Slovenská technická univerzita
Fakulta informatiky a informačných technológií

Tímový projekt I.

Inžinierske dielo

Tím #19 - WAWOT
Bc. Adriana Beňovičová
Bc. Zoltán Csengödy
Bc. Filip Dian
Bc. Ľubomír Fischer
Bc. Andrej Kútny
Bc. Christopher Liebe
Bc. Adam Mikolášek
Bc. Kitti Nagyová
Vedúci projektu: Ing. Ivan Kapustík
Predmet: Tímový projekt I.
Ročník: 2018/2019

Obsah

1. CELKOVÝ POHĽAD NA PROJEKT 1

2. KOMPONENTY 2

2.1. HRÁČ JIM 2

2.1.1. Zisťovanie polohy Jima 2

2.1.2. Kontrola osí x a y 2

2.2. ROZHRANIE NA TESTOVANIE (TESTFRAMEWORK) 3

3. CIELE 4

3.1. ZIMNÝ SEMESTER 4

3.2. LETNÝ SEMESTER 4

4. PRÁCA OSTATNÝCH TÍMOV 5

4.1. TÍM AUSTIN VILLA 5

4.2. TÍM MAGMAOFFENBURG 5

5. POUŽÍVANÉ TECHNOLÓGIE 6

5.1. SERVER 6

5.2. RCSMONITOR 6

5.3. ROBOVIZ 6

5.4. WIKI 6

5.5. STRÁNKA TÍMU 7

6. OPTIMALIZÁCIE 8

6.1. OPTIMALIZÁCIA ODOSIELANIA SPRÁV DO TESTFRAMEWORKU A VÝPIS DO LOGOV 8

6.2. OPTIMALIZÁCIA WORLDMODEL.PROCESSNEWSERVERMESSAGE(PARSEDDATA) 8

6.3. OPTIMALIZÁCIA AGENTMODEL.PROCESSNEWSERVERMESSAGE(PARSEDDATA) 9

6.4. OPTIMALIZÁCIE TRIEDY AGENTMODEL 10

6.5. OPTIMALIZÁCIA TRIEDY LOWSKILLS 10

6.6. VIACVLÁKNOVÁ KOMUNIKÁCIA 11

6.7. KALMANOV FILTER 12

6.7.1. Teória kalmanovho filtra 12

6.7.2. Kalman v projekte 13

6.7.3. Navrhnutá optimalizácia: 14

6.8. ZERO MOMENT POINT 14

6.8.1. Využívanie ZMP v súčasnom riešení 15

7. ORIENTÁCIA HRÁČA 16

7.1. ISINTERSECTION 16

7.2. ISPLUS 17

7.3. isGoalBox 17
7.4. isT 18
8. TESTOVANIE 19
9. ZÁVER 20

1. Celkový pohľad na projekt

Projekt nášho tímu, inteligentný robotický hráč simulovaného futbalu, ktorý je založený na najväčšej súťaži inteligentných autonómnych robotov RoboCup v sebe zlučuje vývoj softvéru umelej inteligencie, výskum robotiky a vývoj aplikácie v rámci fyzikálneho simulátora.

Hráč, jeho pohyby, fyzika a jeho správanie je postavené na stanovených pravidlách 3D simulátora robotického futbalu. Jedná sa o pokračovanie v práci predchádzajúcich tímov, ktoré na vývoji a zdokonaľovaní hráča pracujú od roku 2010. Projekt je okrem predmetu Tímový projekt každoročne aj súčasťou zadaní bakalárskych a diplomových prác študentov našej fakulty. Okrem vytvorenia konkurencieschopného robotického hráča je jedným z hlavných cieľov tohto projektu rozvinúť schopnosti jednotlivých členov a naučiť ich pracovať v tíme. Nové úlohy sú identifikované na pravidelných stretnutiach, počas ktorých má každý možnosť vyjadriť svoj názor a obavy. Príslušník tímu si následne vyberie úlohy, ktoré chce riešiť a svoj progres zverejňuje na zvolenom komunikačnom kanáli.

Synergiou dosiahnutou pri práci tímov ako aj jednotlivcov v rámci individuálnych projektov vzniká už 8. rok znalostná databáza s množstvom akumulovaného know-how, ktoré je využiteľné pri ďalšom vývoji.

2. Komponenty

2.1. Hráč Jim

Náš hráč predstavuje implementáciu agenta – prijíma a vyhodnocuje impulzy zo strany servera pre fyzikálne presnú simuláciu futbalu. Samotné spustenie programu trvá na priemernom počítači približne 15 sekúnd, pričom je potrebné mať nainštalované a spustené programy *RCSSServer* a *RCSSMonitor*.

V zásade sa správa ako bežný futbalový hráč – snaží sa dostať k lopte a kopnúť ju do bránky. Z času na čas spadne, ale snaží sa hneď postaviť. Pohyby zabezpečujú otočné kĺby v jeho končatinách, ich používanie je definované v jeho kóde. Nad pohybmi sú taktiky, ktoré určujú ofenzívne alebo defenzívne správanie v hre.

2.1.1. Zisťovanie polohy Jima

Identifikácia a analýza aspektov, ktoré zabezpečujú výpočet polohy, respektíve orientáciu Jima, je dôležitá pre budúce správne modifikovanie kódu. Identifikovali sme, že správa o polohe hráča sa skladá z nasledujúcich údajov: poloha lopty, poloha hráča, vypočítavanie čiar a následné odoslanie správy do TestFrameworku. Orientácia Jima sa odohráva v triede Vector3D. V triede AgentPositionCalculator je výpočet samotnej polohy agenta. Na základe rozpoznávania preddefinovaných vzorov čiar, hráč vie definovať na čo sa pozerá a následne vyhodnotiť svoju polohu na ihrisku. Trieda LinePatternRecognition poskytuje funkcie pre klasifikáciu vzťahov medzi čiarami a bodmi, ktoré sú Jimom rozpoznávané. Jednotlivé vzory pre identifikovanie polohy neboli všetky implementované.

