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

ROBOCUP – tretí rozmer

(dokumentácia k 3. a 4. šprintu)

Tím č.17 : Tím 17 žije...

Bc. Filip Baďura
Bc. Roman Bilevic
Bc. Tomáš Blaho
Bc. Andrej Bisták
Bc. Peter Holák
Bc. Jozef Macho
Bc. Peter Paššák

Študijný program: Softvérové inžinierstvo/Informačné systémy
Ročník: 1.ročník inžinierskeho štúdia
Predmet: Tímový projekt
Vedúci projektu: Ing. Marián Lekavý, PhD.
Akademický rok: 2011/12

1. Príbeh(„user story“)

Požiadavky zadávateľa, resp. ciele, ktoré si tím určil pre daný šprint.

2. Analýza

2.1.Analýza stavu hráča

Stavom hráča sa v minulosti zaoberalo viacero tímov na našej fakulte. Prvé tímy, ktoré sa ním zaoberali sa venovali ešte 2D robotickému futbalu. Z tímov, ktoré riešili 3D robotický futbal to sú tímy Agenty 007 a Robocopy.

2.1.1. Agenty 007

Stav agenta v tíme Agenty 007 by sa dal rozdeliť na 3 základné stavy, pričom prvý sa dá chápať ako samotný stav agenta, druhý ako pozícia agenta voči lopte a tretí ako pozícia agenta voči súperovej bránke. Stavy agenta zisťujú pomocou funkcií:

- IsAgentFallenOnTheGround – zistí, či je hráč padnutý na zem.
- IsBallInDirectView – zistí, či hráč vidí loptu priamo pred sebou (s odchýlkou 5°).
- IsBallOnTheLeft – zistí, či agent vidí loptu naľavo.
- IsBallOnTheRight – zistí, či agent vidí loptu napravo.
- CanKickTheBall – zistí, či je lopta v dosahu a hráč je schopný do nej kopnúť.
- IsGoalpostInDirectView – zistí, či sa bránka nachádza priamo pred hráčom.
- IsGoalpostOnTheLeft – určuje, či hráč vidí bránku naľavo.
- IsGoalpostOnTheRight – určí, či hráč vidí bránku napravo.

Pričom musia platiť nasledujúce vlastnosti:

- Môže platiť iba jeden predikát z IsBallInDirectView, IsBallOnTheLeft a IsBallOnTheRight.
- Môže platiť iba jeden predikát z IsGoalpostInDirectView, IsGoalpostOnTheLeft a IsGoalpostOnTheRight.
- Predikáty IsGoalpostInDirectView, IsGoalpostOnTheLeft a IsGoalpostOnTheRight uvažujú stred bránky.
- Predikáty IsGoalpostInDirectView, IsGoalpostOnTheLeft a IsGoalpostOnTheRight uvažujú súperovu bránku.
- Predikáty IsBallInDirectView, IsBallOnTheLeft, IsBallOnTheRight, IsGoalpostInDirectView, IsGoalpostOnTheLeft a IsGoalpostOnTheRight neuvažujú Z-ovú súradnicu predstavujúcu výšku, pracujú v dvojrozmernej sústave pri pohľade zhora na ihrisko.

Ako vidieť zaoberajú sa tím, či agent stojí, kde sa voči nemu nachádza lopta a bránka súpera a či dokáže kopnúť do lopty. Neriešia však iných hráčov na ihrisku ako ani to, či hráč dokáže dostreliť až k bráne [1].

2.1.2. Robocopy

Tím Robocopy uvažuje rovnaké stavy ako tím Agenty 007 pričom k funkciám na ich zistenie pridáva aj také, čo majú zistiť, či má hráč loptu, poprípade či je k nej najbližšie a kde na ihrisku sa hráč nachádza [2].

[1]http://labss2.fiit.stuba.sk/TeamProject/2008/team07is-si/dokumentacia3.pdf
[2]http://labss2.fiit.stuba.sk/TeamProject/2009/team15is-si/documents/TIMOVY%20PROJEKT_dokumentacia_final.pdf

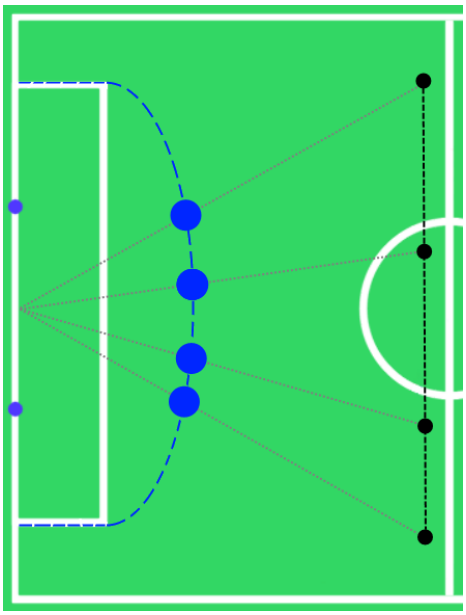
3. Návrh

3.1. Taktické pohyby

➢ **Legenda**:

3.1.1. Obranca

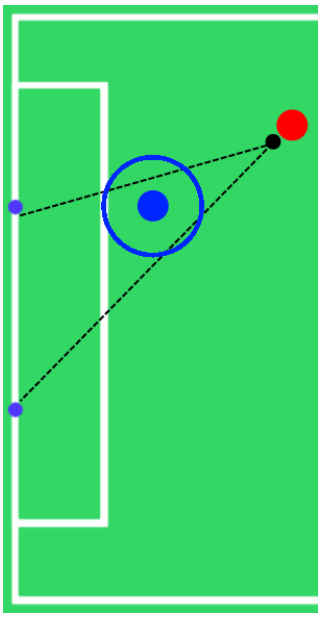
➢ **Kontrolovanie pohybu lopty**



Okolo bránky je istý pomyselný priestor, v ktorom sa pohybuje obranca (pokiaľ nie je súperov hráč s loptou dostatočne blízko bránky). Stále musí sledovať loptu a vypočítavať jej polohu.

