:
4. Pre každý bod sa vypočíta uhol medzi normálovým vektorom a vektorom medzi polohou agenta a bodom. Ak je tento uhol -90° až 90°, agent vidí daný bod, inak sa daný bod odstráni zo zoznamu.
5. Pre každú čiaru sa vypočíta uhol medzi normálovým vektorom roviny a jednotlivými koncovými bodmi čiary (rovnako ako pri bodoch sa počíta uhol medzi normálovým vektorom roviny a vektorom z polohy agenta k bodu). Ak agent vidí oba body, pokračuje sa na ďalšiu čiaru, ak nevidí ani jeden bod, čiara sa odstráni zo zoznamu, ak vidí iba jeden bod, vypočíta sa priesečník roviny s čiarou a aktualizuje sa poloha bodu, ktorý agent nevidí.
6. Vytvorí a vráti sa zoznam bodov a čiar, ktoré vidí agent.

8. Wiki

Bc. Jozef Blažíček, Bc. Matúš Ivanoc

8.1. DokuWiki

Bc. Matúš Ivanoc



Obrázok 21: 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.

8.1. 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.

8.2. Aktualizovanie informácií prvá etapa (zimný semester)

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.

8.2.1. Vytvorené články

Analýza využitia informácii o čiarach v zahraničných tímoch *(analyza_timov)*

- Informácie o využívaní informácií o čiarach zahraničnými tímami

Komunikácia agenta so serverom *(jim)*

- Opisuje priebeh komunikácie agenta so serverom.

Správy zo servera *(jim)*

- Opisuje správy prichádzajúce zo servera.

Zistenie, v ktorej štvrti sa nachádza agent *(jim)*

- Opisuje rozdelenie ihriska na časti a spôsob identifikácia.

Parsovanie čiar
(jim / vylepsenie_zistovania_polohy_hraca_na_zaklade_videnia_ciar)

- 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)

- Opisuje princíp prepočtu relatívnej pozície na globálne súradnice, taktiež opisuje vytvorenú triedu.

Zisťovanie T priesečníkov čiar *(testframework / ciary)*

- Opisuje hľadanie priesečníkov čiar.

Vykresľovanie čiar *(testframework / ciary)*

- Opisuje vykresľovanie čiar v TestFrameworku.

8.2.2. Aktualizované články

Analýza fyzikálneho modelu (jim)

- Aktualizovanie informácií o ihrisku
- Aktualizovanie informácií (tabuľky) nastavenia kĺbov (maximálnych a minimálnych hodnôt) agenta.

8.3. Aktualizovanie informácií druhá etapa (letný semester)

Jozef Blažíček

8.3.1. Vytvorené články

AgentModel.java *(jim/agentmodel)*

- Obsahuje informácie o triede AgentModel.

Analýza matematiky a logiky
(jim/analyza_matematiky_a_logiky)
(nezaradene/analyza_matematiky_a_logiky)

- Výsledky analýzy matematiky a logiky projektu, analýza bola vykonaná začiatkom semestra.

História polohy agenta *(jim/historia_polohy_agenta)*

- Opisuje, kde sa nachádza história, kedy sa aktualizuje a vymaže
- Približuje triedu sk.fiit.jim.agent.models.PositionHistory

Generátor videných objektov *(nezaradene/generator_vydenych_objektov)*

- Vysvetľuje princíp a použitie generátora videných objektov (viď kapitola 7.1. Generátor videných objektov)

9.Záver

Bc. Jozef Blažíček

9.1. 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)

9.1.1. 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.

9.1.2. Wiki

Prešli sme z MediaWIki na DokuWiki, vytvorili štruktúru článkov (ktorá doteraz chýbala), aktualizovali informácie a doplnili naše výsledky (viď kapitola 9.wiki).

9.2. Druhá etapa (Letný semester)

Na začiatku semestra sme na základe získaných informácií zostavili postup (viď kapitola 6. čiary – druhá etapa) ako by mohlo prebiehať vyhodnocovanie polohy na základe čiar a vykonali sme analýzu matematiky a logiky častí projektu, ktoré vyzerali byť dôležité pre daný cieľ (viď Príloha A Jim a Príloha B RoboCupLibrary). Na základe postupu vyhodnocovania sme identifikovali niekoľko scenárov a pripravili unit testy (viď kapitola 7. TDD).

9.2.1. TDD

Začiatkom semestra sme implementovali niekoľko testov. Prípady pokryté testami zahŕňali rôzne situácie vyhodnocovania polohy agenta (rôzne vstupy videných objektov), testovanie mapovania čiar, prepočtu relatívnej polohy bodov na absolútne (viď kapitola 7. TDD).

9.2.2. Čiary

Na základe nadobudnutých poznatkov z predchádzajúcej etapy (Zimný semester), sme vytvorili plán, ako zakomponovať informácie o čiarach do vyhodnocovania polohy agenta. Následne sme tento plán implementovali.

Najskôr sme vytvorili históriu polohy agenta (viď kapitola 6.1. vytvorenie histórie polohy agenta).

Ďalším krokom bolo vytvorenie rozpoznávania vzorov čiar na ihrisku. Vzory reprezentujú niektoré vzťahy medzi čiarami a čiarami a bodmi na ihrisku. (viď kapitola 6.2. Identifikácia vzorov čiar na ihrisku). Identifikácia bola implementovaná statickými funkciami. Funkcie vracajú hodnoty, ktoré presne definujú vzťah objektov (čiar a/alebo kontrolných bodov), ktoré dostali ako argumenty.

Pokúsili sme sa využiť informácie z akcelerometra pre priebežné aktualizovanie polohy agenta (nakoľko tieto informácie dostáva agent v každej správe a neobsahujú šum). Takáto zmena nemala žiaden významný efekt na určenie polohy agenta., preto informácie z akceleromtra iba ukladáme do histórie polohy agenta (do posledného záznamu, vzdialenosť prepočítanú zo zrýchlenia, s tým že pri výpočte $v_0 = 0$), v prípade ak nepridal novú položku do histórie (viď kapitola 6.3. Využívanie Akcelerometra).

Za účelmi rozšírenia vyhodnocovania polohy o informácie z čiar, sme refaktorovali vyhodnocovanie polohy a doplnili triedu reprezentujúcu relatívnu aj absolútnu polohu bodu (viď kapitola 6.4. Refaktorovanie vyhodnocovania polohy agenta). Nakoľko sa rozmery ihriska môžu zmeniť s novou verziou servera, informácie o polohe kontrolných bodov, polomeru stredového kruhu a veľkosti bránkoviska sme tieto informácie presunuli do súboru settings.properties, odkiaľ sa budú načítavať a v prípade potreby, je ich možné zmeniť bez zásahu do kódu (viď. Kapitola 6.5. Načítavanie súradníc zo súboru).

Posledným krokom bola implementácia mapovania čiar (viď kapitola 6.6. Mapovanie bodov čiar) a vyhodnocovanie polohy na základe jednej čiary (postup viď kapitola 6. čiary – druhá etapa).

9.2.3. Wiki

V letnom semestri sme pokračovali s aktualizáciou wiki (viď kapitola 9.wiki).