Rozpoznanie vzorov tvaru “T”, “+”, zistenie či sú body bránkoviskom a zistenie či čiary majú spoločný priesečník sme implementovali v poslednom šprinte zimného semestra. Na základe aktuálneho stavu implementácie sa hráč vie orientovať a zistiť v ktorej časti ihriska sa nachádza aj podľa čiar, ktoré vidí.

2.1.2. Kontrola osí x a y

Pri rôznych pokusoch s implementáciou funkcií v testovacom režime TestFramework v projekte, bola identifikovaná výmena osí X a Y od plánu projektu a bežnej konvencie označovania osí : X horizontálna os, Y vertikálna os. Táto výmena osí v základných výpočtoch

projektu nebola plánovaná a značte sťažovala prácu pri návrhu základných pohybov a funkcii pre pohyb a orientáciu hráča Jima v priestore.

Na základe analýzy fungovania samotného vnímania polohy a prepočítavania vzdialeností zo základných údajov od servera sme dostali predstavu o správnom modeli týchto výpočtov. Náš kód sme pre overenie porovnali aj s kódom tímu UT Austin Villa. Po podrobnej analýze sme zistili, že výmena osí je problémom vo viacerých funkciách ako aj pri výpočte karteziánskych súradníc. Zistili sme, že uhly Phi a Theta, potrebné na určovanie vzdialenosti sú vymenené vo výpočtoch. Rovnako nekorektná implementácia je vo výpočtoch rotácii vektorov. Táto chyba je identifikovaná v triede Vector3D. Výpočty v triede Vector3D sa používajú v rôznych ďalších triedach a identifikovaná chyba v tejto triede ovplyvňuje aj ďalšie výpočty. Jedným z cieľov do budúceho semestra je opraviť túto chybu a ošetriť v kóde všetky jej závislosti.

2.2. Rozhranie na testovanie (TestFramework)

Na získanie detailnejších informácií o hráčovi používame TestFramework. Je to samostatná aplikácia, ktorá s hráčom a serverom komunikuje prostredníctvom správ (tzv. s-výrazov). Do bežného programu pridáva užitočné a podrobnejšie výpisy o stave hry a polohe hráča a predmetov na ihrisku. Taktiež sa v nej dajú samostatne spúšťať jednotlivé pohyby a taktiky na sledovanie zmien po úpravách kódu.

3. Ciele

3.1. Zimný semester

Na jedno z prvých spoločných stretnutí prišiel aj člen z minuloročného tímu a pomohol nám so stiahnutím projektu a nastavením vývojového prostredia. Následne nám zhrnul ich minuloročnú snahu a dal návrhy na zlepšenie. Po dôkladnej analýze sme identifikovali nasledovné nedostatky:

- veľká časť dostupných zdrojov je spotrebovaná hráčom pri získavaní informácii zo servera
- zlá orientácia hráča v rámci ihriska, pri náklonoch hlavy dochádza k chybnej identifikácií polohy
- prehodené osi X a Y počas spracovania obrazových informácii zo servera (segment SEE v S-výrazoch)

Na analýzu zdrojového kódu používame nástroj YourKit Profiler.

3.2. Letný semester

Počas letného semestra sa chceme hlavne zamerať na väčší objem implementovaných úloh. Konkrétnejšie sa zameriame na vylepšenie správania simulovaného hráča k dosiahnutiu lepších výsledkov, ako aj sprehľadnenie fungovania implementácie a rovnako aj opravenie zistených nedostatkov počas letného semestra, ako je napríklad výmena osi X a Y.

4. Práca ostatných tímov

Veľká časť našej práce bola tento semester venovaná štúdiu práce predchádzajúcich tímov, nakoľko sa na tomto projekte pracuje na fakulte už niekoľko rokov. Robotický futbal je predmetom výskumu nielen v rámci predmetu tímový projekt, ale aj mnohých bakalárskych a diplomových prác. Za zmienku stojí práca Matúša Barabása, ktorý sa venoval vylepšeniu genetického algoritmu. Za najprínosnejšie však považujeme implementácie zahraničných tímov, napríklad *UT Austin Villa* alebo *Magma Offenburg*.

4.1. Tím Austin Villa

Tím z The University of Texas in Austin, ktorý je aktuálne vedúcim tímom v RoboCup vyhráva drvivú väčšinu podujatí a zverejnil základný kód svojho robo-hráča bez špecifických taktík. Taktiež ponúka Youtube kanál a viacero odborných článkov popisujúcich ich technické pokroky a významné zmeny na hráčovi a taktike. Úspech tímu spočíva v dlhodobom vývoji rovnakým tímom odborníkov a študentov.

4.2. Tím magmaOffenburg

Tím z Hochschule Offenburg taktiež patrí medzi popredné a aktuálne tímy. Na stránke má zverejnený git repozitár, množstvo aktuálnych publikácií a iného materiálu (užitočné programy). Ich posledný zverejnený zdrojový kód je však z roku 2014.

5. Používané technológie

5.1. Server

V súčasnosti používame server RCSServer3d-0.6.7. Novšími verziami sa nepodarilo kód úspešne spustiť. Komunikácia medzi agentom a serverom prebieha oboma smermi. Agent posiela na server údaje o svojom pohybe a server posiela agentovi naspäť informácie o stave hry.

5.2. RCSMonitor

RCSMonitor slúži na vizualizáciu ihriska, hráča a celkovo celej hry.

5.3. RoboViz

Na počítačoch typu "Mac" sa nepodarilo spustiť RCSMonitor, takže sme používali nástroj RoboViz na vizualizáciu ihriska.

