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

ROBOCUP – tretí rozmer

(dokumentácia k 5., 6., 7., 8., 9. a 10. 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. Analýza 4
1.1. Zistenie pozície/stavu hráčov, ktorých agent vidí 4
1.2. Prispôsobenie pohybov na reťazenie 4
1.2.1. Stav XML pohybov 4
1.3. Sumarizácia pohľadu na hráča 5
1.4. Predikcia lopty 10
1.4.1. Analýza existujúceho riešenia tímu Androids 11
1.4.2. Analýza výpočtov predikcie z diplomovej práce Petra Ertla 11
1.5. Návod 11
1.5.1. Účel návodu 11
1.5.2. Stav úlohy 12
1.5.3. Budúcnosť 12
1.6. High Skill – Hranie futbalu 12
2. Návrh 13
2.1. Zistenie pozície/stavu hráčov, ktorých agent vidí 13
2.2. Vyhodnocovanie herných situácií 13
2.3. Vytvorenie hernej formácie 14
2.4. Zistenie vhodnosti prihrávky 14
2.5. Podpora viacerých hráčov 14
2.5.1. Možnosti implementácie 14
2.6. Vylepšenie GUI 15
2.7. Plánovanie trajektórie 16
2.8. Vylepšenie plánovača 16
2.9. Predikcia lopty 17
2.10. Kráčanie za loptou 18
2.11. Optimalizácia pohybov 18
2.12. Vytvorenie úkrokov 19
2.13. High Skill – Hranie futbalu 19
2.14. Vedenie lopty 19
3. Implementácia 20
3.1. Zistenie pozície/stavu hráčov, ktorých agent vidí 20
3.1.1. Testovanie 21
3.2. Vyhodnocovanie herných situácií 21
3.3. Vytvorenie hernej formácie 22
3.4. Zistenie vhodnosti prihrávky 22
3.5. Zadefinovanie High skillov 23
3.5.1. GetUp 23
3.5.2. Localize 24
3.5.3. Walk 24
3.5.4. Turn 24

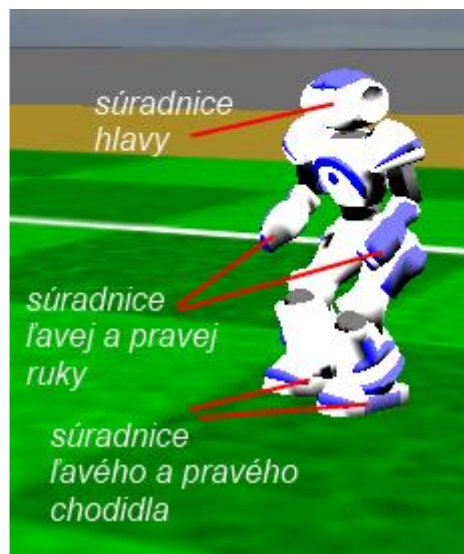
3.5.5. Kick ... 24
3.6. Prispôsobenie pohybov na reťazenie ... 25
3.7. Podpora viacerých hráčov – úprava test framework-u ... 25
3.8. Vylepšenie GUI ... 26
3.9. Správa viacerých agentov ... 26
3.9.1. Úpravy v GUI ... 26
3.9.2. Úpravy v agentovi ... 26
3.10. Profilovanie a optimalizácia kódu ... 27
3.11. Plánovanie trajektórie ... 27
3.12. Vylepšenie plánovača ... 27
3.13. Dokumentácia kódu ... 29
3.14. Predikcia lopty ... 30
3.14.1. Testovanie ... 30
3.15. Kráčanie za loptou ... 30
3.16. Optimalizácia pohybov ... 31
3.17. Vytvorenie nových pohybov ... 32
3.17.1. Možné vylepšenia ... 34
3.18. Grafické zobrazenie v Test frameworku ... 34
3.18.1. Monitorovanie agentov ... 34
3.18.2. Možné dodatočné úpravy ... 34
3.19. Opravy v Test fraweworku ... 35
3.19.1. Rotácia v dátach z monitora ... 35
3.19.2. Správa testov ... 36
3.20. Kop do lopty ... 36
3.20.1. Silný priamy kop do lopty ... 36
3.20.2. Overenie ... 37
3.21. High Skill – Hranie futbalu ... 37
3.21.1. Testovanie ... 37
3.22. Zrýchlenie a spomalenie chôdze ... 37
3.23. Optimalizácia zrýchlenej a spomalenej chôdze ... 38
3.24. Vedenie lopty ... 38

1. Analýza

1.1. Zistenie pozície/stavu hráčov, ktorých agent vidí

V aktuálnej verzii agenta je možné pristupovať k hráčom, ktorých agent vidí a to pomocou triedy WorldModel. Títo hráči sa od seba líšia hlavne tým, že patria do iného tímu, pričom z danej triedy dostaneme zoznam jednotlivých hráčov tímu agenta a zoznam protihráčov. Títo hráči sa od seba odlišujú(resp. mali by sa od seba odlišovať) pomocou ich čísla na „drese", ktoré by malo byť pre hráčov v danom tíme unikátne.

Ďalšími informáciami, ktorými agent dokáže disponovať o ostatných agentoch, ktorých vidí, je okrem iného aj pozícia niektorých častí ich tela. Je to napríklad pozícia hlavy, pozícia ľavej a pravej ruky a pozícia ľavého a pravého chodila(viď obrázok pod textom). Zo súradníc jednotlivých častí tela je teda možné určiť polohu, v akej sa nachádza videný hráč.



Obrázok č.1: Súradnice jednotlivých častí videného hráča

1.2. Prispôsobenie pohybov na reťazenie

Je nutné rozdeliť nekonečné pohyby na časti, tak aby sme mohli reťaziť viacero pohybov, pri pokusoch o to sa vyskytla chyba – hráč bol menej stabilný, treba zistiť o čo ide a ako to opraviť, vo finalizačných – ukončovacích fázach treba zabezpečiť dostanie sa do štandardnej polohy – treba prispôsobiť XML.

1.2.1. Stav XML pohybov

Väčšina pohybov už finalize stavy mala, akurát boli odstránené z niektorých chôdzí, pretože

pri súčasnom stave plánovača sa low skilly zakaždým finalizovali po jedinom vykonaní. O tomto ale nikto nevedel, pretože dokumentácia od minuloročného tímu sa takýmto základným veciam absolútne nevenovala a ich kód bol príliš neprehľadný na to aby sa to dalo pochopiť. Vrátil som teda tieto stavy do walk pohybov z ich predchádzajúcich verzií.

1.3.Sumarizácia pohľadu na hráča

Počas práce na agentovi Jim sme často narážali na problém, že sme veľmi dlho hľadali miesta, kde by sme mali náš kód implementovať alebo ktoré metódy použiť a vôbec, čo máme k dispozícii a s čím máme možnosť pracovať. Často dochádzalo k situácii, že sme strávili veľmi dlhý čas hľadaním potrebných informácií a niekedy bol tento čas dokonca dlhší ako samotná implementácia, keďže neexistovali takmer žiadne popisy ku kódu, resp. prehľad, čo sa kde nachádza. Z tohto dôvodu sme uvážili, že by bolo vhodné takýto prehľad spraviť(aj z dôvodu, že sa na tomto projekte ešte v budúcnosti bude pracovať, a tak by mohol slúžiť ako dobrý základ pre začiatok práce budúcim generáciám).

Zamerali sme sa hlavne na funkcie, ktoré sú najpodstatnejšie z pohľadu ovládania agenta a prístupu k informáciám o okolitom svete. Na základe toho vznikol prehľad najdôležitejších častí kódu, s ktorými sa aj v budúcnosti budú riešitelia RoboCupu stretávať pravdebodobne neustále. V nasledujúcej tabuľke je zhotovený prehľad, ktorý by mal pomôcť hlavne v začiatkoch sa zorientovať v kóde.