10. Zoznam príloh

Príloha	Názov	Opis prílohy
Príloha A	Jim	Analýza vybraných tried z projektu Jim
Príloha B	RoboCupLibrary	Analýza vybraných tried z projektu RoboCupLibrary
Príloha C	Čiary zo servera	Poradie čiar, v akom ich posiela agentovi server

Tabuľka 7: Zoznam príloh

Príloha A:
Jim

Jozef Blažíček, Maryna Kovalenko, Miloš Štefčák

A.1. sk.fiit.jim.agent.models.AgentModel

Jozef Blažíček

Typ: *public*
Implementované rozhrania:
sk.fiit.jim.agent.parsing.ParsedDataObserver
java.io.Serializable
Konštanty:
AgentModel instance
...
Premenné:
double rotationX
double rotationY
double rotationZ
Natočenie agenta v závislosti od jednotlivých osí
Vector3D position
Pozícia agenta na ihrisku
...

Trieda uchováva informácie o aktuálnom stave agenta.

A.1.1. processNewServerMessage

Typ: public
Parametre:
ParsedData data
Informácie získané zo servera

Spracovanie informácií prijatých zo servera:

1. ID hráča
2. Na ktorej strane hrá agent
3. Bola vykonaná zmena strán alebo polčas
4. Aktualizovanie nastavenia otočných kĺbov (updateJointPosition)
5. Nastavenie natočenia agenta z dát z gyroskopu (adjustRotationFor)
6. Nastavenie pozície na základe informácií z akcelerometra (adjustPositionFor)
7. Aktualizovanie natočenia agenta na základe videných bodov (updateRotation a volanie AgentRotationCalculator.updateRotations)
8. Aktualizovanie polohy na základe videných bodov (updatePosition a volanie AgentPositionCalculator.updatePosition)
9. Výpočty pre ZMP (Zero moment point)
10. Kedy naposledy bol videný kontrolný bod.

A.1.2. adjustRotationsFor

Typ: *private*
Parametre:
Vector3D gyroscope
Informácia z gyroskopu

Aktualizuje natočenie agenta na základe informácií z gyroskopu.

A.2. sk...jim.agent.models.AgentPositionCalculator

Miloš Štefčák

Typ triedy: public
Konštanty:
private static final AgentModel agent = AgentModel.getInstance();
informácia o stave agenta
private static final int BAD_POSITION_KOEF = 1;
koeficient zlej pozície agenta, ktorý sa používa pri regresii
private static final int MAX_BAD_POSITIONS = 100;
maximálny koeficient zlej pozície agenta
private static final boolean USING_REGRESSION = false;
konštanta, ktorá určuje, či sa bude používať regresia alebo nie
private static final int MAX_LAST_POSITIONS = 21;
konštanta, ktorá určuje maximálny počet zapamätaných predchádzajúcich pozícií
Premenné:
private static Logger LOG
private static int badPositionCounter
počítadlo pre zlé pozície

Táto trieda sa stará o aktualizovanie polohy agenta. Volá sa v triede „AgentModel“ pri každej novej správe zo serveru. Volá sa v metóde „processNewServerMessage“ a ako vstupný parameter dostáva „data“ typu „ParsedData“.

A.2.1. updatePosition

public static
Návratová hodnota: void
Parametre:
ParsedData data
parsované dáta zo serveru

Metóda slúži na nastavenie agentovej pozície. Ak agent nevidí žiadne fixné objekty, tak sa metóda ukončí. Inak sa prechádzajú všetky „flagy“ a následne podľa toho sa určí pozícia. V tejto metóde sa dá použiť aj regresia, ktorá využíva aj minulé pozície agenta.

A.2.2. regress

private static
Návratová hodnota: *void*

Táto metóda slúži na vypočítanie regresie. Regresia sa ráta v dvoch krokoch samostatne pre x-ovú súradnicu a následne pre y-ovú súradnicu.

1.krok:
Inicializacia:

```
int n = 0;
double[] x = new double[MAXN];
double[] y = new double[MAXN];
double sumx = 0.0;
double sumy = 0.0;
```

Počítanie:

```
for (Vector3D pos : agent.lastPositions) {
    x[n] = pos.getX();
    y[n] = n;
    sumx += x[n];
    sumy += y[n];
    n++;
}
double xbar = sumx / n;
double ybar = sumy / n;
```

2.krok:

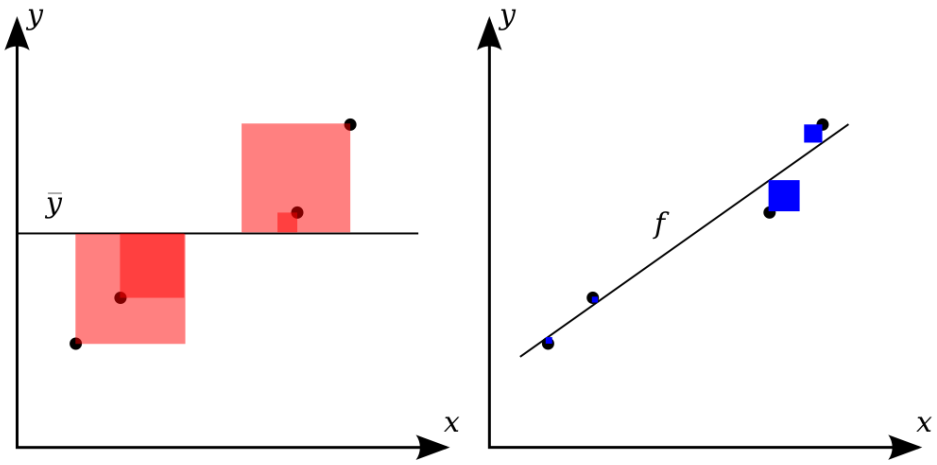
```
double xxbar = 0.0;
double xybar = 0.0;

for (int i = 0; i < n; i++) {
    xxbar += (x[i] - xbar) * (x[i] - xbar);
    xybar += (x[i] - xbar) * (y[i] - ybar);
}
double beta1 = xybar / xxbar;
double beta0 = ybar - beta1 * xbar;
double regresX =  beta1*(n-1)+beta0;
```