Obranca sa pohybuje v tomto priestore tak, aby vždy bol na priamke medzi loptou a stredom bránky. Týmto pohybom minimalizuje priame ohrozenie bránky nakoľko bude schopný v prípade potreby aj padnúť, čím zvýši svoje šance na zastavenie lopty.

- **Bránenie strele**

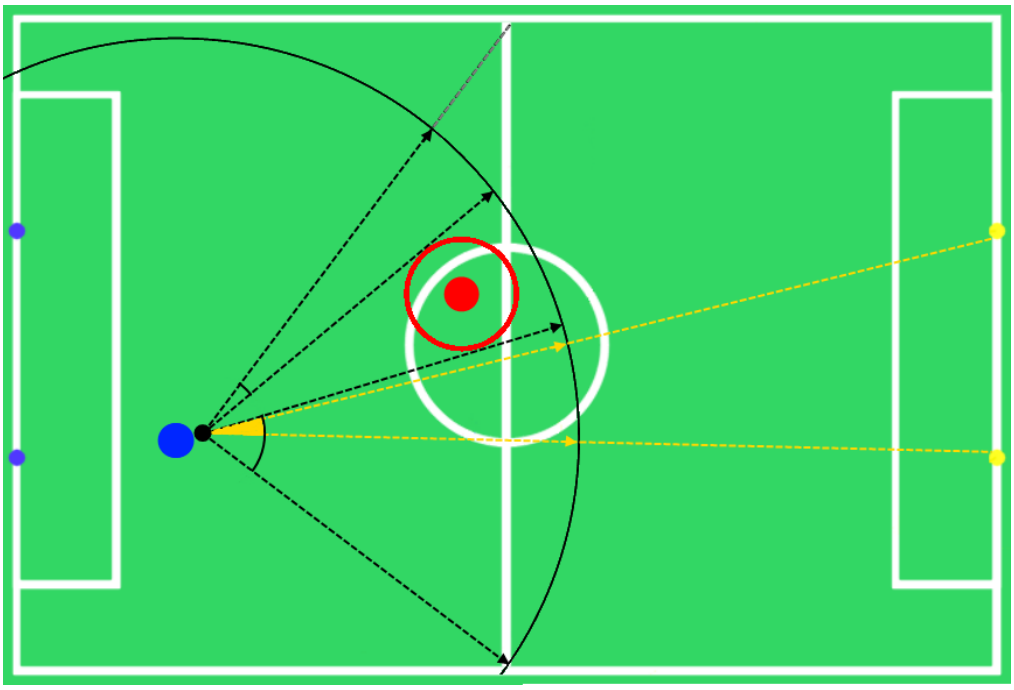


Tento pohyb úzko nadväzuje na predchádzajúci. Ak hráč stojí na priamke tvorenej polohou lopty a stredom bránky, tak má najlepšiu východiskovú pozíciu na zastavenie strely.

Je potrebné vypočítať polohu lopty a všetky možné trajektórie lopty, po ktorých sa môže dostať do bránky.

Obranca môže svoju pozíciu vylepšiť tým, že zníži strelecký uhol tak, že sa priblíži po priamke k protihráčovi s loptou natoľko, aby svojím poľom pôsobnosti zasahoval do všetkých možných smerov strely na bránku.

- **Vykopnutie lopty**

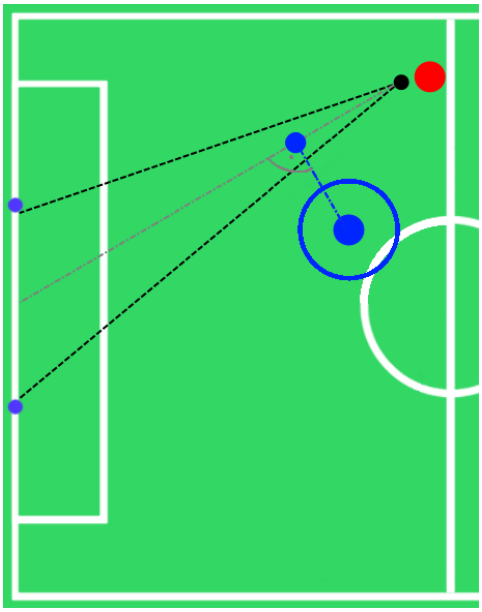


Ak má obranca loptu, jeho hlavnou úlohou je dostať loptu do bezpečia, a teda čo najďalej od bránky. Buď môže nahrať spoluhráčovi, ktorý je možno aj v pozícii ohroziť súperovu bránku alebo bude stačiť len vykopnutie lopty.

Obranca musí vypočítať pozíciu súperov v jeho dosahu (maximálna vzdialenosť kopu do lopty) a musí vedieť, kde je bránka a kde sa nachádza aj on. Z polohy protihráčov zistí, ktorými smermi môže kopať loptu a ktorými nie. Ak sa dá, najlepšie by bolo kopnúť loptu smerom na bránku a ak sa nedá kvôli súperovi, tak čo najďalej.

3.1.2. Stredopoliar

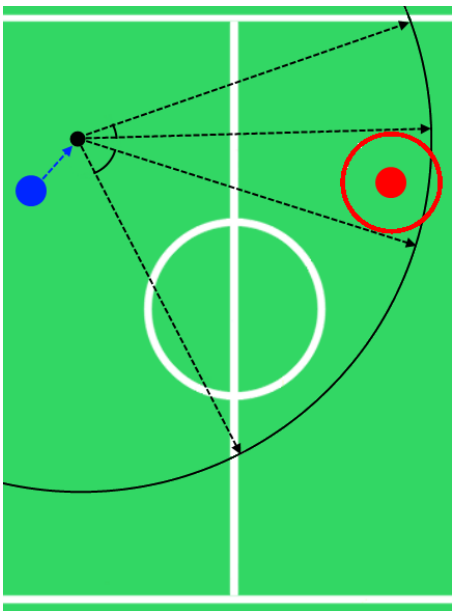
- **Presun a bránenie strele na bránku**



Vypočíta sa poloha lopty a poloha bránky. Vypočíta sa priamka / vektor medzi loptou a bránkou. Následne sa vypočíta kolmica na túto priamku, ktorá prechádza hráčovou pozíciou. Z kolmice a priamky sa vypočíta ich prienik.