Tabuľka č.1: Prehľad základných tried na prácu s agentom a okolím

Trieda	**Atribút/metóda**	**Vstupné parametre**	**Výstup**	**Popis**
AgentInfo	team		String	tím, do ktorého agent patrí
	playerId		int	číslo hráča
	opponentsStates		HashMap <Integer, playerState>	Vráti zoznam videných hráčov súpera a ich stav, teda či ležia alebo stoja
	teammatesStates		HashMap <Integer, playerState>	Vráti zoznam videných spoluhráčov a ich stav, teda či ležia alebo stoja
	getIsBallMine		Boolean	či je agent blízko lopty(má ju v držaní)
	getWhereIsGoal		Boolean	kde sa nachádza bránka voči agentovi(front, back, right, left)

	getWhereIsBall		Boolean	kde sa nachádza lopta voči agentovi(front, back, right, left)
	getIsInRange		Boolean	či je agent na dostrel bránky
	getIsUnderCover		Boolean	či je agent obsadený(protihráči sú v jeho blízkosti)
	getPlayerState	Player, boolean	playerState	Zistí v akom stave je agent(standing, laying)
AgentModel	getJointAngle	Joint		zistenie veľkosti uhlu vybraného kĺbu(*HE1* (-120.0, 120.0), *HE2* (-45.0, 45.0), *RLE1* (-90.0, 1.0),*RLE2* (-45.0, 25.0),*RLE3* (-25.0, 100.0),*RLE4* (-130.0, 1.0),*RLE5* (-45.0, 75.0),*RLE6* (-25.0, 45.0),*RAE1* (-120.0, 120.0),*RAE2* (-95.0, 1.0),*RAE3* (-90.0, 90.0),*RAE4* (-120.0, 120.0), *LLE1* (-90.0, 1.0),*LLE2* (-25.0, 45.0),*LLE3* (-25.0, 100.0),*LLE4* (-130.0, 1.0),*LLE5* (-45.0, 75.0),*LLE6* (-45.0, 25.0),*LAE1* (-120.0, 120.0),*LAE2* (-1.0, 95.0),*LAE3* (-90.0, 90.0),*LAE4* (-120.0, 120.0))
	processNewServerMessage	ParsedData		identifikuje hráča a stranu hráča podľa dát zo servera
	isStanding		Boolean	zisťuje či agent stojí
	isOnGround		Boolean	zisťuje či je agent na zemi
	isLyingOnBack		Boolean	zisťuje či agent leží na chrbte
	isLyingOnBelly		Boolean	zisťuje či agent leží na bruchu
	getRotationX		double	vráti rotáciu v smere osi x
	getRotationY		double	vráti rotáciu v smere osi y
	getRotationZ		double	vráti rotáciu v smere osi z

	getPosition		Vector3D	vráti pozíciu agenta
	getLastDataReceived		ParsedData	vráti posledné obdržané dáta zo servera
	afterAction	Annotation	AgentModel	vráti predikciu budúceho modelu hráča na základe anotácie pohybu
	partialCopy		AgentModel	vytvorí čiastočnú kópiu modelu použiteľnú pre predikciu
DynamicObject	getPosition		Vector3D	vráti pozíciu dynamického objektu
	getRelativePosition		Vector3D	vráti relatívnu pozíciu dynamického objektu
	getLastTimeSeen		double	vráti čas, kedy bol naposledy dynamický objekt videný
	getSpeed		Vector3D	vráti rýchlosť dynamického objektu
	getRelativeSpeed		Vector3D	vráti relatívnu rýchlosť dynamického objektu
Environment Model	GAME_TIME		double	Odohraný čas bez počítania v čase aktivity módu BEFORE_KICK_OFF
	SIMULATION_TIME		double	čas simulácie od štartu servera
	PLAY_MODE		PlayMode	hrací mód
	TIME_STEP		double	časová jednotka, o ktorú sa inkrementuje hrací čas(časové kvantum), aktuálne 0,02 sekundy
	version		Version	aktuálna verzia servera
	processNewServerMessage	ParsedData		nastaví čas hry, čas simulácie a hrací mód podľa dát zo servera
Player	isTeammate		boolean	zistí či je videný hráč spoluhráčom alebo protihráčom
	getHead		Vector3D	vráti pozíciu hlavy videného agenta
	getLlowerarm		Vector3D	vráti pozíciu ľavej ruky videného agenta
	getRlowerarm		Vector3D	vráti pozíciu pravej ruky videného agenta

	getLfoot		Vector3D	vráti pozíciu ľavej nohy videného agenta
	getRfoot		Vector3D	vráti pozíciu pravej nohy videného agenta
	getNumber		int	vráti číslo videného agenta
	getAbsoluteRotation		double	vráti absolútnu rotáciu videného agenta vzhľadom na os x podľa polohy jeho nôh
	getRelativeRotation		double	vráti relatívnu rotáciu videného agenta vzhľadom na agenta, ktorý ho vidí
	getDistanceFromBall		double	vráti vzdialenosť videného agenta od lopty
	getIsInRange		boolean	zistí či je videný agent v dosahu streľby na bránku
WorldModel	processNewServerMessage	ParsedData		identifikuje videných hráčov a loptu
	calculateBallPosition		ParsedData	nastaví relatívnu pozíciu lopty
	getBall		DynamicObject	vráti objekt lopty ako dynamický objekt
	getTeamPlayers		List<Player>	vráti zoznam videných spoluhráčov
	getOpponentPlayers		ArrayList <Player>	vráti zoznam videných protihráčov
	ballAfterAction	Annotation	DynamicObject	vráti predikciu budúcej polohy lopty vypočítanú na základe anotácie k pohybu
	isInRange	Player		vypočíta či je uvedený videný hráč v dosahu streľby na bránku
LowSkill	name		String	Názov LowSkill-u
	initialPhase		String	Začiatočná fáza LowSkill-u
	activePhase		String	Aktívna fáza LowSkill-u
	executeFinalisation			Naštartuje finalizáciu Low Skillu, nastaví sa finalizačná fáza
	reset			Nastaví všetky atribúty na iniciálnu hodnotu, ako na začiatku pohybu

	step			Nastaví sa nasledujúca fáza pohybu, ak je pohyb vo finálnej fáze, vykoná sa finalizácia
	canFinalize		boolean	Zistí či je možná finalizácia pohybu
LowSkills	addSkill	LowSkill		Vloží LowSkill do cache zoznamu LowSkill-ov
	exists	String		Zistí či je LowSkill so zadaným názvom v cache zozname LowSkillov
	get	String	LowSkill	Vráti LowSkill podľa zadaného mena
	reset			Nastaví aktívnu fázu všetkých LowSkill-ov v cache zozname na null
	getAll		List<LowSkill>	Vráti zoznam všetkých LowSkill-ov nachádzajúcich sa v cache zozname
Phase	name		String	Meno fázy
	next		String	Ďalšia fáza v poradí vykonávania pohybu, v prípade, že je isFinal nastavené na true, vynecháva sa
	effectors		List<EffectorData>	Zoznam pohybov kĺbov vykonávaných paralelne
	duration		double	Čas, ktorý by mala fáza trvať zaokrúhlený na násobok 20 milisekúnd
	finalizationPhase		String	Atribút finalizácie validný iba v prípade, že isFinal je nastavené na true
	isFinal		boolean	Určuje či je daná fáza posledná z cyklu fáz pohybu, ak áno, vykonávanie cyklu môže byť prerušené
	skipIfFlag		SkipFlag	Ak je true, táto fáza sa nevykonáva
	setTrueFlag		SkipFlag	LowSkill nastavuje na true, keď sa fáza

				úspešne dokončila
	setFalseFlag		SkipFlag	LowSkill nastavuje na false pred vykonaním tejto fázy
Phases	addSkill	Phase		Vloží fázu do cache zoznamu fáz
	exists	String		Zistí či je fáza so zadaným názvom v cache zozname fáz
	get	String	Phase	Vráti fázu podľa zadaného mena
	reset			Vymaže cache zoznam fáz
	getAll		List<Phase>	Vráti zoznam všetkých fáz nachádzajúcich sa v cache zozname
HighSkill	name		String	
	state		HighSkill State	Stav, v ktorom sa HighSkill nachádza(INITIAL_STATE,EXECUTING_STATE, FINALIZING_STATE, END_STATE)
	execute			Vykoná sa HighSkill
	pickLowSkill		LowSkill	Vráti ďalší LowSkill, ktorý sa má v rámci daného HighSkill-u vykonať, ak sa vráti null, vykoná sa finalizácia aktuálneho LowSkill-u a HighSkill sa ukončí
	checkProgress			Pokiaľ sa agent nachádza v neočakávanej polohe počas vykonávania HighSkill-u, tento sa ukončí

1.4.Predikcia lopty

Predikcia pohybu lopty sa ráta z rýchlosti a smeru lopty. Ako sa predikcia ráta nájdeme v bakalárke niektorého zo študentov, ktorými sme sa už zaoberali (od Ing. Lekavého), treba dať pozor na rôzne kopy, napríklad kop oblúčikom má iný faktor tlmenia, čiže treba rátať aj s vertikálnou a aj s horizontálnou zložkou, taktiež treba vyrátať pravdepodobnosť pohybu

súperovho hráča, aj s použitím ukladania jeho predchádzajúcich pohybov.