5.4. Wiki

Jedným zo základných kameňov projektu robotického hráča je uchovávanie a zdokonaľovanie znalostnej bázy. Znalosti, výsledky analýz, dokumentácia k novým implementáciám, návody na prácu s programom a množstvo ďalšieho materiálu, ktorý umožňuje efektívne pokračovanie v práci na tomto projekte sú uložené a zorganizované na wiki, ktorú vytvoril tím A55 Kickers (akad.rok 2012/2013). Úlohou nášho tímu v tomto ako aj ďalšom semestri je aktualizácia článkov a dopĺňanie chýbajúcich analýz k jednotlivým častiam problematiky.

Momentálne pozostáva wiki z týchto častí:

1. Úvod - všeobecné informácie o projekte, úvod do problematiky
2. Jim - informácie k programu a jeho súčastí
    a. Komunikácia - priebeh komunikácie so serverom, vysvetlenie tried a analýza efektivity
    b. Poloha – vyhodnocovanie polohy agenta, kalmanov filter, videnie a funkcie okolo neho
    c. Pohyby - analýza pohybov, high skills, low skills, taktiky hry
    d. Svet - analýza fyziky prostredia simulátora, matematické funkcie v hre, grafy výstupov z hry

3. Analýza tímov - opis jednotlivých predchádzajúcich tímov FIIT a konkurenčných tímov, ich predností a výhod v hre, informácie k dostupným stránkam tímov a ich zdrojových kódov
4. Návody - vypracované dokumenty k spúšťaniu programu, inštalačné príručky, návody na prácu s potrebnými programami
5. TestFramework - návod a podrobná užívateľská príručka k TestFramework programu
6. Wiki - návody k ovládaniu wiki, import a export dokumentov
7. Záverečné práce - súhrn a analýza jednotlivých bakalárskych a diplomových prác k téme robotického hráča simulovaného futbalu
8. Archív - vyradené časti wiki, staré implementácie v ruby, staré návody a iné archívne dokumenty k projektu

Náš tím sa snaží o komplexnú generálnu aktualizáciu a sprehľadnenie wiki. Okrem vypracovania návodov na inštaláciu v systémoch Windows a OS X sme aktualizovali analýzy a rozšírili ich. Vo výsledku by mala byť wiki v budúcnosti primárnym zdrojom informácií pre výskum a implementáciu nových funkcií. Analyzovanie problematík pomôže budúcim tímom urýchliť úvodnú fázu vstupu do projektu.

5.5. Stránka tímu

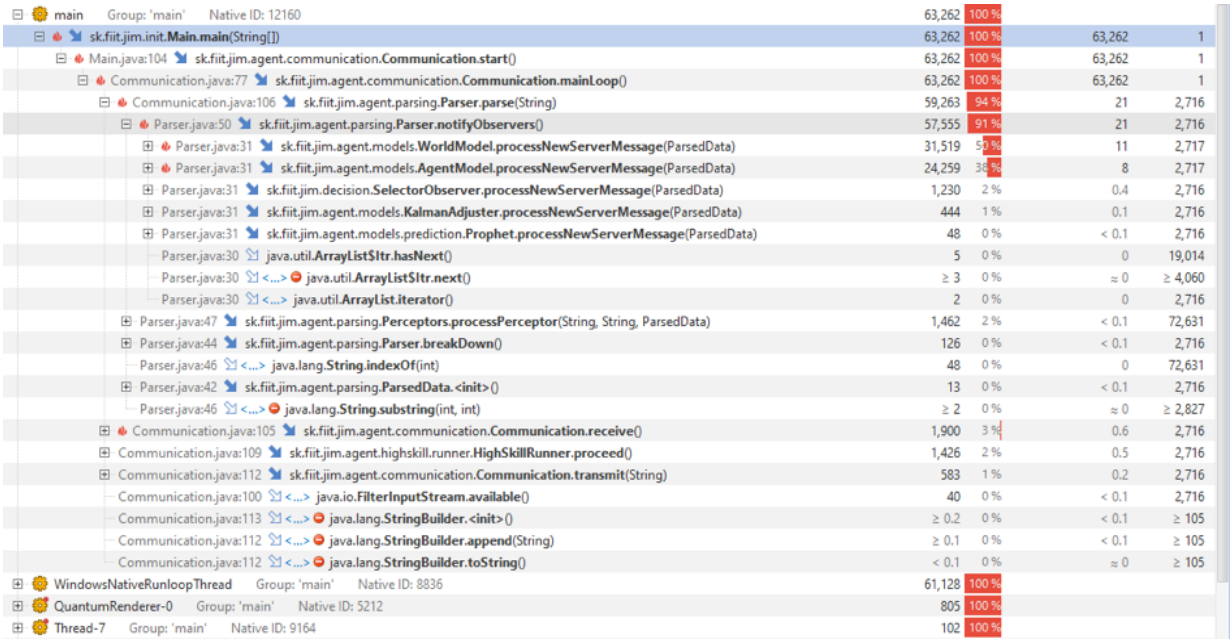
Okrem wiki uverejňujeme časť informácií, napr. zápisnice zo stretnutí, sumarizácie šprintov a zopár ďalších užitočných odkazov aj na našej webovej stránke[1]. V blízkej budúcnosti plánujeme, podobne ako ostatné tímy, pridať sekciu s významnými míľnikmi. Stránka je nastavená tak, aby boli nové dokumenty v súborovom systéme servera rozpoznané automaticky, čo šetrí čas.

[1] http://team19-18.studenti.fiit.stuba.sk

6. Optimalizácie

6.1. Optimalizácia odosielania správ do TestFrameworku a výpis do logov

Na základe analýzy pomocou nástroja Yourkit Java Profiler boli identifikované kusy kódu, ktoré sú v riešení výpočtovo najnáročnejšie. Identifikované metódy bolo potom možné upraviť, aby sa nevykonávali časti, ktoré nie sú pri niektorých úlohách nevyhnutné.