Po vypočítaní prieniku priamky s kolmicou sa hráč začne pohybovať v tomto smere, aby čo najskôr zabránil možnej strele na bránku.

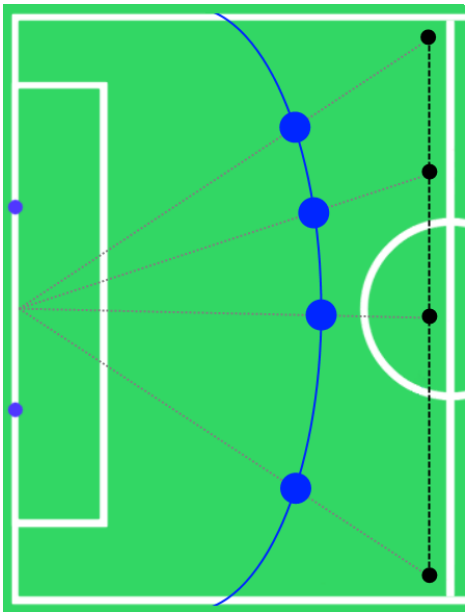
- **Vykopnutie lopty**



Ak má stredopoliar loptu, tak ju môže nahrať spoluhráčovi, ktorý je možno aj v pozícii ohroziť súperovu bránku alebo bude stačiť len vykopnutie lopty.

Stredopoliar musí vypočítať pozíciu súperov v jeho dosahu (maximálna vzdialenosť kopu do lopty) a musí vedieť, kde je bránka a kde sa nachádza aj on. Z polohy protihráčov zistí, ktorými smermi môže kopať loptu a ktorými nie. Ak sa dá, najlepšie by bolo kopnúť loptu smerom na bránku a ak sa nedá kvôli súperovi, tak čo najďalej.

➢ **Kontrolovanie pohybu lopty**



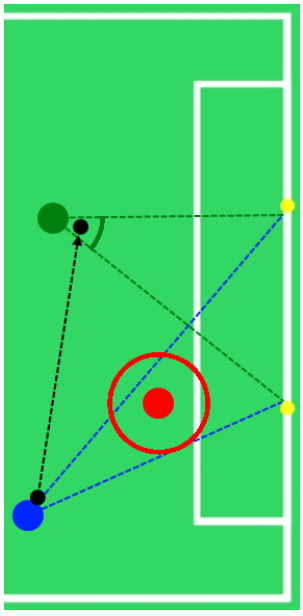
Okolo bránky je istý pomyselný priestor aj pre stredopoliarov, v ktorom sa pohybujú (pokiaľ nie je súperov hráč s loptou dostatočne blízko bránky). Stále musí sledovať loptu a vypočítavať jej polohu.

Stredopoliar sa pohybuje v tomto priestore tak, aby vždy bol na priamke medzi loptou a stredom bránky. Týmto pohybom minimalizuje priame ohrozenie bránky nakoľko bude schopný v prípade potreby aj padnúť, čím zvýši svoje šance na zastavenie lopty.

Pokiaľ sa nachádza na tejto priamke, môže sa začať pohybovať smerom k lopte a získať ju.

3.1.3. Útočník

➢ **Prihrávka na gól**

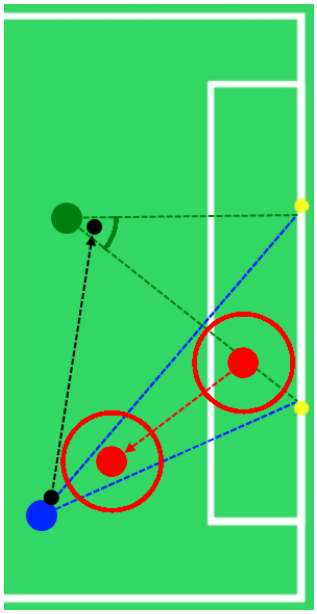


Najprv sa vypočíta, ktorý z hráčov útočiacich na bránku má „lepší výhľad“ na bránku, teda ktorý z nich má väčšiu možnosť skórovať gól. Do úvahy treba taktiež brať tzv. „pole pôsobnosti“ brankára, t.j. nejaký kruh okolo brankára, v ktorom môže zasiahnuť či už pádom, posadením sa alebo inak.

Ak má spoluhráč (hráč bez lopty) lepší výhľad, tak sa zistí jeho presná pozícia, vzájomná vzdialenosť medzi ním a hráčom s loptou a vypočíta sa rýchlosť a smer kopu do lopty.

Hráč s loptou sa k nej postaví v správnom smere a loptu odkopne spoluhráčovi.

- **Vyčkanie na prihrávku**



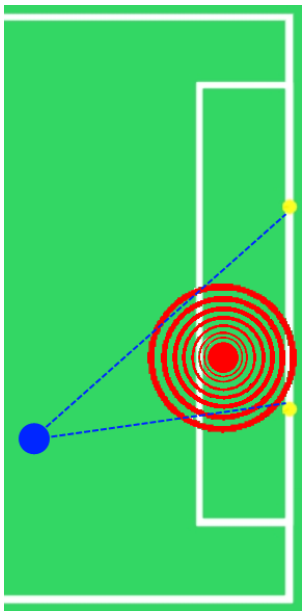
Postup je rovnaký ako v predchádzajúcom prípade (prihrávka na gól), no okrem toho sa vypočíta vzdialenosť medzi hráčom s loptou a brankárom.

Hráč chvíľu počká, znovu odmeria túto vzdialenosť a zistí, či sa brankár hýbe smerom k nemu.

Ak áno, tak počká, kým sa táto vzdialenosť nezastaví alebo kým brankárove „pole pôsobnosti“ nie je „dostatočne“ blízko k trajektórii možnej prihrávky, čiže pokiaľ nepretína túto trajektóriu.