1.4.1. Analýza existujúceho riešenia tímu Androids

Tím Androids sa pokúsili o vytvorenie predikcie pohybov hráča aj lopty. Počas analýzy ich riešenia sme už v ich komentároch našli, že sa im nepodarilo túto úlohu dokončiť. Dokumentácia k týmto častiam kódu neexistovala. Preto sme sa snažili zistiť, čo ich metódy robia. Zistili sme, že implementované metódy vracajú úplne zlé čísla a ďalej sme sa snažili zistiť, kde spravili chybu. Po kontrole všetkých použitých volaní sme odhalili chybu v metóde getSpeed, ktorá bola implementovaná v triede dynamicObject. Rátanie tejto rýchlosti bolo úplne nevhodné. Odhalili sme aj ďalšie chyby, ktoré mali v implementácii. V počítaní predikcie lopty nerátali s trením a predikcia hráča fungovala iba v niektorých pripadoch.

1.4.2. Analýza výpočtov predikcie z diplomovej práce Petra Ertla

V diplomovej práci sa Bc. Peter Ertl zaoberal robotickým brankárom. Dôležitou súčasťou bolo teda aj predvídanie pohybu lopty. Navrhol tri modely:

1. model s konštantnou odporovou (trecou) silou
2. model s odporovou silou priamo úmernou rýchlosti objektu
3. model s oboma silami

Na základe testovania zistil, že prvý model bol veľmi nepresný, a v treťom modeli parameter pre konštantný odpor nemá žiaden dopad. Druhý a tretí model boli takmer ekvivalentné. Preto sa rozhodol pre použitie druhého modelu.

$v_x (t) = v_{x0}\, e^{-c1 * t}$ $\qquad$ $x(t) = (v_{x0} / c_1) \; * \; (1 - e^{-c1 * t})$

Hodnoty konštanty c sa pohybovali v rozmedzí 1,13 až 1,19, najčastejšie okolo 1,17.

1.5. Návod

1.5.1. Účel návodu

Keďže nedostatok dokumentácie je dlhodobým problémom v tomto projekte, rozhodli sme sa tento problém vyriešiť napísaním návodu ku kódu, ktorý by každému pomohol v zorientovaní sa v tých častiach kódu, na ktorých bežne nepracuje. S týmto sa spája doplnenie podrobnejších Javadoc komentárov k všetkým netriviálnym metódam, na ktoré bude návod vo

veľkej miere odkazovať. Takisto sa bude odkazovať z Javadoc komentárov do návodu.

1.5.2. Stav úlohy

Návod sa nachádza na wiki, skladá sa z viacerých wiki stránok, vstupnou je http://team17-11.ucebne.fiit.stuba.sk/wiki/N%C3%A1vod. V súčasnosti je popísaný agent a team framework so všetkými ich časťami, chýba ešte zhrnutie, ktoré dá čitateľovi miesto kde začať. Takisto treba updatnúť niekoľko detailov, ktoré sa priebežne stali neaktuálnymi.

Takisto treba vyriešiť, ako posunúť návod budúcoročným tímom, keďže server, na ktorom sa wiki nachádza prestane po skončení predmetu existovať. V súčasnosti máme skript na export statickej verzie wiki stránok, no lepšie asi bude poskytnúť aj dump databázy, aby sa dalo jednoducho vo wiki pokračovať.

1.5.3. Budúcnosť

Kód, ktorý návod popisuje, sa samozrejme v priebehu času mení, takže treba návod postupne aktualizovať a dopĺňať. Prinajmenšom treba označovať časti, ktoré už nie sú aktuálne. Preto je dobré umiestňovať veľkú časť popisu metód a tried práve do javadoc komentárov, kde je väčší predpoklad, že budú aktualizované zároveň s kódom.

1.6.High Skill – Hranie futbalu

Počas semestra sme vytvorili high skill kráčania za loptou, otáčania za loptou a dokopania lopty do brány. Bolo potrebné vylepšiť tento high skill tak, aby sme dokázali hrať futbal. A teda hráč musí vedieť nie len prísť za loptou a hľadať ju, ale hráč sa musí vedieť aj rozhodovať, aby sa hra podobala na futbal.

Hlavnými High Skillmi, ktoré sme vytvorili sú teda hľadanie lopty, kráčanie za loptou a dokopanie lopty do brány. Rozhodovanie, ktoré z nich sa majú použiť je implementované v pláne. V ňom sa teda rozhoduje na základe podmienok vyhodnotených z modelu sveta, čo hráč bude vykonávať, a teda ak leží, tak sa bude vykonávať vstávanie, ak nevidí loptu, tak sa bude vykonávať jej hľadanie, aj vidí loptu, tak sa ňou bude kráčať, ak je v dostatočnej vzdialenosti, tak sa na ňu bude nastavovať, ak má loptu pod nohou, tak do nej bude kopať. Bolo by potrebné vytvoriť funkcie pre podmienky, ak nevie pozície všetkých spoluhráčov, tak ich nájde, ak je hráč najbližšie k lopte tak ide k lopte, aj hráč nie je najbližšie k lopte tak kráča do formácie, prípadne ak to bude možné, tak ak hráč je druhý najbližší k lopte, tak bude si nabiehať na prihrávku od prvého.

2. Návrh

2.1. Zistenie pozície/stavu hráčov, ktorých agent vidí

Je potrebné určiť, v akom stave, resp. pozícii, sa hráč nachádza, teda či stojí alebo je na zemi(resp. sa dá povedať, že nás zaujíma či stojí alebo nie, pretože napríklad keď chceme prihrať videnému hráčovi, je potrebné vedieť či stojí, teda či sa mu oplatí adresovať nahrávku). Podľa súradníc získaných z častí tela videného agenta sme sa rozhodli vypočítať stav agenta a v konečnom dôsledku nám vznikli dva stavy videného agenta a nimi sú:

- stav, kedy agent stojí
- stav, kedy agent nestojí a je pravdepodobne na zemi(ale môže sa napríklad aj zdívhať)

V rámci dostupných údajov, ktoré môžeme získať zo strany servera o pozícii častí tela videných agentov sme sa rozhodli stav agenta určovať pomocou z-ovej súradnice polohy hlavy a chodidiel. Rozdiel týchto dvoch súradníc je smerodajný pre určenie stavu/pozície agenta.

2.2. Vyhodnocovanie herných situácií

Pre výber toho správneho správania pri hraní futbalu je potrebné rozlišovať okrem iného aj herné situácie, podľa ktorých sa agenti budú rozhodovať, ktorý high skill práve vykonávať. Preto boli navrhnuté niektoré základne herné situácie a spôsob akým sa budú tieto situácie rozlišovať.

Cieľom je na základe polohy hráčov a polohy lopty rozlišovať viaceré herné situácie, v ktorých sa tím nachádza. Základom bude určiť, v ktorej časti ihriska sa ako spoluhráči tak aj súperovi hráči nachádzajú. Samozrejme dôležitým faktorom je aj poloha lopty, a to ktorý tím ma loptu v držaní.

V prípade, že bude na zistenie hernej situácie potrebné získať „vlastníka“ lopty a túto loptu nebude mať v držaní ani jeden z tímov použije sa výpočet predikcie, ná základe ktorého sa určí, ktorý tím sa dostane k lopte ako prvý za určitý časový úsek.

Na základe aktuálnej polohy hráčov a toho, kto má práve loptu v držaní sme navrhli tieto základné herné situácie:

- *OffensiveSituation* – sme na súperovej polovici a loptu máme v držaní
- *DeffensiveSituation* – súper je na našej polovici a má loptu v držaní
- *StartAttackSituation* – sme na vlastnej polovici a mám loptu v držaní

- *EnemyStartAttackSituation* – súper je na svojej polovici a má loptu v držaní

V rámci situácií, keď my alebo súper zakladá útok (*StartAttackSituation* a *EnemyStartAttackSituation*) chceme ešte rozlišovať na základe polohy protihráčov, či je mužstvo pri jeho zakladaní pod tlakom alebo nie.

2.3. Vytvorenie hernej formácie

Tak ako v reálnom futbale existujú nejaké základne herné formácie, tak by mali existovať aj v robotickom, aby mali všetci hráči určené svoje miesta. To by malo zaručiť aby sa nestalo, že všetci hráči budú na jednej malej časti ihriska a zvyšok ihriska ostane neobsadený.

Pôvodne sme chceli navrhnúť aby hráči obsadzovali pozície vo formácií podľa toho, ku ktorej sú najbližšie. Tu by však vznikali problémy keďže hráč má informácie iba o spoluhráčovi, ktorého vidí a to by znamenalo, že by sa častokrát stávalo, že viacerí hráči by chceli obsadiť jednu pozíciu.