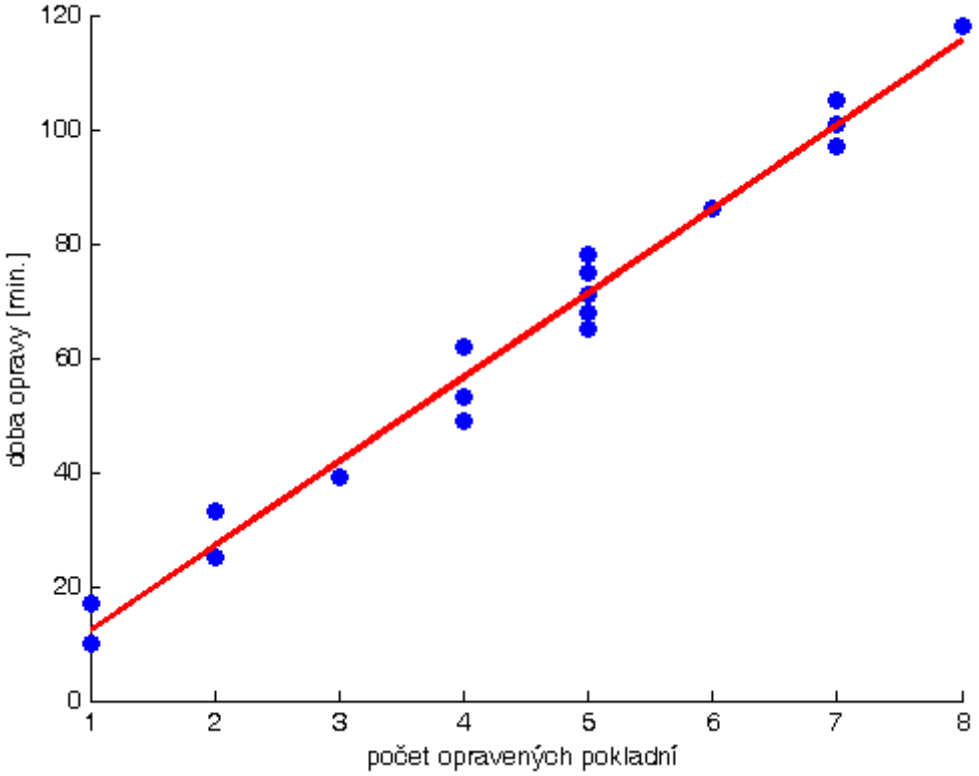
To iste sa vyráta aj pre y-ovú súradnicu a následne sa agentovi nastaví pozícia s vyrátanými parametrami.

```
agent.position = agent.position.setX(regresX).setY(regresY);
```

Princíp regresie: regresia slúži na predpovedanie ďalšej hodnoty, ktorá je úzko spätá s predchádzajúcimi hodnotami. To znamená, ak mám nejaký zoznam hodnôt, určí sa priemer týchto hodnôt a ďalšia hodnota sa vyráta z priemernej hodnoty. Rátanie je rozdelené do menších štvorcov a snažíme sa v konečnom dôsledku dostať „nejakú priamku“ – trajektóriu pohybu hráča.



Obrázok 22: lineárna regresia metódou najmenších štvorcov



Obrázok 23: výsledok regresie zobrazených bodov

Odkazy:
https://en.wikipedia.org/wiki/Coefficient_of_determination
http://www.ujfi.fei.stuba.sk/fyzika/navody/regresia-dodatok_20101022.pdf

A.3. sk...jim.agent.models.AgentRotationCalculator

Jozef Blažíček

Konštanty:

private static int FLAGS_TO_COMPUTE_ROTATION = 3

Počet bodov potrebných pre výpočet otočenia agenta

private AgentModel agent

Informácie o aktuálnom stave agenta. (sk.fiit.jim.agent.models.AgentModel)

Trieda je (aspoň by mala byť) určená na výpočet otočenia agenta na základe pevných bodov. Rotácia sa počíta na základe troch bodov.

A.3.1. Konštruktor

Jozef Blažíček

Typ: *public*
Parametre:

AgentModel agent

Aktuálny stav agenta

Vytvorí objekt a nastaví hodnotu konštanty agent na predaný parameter.

A.3.2. updateRotations

Maryna Kovalenko

Typ: *public*
Parametre:

ParsedData data

Táto metóda aktualizuje rotáciu agenta v aktuálnom AgentModel na základe dát zo servera.
Rotácia sa vykoná, ak z jednej strany (sameSideFlags) agent vidi minimálne 3 vlajky.
Vytvoria sa a naplnia sa 2 zoznamy pozícií vlajok (globálnych a vnímaných agentom), a pošlú sa ako argument do následne zavolanej metóde calculateRotation.

A.3.3. getFlagsOfSideWithMoreFlagsSeen

Jozef Blažíček

Typ: *private*
Návratová hodnota: *Map<Vector3D, Vector3D>*
Parametre:

Map<FixedObject, Vector3D> fixedObjects

Videné kontrolné body nemapované na ich globálne súradnice (kľúčom sú globálne hodnoty).

Funkcia vracia vlajky zo strany, na ktorej je ich viac.
Vlajky sú rozdelené do dvoch strán („our“ – ľavá strana ihriska/ záporná x-ová súradnica a their – pravá strana ihriska / kladná x-ová súradnica).
Ak agent vidí menej ako FLAGS_TO_COMPUTE_ROTATION (3) bodov, vráti sa prázdna mapa, inak sa vráti mapa s hodnotami strany, kde je viac videných bodov. Kľúčom je opäť reálna pozícia vlajky.

A.3.4. calculateRotation

Maryna Kovalenko

Typ: private
Parametre:

List<Vector3D> absolute

zoznam absolútnych pozícií vlajok

List<Vector3D> seen

zoznam vnímaných agentom pozícií vlajok

Vypočíta rotáciu agenta na základe 3 vlajok, ktoré vidí agent.

Usporiadajú sa hodnoty absolútnych pozícii vlajok pomocou metódy orderToFormAxes(absolute).

Hodnoty y1,y2 a z1 sa nastavia na základe zoradenia pomocou metódy *orderToFormAxes*

y1,y2 sa používajú na vytvorenie Y-ovej osí

z1, z2 sa používajú na vytvorenie Z-ovej osí

double rotationX - natočenie agenta v závislosti od osi X

double rotationY - natočenie agenta v závislosti od osi Y

double rotationZ - natočenie agenta v závislosti od osi Z

Vector3D	Z_2	$Z_1 - \frac{(Y_2 - Y_1) \times (Z_1 - Y_1) \times (Y_2 - Y_1)}{(Y_2 - Y_1) * (Y_2 - Y_1)}$
Vector3D	Z_{axis}	$Z_1 - Z_2$
Vector3D	Z_{axis}	$-Z_{axis}$ ak absolute.order[2].Z = 0
Vector3D	X_{axes}	$Y_{axes} \times Z_{axes}$
Vector3D	Y_{axes}	$-(Y_2 - Y_1)$
double	rotationX	$\text{asin}(-xAxis.Z)$
double	rotationZ	$atan2(\frac{-xAxis.rotateOverX.Y}{\cos(rotationX)}, \frac{\text{xAxis.rotateOverX.X}}{\cos(\text{rotationX})})$
double	rotationY	$atan2(\frac{-yAxis.rotateOverX.Z}{\cos(rotationX)}, \frac{\text{zAxis.rotateOverX.Z}}{\cos(\text{rotationX})})$

Uhol rotácie musí patriť intervalu $(0, 2\pi)$, čo sa dosahuje pomocou metódy normalize.