Ak je to dostatočne blízko, tak prihrá loptu.

- **Strela na bránku**

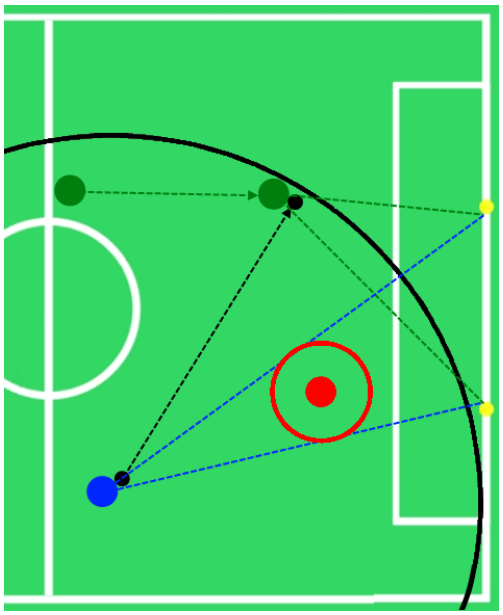


Vypočíta sa poloha brankára.

Zistia sa možné trajektórie streľby na bránku, čiže akési vektory striel.

Zistí sa, ktorá z týchto polpriamok sa najmenej priblíži k polohe brankára, t.j. ktorej vzdialenosť voči brankárovi je najmenšia a podľa toho vystrelí na bránku.

- **Presun na miesto**



Zistí sa poloha lopty a vypočíta sa jej dostrel/dosah, t.j. kam ju je hráč schopný dokopnúť najďalej.

Zistí sa poloha brankára a poloha spoluhráča.

Vypočíta sa, z ktorej polohy v rámci „kruhu dosahu“ je najlepšia šanca na streľbu berúc do úvahy: - čas presunu spoluhráča k nej, - čas jej presunu na konkrétne miesto, - možná pozícia brankára po tomto čase.

Ak sa zistí, že spoluhráč by mal lepšiu šancu streliť gól, tak hráč s loptou mu prihrá loptu a on k nej zatiaľ príde a potom do nej kopne.

3.2. Správa anotácií

Pre využitie anotácií pri predikcii a plánovaní je nutné zabezpečiť určitým spôsobom ich načítavanie a správu. Tento účel plní trieda AnnotationManager.

Pri spustení po načítaní pohybov sa načítajú anotácie – všetky xml súbory z moves/annotations do štruktúry, ktorá ich priraďuje k pohybom ktoré popisujú.

Kvôli potrebe jednoznačnej identifikáciu anotácií podľa mena, a faktu, že jeden pohyb môže mať viac anotácií (čo vylučuje jednoduché použitie mena pohybu) pribudla k objektu anotácie vlastnosť id. Tá sa určuje z názvu súboru, nie je špecifikovaná vnútri xml.

Trieda AnnotationManager umožňuje získanie konkrétnej anotácie podľa id a získanie zoznamu anotácií pre určitý pohyb na základe mena pohybu.

3.3. Výpočet budúcich modelov

Cieľom bolo navrhnúť a implementovať funkcionalitu umožňujúcu zistiť predpokladaný budúci stav sveta. Toto sa vykonáva na základe anotácií k pohybom, ktoré obsahujú popis toho, aký efekt má daný pohyb na hráča a loptu.

Súčasný model agenta a sveta je zahrnutý v triedach:

- AgentModel – stav agenta, jeho poloha a orientácia v ihrisku, stavy kĺbov a ďalších vlastností (či leží, stojí, atď)

- WorldModel – stav sveta, vrátane stavu lopty, stavu agenta a stavu ostatných hráčov
- DynamicObject – lopta – obsahuje jej absolútnu a relatívnu pozíciu a rýchlosť

Boli pridané metódy do tried príslušných modelov, ktoré na základe ich stavu a danej anotácie vrátia novú inštanciu triedy modelu, ktorá bude obsahovať predpokladaný budúci stav.

Funkcionalitu zabezpečujú metódy:

- AgentModel.afterAction(Annotation annotation)– berie ako argument anotáciu a vracia inštanciu triedy AgentModel, ktorá obsahuje predpokladaný budúci stav agenta vypočítaný na základe anotácie. Vychádza sa z kópie súčasného stavu agenta, ktorá je vrátená metódov AgentModel.partialCopy(). Tá vracia kópiu modelu agenta, ktorá však neobsahuje všetky dáta – len tie, ktoré sa využijú. Predpokladá sa totiž veľa volaní metódy afterAction za sebou, čo by v prípade použitia kompletnej hlbokej kópie mohlo rýchlo zvýšiť pamäťové nároky agenta.
- WorldModel.ballAfterAction(Annotation annotation)– vracia objekt triedy DynamicObject, ktorý obsahuje predpokladaný budúci stav lopty. Predpokladá sa, že posledný čas videnia lopty bol čas skončenia pohybu, pretože anotácie toto bližšie nešpecifikujú.

Jednotlivé metódy sú v kóde podrobne okomentované.

3.4. Vylepšenie plánovania s využitím predikcie

Cieľom bolo prerobiť rozhodovanie plánovača tak, aby agent robil rozhodnutia na základe predpokladaného budúceho stavu sveta a agenta samotného. Príkladom by mohlo byť kráčanie k miestu, kde lopta bude, namiesto toho, kde sa nachádza teraz. Taktiež tak vznikne možnosť zistiť, či agent na určité miesto stihne prísť do určitého času.

K tomuto účelu bolo treba z veľkej časti prepísať plánovancí skript. Aby ostal agent funkčný a zároveň som mohol pracovať na novom pláne, ktorý je z veľkej časti previazaný so zatiaľ nie úplne funkčnými časťami (HighSkills, nie je dostatok anotácií, predikcia nie je dostatočne otestovaná), bol nový pre nový plán vytvorený skript plan_new.rb, ktorý začal ako kópia skriptu plan.rb a od tohto stavu som ho upravoval. Nejedná sa však o samostatný nový plánovač – všetky zmeny budú neskôr integrované do pôvodného plánovača.