Preto sme navrhli priraďovanie pozícií vo formácií podľa ID (číslo dresu) jednotlivých hráčov. To znamená že pozície sú očíslované a hráč podľa svojho čísla bude vedieť, na ktorú pozíciu ma ísť. Pozície bud taktiež zoradené podľa priority, ktorá určuje poradie obsadzovania týchto pozícií, čo zabezpečí, že jeden zoznam pozícií bude možné používať pre rôzny počet hráčov.

2.4. Zistenie vhodnosti prihrávky

Pri hraní futbalu dochádza k momentom, kedy je hráč, ktorý práve kontroluje loptu v situácii, v ktorej je lepšie posunúť loptu spoluhráčovi vo vhodnejšej pozícii. Môže sa jednať jednak o spoluhráča bližšie k bránke, alebo spoluhráča nepokrytého protihráčmi, či jedoducho o snahu rýchlejšie presunúť hru. V takejto situácii hráč potrebuje vyhodnotiť aktuálnu situáciu na ihrisku ako aj jej možnú zmenu počas presunu lopty, aby prihrával naozaj spoluhráčovi v lepšej pozícii a aby počas prihrávky súper loptu nedokázal zachytiť.

2.5. Podpora viacerých hráčov

Podpora viacerých hráčov – treba spraviť také zmeny, aby bolo možné jednoducho spúšťať viacerých hráčov naraz.

2.5.1. Možnosti implementácie

Potreba tejto podpory vyplynula z faktu, že sme sa dostali do fázy, kde implementujeme

funkcionalitu ktorá súvisí napríklad s obchádzaním iných hráčov, formáciami a pod., kde na testovanie treba mať viacero hráčov naraz. Doteraz sa spúšťali ručne, čo je veľmi zdĺhavé a málo flexibilné. Preto je potrebné zabudovať túto možnosť priamo do GUI a spúšťanie a ovládanie maximálne uľahčiť.

Rozhodoval som sa medzi dvoma možnosťami ako túto funkcionalitu implementovať:

- do kódu agenta – jeden proces „agenta" by mohol vlastniť viacero agentov na serveri, pričom GUI by bolo upravené tak aby toto odrážalo. Výhodou by bola jednoznačne oveľa vyššia rýchlosť pridávania agentov a celkovo menšia náročnosť na zdroje, keďže značná časť z nich by mohla byť zdieľaná. Nevýhodou by ale bola potreba rozsiahlych úprav kódu, napríklad singleton triedy pre modely by museli byť upravené na triedy s viacerými inštanciami – jednu pre každého agenta. Toto by výrazne ovplyvnilo aj budúci vývoj kódu agenta. Navyše na súťaži môže aj tak jeden proces ovládať len jedného agenta, takže takéto riešenie by bolo užitočné len na testovanie.
- do test frameworku – test framework by spúšťal samostatné procesy agentov – toto bolo v test frameworku plánované už predtým, takisto existovala určitá implementácia, tá však bola do značnej miery nevyhovujúca, alebo presnejšie povedané, nekompletná. Jedinou nevýhodou tohto riešenia je veľmi dlhý čas potrebný na spustenie procesu agenta, ktorého veľmi spomaľuje závislosť na JRuby.

Rozhodol som sa teda implementovať túto funkcionalitu do test frameworku. Veci týkajúce sa GUI pre podporu tejto správy popisujem v dokumentácií k úlohe Vylepšenie GUI.

2.6. Vylepšenie GUI

Je potrebné zmeniť GUI tak, aby vyhovovalo momentálnym potrebám práce so serverom a hráčmi. Staré GUI test frameworku nemalo mnoho funkcií, preto nebolo príliš organizované – všetky veci boli nahádzané vedľa seba. Členovia druhého tímu medzitým začali vývoj iného GUI. Toto malo rozhranie s tabmi a logovacím výstupom na spodku. Z veľkej časti však bolo nefunkčné (napr. logovali sa veci zo Swingu, no nie niektoré veci z test frameworku; nescrollovalo sa, nebolo vidieť celý výstup monitorovania...) a jeho kód bol veľmi neprehľadný (žiadne komentáre, funkcionalita ovládacích prvkov implementovaná inline priamo medzi hromadou kódu vygenerovaného GUI builderom...).

Toto GUI som zobral ako základ a začal som ho výraznejšie upravovať a dopĺňať do neho

novú funkcionalitu. GUI je tvorené nástrojom Jigloo SWT/Swing GUI Builder, no kód ktorý generuje sa dá parsovať napr. aj GUI builderom dodávaným priamo s Eclipse (nie je však dobré miešať dokopy viac nástrojov).

2.7. Plánovanie trajektórie

Cieľom tejto úlohy je umožniť agentovi plánovanie trajektórie pohybu z bodu A (reprezentovaného súradnicami ihriska a natočením vzhľadom na osi ihriska) do bodu B na základe dostupných anotácií k pohybom.

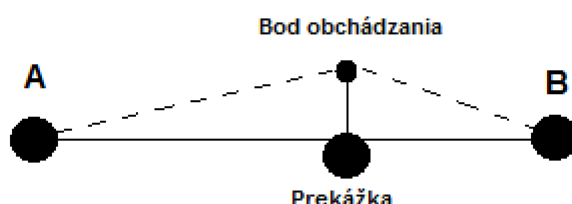
Agent bude plánovať trajektóriu v troch fázach. V prvej fáze vypočíta uhol medzi pôvodným smerom natočenia a vektorom k bodu B. Z dostupných anotácií vytvorí rad pohybov, po ktorých vykonaní sa otočí o požadovaný uhol pričom sa vždy snaží otočiť o čo najväčší možný uhol. V druhej fáze naplánuje agent presun z bodu A do bodu B. Na presun na dlhšie vzdialenosti bude určený jeden pohyb, ktorý bude najefektívnejší z pohľadu rýchlosti presunu a minimálnej odchýlky (aby bol pohyb priamočiary). Keď sa agent priblíži k bodu B tak, že vzdialenosť medzi ním a bodom B bude menšia a ako dĺžka kroku pohyb určeného na presun, tak z dostupných anotácií vytvorí rad pohybov tak , aby sa dostal presne do bodu B. Tretia fáza pohybu je rovnaká ako prvá, pričom sa vypočíta uhol medzi vektorom pohybu a požadovaným koncovým natočením. Bude zavedená prípustná odchýlka pri otáčaní a presunu, aby sa zabránilo vykonávaniu príliš veľkého počtu pohybov, keď sa bude chcieť agent otočiť o požadovaný uhol alebo sa dostať na požadované súradnice.

Pri výpočtoch trajektórie je určená povolená odchýlka, aby sa zabránilo vykonávaniu príliš veľkého počtu pohybov, ak by chcel agent dosiahnuť presných súradníc alebo natočenia. Aby sa dosiahlo čo najefektívnejšej trajektórie z pohľadu rýchlosti jej vykonania, tak je týchto povolených odchýlok zavedených viacero. Trajektória sa pritom vypočíta pre každú odchýlku zvlášť a vyberie sa tá, ktorej súčet dôb vykonania pohybov je najkratší. Pri akejkoľvek odchýlke sa v rámci priamočiareho pohybu v pravidelných vzdialenostných intervaloch upravuje rotácia agenta tak, aby smeroval k bodu B.

2.8. Vylepšenie plánovača

Cieľom vylepšenia plánovača trajektórie je vytvorenie plánu sekundárnej trajektórie (tzv. plán B), ktorá sa bude vykonávať, pokiaľ bude čas jej vykonania kratší ako čas vykonania primárnej trajektórie. Ďalšou časťou úlohy je návrh a implementácia obchádzania prekážok na trajektórii.

Pokiaľ sa zistí, že sa na trajektórii nachádza prekážka (v podobe hráča), tak bude táto trajektória prepočítaná. Vypočíta sa bod obchádzania, ktorý sa bude z hráčovho pohľadu nachádzať naľavo od prekážky a vypočíta sa trajektória prechádzajúca týmto bodom. Pokiaľ sa bude aj na tejto trajektórii nachádzať prekážka, tak sa vypočíta bod obchádzania napravo od pôvodnej prekážky a k nemu príslušná trajektória. Ak sa aj na tejto trajektórii nachádza prekážka, tak sa rovnakým spôsobom vypočítavajú body obchádzania pre ďalšie zistené prekážky dovtedy, kým sa nenájde voľná trajektória. Bod obchádzania taktiež nesmie ležať mimo ihriska.