Obrázok 1Pôvodná výpočtová zložitosť

Konkrétne ide o zmeny v metódach *WorldModel.processNewServerMessage(ParsedData)* a *AgentModel.processNewServerMessage(ParsedData),* ktorých pôvodnú zložitosť vidno na obr. č. 1. Je to 50% resp. 38%.

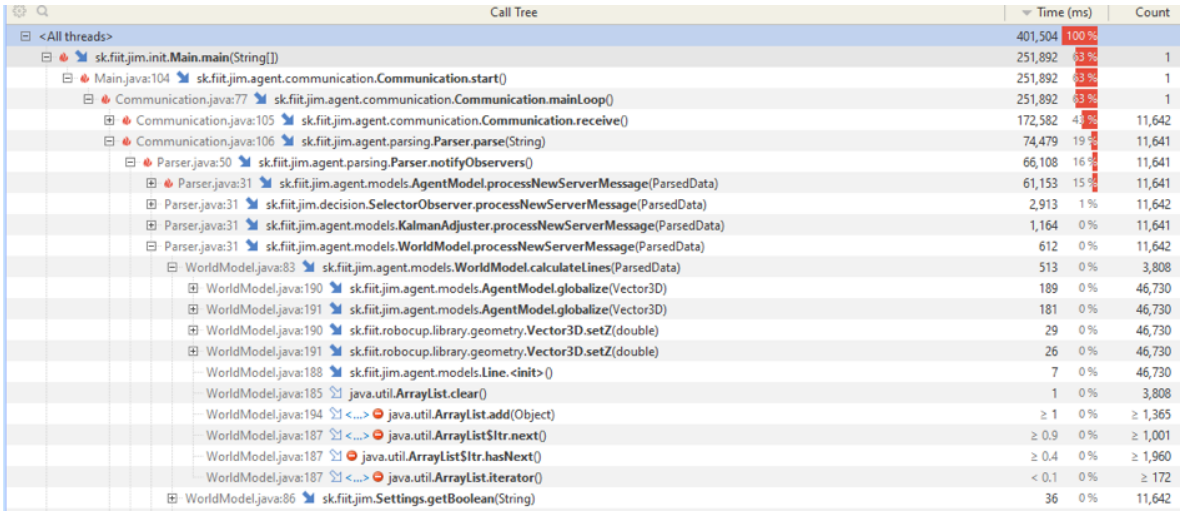
Po analyzovaní kódu týchto metód sa ukázalo, že vytvárajú dáta typu String aj v prípadoch, kedy sa tieto dáta ďalej nepoužijú. V prvom prípade vytvorenie správy, ktorá sa pošle do TestFrameworku a v druhom vytvorenie logového výpisu.

6.2. Optimalizácia WorldModel.processNewServerMessage(ParsedData)

V tomto prípade sa v triede *WorldModel* zbytočne vytvára objekt typu *Message*, kde sa pre vytvorenie používa *ObjectOutputStream*, ktorý zaberá dosť času. Táto správa sa následne pošle

do metódy na odoslanie do TF, kde sa ale odoslanie vykoná iba v prípade, že je TF zapnutý. Ide konkrétne o *if (Settings.getBoolean("TestFramework_monitor_enable")).*

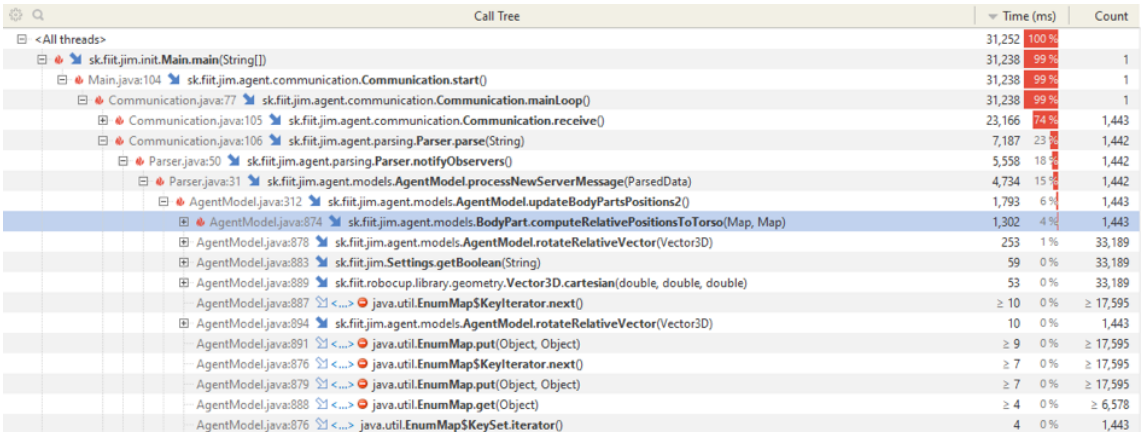
Aby sme zabránili zbytočnému vytváraniu správy, túto podmienku sme dali rovnako aj pred vytváranie správy a následné volanie metódy na odoslanie. Funkcionalita odosielania správ do TF by teda mala byť zachovaná, ale podarilo sa nám skresať náročnosť o značnú časť, čo vyplýva z obr. č. 2.