3.1.4. Nekonečné low skilly

Starý plánovač bol založený takmer úplne na LowSkill objektoch. Tento plánovač využíva prevažne HighSkill objekty, napr. skill Walk. Problémom je ale, že keďže HighSkill okrem iného vykonáva sériu LowSkillov, nemôžeme využiť niektoré naše LowSkills, ktoré nemajú koncový stav a idú donekonečna.

3.1.5. Rozhodovanie

V súčasnom stave išlo hlavne o úpravu kostry rozhodovania tak, aby sa do nej mohli dopĺňať údaje zistené na základe predikcie a proof-of-concept ukážku. Činnosti pri páde boli ponechané, bolo ale vytvorené plánovanie pohybu k lopte, ktoré je pravidelne preplánovavané v závislosti od toho ako sa poloha lopty mení. Neplánuje sa teda celá nasledujúca činnosť, ale len jej istá časť, ktorá je po dokončení nasledovaná ďalšou, alebo môže byť prerušená ak sa hodnoty odchýlia natoľko, že sa neoplatí pokračovať.

4. Implementácia

4.1. Implementácia stavu hráča

Stav hráča, ako to bolo vidieť aj u predchádzajúcich tímov sa skladá z rôznych na sebe nezávislých častiach. V našom prípade sme sa rozhodli implementovať niektoré stavy a informácie, ktoré nám môžu pomôcť pri vyhodnocovaní situácie na ihrisku a sú to tieto:

- state
- whereIsBall
- whereIsGoal
- isBallMine
- isInRange
- isUnderCover

Stavy aj konštanty, ktoré sa pri vyhodnocovaní používajú, sa nachádzajú v triede AgentInfo.

4.1.1. state

Stav state predstavuje názov pohybu, ktorý agent aktuálne vykonáva, alebo pozíciu v akej sa jeho telo nachádza v prípade pádu na zem.

4.1.2. whereIsBall

Stav whereIsBall predstavuje, na ktorej strane sa aktuálne nachádza lopta vzhľadom na relatívnu orientáciu hráča. Tento stav môže nadobúdať hodnoty z množiny front, left, right, back pre pozície lopty na jednotlivé strany od hráča. Na vyhodnotenie stavu slúži metóda triedy AgentInfo s názvom whereIsBall().

4.1.3. whereIsGoal

Stav whereIsGoal predstavuje, na ktorej strane sa aktuálne nachádza súperova bránka, vzhľadom na relatívnu orientáciu hráča. Tento stav môže nadobúdať hodnoty z množiny front, left, right, back pre pozície bránky na jednotlivé strany od hráča. Pozícia bránky sa počíta pre jej stred. Na vyhodnotenie stavu slúži metóda triedy AgentInfo s názvom whereIsGoal().

4.1.4. isBallMine

Stav isBallMine predstavuje, či je hráč v určitej blízkosti od lopty, ktorá sa vyhodnocuje podľa konštanty *BALL_IN_CONTROLL*, v prípade, že sa agent nachádza v stanovenej alebo menšej vzdialenosti k lopte, nadobúda stav hodnotu true, čo znamená, že je vyhodnotený ako pravivý.

V opačnom prípade nadobúda hodnotu false. Na vyhodnotenie stavu slúži metóda triedy AgentInfo s názvom isBallMine().

4.1.5. isInRange

Stav isInRange predstavuje, či je hráč v pozícii, z ktorej dokáže strelou ohroziť súperovu bránku. Pozícia sa vyhodnocuje na základe hodnoty premennej ourGoalRange, ktorá sa inicializuje z konštanty OUR_GOAL_RANGE a následne môže byť menená počas zápasu. Ak sa hráč dostane do časti ihriska, z ktorej ako už bolo povedané, dokáže dostreliť do súperovej bránky, tento stav sa vyhodnotí ako pravdivý a nadobudne hodnotu true. V inom prípade má hodnotu false. Na vyhodnotenie stavu slúži metóda triedy AgentInfo s názvom isInRange().

4.1.6. isUnderCover

Stav agenta isUnderCover predstavuje situáciu, kedy v tesnej blízkosti agenta nachádza jeden alebo viacerí hráči súpera. Hodnota nebezpečnej vzdialenosti sa stanovuje na základe konštanty PLAYER_UNDER_COVER. V takejto situácii má stav hodnotu true. V prípade, že sa v stanovenej blízkosti agenta nenachádza žiadny súperiaci hráč, stav má hodnotu false. Na vyhodnotenie stavu slúži metóda triedy AgentInfo s názvom isUnderCover().

4.2.Implementácia vyhodnotenia hráčov na dostrel bránky

Pri taktike má význam aj rozmiestnenie ostatných hráčov na ihrisku a vyhodnotenie možných výhod a nebezpečenstiev, ktoré z toho vyplývajú. Overovanie, či sú jednotlivý hráči na dostrel súperovej bránky, môže mať využitie vo viacerých situáciách.

Toto vyhodnotenie prebieha po vypočítaní globálnych pozícií ostatných hráčov na ihrisku. V cykle sa prejdú všetci hráči a pre každého sa vyhodnotí stav, či sa nachádza, alebo nenachádza na dostrel súperovej bránky. Vyhodnotenie je odlišné pre spoluhráčov a protihráčov, keďže je veľmi pravdepodobné, že budú mať kvalitatívne odlišnú techniku kopania a hráči jedného tímu môžu dokopnúť do bránky z väčšej vzdialenosti ako hráči z druhého tímu.

Vzdialenosť, ktorá sa vyhodnocuje ako nebezpečná sa načítava z premenných triedy AgentInfo a ide o premenné ourGoalRange a enemyGoalRange, ktoré su inicializované hodnotami z konštánt OUR_GOAL_RANGE a ENEMY_GOAL_RANGE. Premenné môžu byť následne upravované počas zápasu, ak nastane taká potreba.

Stav jednotlivých agentov pre situáciu, keď sú na dostrel súperovej bránky je uložený v premennej isInRange v triede Player.