Obrázok č.2: Obchádzanie prekážky zľava

Bod obchádzania sa vypočíta pomocou vektora kolmého na vektor trajektórie so začiatkom v súradniciach prekážky o dĺžke 0,5 metra, čo je vzdialenosť dva a pol násobku polomeru priemetu robota na hraciu plochu. Táto vzdialenosť by mala byť postačujúca na obídenie hráča.

2.9.Predikcia lopty

Rozhodli sme sa implementovať metódy predikcii do balíka, ktorý na to vytvoril tím Androids – sk.fiit.jom.agent.models.prediction. V tomto balíku sa nachádza trieda Prophet.java, kde boli implementované chybné predikcie, ktoré sme vložili na koniec triedy do komentára.

Pre naše riešenie bude potrebné triedu spúšťať vždy pri primaní informácii zo servera. Keďže to sa vykonáva veľmi často, predikcie budeme počítať len každú sekundu, na základe času hry, uloženého v premennej EnvironmentModel.Game_Time.

Budeme musieť vytvoriť tri metódy:

1. predikcia pohybu spoluhráčov – calcTeamPlayerPrediction,
2. predikcia pohybu súperov – calcOpponentPlayerPrediction,
3. predikcia pohybu lopty – calcBallPrediction.

Na počítanie predikcii hráčov a lopty použijeme informácie z WorldModelu, z kade získame pozície hráčov a lopty. Pre výpočet predikcie hráčov a lopty budeme používať pozície vo vektorovom tvare. Na počítanie s vektormi použijeme matematické metódy, ktoré

sú už v projekte zahrnuté – add, divide, multiply, subtract.

2.10. Kráčanie za loptou

V prvých dvoch šprintoch sme vytvorili high skill kráčania za loptou. Je potrebné vylepšiť tento high skill tak, aby sme dokázali hrať futbal. A teda hráč musí vedieť nie len prísť za loptou, ale aj ju hľadať keď ju nevidí.

V existujúcom riešení bolo upravené spracovanie high skillov na štyri stavy – úvodný, vykonávanie, finalizovanie a skončený. Finalizácia fázy spočíva buď v skončení, alebo výbere ďalšieho skillu. Hráč dokáže prísť k lopte a kopnúť do nej. V správaní hráča sa objavilo ešte niekoľko chýb. Pre našu úlohu je hlavná chyba, že keď hráč loptu nevidí, tak je nehľadá, je iba náhoda, ak ju znova uvidí, tak ide k nej.

Podstatným pre návrh riešenia bolo zistenie, ako presne momentálna situácia funguje, čo bolo v neokomentovaných ruby skriptoch dosť problémové. Po niekoľkonásobnom testovaní sme prišli na to, že by bolo vhodné v prvej fáze vytvoriť samotnú funkcionalitu, a to otáčanie za loptou ak ju nevidíme. Na toto by sme mohli vytvoriť metódu, ktorá by nám vracala, ako dlho nevidíme loptu v sekundách. Na otáčanie použijeme hotový pohyb otáčania o dvadsať stupňov, bolo by však vhodné, keby bol vytvorený pohyb otáčania o deväťdesiat stupňov, aby sme na tri až štyri otočenia pokryli celých 360 stupňov. V druhej fáze by bolo vhodné vyčleniť funkcionalitu z plánu zvlášť do high skillu, aby sa v prípade potreby dala použiť aj na iné účely.

2.11. Optimalizácia pohybov

Tím Androids vytvoril pomerne dosť pohybov, ktoré využívali pri svojom minuloročnom projekte. Nimi vytvorené pohyby boli funkčné, no niektoré potrebovali mále úpravy, či už v súvislosti s ich rýchlosťou, alebo stabilitou. Keďže táto úloha bola podelená medzi oba tímy, tak mojou úlohou bolo optimalizovanie pohybov otáčania o 5°, 20° i 90°.

Po analyzovaní aktuálneho stavu týchto otočení som prišiel k záveru, že tieto pohyby sú navrhnuté a implementované dobre, takže nebolo nutné ich nejak zásadne meniť, či vytvárať úplne nové pohyby. Niektoré ich fázy sú i napriek tomu zbytočne dlhé a niektoré za sebou idúce fázy sa dali spojiť do jednej a tým tiež znížiť celkový čas vykonania pohybu.

2.12. Vytvorenie úkrokov

Po analýze pohybov som prišiel k záveru, že sa neoplatí ich upravovať, ale radšej vytvoriť nové pohyby. Tieto pohyby vznikli z pohybu otočenia o 20° (turn_right_cont_20). Podarilo sa mi vytvoriť pohyb úkrokov (stepright_new a stepleft_new), ktoré sú výrazne rýchlejšie ako pôvodné pohyby. Tak isto som vytvoril pohyb menších úkrokov (stepright_new_smaller a stepleft_new_smaller), ktoré sa môžu využiť pri potrebe malého posunu vbok a presnejšieho nastavenia sa hráča.

Oba typy úkrokov je možné vylepšiť, prípadne vytvoriť iný spôsob ich vykonania. Pre naše účely však postačujú aktuálne vytvorené pohyby.

2.13. High Skill – Hranie futbalu

Ako prvú sme sa rozhodli implementovať metódu nájdenia spoluhráčov. Ak vieme zistiť polohy všetkých spoluhráčov, vieme sa lepšie rozhodovať, a nebude dochádzať zbytočným konfliktom. Samotné hľadanie spoluhráčov by malo prebiehať otáčaním sa hráča.

Ako druhú budeme potrebovať metódu, ktorá vracia či je hráč najbližšie svojím postavením k lopte alebo nie. Táto metóda nám jasné urči, či má hráč kráčať k lopte a bojovať o ňu, alebo či sa má postaviť do formácie. Taktiež sme navrhli metódu, ktorá vyberá hráča, ktorý je druhý nebližší k lopte, ten si bude nabiehať medzi hráča, ktorý smeruje za loptou a teda je prvý najbližší k lopte a bránu, pretože hráč sa bude snažiť kopať loptu smerom na bránu.

2.14. Vedenie lopty

Základom bude vytvorenie pohybu pre vedenie lopty. Tento pohyb by mal byť vytvorený tak, aby si agent loptu predkopával pred sebou. Zároveň pri tomto pohybe lopta musí byť stále v blízkosti agenta.

Tento pohyb sa použije ako základ pre vytvorenie high skillu, ktorý zabezpečí celkové vedenie lopty. Celkové vedenie musí obsahovať výber vhodného pohybu tak, aby mal agent loptu stále pred sebou.

3. Implementácia

3.1. Zistenie pozície/stavu hráčov, ktorých agent vidí

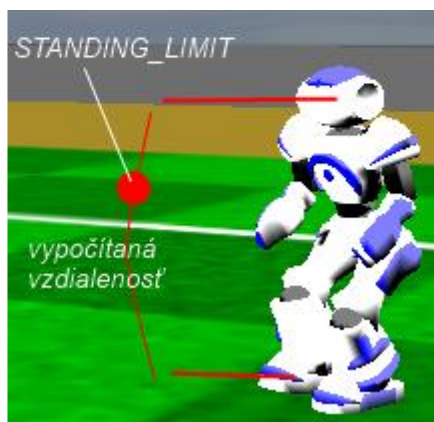
Podľa návrhu bola implementovaná logika rozhodovania pre stav/pozíciu videného agenta. V riešení sa po určitej časovej jednotke neustále obnovujú informácie, ktoré agent vidí, a teda aj stav videných agentov. V budúcnosti by sa časová jednotka obnovovania stavu hráčov mala presne stanoviť, pretože nie je potrebné, aby hráč vypočítaval tieto údaje nestále, keďže veľa podobných výpočtov potom zbytočne spomaluje činnosť agenta.