Obrázok 2Výpočtová zložitosť metód po vypnutí posielania správ do TF

6.3. Optimalizácia AgentModel.processNewServerMessage(ParsedData)

V rámci tejto metódy sa volajú zmeny v chovaní a stave hráča na základe spracovaných dát zo servera. Konkrétne v metóde *updateBodyPartsPositions2(),* ktorá je počas spracovávania volaná sa nachádza vypisovanie do logu o stave hráča. Tento výpis sa zapína v GUI pomocou tlačidla AGENT_MODEL. Aj keď je tento výpis vypnutý, správa sa vytvára pomocou StringBuilder-u, čo predstavuje značné spomalenie. Aby sme sa zbytočnému vytváraniu správy vyhli, bol do triedy *Settings* pridaný atribút „logBodyPartsPosition“, ktorý určuje, či sa má logovanie použiť. V kóde je potom podmienka, kde sa kontroluje, či má daný atribút hodnotu true/false a až potom je samotný výpis do logu. Nastavenie atribútu sa deje defaultne pri spustení programu, kedy sa načíta uložená konfigurácia logovania a prepína sa pri vypnutí/zapnutí atribútu na logovanie AGENT_MODEL v GUI. Touto zmenou sme dosiahli ďalšie ušetrenie výkonu CPU.



Obrázok 3Výpočtová zložitosť metód po vypnutí logovania

6.4. Optimalizácie triedy AgentModel

Cieľom úlohy bolo optimalizovanie triedy *sk.fiit.jim.agent.models.AgentModel* na základe analýzy s Yourkitom, ktorá prvotne ukázala, že niektoré metódy v nej sú pomerne výpočtovo náročné. V tejto triede sa nachádza aktualizovanie stavu hráča, nastavenia jeho častí tela a polohy. Väčšinou ide o vektorové a podobné výpočty, ktoré sa viac optimalizovať zrejme nedajú. V triede sa ale nachádza mnoho výpisov do logov, ktoré sa volajú aj keď nie sú zapnuté. Časť týchto výpisov už bola vypnutá na základe flagu v rámci úlohy v minulom šprinte (metóda *updateBodyPartsPositions2()*), ale niektoré výpisy, ktoré tiež zaberali nepatrný čas ostali zapnuté. Sú to výpisy v metódach *updateCenterOfMass()* a *updateZeroMomentPoint()*. V rámci úlohy boli tieto výpisy takisto vypnuté, takže sa ušetrilo niekoľko percent z celkového času vykonávania. Vypnutie logov, keď ich nie je potreba ušetrilo ~2% času.

6.5. Optimalizácia triedy LowSkills

Cieľom tejto úlohy bolo optimalizovanie zdrojových kódov tried nachádzajúcich sa v priečinku *sk.fiit.jim.agent.moves* na základe analýzy nástroja Yourkit. Tento nástroj ukázal, že niektoré metódy v triedach sú pomerne výpočtovo náročné. V týchto triedach sa nachádza výber low skill-u hráča na základe aktuálnej vybranej taktiky a stavu. Zistili sme, že v triede *LowSkill* sa nachádzajú Log výpisy, ktoré po ošetrení podmienkou *if(Setting.getBoolean(„lowSkillsLogger“))* sa znížila výpočtová náročnosť tejto triedy na 0%. Log výpisy sa odstránili v metódach *step()* a *setNewActionPhase()*. V triede *SkipFlag* sa upravil zdrojový kód v metóde *equals()* na základe odporúčania prostredia InteliJ. Táto úprava zahŕňala zjednodušenie podmienok IF.

Na základe analýzy sa nám podarilo znížiť výpočtovú náročnosť agenta Jim o 7%, konkrétne v triede *LowSkill*v metódach *step()* a *setNewActionPhase()*.

6.6. Viacvláknová komunikácia

Prijímanie a spracovanie informácii zo servera zaberá takmer polovicu zdrojov. Rozhodli sme sa preto oddeliť komunikáciu do vlákna, pričom môže byť spustených aj viacero vlákien naraz. Náš prvý pokus výrazne zrýchlil beh programu, ale spôsobil časté pády hráča. **Error! Reference source not found.** zobrazuje 10 meraní výpočtovej náročnosti pred a po zmene implementácie. Celkovo sa zvýšila rýchlosť o 9,7%.

Číslo merania	Pred zmenou implementácie	Po zmene implementácie
1	48%	45%
2	49%	41%
3	48%	39%
4	47%	36%
5	46%	38%
6	45%	37%
7	51%	35%
8	42%	34%
9	58%	34%
10	47%	45%
priemer	48,1%	38,4%

Tabuľka 1 Porovnanie výpočtovej náročnosti

Následne boli do kritickej oblasti, v ktorej sú správy preposlané observerom, pridané zámky, aby sme zabránili neželanému prepisovaniu dát. Je potrebné podotknúť, že s implementáciou tejto úlohy sa začalo až ku koncu semestra a nemali sme dostatok času na podrobné porovnanie s predchádzajúcou verziou.

6.7. Kalmanov Filter

Kalmanov filter je algoritmus, ktorý používa sériu meraní pozorovaných v priebehu času, obsahujúcich štatistický šum a iné nepresnosti na odhad premenných v širokom spektre procesov. Je široko používaný pre spracovanie signálov, navigáciu a iné úlohy. Predpoklad pre využitie kalmanovho filtra je linearizovaný systém, alebo systém transformovaný na lineárny.

V projekte sa kalmanov filter používa na redukciu šumu, a teda presnejšie určenie pozície objektov, ktoré hráč vidí, teda pozície lopty a pozície pevných bodov.