A.3.5. orderToFormAxes

Jozef Blažíček

Typ: *private*
Návratová hodnota: *int[]*
Parametre:

List<Vector3D> absolute

Metóda vracia pole troch hodnôt. Určí, ktoré dva body sa použijú na vytvorenie y-ovej (nultá a prvá hodnota poľa) osi a ktorý bod bude použitý na vytvorenie z-tovej osy (kolmej na y-ovú).

prvý.z == druhý.z	prvý.y > druhý.y	0,1,2
	prvý.y <= druhý.y	1,0,2
prvý.z == tretí.z	prvý.y > tretí.y	0,2,1
	prvý.y <= tretí.y	2,0,1
druhý.y > tretí.y		1,2,0
Inak		2,1,0

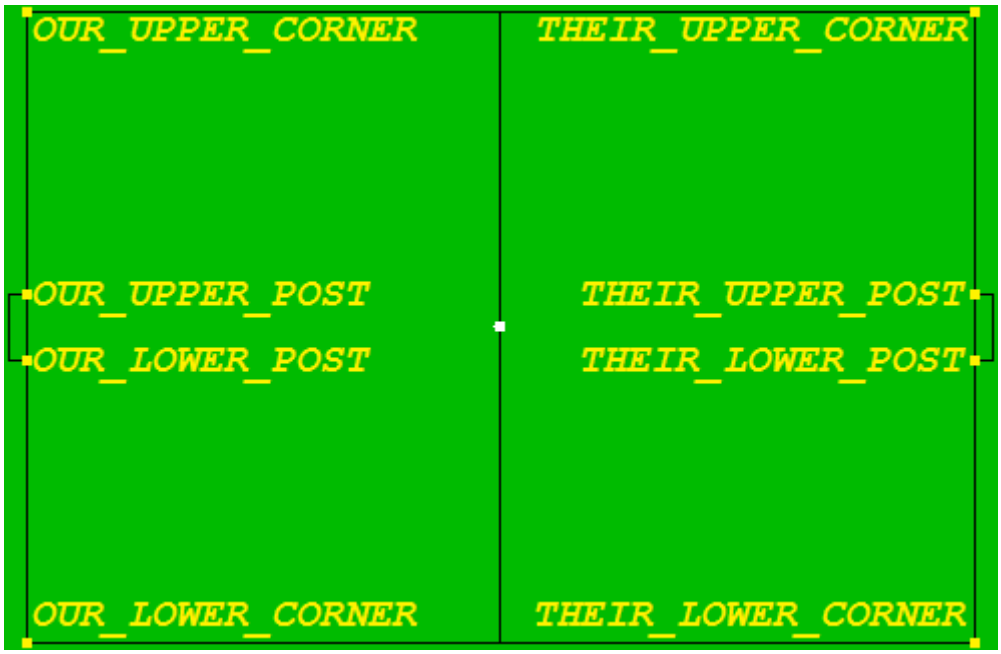
* Podmienky sa vyhodnocujú v poradí, v akom sú uvedené (zhora na dol), vracia sa prvá zhoda.

A.4. sk.fiit.jim.agent.models.FixedObject

Jozef Blažíček

Typ: *public enum*
Hodnoty:

OUR_LOWER_CORNER
OUR_UPPER_CORNER
OUR_UPPER_POST
OUR_LOWER_POST
THEIR_UPPER_CORNER
THEIR_LOWER_CORNER
THEIR_UPPER_POST
THEIR_LOWER_POST



Obrázok 24: Mapovanie hodnôt (objektov) FixedObject na kontrolne body ihriska

A.4.1. positions_0_6_7, positions_0_6_5, positions_0_6_2

Typ: *private static*
Návratová hodnota: *Map<FixedObject, Vector3D>*

Metódy, ktoré vytvoria mapu kontrolných bodov so súradnicami v závislosti od verzie serveru (volanej funkcie)

A.4.2. namesInServerMessages

Typ: *private static*
Návratová hodnota: *Map<String, FixedObject >*

Mapovanie správ zo servera (s-výrazov) na objekty typu FixedObject

A.4.3. fromServerId

Typ: *public static*
Návratová hodnota: *FixedObject*
Parametre:
String id
ID kontrolného bodu v notácii servera (typ s-výrazu)

Vráti objekt požadovaného kontrolného bodu.

A.5. sk.fiit.jim.agent.parsing.ParsedData

Maryna Kovalenko

Typ triedy: *public*
Premenné:
public double SIMULATION_TIME
čas od spustenia servera
public double GAME_TIME
čas behu hry
public Integer PLAYER_ID
id agenta
public Boolean OUR_SIDE_IS_LEFT
strana ihriska, na ktorej hrá agent
public EnvironmentModel.PlayMode playMode
public Map<Joint, Double> agentsJoints
public Vector3D gyroscope
informácia z gyroskopu
public Vector3D accelerometer
informácia z accelerometru
public ForceReceptor forceReceptor
public Vector3D ballRelativePosition
public Map<FixedObject, Vector3D> fixedObjects
public List<PlayerData> otherplayers
public List <ParsedLineWithFlags> lines
zoznam čiar v poradí, v ktorom ich dostáva agent zo servera
public Boolean bumper
public HearReceptor hearReceptor

Trieda je určená na uchovávanie informácii získaných zo správ zo servera.

Príloha B: RoboCupLibrary

Ján Ďurica, Jakub Chalachán, Matúš Ivanoc

B.1. sk.fiit.robocup.library.geometry.Angles

Ján Ďurica

Typ triedy: *public final*
Premenné:
private double from
určuje koniec intervalu uhlov (v radiánoch) - východzia hodnota = 0.0
private double to
určuje koniec intervalu uhlov (v radiánoch) - východzia hodnota = 0.0

Táto trieda poskytuje funkcie na prácu s uhlami. Reprezentuje interval medzi 2 uhlami. Tie sú pôvodne v radiánoch, no dajú sa previesť na stupne pomocou metódy setDegree(). Obsahuje 2 konštruktory, jeden prázdny a jeden s parametrami double *from* a double *to*, ktorý priradí lokálnym rovnomenným premenným hodnoty uhlov v stupňoch.

B.1.1. setRadian

Typ metódy: *public*
Návratová hodnota: *void*
Parametre:
double from
double to
uhly (v radiánoch) určujúce interval uhlov

Táto metóda priradí lokálnym rovnomenným premenným hodnoty uhlov v radiánoch.

B.1.2. setDegree

Typ metódy: *public*
Návratová hodnota: *void*
Parametre:
double from
double to
uhly (v radiánoch) určujúce interval uhlov

Táto metóda priradí lokálnym rovnomenným premenným hodnoty uhlov v stupňoch.
Použitý matematický výpočet: uhol v stupňoch = uhol v radiánoch / 180 * π

B.1.3. include

Typ metódy: *public*
Návratová hodnota: *boolean*

Parametre:
double include_angle

Táto metóda vráti hodnotu true, ak sa vstupný parameter nachádza v intervale (from, to). Ak nie, vráti hodnotu false.

B.1.4. angleDiff

Typ metódy: *public static*
Návratová hodnota: *double*
Parametre:
double first
double second

Táto metóda vráti menšiu hodnotu z normalizovaných rozdielov (z intervalu <0; 2π>) vstupných parametrov v radiánoch.

B.1.5. angleDiffInDeg

Typ metódy: *public static*
Návratová hodnota: *double*
Parametre:
double first,
double second

Táto metóda vráti menšiu hodnotu z normalizovaných rozdielov vstupných parametrov v stupňoch.

B.1.6. normalize

Typ metódy: *public static*
Návratová hodnota: *double*
Parametre:
double angle

Táto metóda vráti uhol v normalizovanej forme, t.j. v intervale <0; 2π>. Ak je vstupný uhol už v tomto intervale, tak sa vráti nezmenený. Ak nie, vypočíta sa takýmto spôsobom:
closest = angle - (2π * (angle / 2π)) Ak je vypočítaná hodnota menšia ako nula, pripočíta sa k nej hodnota 2π a vráti sa. Ak nie je, tak sa rovno vráti.