V triede AgentInfo boli pridané dva zoznamy a to zoznam stavov spoluhráčov a protihráčov, v oboch sa k hráčom pristupuje pomocou ich identifikačného čísla, ktoré by mali mať v tíme jednoznačné, keďže to predpisujú aj pravidlá futbalu. V zoznamoch samozrejme nie sú všetci hráči, ale len tí, ktorých agent vidí. Do zoznamov sa postupne pridávajú záznamy o jednotlivých hráčoch, pričom sa pre každého počíta jeho stav, v akom sa nachádza. Ako sa uvádza v návrhu, stav je určený pomocou rozdielu Z-ovej súradnice hlavy a chodidiel, pričom rozhodovanie a stavy sú nasledovné:

- STANDING – hráč je v tomto stave(teda stojí), ak platí aspoň jedna z podmienok:
  - Z(hlava) – Z(ľavé chodidlo) > STANDING_LIMIT
  - Z(hlava) – Z(pravé chodidlo) > STANDING_LIMIT

    , kde Z(hlava) je Z-ová súradnica hlavy, Z(chodidlo) je Z-ová súradnica daného chodila a STANDING_LIMIT je konštanta
- LYING – hráč je v tomto stave(teda nestojí), ak neplatí ani jedna z vyššie uvedených podmienok

Do výpočtu bola zahrnutá aj konštanta STANDING_LIMIT, podľa ktorej určujeme či daný stav spadá do kategórie STANDING alebo LYING. Táto konštanta sa v prípade potreby v budúcnosti môže zmeniť, pokiaľ by z nejakého dôvodu server posielal odlišné hodnoty pre jednotlivé súradnice v oblasti osi Z. Momentálne je veľkosť konštanty odvodená z niekoľkých desiatok pokusov a dosahuje približne hodnotu uvedenú na obrázku nižšie.



Obrázok č.3: Vzdialenosť rozdielu polohy hlavy a chodidiel na osi Z. Zobrazenie konštanty STANDING_LIMIT na vypočítanej vzdialenosti

3.1.1. Testovanie

Pri testovaní sa boli dosiahnuté očakávané výsledky správneho určenia polohy videných agentov, pričom v približne 3-4% prípadov vznikla situácia, že agent zle určil stav videného hráča. To je pravdepodobne spôsobené určitými chybami, ktoré sú vnášané do údajov na strane servera.

3.2. Vyhodnocovanie herných situácií

V rámci implementácie sme sa snažili vytvoriť spôsob ako rozlišovať všetky navrhnuté herné situácie. Pri situáciach, kedy útočíme resp. kedy bránime (*OffensiveSituation* a *DeffensiveSituation*) sme pridali aj určenie v akom počte útočíme, proti akému počtu súperových obrancov, resp. aký počet súperových útočiacich hráčov útočí proti akému počtu našich brániacich hráčov.

Pri určovaní herných situácií sa ako prvé vyhodnotí vlastník lopty, ak nikto loptu v danej chvíli nevlastní zavolá sa predikcia. Následne sa zavolá ďalšia metóda, ktorá podľa polôh hráčov, ktorí sú uložení v zozname vyhodnotí na ktorej polovici sa nachádzajú a z toho počet útočníkov a obrancov. V rámci situácií zakladania útokov sa ďalej na základe polohy hráčov brániaceho mužstva určí, či je mužstvo zakladajúce útok pod tlakom súpera alebo nie (*NoPressure* alebo *UnderPressure*). Aj v týchto prípadoch sa volá aj predikcia polohy hráčov.

Za vlastníka lopty sa považuje hráč, od ktorého je lopta vo vzdialenosti menej ako 1 meter. Hráč je pod tlakom ak je od neho súper vo vzdialenosti menej ako 2 metre.

Najväčším problémom je, že v súčasnom stave projektu hráči nedokážu medzi sebou komunikovať. Preto bola implementácia zatiaľ vytvorená tak ,že polohy hráčov aj lopty vyhodnocuje jeden hráč, ktorý má, ak je to možné všetkých ostatných pred sebou. To bol aj jediný spôsob ako otestovať funkčnosť vytvoreného rozlišovania herných situácií.

3.3. Vytvorenie hernej formácie

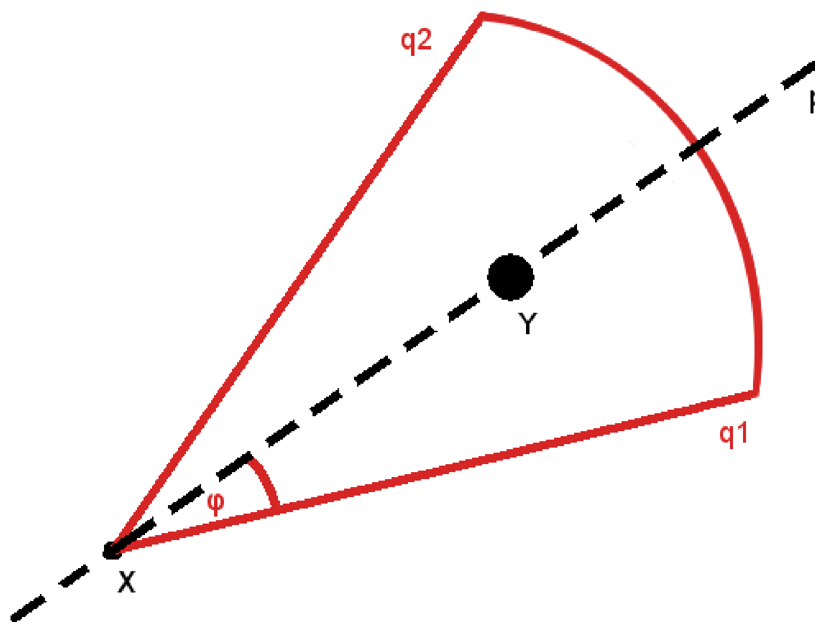
V rámci implementácie bolo najskôr treba vytvoriť high skill, ktorý by zabezpečoval prejdenie hráča na určitú pozíciu, v tomto prípade pozícia vo formácií. Tento high skill bol vytvorený úpravou high skillu prejdenia k lopte (viď. Kapitola TODO: tu potom napíšte kapitolu v ktorej to bude). Funkcia vytvorená v ruby *goto_formation_position*, zabezpečí prejdenie hráča na určitú pozíciu, resp. do okruhu 0,5 metra od pozície. Táto funkcia je volaná v plánovači, až kým hráč nedosiahne požadovanú pozíciu. To či ju dosiahol sa vyhodnocuje na základe pozície hráča, ktorú nám vráti metóda v triede AgentInfo *getPosition()*.

Jednotlivé pozície sú uložené v poly typu 3DVector. Ich poradie v akom sú uložené v tomto poli určuje zároveň aj priority pozícií (prvá pozícia – najvyššia priorita) a aj to hráč s ktorým číslom má túto pozíciu zaujať (prvá pozícia – hráč s číslom 1).

Súradnice *y* jednotlivých pozícií vo formácii sú pevne dané a súradnica *x* (po dĺžke ihriska) je vypočítavaná podľa polohy lopty. Tu sme museli vytvoriť obmedzenia tak, aby sa nestalo, že hráči sa dostanú mimo hranice ihriska. To znamená, že ak sa lopta dostala do malej blízkosti k bránke hráči vytvoria formáciu pred bránkou na tej polovici, na ktorej je lopta.

3.4. Zistenie vhodnosti prihrávky

Pri vyhodnotení vhodnosti prihrávky sa berie do úvahy miesto odkiaľ má prihrávka smerovať a miesto kde má prihrávka skončiť. Z východzieho bodu sa následne zoberie výsek kružnice s osou precházajúcou cieľovým bodom prihrávky. Uhol výseku záleží od nastaveného uhla φ a od polomeru kružnice, ktorý je momentálne stanovený na 1,5 násobok dĺžky prihrávky. Východzí bod prihrávky je na obrázku nižšie znázornený ako bod X a cieľový bod ako bod Y. Následne sa v tomto území, ktoré je na obrázku nižšie vyznačené červenou farbou zisťuje pritomnosť súperových hráčov. V prípade, že sa v území nenachádza ani jeden protihráč, prihrávka sa vyhodnotí ako vhodná, v opačnom prípade ako nevhodná.



Obrázok č.4: Zistenie vhodnosti prihrávky

Do budúcna by sa do vyhodnocovania vhodnosti prihrávky mohli pridať ďalšie prvky ako napríklad vyhodnotenie stavu súperových hráčov, keďže napríklad hráči, ktorí ležia na ihrisku s veľkou pravdepodobnosťou prihrávku neohrozia ak teda neležia priamo v jej dráhe. Takisto aj hráči otočení chrbtom. Tiež by bolo možné upraviť tvar územia, ktoré sa pri prihrávke vyhodnocuje napríklad na tvar pripomínajúci kvapku, kde by sa brala do úvahy nie vzdialenosť od vychodzieho bodu, ale od cieľového bodu.

3.5. Zadefinovanie High skillov

Pri tvorbe základného plánu hry, podľa ktorého má agent postupovať, je nutné zadefinovať aspoň najjednoduchšie high skilly a vytvoriť aspoň ich jednoduché prevedenie. Medzi high skilly, ktoré boli identifikované ako základné, boli určené nasledovné:

- **GetUp** – pre postavenie sa zo zeme
- **Localize** – pre nájdenie lopty na ihrisku
- **Walk** – pre chôdzu za na pozíciu na ihrisku
- **Turn** – pre natočenie sa na bránku
- **Kick** – pre kopnutie do lopty

3.5.1. GetUp

Hráč vyhodnotí situáciu, kedy sa nachádza na zemi a následne sa pomocou tohoto high skillu postaví opäť na nohy.