6.7.1. Teória kalmanovho filtra

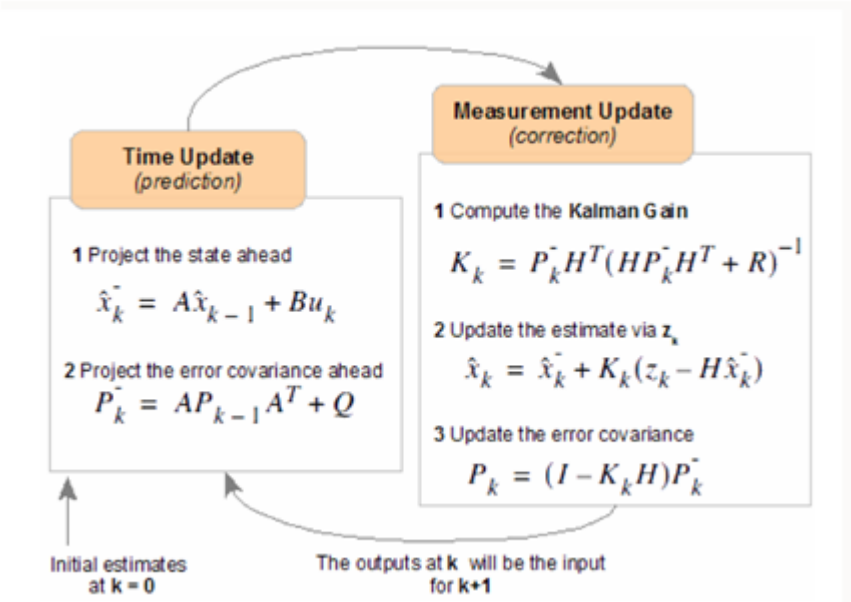
6.7.1.1. Lineárne systémy

V uvedených rovniciach pre lineárne systémy platí:

- A,B a H sú matice
- k je časový index
- x je stav systému
    - vektor x obsahuje všetky dostupné informácie o systéme
    - nie je možné ho merať
- P je matica - predikovaná kovariancia stavu
- K je matica - optimálny kalman zisk
- u je známy vstup
- w je procesný šum
- z je merací šum

6.7.1.2. Rovnice kalmanovho filtra:

Kalmanov filter sa delí na 2 fázy: Predikčnú, počas ktorej sa robí odhad ako predikcia z predchádzajúceho najlepšieho odhadu plus korekcia. Táto fáza obsahuje dve rovnice. Druhá fáza je korekčná fáza, ktorá obsahuje tri rovnice. Ich tvar môžeme vidieť na obrázku č.4.



Obrázok 4 Rovnice Kalmanovho filtra

6.7.2. Kalman v projekte

6.7.2.1. Starý Kalmanov Filter

Tento kalman bol implementovaný v roku 2010 tímom Androids. Využívajú sa v ňom triedy KalmanForVector, ktorý je počítaný pomocou triedy KalmanForVariable aplikovanej na všetky súradnice vektora.

- KalmanForVariable slúži na výpočet linearno-kvadratického Gaussovho regulátora pre premennú - v tomto prípade súradnicu
- KalmanForVector slúži na aplikovanie kalmanovho filtra na vektor
- KalmanAdjuster je trieda slúžiaca na výpočet polohy objektov na ihrisku použitím vyššie uvedených tried

Tento kalmanov filter nie je optimálny, a preto sa ho v roku 2017 pokúsil nahradiť tím BAREKO.

6.7.2.2. Nový Kalmanov Filter

Kalmanov filter je implementovaný v triede KalmanReal, v balíku sk.fiit.jim.agent.kalman. Tento kalmanov filter využíva matice a vektory definované v teoretickej časti a rovnice implementuje podľa obrázku 4. V triede sa nachádzajú dve hlavné metódy:

- public void predict()
    - táto metóda obsahuje rovnice prvej predikčnej fázy kalmanovho filtra

- public void update(RealVector z)
    - táto metóda obsahuje rovnice z druhej korekčnej fázy kalmanovho filtra

Tento kalmanov filter je tiež volaný v triede KalmanAdjuster. Matice a vektory sa tiež inicializujú v tejto triede. Ich hodnoty sa nedajú presne overiť, keďže hodnoty v maticiach sú unikátne pre každý problém, preto ich tím BAREKO testoval na základe im dostupných informácií a metódou pokus - omyl. Podrobné informácie o testovaní hodnôt matíc sa nachádza v inžinierskom diele tímu BAREKO.

6.7.3. Navrhnutá optimalizácia:

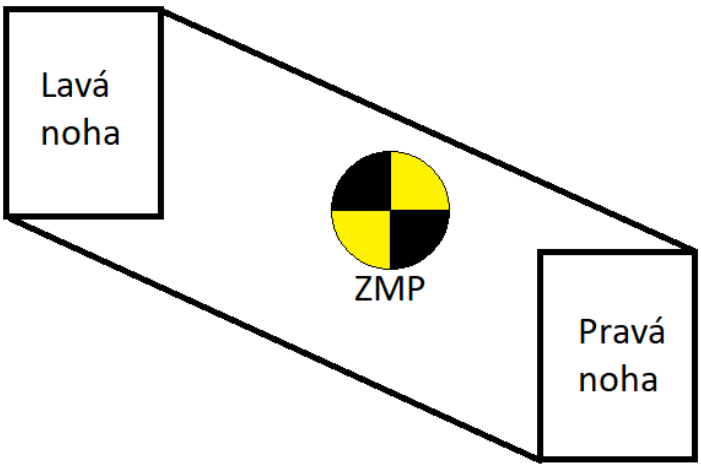
Nový kalman sa v projekte využíva iba na upravovanie pozície lopty. Upravovanie pozície pevných bodov prebieha stále pomocou starého kalmana, ktorý dosahuje horšie výsledky ako nový kalman implementovaný pomocou matíc. Preto hlavnou navrhnutou optimalizáciou bude aplikovanie nového kalmana aj na pevné body, upravenie hodnôt matíc a vektorov pre tieto pevné body, a následné testovanie. Ďalším problémom je opakovaná inicializácia nového kalmana, ktorý sa inicializuje pred každým výpočtom a tým pádom sa vypočítané hodnoty nepoužívajú v ďalších krokoch.

```
 */
public void processNewServerMessage(ParsedData data) {
    initMatrixes();
    this.now = data.SIMULATION_TIME;
    adjustBallPosition(data);
    adjustFixedPointsPosition(data.fixedObjects);
}
```

Obrázok 5 Inicializácia sa vykonáva vždy pred aplikovaním filtra

6.8. Zero moment point

Zero moment point špecifikuje bod, v ktorom dynamická reakcia síl v kontakte s chodidlom a zemou neprodukujú žiadny moment v horizontálnom smere. Stabilita počas všetkých pohybov si vyžaduje umiestnenie ZMP vnútri podporného polygónu. Podporný polygón je vykreslený ako konvexný trup v okolí chodidla a zeme.