B.2. sk.fiit.robocup.library.geometry.Circle

Ján Ďurica

Typ triedy: *public final*
Premenné:
private Point2D center
reprezentuje stred kruhu ako bod v dvojrozmernom priestore
private double radius
reprezentuje polomer kruhu

Táto trieda reprezentuje kruh ako matematický 2D útvar. Obsahuje 2 konštruktory, jeden bez parametrov, ktorý vytvorí kruh so stredom so súradnicami [0;0] a polomerom 0. Druhý, s parametrami Point2D *center*, double *radius* vytvorí kruh ako objekt s danými parametrami.

B.2.1. getCenter

Typ metódy: *public*
Návratová hodnota: *Point2D*

Metóda vráti stred kruhu.

B.2.2. getRadius

Typ metódy: *public*
Návratová hodnota: *double*

Metóda vráti polomer kruhu.

B.2.3. toString

Typ metódy: *public*
Návratová hodnota: *String*

Metóda preťažuje klasickú metódu toString vo formáte "Circle [center=" + center + ", radius=" + radius + "]", kde center je stred kruhu a radius je jeho polomer.

B.3. sk.fiit.robocup.library.geometry.Line2D

Ján Ďurica

Typ triedy: *public*
Premenné:

private double x1
reprezentuje x-ovú súradnicu prvého bodu čiary
private double x2
reprezentuje x-ovú súradnicu prvého bodu čiary
private double y1
reprezentuje y-ovú súradnicu druhého bodu čiary
private double y2
reprezentuje y-ovú súradnicu druhého bodu čiary
private double a
reprezentuje koeficient *a* z rovnice priamky ax + by + c = 0
private double b
reprezentuje koeficient *b* z rovnice priamky ax + by + c = 0
private double c
reprezentuje koeficient *c* z rovnice priamky ax + by + c = 0
private double nx
reprezentuje x-ovú súradnicu normálového vektoru (kolmého na čiaru)
private double ny

reprezentuje y-ovú súradnicu normálového vektoru (kolmého na čiaru)

private double kx

reprezentuje x-ovú súradnicu smerového vektoru

private double ky

reprezentuje y-ovú súradnicu smerového vektoru

Táto trieda reprezentuje čiaru ako matematický 2D útvar. Obsahuje 2 konštruktory, kde na vytvorenie čiary použijeme buď 2 body (ich x a y súradnice), alebo 1 bod a smerový vektor. Sú tu použité základné matematické výpočty pre výpočet smerového vektoru a normálového vektoru a nakoniec aj samotných koeficientov a, b a c.

```
kx = x2 - x1;
ky = y2 - y1;

nx = -ky;
ny = kx;

a = nx;
b = ny;
c = -(a * x1 + b * y1);
```

B.3.1. getNormalVector

Typ metódy: *public*
Návratová hodnota: *Point2D*

Metóda vráti normálový vektor ako 2D bod.

B.3.2. solveGeneralEqation

Typ metódy: *public*
Návratová hodnota: *double*
Parametre:

double x
double y

Metóda vráti výsledok rovnice $a*x + b*y + c$.

B.3.3. getCircleIntersection

Typ metódy: *public*
Návratová hodnota: *List<Point2D>*
Parametre:

Circle c

Metóda vráti zoznam bodov danej priamky, ktoré pretínajú kruh zo vstupného parametru. Najprv sa vypočíta euklidovská vzdialenosť medzi 2 bodmi čiary ako odmocnina zo súčtu druhých mocnín ich rozdielov x a y súradníc. Matematicky je to vyjadrené takto:

$$LAB = \sqrt{(\mathrm{Bx} - \mathrm{Ax}) * (\mathrm{Bx} - \mathrm{Ax}) + (\mathrm{By} - \mathrm{Ay}) * (\mathrm{By} - \mathrm{Ay})}$$

Následne sa vypočíta smerový vektor D z bodu A do bodu B.

```
Dx = (Bx - Ax) / LAB
Dy = (By - Ay) / LAB
```

Potom sa počíta *t*, čo je najbližší bod čiary od stredu kruhu, vypočíta sa takto:

```
t = Dx * (c.getCenter().getX() - Ax) + Dy * (c.getCenter().getY() - Ay)
```

Takto sa môže vypočítať bod E, ktorý leží na čiare a je najbližšie ku kruhu.

```
Ex = t * Dx + Ax
Ey = t * Dy + Ay
```

Ďalej sa vypočíta euklidovská vzdialenosť LEC bodu E od kruhu (podobne ako pri LAB). Ak je táto vzdialenosť menšia ako polomer kruhu, tak sa vypočítajú 2 priesečníky čiary s kruhom. Najprv sa vypočíta dt, čo je (euklidovská) vzdialenosť t od prieniku čiary a kruhu. Potom sa počítajú spomínané priesečníky F a G.

```
Fx = (t - dt) * Dx + Ax
Fy = (t - dt) * Dy + Ay
Gx = (t + dt) * Dx + Ax
Gy = (t + dt) * Dy + Ay
```

Nakoniec sa tieto 2 body pridajú do zoznamu, ktorý sa vráti.
Ak vzdialenosť nebola menšia ako polomer kruhu, ale rovná, tým pádom je čiara dotyčnica ku kruhu a vráti sa zoznam obsahujúci len bod E. V iných prípadoch sa vráti prázdny zoznam.

B.3.4. toString

Typ metódy: *public*
Návratová hodnota: *String*

Metóda preťažuje klasickú metódu toString vo formáte " Line2D [x1=" + x1 + ", y1=" + y1 + ", x2=" + x2 + ", y2=" + y2 + ", a=" + a + ", b=" + b + ", c=" + c + ", nx=" + nx + ", ny=" + ny + ", kx=" + kx + ", ky=" + ky + "]", kde sa využívajú lokálne premenné popísané v časti 0.

B.3.5. gettre

Typ metódy: *public*
Návratová hodnota: *double*

V triede sa nachádzajú metódy pre prístup ku všetkým lokálnym premenným (x1, y1, x2, y2, a, b, c, nx, ny, kx, ky).

B.4. sk.fiit.robocup.library.geometry.MEC

Ján Ďurica

Typ triedy: *public*

Táto trieda slúži na počítanie najmenšieho ohraničujúceho kruhu (angl. Minimal Enclosing Circle) z daných bodov.

B.4.1. minEnclosingCircle

Typ metódy: *public static*
Návratová hodnota: *Circle*
Parametre: *List<Point2D> points*

Táto metóda vráti zo zoznamu bodov najmenší ohraničujúci kruh. Volanie tejto metódy podnecuje volanie pomocnej rekurzívnej metódy minCircle, ktorá je opísaná nižšie.

B.4.2. minCircle

Typ metódy: *public static*
Návratová hodnota: *Circle*
Parametre:
int n
Point2D[] p
int m
Point2D[] b

Táto metóda slúži na výpočet stredu a polomeru najmenšej kružnice, ktorá obsahuje body zo vstupného poľa bodov. Pri volaní z predchádzajúcej metódy minEnclosingCircle sa zavolá 4x, až sa v poslednom volaní zavolá metóda findCenterRadius s 3 bodmi tvoriacimi najmenší obklopujúci kruh.