4.3. *Optimalizácia chôdze I.*

4.3.1. Chôdza vpred

V tejto kapitole sme sa venovali optimalizácii chôdze, ktorá bola veľmi neefektívna, až zlá. Na obrázkoch nižšie je vidieť, že chôdza je nielen veľmi pomalá, ale je aj nepresná a nestabilná. Hráč sa takisto nepohyboval rovno, ale robil veľký oblúk a tým pádom sa jeho schopnosť kráčať vpred veľmi znižuje.

Obrázok č.: Chôdza walk_fine_fast1 – pohľad zhora

Obrázok č.: Chôdza walk_fine_fast1 – pohľad zboku

Na pohľade zhora je zreteľné, že sa agent hýbe neprimerane doľava, pričom má ísť rovno. Na pohľade zboku zase vidno, ako ďaleko je agent schopný s danou implementáciou pohybu presť, čo je skutočne veľmi málo vzhľadom na rozmery ihriska. Preto sme tento pohyb museli optimalizovať vylepšujúc smer pohybu, teda aby sa agent nevychyľoval zo smeru pohybu, a takisto, aby prešiel za ten istý čas väčšiu vzdialenosť.

Tieto ciele sa nám podarilo úpravou jednotlivých fáz nakoniec docieliť, pričom dosiahnuté výsledky sú pre rovnaké pohľady na agenta viditeľné na nasledujúcich obrázkoch.

Obrázok č.: Chôdza walk_fine_fast1po úprave – pohľad zhora

Obrázok č.: Chôdza walk_fine_fast1 po úprave – pohľad zboku

Na pohľade zhora je vidieť, že agent sa oproti predošlému pohybu už pri kráčaní tak nevychyľuje zo smeru pohybu a kráča takmer priamo. Na druhom obrázku je zase vidieť, že agent prejde za rovnakú časovú dobu(v tomto prípade 70 sekúnd), prejde vyše dvojnásobnú vzdialenosť, čo je samozrejme spôsobené aj jeho priamou chôdzou,a tým, že nerotuje do strany. Navyše je tento pohyb stabilný a pri opakovanom spúšťaní sa agent pohyboval stabilne a nepadal.

4.3.2. Chôdza vzad

Chôdza vzad mala jeden veľký problém, že pri kráčaní hráč stále rotoval svoj pohyb a stáčal sa do špirály miesto toho, aby šiel priamo dozadu. Jeho rýchlosť síce nebola príliš veľká, no takto bol ten pohyb pôvodne aj implementovaný, aby bol len posunom o malú vzdialenosť, preto sme sa zamerali iba na priamosť tohto pohybu. Predchádzajúca verzia je znázornená na nasledujúcom obrázku.

Obrázok č.: Chôdza walkback2 – pohľad zboku

Po zmene zdrojového kódu a úprave jednotlivých fáz pohybu sme dosiahli oveľa presnejší pohyb vzad, ktorý už nevytváral špirálu. Smer pohybu sa síce odchyľoval od kolmice na začiatok pohybu, ale táto odchýlka nie je vzhľadom na krátkosť kroku až tak podstatná.

4.4. Optimalizácia chôdze II.

V tejto kapitole sme sa venovali optimalizácii chôdze, ktorá bola veľmi neefektívna, až zlá.

Tento dojem sme nadobudli po zhliadnutí jednotlivých typov chôdze. Po odovzdaní tejto časti dokumentácie sme však prišli na to, že uvedené pohyby nie sú vlastne tie, ktoré tím Androids implementoval v minulom roku a použil v súťaži Robocup na FIIT, resp. tieto pohyby obsahovali isté časti kódu, ktorých vplyv na vykonávanie cyklických pohybov bol markantný a tieto sa vykonávali nielen oveľa pomalšie, ale aj nesprávne. Preto sme v ďalšom šprinte kontaktovali tím Androids a tento problém riešili. Išlo o to, že v každom cyklickom pohybe boli vykonané zmeny, ktoré neboli premietnuté do poslednej verzie v SVN, ktorú sme využili ako základ nášho projektu. Zo zistených skutočností sme vykonali v implementácii pohybov zmeny, ktoré napravili nesprávne vykonávanie pohybov, a tak sme museli aj znovu implementovať optimalizáciu pohybov.

4.4.1. Chôdza vpred walk_fine_fast2

Ako prvú sme sa snažili optimalizovať chôdzu, ktorú tím Androids implementoval naposledy, a ktorá bola aj použitá v súťaži. Táto chôdza je veľmi stabilná a dosahuje aj dobré výsledky, čo sa týka smeru chôdze. Vzhľadom na tieto vlastnosti chôdze bolo dosť náročné implementovať nejaké optimalizačné zmeny, ktoré by vylepšili tento pohyb. Kedže bol priamy, týmto smerom sme sa nevydali a sústredili sme sa na rýchlosť pohybu. Túto rýchlosť sme sa snažili zlepšiť hlavne skrátením vykonávania jednotlivých fáz. Nasledujúca tabuľka je vyhodnotením spúšťania tohto pohybu v pôvodnej podobe, pričom výsledky boli takmer identické, a teda možno konštatovať, že tento pohyb je skutočne veľmi dobre spracovaný a je mimoriadne stabilný. Výsledky prezentované v tabuľkách nižšie predstavujú čas, za ktorý sa dostal agent z počiatočnej polohy do polovice ihriska(aj v prípade, že šiel šikmo).

Tabuľka č.: Výsledky spúšťania chôdze walk_fine_fast2

Číslo pokusu	1	2	3	4	5
Výsledný čas[s]	66	63	63	63	64

Priemerný čas, za ktorý tento pohyb bol vykonaný, bol **63,8 s**.