Obrázok 6 Podporný polygón

V prípade vychýlenia ZMP z podporného polygónu robot stráca stabilitu a nastáva pád.

6.8.1. Využívanie ZMP v súčasnom riešení

Výpočet ZMP prebieha v triede AgentModel. Obsahuje dve podstatné metódy pre výpočet ZMP: updateZeroMomentPoint() a updateCenterOfMass(). O využívaní ZMP rozhoduje boolean premenná, ktorá ak je zapnutá, umožňuje využitie ZMP pri rôznych spôsoboch chôdze.

Výpočet ZMP prebieha v kóde nasledovne:

```
x = -1 * (center.getX() - (center.getZ() * lastAccelerometer.getX()) /
(lastMomentum.getZ()));
y = -1 * (center.getY() - (center.getZ() * lastAccelerometer.getY()) /
(lastMomentum.getZ()));
zeroMomentPoint = Vector3D.cartesian(x, y, z);
```

ktorý bol s najvyššou pravdepodobnosťou odvodený od týchto vzorcov:

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

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

Robot má accelometer v strede trupu, no výpočet v kóde prebieha z hodnoty Cener of Mass (ťažisko robota). Vzorec taktiež neobsahuje gravitačné zrýchlenie.

7. Orientácia hráča

Nasledujúce implementácie metód napomáhajú hráčovi rýchlejšiu orientáciu na ihrisku. Tieto metódy ešte nie sú použité v orientácii hráča, t.j. ich implementácia bude zapracovaná neskôr.

7.1. isIntersection

Cieľom úlohy je implementovať metódu *isIntersection()*, ktorá zisťuje, či majú dve úsečky spoločný priesečník v tvare + alebo T. Táto funkcia bola predošlým tímom identifikovaná ako potrebná a doposiaľ neimplementovaná. Na jej základe môže hráč lepšie určiť svoju polohu – podľa relatívnych súradníc, ktoré dostane od servera vypočíta, o aký tvar úsečiek ide a podľa vzdialenosti zistí, kde sa nachádza.

Pre naplnenie úlohy boli implementované dve metódy v triede LinePatternRecognition, metóda *isIntersection() a getSegmentIntersect()*. Správnosť týchto metód je otestovaná unit testom v tiede LinePatternRecognitionTest.

public static boolean isIntersection(ParsedLineWithFlags, ParsedLineWithFlags)

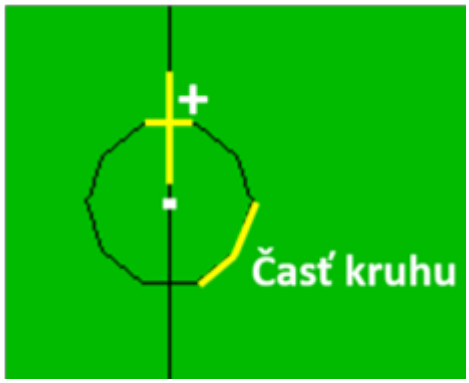
V tejto metóde sa najskôr zistí, či dané úsečky, ktoré prídu ako parameter (v tvare x1, y1, x2, y2) majú priesečník a to pomocou vstavanej metódy *intersectsLine()* triedy Line2D (java.awt trieda, nie trieda z projektu). Ak úsečky majú priesečník, pomocou metódy *getSegmentIntersect()*sa vypočítajú jeho súradnice a následne sa zisťuje, či nejde o úsečky tvoriace roh. To sa vyhodnocuje na základe porovnania súradníc priesečníka a jednotlivých bodov obidvoch úsečiek. Metóda má návratovú hodnotu typu boolean, ktorá nadobúda hodnoty podľa toho, či majú úsečky priesečník a či sa jedná o roh.

public static Point2D getSegmentIntersect(java.awt.geom.Line2D, java.awt.geom.Line2D)

Pomocou tejto metódy sa vypočítajú súradnice priesečníka dvoch úsečiek metódou na základe vektorového súčinu. Metóda vracia priesečník reprezentovaný triedou Point2D, v ktorej sú jeho súradnice x a y. Ak dané úsečky priesečník nemajú, metóda vráti null.

7.2. isPlus

Implementovali sme funkciu *isPlus*(), ktorá zisťuje pomocou metódy *isIntersection()*, či dané dve čiary sa pretínajú a zároveň netvoria roh. Zároveň sa zisťuje, že ak sa pretínajú dané dve čiary tak nech netvoria vzor „T“, toto sa ošetruje volaním funkcie *isT()*, ktorá vracia hodnotu nula ak dve čiary netvoria vzor „T“. Ďalej sa zisťuje či existuje bod priesečníka pomocou metódy *getSegmentIntersect()* medzi dvomi čiarami. Bod musí existovať. Ak sa všetky predchádzajúce podmienky splnia tak vzor plus existuje. Stredová čiara je stále prvá čiara, ktorá sa posiela zo servera. Do funkcie *isPlus()* sa majú posielať iba čiary 1, 14, 19 pretože ináč sa nedá zistiť, že ktorá je stredová čiara. Inštancia čiary nemá informáciu v sebe o type čiary alebo poradí. Toto sa má ošetriť pred volaním samotnej funkcie pri spracovaní odpovede zo servera.