Vyhodnocujú sa dve najpravdepodobnejšie pozície v ktorých môže ležať a to je pozícia pri ktorej hráč leží na chrbte a druhá, pri ktorej leží na bruchu. Pre tieto sú zadefinované pohyby na postavenie sa. Ak sa hráč nachádza v inej polohe, pokúsi sa vstať ako z polohy na chrbte, keďže pri vstávaní z chrbta sa najskôr preklápa na brucho a až potom vstane, je veľká pravdepodobnosť, že sa mu podarí vstať.

3.5.2. Localize

Pri situácii, kedy hráč nevidí loptu dlhšie ako 5 sekúnd sa zavolá nasledujúci high skill, pričom sa agent otáča okolo osi, až kým opať nevidí loptu.

3.5.3. Walk

Hr áč pri tomto high skille kráča na určitú pozíciu, pričom dokáže meniť smer. Pri dosiahnutí cieľovej pozície, prípadne páde automaticky ukončí vykonávenie high skillu.

Aktuálne je tento high skill rozdelený na dve fázy a to fázu, kedy je hráč ďalej od lopty a fázu, kedy je pri lopte blízko.

V prvej spomínanej fáze sa snaží k lopte priblížiť. Podľa pozície lopty sa hráč buď natáča k lopte, alebo k nej kráča, keď sa k lopte dostane na vzdialenosť stanovenú ako blízku prechádza do hruhej fázy a zas naopat.

V druhej fáze má priestor okolo seba rozdelený na 9 zón a podľa toho, kde sa lopta nachádza robí pohyby, aby sa dostal na vhodnú pozíciu a bol následne schopný napríklad vystreliť. Pri pohybe v blizkosti používa iba kráčanie vpred, vzad a do strán..

3.5.4. Turn

Pri potrebe natočiť sa na bránku, napríklad pri strele, sa hráč pomocou akýchsi úkrokov postupne smeruje až kým nedosiahne vhodné natočenie. Tieto úkroky sú navrhnuté tak, aby sa hráč dokázal nasmerovať a pritom sa priveľmi nevzdialil od lopty. V ideálnom prípade, by mal zostať vo vhodnej pozícii na strelu.

3.5.5. Kick

Kopnutie lopty. Hráč pri vhodnej pozícii lopty a správnom natočení na brávku vyhodnocuje situáciu ako vhodnú na kopnutie do lopty. Podľa vzdialenosti od lopty sa rozhoduje, či je ešte potrebné k lopte dokročiť, ak áno urobí to. V prípade, že na loptu dočiahne, rozhoduje sa ktorou nohou vystrelí podľa toho ako pred ním leží lopta

3.6. Prispôsobenie pohybov na reťazenie

Kód riešiaci spracovanie high skillu v agentovi bol nepochopiteľnou spleťou asi piatich rôznych premenných a ich všemožných kombinácií bez akéhokoľvek vysvetlenia. Prepísal som ho teda tak, aby používal stavy. High skill sa môže nachádzať v jednom zo 4 stavov (initial, executing, finalizing, ended) a spracovanie v každom z nich, ako aj prechody medzi stavmi sú pomerne priamočiare. Podrobnejšie som to popísal na wiki v sekcii http://team17-11.ucebne.fiit.stuba.sk/wiki/Pl%C3%A1novanie_a_vykon%C3%A1vanie_pohybov.

3.7. Podpora viacerých hráčov – úprava test framework-u

Test framework obsahuje triedu AgentManager slúžiacu na správu agentov - tých ktorých spustil test framework, aj tých, ktorí boli spustení externe a následne sa k nemu pripojili. Agenta reprezentuje objekt triedy AgentJim. Príkaz na spustenie agenta sa udáva v konfiguračných súboroch test frameworku.

Najprv bolo treba rozdeliť veľmi nejasne oddelené vlastnosti tímu agenta a strany na ktorej hrá. Sú to dve odlišné veci, no v kóde boli častokrát zmiešané dohromady, v dôsledku čoho test framework správne fungoval len s agentami v tímoch nazvaných Left a Right (čo sú aj názvy strán). Toto neskôr vyriešili členovia druhého tímu, ako dočasné riešenie som použil agentov s uvedenými názvami tímov.

Ďalej bolo treba odhadnúť voľné porty pre TFTP server agentov. Pri spustení viacerých agentov na jednom stroji totiž nemôžu používať ten istý port. Na to som vytvoril metódu getFreeTFTPPort v triede AgentManager, ktorá vráti najbližšie číslo väčšie alebo rovné 3070, ktoré nie je doteraz obsadené na danej IP adrese. Napríklad ak už AgentManager má agentov s TFTP serverom na portoch 3070 a 3071, ktorý je pripojený z IP adresy 127.0.0.1, metóda getFreeTFTPPort pre IP adresu 127.0.0.1 vráti 3072. Pre inú IP adresu by ale vrátila 3070.

Podobne bolo treba určiť najnižšie číslo uniformy v danom tíme. Na to je metóda getFreeUniform. Bola pridaná podpora command line argumentov do agenta tak, aby sa dali upravovať ľubovolné settings podľa ich názvu. Takýto istý formát argumentov používal už predtým test framework. Agenti sa spúšťajú bez GUI. Bolo implementované zachytávanie štandardného výstupu agentov, pre možnosť jeho následného zobrazenia v GUI test frameworku.

3.8. Vylepšenie GUI

GUI nemá prvky umiestnené absolútne ale používa layouty. To umožňuje zväčšovanie a zmenšovanie, užitočné napr. pri potrebe mať naraz na obrazovke väčšie množstvo informácií. Okrem tabu s monitorovaním, ktorý umožňuje aj zmenu pozície lopty a agenta, a v zásade je kópiu starého GUI, je prítomný aj tab pre anotátor a tab pre správu viacerých agentov.

Na začiatok som implementoval len možnosť pridávať nových agentov. Po pridaní nového agenta sa GUI zablokuje až kým sa tento agent nepripojí. Trvá to pomerne dlhý čas.

Implementoval som triedu AgentComboModel, ktorá je modelom pre combo boxy obsahujúce vždy aktuálny zoznam pripojených agentov. Objekty tejto triedy sú vždy informované o nových agentoch od triedy AgentManager pomocou rozhrania IAgentManagerListener, ktoré som takisto pridal v tejto úlohe. Jeden takýto combo box som umiestnil pri ovládanie pozície agenta.

3.9. Správa viacerých agentov

Bolo treba vytvoriť podporu pre správu a monitorovanie viacerých agentov.

3.9.1. Úpravy v GUI

Pridal som podporu preplánovania (vrátane zmeny plánu), nového načítania pohybov a odoberania agentov. Ovládanie agenta z test frameworku sa tak vyrovnalo ovládaniu agenta priamo z jeho GUI – všetko čo sa dá na agentovi ovládať z jeho GUI sa teraz dá ovládať aj z test frameworku. Tieto funkcie sú riešené vykonaním určitého ruby kódu na strane agenta. Tento ruby kód má plne pod kontrolou test framework, čo nie je dobré, keďže to porušuje princíp enkapsulácie – test framework je tým priamo závislý na konkrétnej vnútornej implementácii agenta.

Ďalej som pridal zobrazenie výstupu vybraného agenta.

Okrem toho bolo pridaných viacero komentárov k triedam týkajúcich sa podpory viacerých agentov.

3.9.2. Úpravy v agentovi

V agentovi bolo treba zmeniť pripájanie sa k test frameworku tak, aby sa pripojil len vtedy keď už má od servera priradenú stranu. Inak si test framework myslel že hrá stále na strane Left.

3.10. *Profilovanie a optimalizácia kódu*

Pri paralelnom spustení viacerých agentov ide server pomaly. Mal som sa pokúsiť znížiť záťaž kladenú agentom na server.

Profilovanie som vykonal nástrojom VisualVM. Vyplynulo z nich, že cca. 90% času agent čaká na blokujúce operácie (prijímanie/odosielanie dát na server), čo sa dá považovať za idle stav z dôvodu toho, že už nič iné nemá na práci. Väčšinu zvyšného času zabrala metóda parseEval z JRuby. Je teda zrejmé, že vykonávanie kódu obsiahnutého v ruby skriptoch (plánovač, high skilly) zaberá najviac času. Nateraz sme sa rozhodli príliš sa optimalizáciou nezaoberať.