B.4.3. findCenterRadius

Typ metódy: *public static*
Návratová hodnota: *Circle*
Parametre:
Point2D p1
Point2D p2
Point2D p3

Táto metóda na základe vstupných 3 bodov tvoriacich kruh vypočíta jeho stred a polomer. To sa počíta takto:

```
x = (p3.getX() * p3.getX() * (p1.getY() - p2.getY()) + (p1.getX() * p1.getX() + (p1.getY() - p2.getY()) *
    (p1.getY() - p3.getY())) * (p2.getY() - p3.getY()) + p2.getX() * p2.getX() * (-p1.getY() + p3.getY()))
/ (2 * (p3.getX() * (p1.getY() - p2.getY()) + p1.getX() * (p2.getY() - p3.getY()) + p2.getX() * (-p1.getY()
    + p3.getY())))

y = (p2.getY() + p3.getY()) / 2 - (p3.getX() - p2.getX()) / (p3.getY() - p2.getY())
* (x - (p2.getX() + p3.getX()) / 2)
```

Stred kruhu má teda tieto súradnice x a y. Polomer kruhu sa vypočíta ako (euklidovská) vzdialenosť jedného bodu z kružnice od stredu, tu sa používa bod p1. Nakoniec sa vráti kruh so známym stredom aj polomerom.

B.5. sk.fiit.robocup.library.geometry.Point2D

Jakub Chalachán

Typ triedy: public
Premenné:
private double x
určuje x-ovú súradnicu
private double y
určuje y-ovú súradnicu

Táto trieda reprezentuje bod s dvoma súradnicami. Obsahuje konštruktor, ktorý ma 2 parametre, konkrétne double *x* a double *y*. Trieda obsahuje základné funkcie get a set pre vyššie spomínané premenné.

B.5.1. interpolate

Typ metódy: *public static*
Návratová hodnota: *Point2D*

Point2D *from* – začiatočný bod, Point2D *to* – koncový bod, double *timelinePosition* – premenná na posunutie

Táto metóda interpoluje dva body, teda zisťuje v podstate, kam sa posunie bod v určenom smere.

B.5.2. distance

Typ metódy: *public*
Návratová hodnota: *double*
Parametre:
Point2D p

Táto metóda zisťuje vzdialenosť medzi prvým a druhým bodom, pričom prvý bod je definovaný globálne (aké ma súradnice??).

B.5.3. toString

Typ metódy: *public*
Návratová hodnota: *String*

Metóda preťažuje klasickú metódu toString vo formáte " Vector2 [x=" + x + ", y=" + y + "]".

B.6. sk.fiit.robocup.library.geometry.Point3D

Jakub Chalachán

Typ triedy: public
Premenné:
private double x
určuje x-ovú súradnicu
private double y
určuje y-ovú súradnicu
private double z
určuje z-ovú súradnicu

Táto trieda reprezentuje bod s troma súradnicami. Obsahuje konštruktor, ktorý ma 3 parametre, konkrétne double *x*, double *y* a double *z*. Trieda obsahuje základné funkcie get a set pre vyššie spomínané premenné.

B.6.1. toString

Typ metódy: *public*
Návratová hodnota: *String*

Metóda preťažuje klasickú metódu toString vo formáte " %1$.4f %2$.4f %3$.4f", x, y, z".

B.7. sk.fiit.robocup.library.geometry.Vector2D

Jakub Chalachán

Typ triedy: *public*
Premenné:
private double x
reprezentuje x-ovú súradnicu prvého bodu čiary
private double y
reprezentuje y-ovú súradnicu druhého bodu čiary

Táto trieda reprezentuje 2D vektor a definuje možné operácie s nimi. Obsahuje jeden prázdny konštruktor, jeden konštruktor s 2 vstupnými parametrami – double *x*, double *y* a jeden konštruktor s dvojicou súradníc. Trieda obsahuje základné funkcie get a set pre vyššie spomínané premenné.

B.7.1. angle

Typ metódy: *public*
Návratová hodnota: *double*
Parametre: *Vector2D v*

Metóda vráti uhol, ktorý zvierajú 2 vektory, prostredníctvom príslušnej matematickej operácie.

B.7.2. equals

Typ metódy: *public*
Návratová hodnota: *boolean*
Parametre:
Object obj

Metóda porovná vstupný objekt s vektorom, pričom vráti *true* ak je identický alebo zhodný len s malou odchýlkou (0,01). V iných prípadoch vráti *false*.

B.8. sk.fiit.robocup.library.geometry.Vector3D

Typ triedy: *public*
Konštanty:
private static final Vector 3D ZERO_VECTOR
predstavuje nulový 3D vektor so zadefinovanými karteziánskymi súradnicami
public static final Vector3D X_AXIS
predstavuje 3D vektor so zadefinovanými karteziánskymi súradnicami, pričom x-ová súradnica je rovná 1

public static final Vector3D Y_AXIS
predstavuje 3D vektor so zadefinovanými karteziánskymi súradnicami, pričom y-ová súradnica je rovná 1
public static final Vector3D Z_AXIS
predstavuje 3D vektor so zadefinovanými karteziánskymi súradnicami, pričom z-ová súradnica je rovná 1

Premenné:

private double x
x-ová súradnica v karteziánskej sústave
private double y
y-ová súradnica v karteziánskej sústave
private double z
z-ová súradnica v karteziánskej sústave
private double r
sférická premenná v jednotkách radiánov
private double phi
sférická premenná v jednotkách radiánov
private double theta
sférická premenná v jednotkách radiánov

Táto trieda reprezentuje 3D vektor a definuje možné operácie s nimi. Obsahuje jeden prázdny konštruktor, pričom je povedané, že vytvárať 3D vektor je možné len cez jednu z metód a nie priamo cez konštruktor. Trieda obsahuje základné metódy get a set pre vyššie spomínané premenné.

B.8.1. cartesian

Typ metódy: *public static*
Návratová hodnota: *Vector3D*
Parametre:
Point2D xy
double z

Táto metóda vytvorí 3D vektor zo vstupného 2D bodu a tretej súradnice z, reprezentujúcej tretiu súradnicu v priestore.

B.8.2. cartesian

Typ metódy: *public static*
Návratová hodnota: *Circle*
Parametre:
double x
double y
double z

Táto metóda vytvorí 3D vektor zo vstupných súradníc v priestore, pomocou prepočtu karteziánskych súradníc na sférické.

B.8.3. calculateSpherical

Typ metódy: *public static*

Návratová hodnota: *void*

Táto metóda vypočíta sférické súradnice z karteziánskych súradníc.

B.8.4. calculateCartesian

Typ metódy*:* *public static*
Návratová hodnota: *void*

Táto metóda vypočíta karteziánske súradnice zo sférických súradníc.

B.8.5. add - metódy

Typ metódy: *public*
Návratová hodnota: *Vector3D*
Parametre: *double*

Tieto metódy pripočítavajú dĺžku ku konkrétnym reprezentáciám (karteziánska/sférická) súradníc, resp. pripočítanie celého vektora.