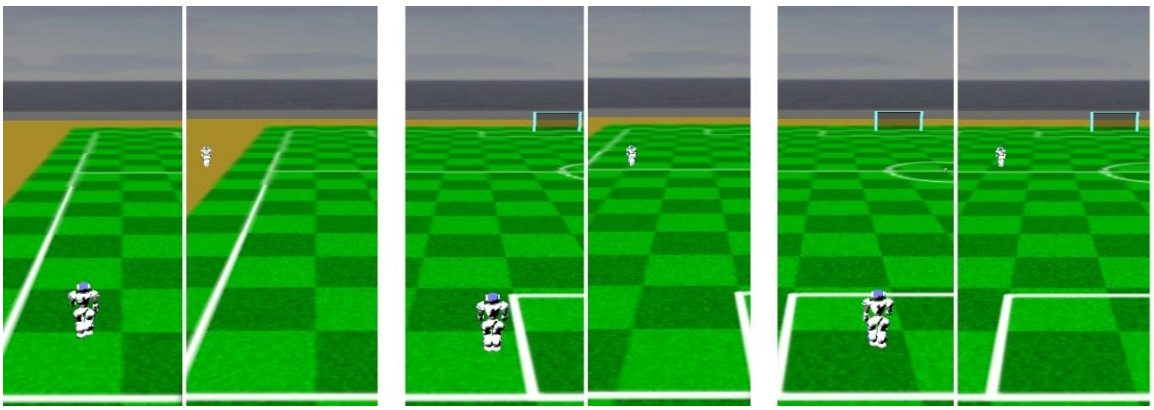
Pri implementácii optimalizačných zmien sme sa teda zamerali na zrýchlenie pohybu. Po vykonaní zmien v kóde sa tieto zmeny prejavili veľmi pozitívne, čo sa týka zrýchlenia pohybu, avšak vzhľadom na zrýchlenie sa dostavili aj nežiadúce účinky v podobe odchýlenia smeru, v niektorých prípadoch dokonca k strate stability a občasného pádu. Pád sa vyskytoval približne pri 17% spustení pohybu. Výsledky sú prezentované v nasledovnej tabuľke, koncipovanej podobne, ako predchádzajúca.

Tabuľka č.: Výsledky spúšťania po optimalizácii chôdze walk_fine_fast2

Číslo pokusu	1	2	3	4	5	6	7	8	9	10	11
Výsledný čas[s]	49	48	50	48	pád	47	49	pád	50	51	49

Priemerný čas, za ktorý sa tento pohyb vykonal(odhliadnuc od pádov), bol **49 s.**

Ako je vidieť, rýchlosť pohybu sa podarilo dosť dobre vylepšiť, avšak stabilita sa zhoršila. Rovnako sa zhoršila aj priamočiarosť pohybu, čo je znázornené na nasledujúcom slede obrázkov.



Obrázok č.: Chôdza walk_fine_fast2 po úprave – pohľad zhora

Na základe týchto zistení je možné skonštatovať, že tento pohyb by sa dal využiťhlavne na prekonávanie kratších vzdialeností, pričom by občas bolo potrebné skorigovať smer agenta. Je tu ale takisto priestor na ďalšie vylepšenie udržania priamočiareho smeru, čím by optimalizácia predstavovala na dĺžku ihriska zrýchlenie oproti predchádzajúcej implementácii približne o 13 sekúnd.

4.4.2. Chôdza vpred walk_fine_fast1

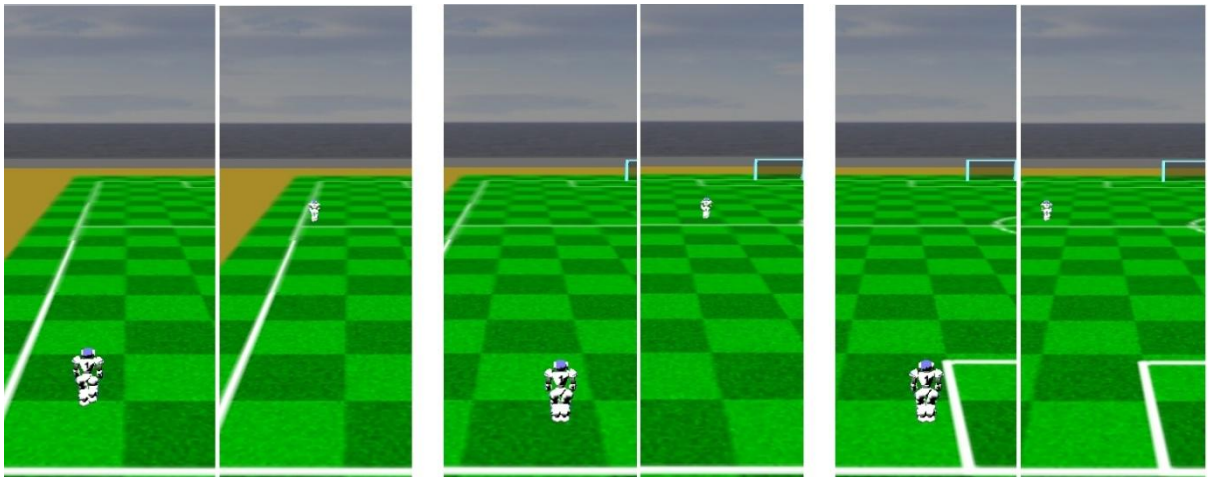
Druhou chôdzou, ktorú sme skúšali optimalizovať bola chôdza, z ktorej vychádzala predchádzajúca. Preto sme očakávali, že tu taký priestor na vylepšenie nenájdeme, avšak opak sa stal pravdou a po vhodnej optimalizácii sme dosiahli lepšie výsledky ako v predchádzajúcom prípade. V tomto prípade bol pohyb pri spustení veľmi nestabilný, čo sa týka priamočiarosti a rýchlosť tohto pohybu bola taktiež veľmi nevyhovujúca. Po spustení sa robot začal nekoordinovane točiť na jednu či druhú stranu a iba vo veľmi zriedkavých prípadoch sa dostal až do polovice ihriska, ktorá bola métou pre testovanie našich pohybov. Keď sa tam už agent dostal, trvalo mu to približne 102 sekúnd.