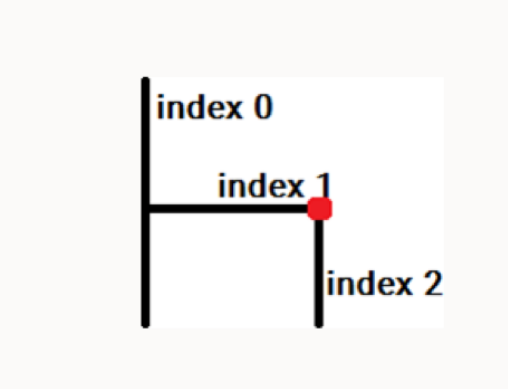


Obrázok 7 Vzor plus

7.3. isGoalBox

Metóda *isGoalBox(),* ktorej argumentom sú tri čiary tyu *ParsedLinesWithFlags* je metóda, ktorá slúži na zistenie, či tri vstupujúce čiary sú bránkovisko. Keďže sa nevie, ktorá čiara je ktorá, je treba zistiť, medzi ktorými dvoma čiarami sa nachádza vzor roh (funkciou *isCorner*). Tieto dve čiary sú čiary bránkoviska - šestnástky. Následne sa pre obe tieto čiary skontroluje, či s treťou čiarou netvoria vzor T (funkciou isT). Ak sa zistí že sa jedná o bránkovisko, tak sa vráti usporiadaná trojica, kde na prvej pozícii (indexe 0) je číslo zadnej čiary, na druhej pozícii (indexe 1) je číslo hrany tvoriacou T so zadnou čiarou, a na tretej pozícii (indexe 2) je číslo poslednej, prednej čiary.

Čiara na indexe 1 má priradené záporné číslo ak spoločný bod čiar bránkoviska je druhým bodom bočnej čiary, a kladné číslo ak sa jedná o prvý bod. Podobne pre čiaru na indexe 2 platí, že ak je tento spoločný bod druhým bodom, vrátená hodnota bude záporná, a ak prvým bodom, tak hodnota bude kladná.

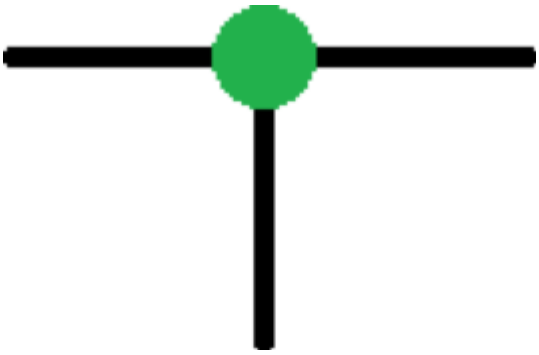


Obrázok 8 Bránkovisko

Metóda teda môže vrátiť 6 rôznych permutácií poľa [1,2,3], pričom druhá a tretia hodnota môžu byť záporné - celkovo 24 rôznych hodnôt, plus *null* v prípade, že sa nejedná o bránkovisko.

7.4. isT

Cieľom bolo pridanie metódy *isT()* do triedy *sk.fiit.jim.agent.models.LinePatternRecognition.* Metóda zisťuje, či dané dve čiary majú spoločný bod a či to je "T", ako ukazuje nasledujúci obrázok.



Obrázok 9 Vzor "T"

8. Testovanie

Implementácia jednotlivých úloh bola priebežne testovaná. Úlohy, ktoré sa zameriavali na zlepšenie výpočtovej náročnosti boli otestované nástrojom YourKit, kde sa odmerala viackrát výpočtová náročnosť pred zmenou implementácie, následne sa nahral nový zdrojový kód a rovnako veľa krát sa odmerala výpočtová náročnosť. Merania boli spriemerované a mohli sme vysloviť záver o koľko sa zlepšila, resp. zhoršila výpočtová náročnosť Jima. Každá jedna takáto úloha má vytvorený dokument testovania, kde je opísané testovanie, testovacie dáta, namerané hodnoty, porovnanie, výsledky a záver.

Úlohy, ktoré riešili implementáciu novej metódy sa testovali jednotkovými testami. Vytvorené jednotkové testy boli všetky na 100% úspešné a nahrali sa na bitbucket. Úlohy testované jednotkovými testami majú vytvorené taktiež dokumenty testovania, kde sú opísané jednotkové testy, vstupy, výstupy a záver.

9. Záver

Strávili sme veľké množstvo času analyzovaním práce predchádzajúcich tímov z našej fakulty ako aj tímov zo zahraničia. V zdrojovom kóde bolo identifikovaných mnoho potenciálnych vylepšení vo forme TODO komentárov, pričom časť z nich sa už dostala aj do backlog-u.

Pre mnohých z nás bola toto prvá skúsenosť s metodikou Scrum a prácou vo väčšom tíme. Riešenie úloh sprevádzali pochopiteľne aj chyby. Spočiatku sme mali vytvorené malé a následne príliš veľké množstvo úloh, ktoré sme kvôli povinnostiam na iné predmety nestíhali. Niektoré úlohy neboli kvôli nedostatočnej komunikácii vypracované správne. Z toho dôvodu sme si zaviedli vlastné DoR (skratka z angl. *definition of ready*) a na task-och pracujeme až po vzájomnej konzultácii. Máme zavedené metodiky a všetky významné zmeny opisujeme v sprievodných dokumentoch. Z doterajších chýb sme sa snažili v maximálnej možnej miere poučiť, aby sme toho budúci semester stihli najmä po implementačnej stránke viac. Pomôže nám k tomu aj väčšia znalosť v riešenej problematike. Rovnako sme si viac osvojili aj odhadovanie času riešenia úloh, ktoré zo začiatku nebolo presné, čo narúšalo aj celkový proces práce, definovania a vypracovania nových úloh. Vďaka tejto skúsenosti sme v tíme začali pravidelne pridávať nové úlohy do backlogu, ktoré sú na pláne a na ktorých je možné začať pracovať v prípade, že všetky úlohy sa stihli s predstihom už počas šprintu.