B.8.6. substract

Typ metódy: *public*
Návratová hodnota: *Vector3D*
Parametre:
Vector3D anotherVector

Táto metóda odčíta vstupný vektor od globálne definovaného vektora a vráti výsledný vektor.

B.8.7. multiply

Typ metódy: *public*
Návratová hodnota: *Vector3D*
Parametre:
Vector3D *anotherVector*

Táto metóda vynásobí súradnice vstupného vektoru a súradnice globálne definovaného vektora a vráti výsledný vektor.

B.8.8. divide

Typ metódy: *public*
Návratová hodnota: *Vector3D*
Parametre:
Vector3D anotherVector

Táto metóda vydelí súradnice vstupného vektoru a súradnice globálne definovaného vektora a vráti výsledný vektor.

B.8.9. negate

Typ metódy: *public*
Návratová hodnota: *Vector3D*

Táto metóda otočí globálne definovaný vektor do opačného smeru a vráti výsledný vektor.

B.8.10. toUnitVector

Typ metódy: *public*
Návratová hodnota: *Vector3D*

Táto metóda vráti vektor dĺžky 1.

B.8.11. rotateOverSúradnica - metódy

Typ metódy: *public*
Návratová hodnota: *Vector3D*
Parametre:
double súradnica

Tieto metódy vracajú vektory, ktoré boli otočené okolo konkrétnej súradnicovej osi.

B.8.12. crossProduct

Typ metódy: *public*
Návratová hodnota: *Vector3D*
Parametre:
Vector3D anotherVector

Táto metóda vracia vektorový súčin 2 vektorov vo výslednom vektore.

B.8.13. dotProduct

Typ metódy: *public*
Návratová hodnota: *Vector3D*
Parametre:
Vector3D anotherVector

Táto metóda vracia skalárny súčin 2 vektorov vo výslednom vektore.

B.8.14. asPoint3D

Typ metódy: *public*
Návratová hodnota: *Point3D*

Táto metóda vráti definovaný vektor ako (počiatočný) bod s 3 súradnicami.

B.8.15. getXYDistanceFrom

Typ metódy: *public*
Návratová hodnota: *double*
Parametre:
Vector3D b

Táto metóda vráti vzdialenosť vstupného vektora od definovaného vektora.

B.8.16. toString

Typ metódy: *public*
Návratová hodnota: *String*

Metóda preťažuje klasickú metódu toString vo formáte " "x, y, z: [%.2f, %.2f, %.2f] r, phi, theta: [%.2f, %.2f, %.2f]", x, y, z, r, phi, theta".

B.8.17. equals

Typ metódy: *public*
Návratová hodnota: *boolean*
Parametre:
Object obj

Metóda porovná vstupný objekt s vektorom, pričom vráti true, ak je identický alebo zhodný len s malou odchýlkou (0,01). V iných prípadoch vráti false.

B.8.18. normalize

Typ metódy: *public*
Návratová hodnota: *Vector3D*

Metóda vráti normálový vektor.

B.8.19. rotateOver

Typ metódy: *public*
Návratová hodnota: *Vector3D*
Parametre:
Vector3D axis
double angleInRad

Metóda vráti vektor, ktorý je otočený okolo vstupného vektora *axis* vo vstupnom uhle *angleInRad*.

B.8.20. flatten

Typ metódy: *public*

Návratová hodnota: *Vector3D*

Metóda vráti vektor, pričom vynuluje jeho z-ovú (priestorovú) súradnicu.

B.9. Kalmanov filter

Matúš Ivanoc

Kalmanov filter je algoritmus, ktorý z dát zaťaženými nepresnosťami a šumom, odhaduje neznáme hodnoty premenných. Využíva k tomu nielen naposledy namerané dáta a model systému, ale tiež vektor údajov o predchádzajúcom stave systému. Kalmanov filter je široko využívaný pre spracovanie signálov, navigáciu a iné úlohy.
Kalmanov filter hľadá optimálny faktor zmeny pre nasledujúci stav meranej veličiny. Vychádza sa z predpokladu, že nevieme presne veličinu zmerať ani odhadnúť so 100% istotou. Predpoklad je mať buď linearizovaný model správania, alebo v pokročilej verzii algoritmu sa meraný model linearizuje. Do úvahy sa berú tiež predchádzajúce stavy a ich odhady.

podrobnejšie info tu: http://bilgin.esme.org/BitsAndBytes/KalmanFilterforDummies
https://www.cl.cam.ac.uk/~rmf25/papers/Understanding%20the%20Basis%20of%20the%20Kalman%20Filter.pdf

B.10. sk.fiit.robocup.library.math.KalmanForVariable

Matúš Ivanoc

Typ triedy: *public*
Premenné:

private double x_est_last = Double.NEGATIVE_INFINITY
predchádzaujúci odhada nastavený na záporné nekonečno
private double P_last
private double Q
kovariant chyby merania
private double R
kovariant meranej veličiny
private double K
Kalman gain
private double P
predchádzajúci odhad
private double P_temp
private double x_temp_est
private double x_est
výstupný odhad X veličiny podľa Kalmanovho algoritmu

Kalman filter tracing a single observable variable

B.10.1. update()

Typ metódy: *public*
Návratová hodnota: *double*
Parametre:

double observed

Metóda pre update pôvodnej hodnoty zo správy pomocou Kalmanovho algoritmu. Q a R sú konštanty pre filter sa berú z **sk.fiit.jim.Settings** a tie sa získavajú v praxi len experimentom.

B.11. sk.fiit.robocup.library.math.KalmanForVector

Matúš Ivanoc

Typ triedy: *public*
Premenné:
private KalmanForVariable x
x súradnica
private KalmanForVariable y
y súradnica
private KalmanForVariable z
z súradnica

Trieda pre 3D vektor miesto jedného bodu/hodnoty. Metóda update() spracováva 3 súradnice.

B.11.1. update()

Typ metódy: public
Návratová hodnota: Vector3D
Parametre:
Vector3D *observed*

Metóda pre optimalizáciu 3D vektora pomocou Kalmanovho filtra. Aktualizujú sa hodnoty postupne pre x, y a z súradnice a vracia sa výsledný 3D vektor.

B.12. sk.fiit.jim.agent.models.KalmanAdjuster

Matúš Ivanoc

Typ triedy: *public*
Implementuje *ParsedDataObserver*

Trieda upravuje sledované koordináty lopty a zástaviek agentom pomocou Kalmanovho filtra Snaha je znížiť zašumenie týmto spôsobom pred ďalšími výpočtami. V projekte je implementovaný ako observer KalmanAdjuster, ktorý je naviazaný na prijatú správu zo servera. Implementované je to spôsobom, že sa automaticky optimalizuje/odhaduje poloha lopty a pevné rohy ihriska – vlajky.

B.12.1. processNewServerMessage(ParsedData data)

Typ metódy: *public*
Návratová hodnota: *void*
Parametre: ParsedData *data*

Public metóda pre spracovania dát z novej správy zo servera. Spúšťa sa optimalizácia zašumenia polohy lopty na ihrisku a poloha rohových bodov.

B.12.2. adjustBallPosition(ParsedData data)

Typ metódy: *private*
Návratová hodnota: *void*
Parametre:
ParsedData data

Aplikuje Kalmanov filter pre polohu lopty, ak jej posledná videná poloha je staršia než 250ms. Výsledná hodnota sa ukladá späť do správy zo servera.