Po optimalizácii tohto pohybu sa dosiahlo výrazné zlepšenie nielen priamočiarosti kráčania, ale aj v rýchlosti pohybu, čo sa prejavilo zlepšením času prechodu do testovanej oblasti, teda polovice ihriska. Výsledky sú obsiahnuté v nasledujúcej tabuľke a takisto aj v nasledujúcom obrázku, ktorý ilustruje aj približnú odchýlku od požadovaného smeru chôdze.

Tabuľka č.: Výsledky spúšťania po optimalizácii chôdze walk_fine_fast1

Číslo pokusu	1	2	3	4	5	6	7	8	9	10	11
Výsledný čas[s]	43	42	44	42	44	43	42	43	44	43	44

Priemerný čas, za ktorý tento pohyb bol vykonaný, bol **43, 09 s**.



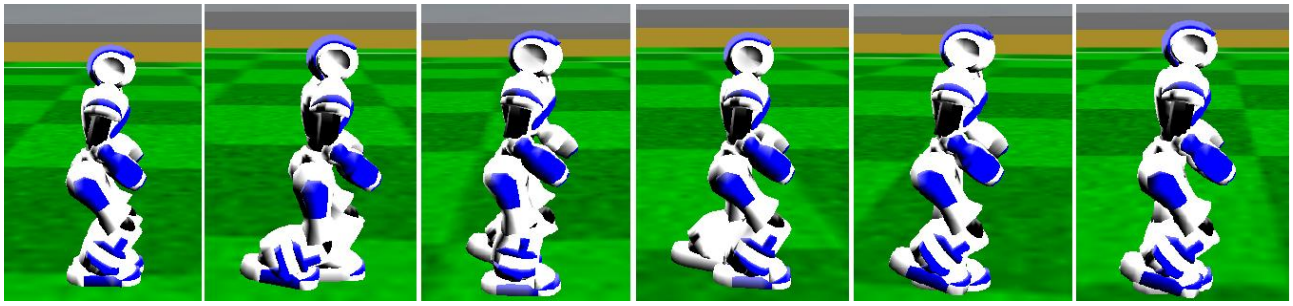
Obrázok č.: Chôdza walk_fine_fast2 po úprave – pohľad zhora

Vylepšenie v oblasti rýchlosti je oproti chôdzi walk_fine_fast2 v pôvodnej verzii o cca 20 sekúnd, pričom dosahujeme prijateľnú priamočiarosť pohybu, čím sa tento pohyb stáva veľmi dobre použiteľným na prekonávanie dlhých vzdialeností pri potrebe prejsť ich čo najrýchlejšie. Navyše pri testovaní bola dosiahnutá 100%-ná stabilita, čo sa týka udržania agenta v polohe stoja, resp. chôdze, teda agent ani raz pri testovaní nespadol.

4.5. Chôdza dozadu

Táto chôdza dozadu je výraznejšie rýchlejšia a stabilnejšia ako chôdze vytvorené tímom Androids v minulom roku. Jej začiatočná fáza vznikla z prvej fáze chôdze dopredu. Vzhľadom na to že tento pohyb je pomerne rýchly a stabilný je vhodný na prekonávanie väčších vzdialeností. Robot je pri tejto chôdzi celý čas mierne prikrčený. Samotný krok pozostáva zo 6 fáz a je zobrazený na obrázku.

- robot sa prikrčí
- vykroči pravou nohou dozadu
- pravú nohu mierne vystrie a pohyb pokračuje symetricky pre ľavú nohu



Obrázok č.: Chôdza dozadu – pohľad zboku

Overenie

Pohyb pri niekoľko násobnom testovaní bol stabilný a jeho dráha priama. Ak vznikli odchýlky tak len veľmi minimálne a to pravdepodobne kvôli chybám zo servera. Celý pohyb trvá 2400 ms, čo znamená že krok jednou nohou polovicu tohto času. Pri pokuse o zrýchlenie jednotlivých fáz agent dokázal udržať stabilitu, avšak pri tomto zrýchľovaní sa agent viac vychyľuje z dráhy.

4.6. Parsovanie a serializácia anotácií

Reprezentácia anotácií bola implementovaná v jazyku Java. Trieda *Annotation* predstavuje dátovú štruktúru anotácie presne podľa návrhu. Ďalšie triedy predstavujú čiastkové dáta anotácie. Konkrétne trieda *State* predstavuje stav robota na začiatku alebo konci vykonávaného pohybu, trieda *Joint* predstavuje kĺb robota (jeho názov a uhol otočenia) , trieda *Axis* predstavuje súradnicový systém a trieda *Values* obsahuje samotné hodnotu parametrov (minimálnu, maximálnu a priemernú hodnotu).

Pre prácu s XML súbormi bolo použité API JAXP DOM3. Všetky metódy pre parsovanie a serializáciu sú statické, aby sa zaistilo ich jednoduché použitie v akejkoľvek časti kódu.

Na parsovanie XML súboru slúži trieda *XMLParser*. Jej metóda *parse(File file)* najprv vytvorí objektový model dokumentu. Potom overí, či XML dokument zodpovedá XML schéme anotácie, ktorá je uložená v súbore *framework.xsd*. Na toto overenie sa použije metóda *check(File f).* Pokiaľ je overenie úspešné, tak sa údaje z jednotlivých elementov objektového modelu XML súboru skopírujú do premenných inštancie triedy *Annotation.* Po úspešnom prekopírovaní sa tento objekt vráti ako návratová hodnota metódy *parse(File file).*

Na serializáciu anotácie do XML súboru slúži trieda *XMLCreator*. Metóda *serialize(String file, Annotation annotation)* má ako argumenty názov výstupného súboru a objekt anotácie, ktorá sa má do tohto súboru serializovať. Najprv sa vytvorí objektový model dokumentu pomocou metódy *createXML(Annotation annotation).* Z jednotlivých

premenných anotácie, ktorá je argumentom tejto metódy, sa vytvoria príslušné elementy a vzťahy medzi nimi tak, aby zodpovedali hore uvedenej XML schéme. Pomocou triedy *Transformer*, ktorá je súčasťou JAXP DOM3, sa následne vykoná samotná serializácia.