B.12.3. adjustFixedPointsPosition()

Typ metódy: *private*
Návratová hodnota: *void*
Parametre:
Map<FixedObject, Vector3D> fixedObjects

Aplikuje Kalmanov filter pre pevný bod na mape. Vstupný parameter je hash mapa objektov typu ihrisková vlajka a vektor 3D súradníc ich polohy. Spracované hodnoty sa vracajú späť do spracovanej správy zo servera.

B.12.4. freshKalman()

Typ metódy: *private*
Návratová hodnota: *KalmanForVector*

Konštanty pre filter sa berú z **sk.fiit.jim.Settings** a vracia objekt KalmanForVector().

B.12.5. isObsolete()

Typ metódy: *private*
Návratová hodnota: *boolean*
Parametre:
double when

Metóda vracajúca *true,* ak je spracovaná správa zo servera staršia než 250ms. Porovnáva sa čas *when* s aktuálnym časom simulácie.

B.13. sk...library.math.MathExpressionEvaluator

Matúš Ivanoc

Typ triedy *public*
Premenné:
private final String expression
reťazec výrazu

Transformuje matematický výraz vo formáte reťazca do číselného výsledku
Príkald: new{@link MathExpressionEvaluator}("7+5").getInt() == 12

B.13.1. getInt()

Typ metódy: *public*
Návratová hodnota: *int*

Vracia integer hodnotu reťazce *expression*. Vychádza z návratovej hodnoty getDouble(), ktorú pretypuje.

B.13.2. getDouble()

Typ metódy: *public*
Návratová hodnota: *double*

Vyhodnocuje reťazec *expression* ako double hodnotu.

B.14. sk...library.math.TransformationMatrix

Matúš Ivanoc

Typ triedy: *public*
Premenné:
private double values
private static TransformationMatrix identity

Trieda reprezentujúca operácie na transformáciu trojrozmernej projekcie. Využíva sa to v TestFrameworku pri vykresľovaní viacerých komponentov na 2D plochu.

B.14.1. toString()

Typ metódy: *public*
Návratová hodnota: *String*

Preťažená metóda toString(), ktorá upravuje výstupný formát double čísel do [ ] a na pevný počet miest.

B.14.2. compareWith()

Typ metódy: *public*
Návratová hodnota: *boolean*
Parametre:
TransformationMatrix matrix

Porovnávanie dvoch matíc medzi sebou, vracia *true,* ak sú matice zhodné, inak vracia *false.*

B.14.3. multiply()

Typ metódy: *public*
Návratová hodnota: *TransformationMatrix*

Parametre:
TransformationMatrix matrix

Násobenie dvoch báz (matíc) a vracia novú vypočítanú maticu.

B.14.4. getTranslation()

Typ metódy: public
Návratová hodnota: Vector3D

Výpočet posunu ako 3D vektor.

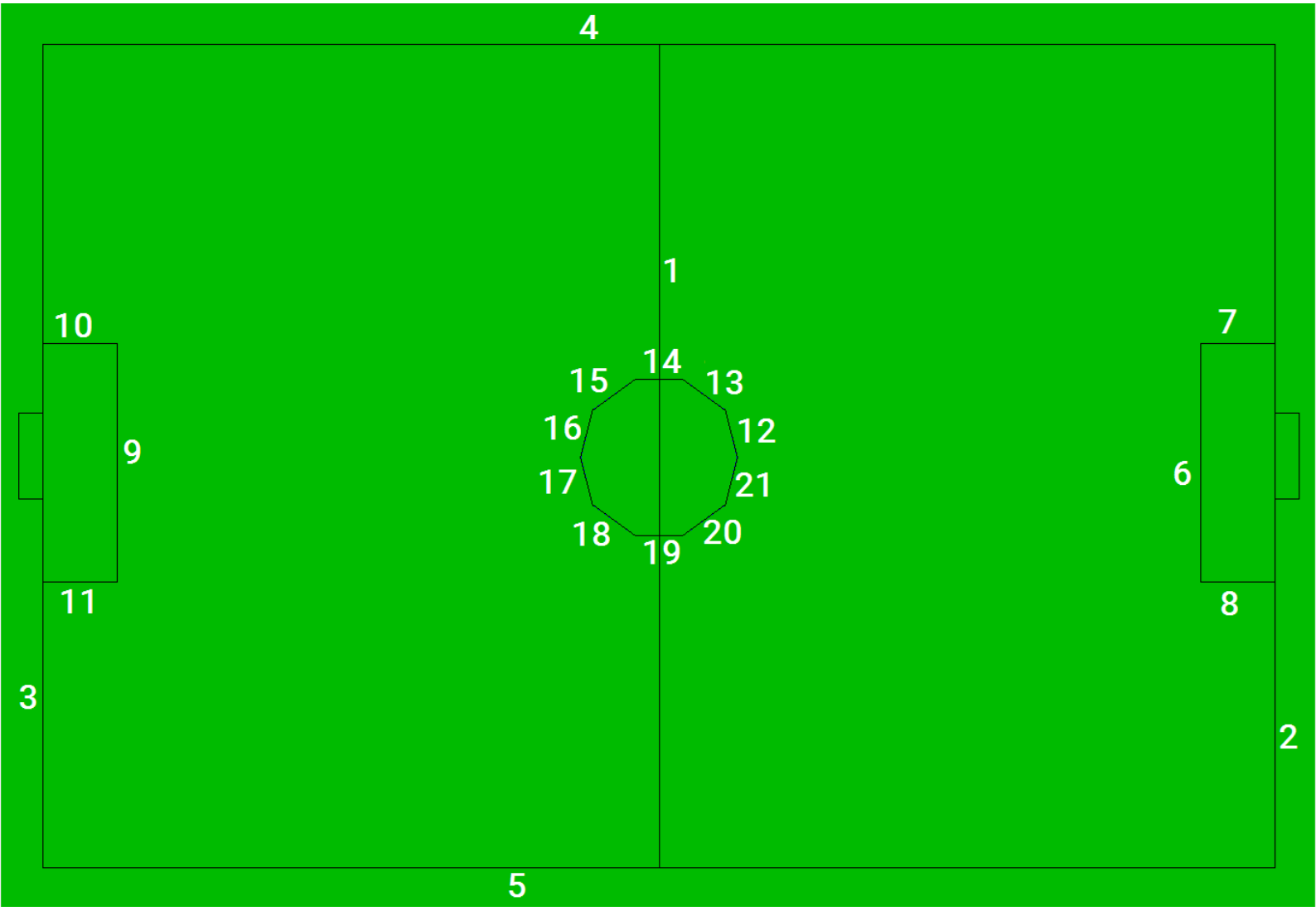
B.14.5. getRotation()

Typ metódy: *public*
Návratová hodnota: *Vector3D*

Výpočet rotácie ako 3D vektor.

Príloha C:
Čiary zo servera

Poradie čiar (id čiar), v akom ich dostáva agent zo servera. Agent dostáva informácie o čiarach, čo vidí, zoradené od najmenšieho podľa príslušného id (čísla pri čiare na obrázku).



Obrázok 25: Poradie čiar, v akom prichádzajú zo serveru