3.11. *Plánovanie trajektórie*

Bola vytvorená trieda TrajectoryPlanner. Po vytvorení objektu tejto triedy sa vytvoria dva zoznamy anotácií. Jeden zoznam sa skladá z anotácií k pohybom typu „rotation“ , ktoré budú využívané pre otáčanie agenta okolo vlastnej osi a druhý zoznam tvoria anotácie k pohybom typu „walk“ určené na presun po ihrisku. Oba zoznamy sú usporiadané podľa hodnoty podstatného atribútu (rotation – rotácia okolo osi z, walk – dĺžka posunutia po osi x) od najvyššej po najnižšiu. Metóda plan() je zodpovedná za naplánovanie trajektórie. Ako prvé vypočíta potrebné uhly otáčania a dĺžku presunu po ihrisku. Následne volá metódy rotate() a walk(), ktoré majú ako argument uhol alebo vzdialenosť a sú zodpovedné za samotný výber pohybov, ktoré budú viesť k želanému otočeniu/posunutiu. Obe metódy potom vkladajú túto postupnosť pohybov do zásobníka typu FIFO, ktorý predstavuje samotnú naplánovanú trajektóriu.

3.12. *Vylepšenie plánovača*

Výpočet sekundárnej trajektórie prebieha zároveň s výpočtom primárnej trajektórie, s tým, že akceptovateľná odchýlka primárnej trajektórie je 0,5 stupňa a sekundárnej 2,5 stupňa. Pri týchto odchýlkach sa pri experimentovaní dosahovali časovo najvýhodnejšie trajektórie.

Bola pridaná trieda *Obstacles,* pre prácu s prekážkami v trajektórii. Hlavnou metódou tejto triedy je *checkIntersection()*, ktorá kontroluje prienik trajektórie so všetkými prekážkami uvedenými formou zoznamu ako argument tejto metódy. Pokiaľ je nájdený prienik, tak táto metóda vráti súradnice prekážky. Ďalej obsahuje metódy na výpočet bodu obchádzania prekážky zľava a sprava. Tento bod sa vypočíta pomocou vektora kolmého na vektor trajektórie so začiatkom v súradniciach prekážky o dĺžke 0,5 metra, čo je vzdialenosť dva

a pol násobku polomeru priemetu robota na hraciu plochu. Táto vzdialenosť by mala byť postačujúca na obídenie hráča. Trieda *Obstacles* obsahuje taktiež metódu *checkInfield(),* ktorá kontroluje, či sa daný bod nachádza vnútri hracieho poľa.

Bola pridaná trieda *Trajectory*, ktorá uchováva a sprístupňuje zoznam pohybov tvoriacich trajektóriu. Taktiež uchováva prijateľnú odchýlku danej trajektórie.

Výpočet sekundárnych trajektórií už neprebieha súčasne, ale pre každú prijateľnú odchýlku sa metóda *plan()* vykoná samostatne. Kvôli akceptovaniu ľubovoľnej odchýlky sa trajektória po prejdení ¾ vzdialenosti prepočítava a v prípade vychýlenia opravuje.

Bola pridaná trieda *TrajectoryRealTime,* ktorá získava informácie o pozícii a natočení agenta z inštancie triedy *AgentModel* a používa ich pri volaní konštruktora triedy *TrajectoryPlanner* ako začiatok trajektórie. Do triedy *Obstacles* bola pridaná metóda *getRealObstacles(),* ktorá z inštancie triedy *WorldModel* získava pozície všetkých hráčov na ihrisku a vracia ich ako zoznam prekážok.

Bol vytvorený súbor *trajectory.rb* predstavujúci high skill pre tvorbu trajektórie. Obsahuje triedu *Trajectory*, ktorej konštruktor obsahuje cieľové súradnice a natočenie. Po zavolaní tohto high skillu sa vytvorí inštancia triedy *TrajectoryRealTime* s argumentmi prenesenými z konštruktora. Následne sa v cykle získavajú názvy pohyb zo zásobníka predstavujúceho trajektóriu.

Trieda *TrajectoryRealTime* získava informácie pre výpočet trajektórie priamo z modelu agenta a modelu sveta. Konkrétne sú to aktuálne súradnice pozície hráča a aktuálne natočenie hráča, ktoré slúžia ako začiatok trajektórie a pozície všetkých hráčov na ihrisku, ktoré slúžia ako prekážky v trajektórii.

Plánovanie trajektórie je do veľkej miery závislé od presnosti anotácií k pohybom a od presnosti samotných pohybov. V súčasnom stave je plánovanie trajektórie pomerne nepresné (agent sa po vykonaní pohybov trajektórie väčšinou nenachádza v akceptovateľnej odchýlke od požadovanej pozície), takže môže slúžiť na hrubé priblíženie sa k požadovaným súradniciam.

Vytvorený high skill má potenciál byť v budúcnosti využitý inými high skillmi, napríklad na presunutie agenta do takticky výhodnej pozície. Rozhodne však nie je vhodný pre plánovanie trajektórie vyžadujúcej veľkú presnosť (napríklad nastavenie sa k lopte pred kopom). V budúcnosti bude vhodné spresniť obchádzanie prekážky s využitím konvexného obalu prekážky.

3.13. *Dokumentácia kódu*

V zdedenej verzii agenta Jim bolo obrovským problémom, že mnohé konštanty, metódy a dokonca aj triedy boli neokomentované. V takomto kóde bola veľmi náročná navigácia za potrebnými metódami a často sa nevedelo či a ako sa vôbec dané časti kódu používajú. Vznikla potreba pridať do kódu javadoc komentáre, ktoré by programátorovi pomohli zorientovať sa v jednotlivých triedach a získať informácie o jednotlivých častiach kódu, s ktorými pracuje. Na tento účel boli do kódu Jim doplnené komentáre k jednotlivým častiam kódu. Vytvoril sa javadoc k dôležitým public a protected metódam, atribútom tried a takisto, čo je veľmi dôležité, ku konštantám.

Je potrebné uviesť, že tu sa nekončí práca v tejto oblasti a je mimoriadne dôležité, až nevyhnutné, aby sa pridávaný kód **OKAMŽITE OKOMENTOVAL**. Kód by sa mal udržiavať nie len pre budúce generácie, ale aj pre potreby ostatných členov tímu a aj samotného autora kódu, pretože aj ten po určitom čase zabúda na niektoré veci. Na začiatku riešenia tohto projektu bude mať potom nasledovník predchádzajúcich riešiteľov jednoduchšiu situáciu a bude sa môcť sústrediť na vylepšovanie samotnej hry namiesto niekoľotýždňového oboznamovania sa s kódom.

```
/**
 * Whether the player's side was already assigned by the server
 */
public static boolean hasAssignedSide = false;
/**
 * Distance used in ball control calculation.
 * If x and y distance between agent and ball is less
 * than this value, agent thinks he controls the ball.
 */
public static final double BALL_IN_CONTROL = 0.55;
/**
```

Obrázok č.5: Ukážka okomentovaného public atribútu

```
//added by team17
//vracia predikciu buducej polohy lopty vypocitanu na zaklade anotacie k pohybu
/**
 * Returns ball as dynamic object with new values
 * based on specified Annotation execution saying how object changes after
 * Annotation execution. This is a part of prediction
 * as these values are future values of object.
 *
 * @param annotation
 * @return
 */
public DynamicObject ballAfterAction(Annotation annotation) {
    DynamicObject newBall = ball.clone();
    double avgElapsed = annotation.getDuration().getAvg();
    //predpoklada sa ze loptu hrac videl v okamihu ukonceni pohybu
    //ak tomu tak nie je, treba tuto informaciu pridat do anotacii
    double ballTime = EnvironmentModel.SIMULATION_TIME + avgElapsed;
    //kedze pozicia lopty v anotacii je dana z pohladu hraca, treba vektor jej zmeny otocit
    //o rotaciu (orientaciu) hraca
```

Obrázok č.6: Ukážka okomentovanej public metódy

3.14. Predikcia lopty

Pozície hráčov dostaneme v dvoch poliach z WorldModelu, zvlášť pre spoluhráčov a zvlášť pre protihráčov. Princíp počítania bude taký istý pre obe, ale aj tak je lepšie ich nechať v oddelených metódach. Riešenie spočíva v tom, že pre každého hráča v poli počítame jeho pozíciu kde sa bude nachádzať o niekoľko sekúnd. Po sekunde ukladáme dve premenné, terajšiu a minulú pozíciu. Odčítaním vektorov dostaneme vektor rýchlosti (keďže to rátame každú sekundu). Ak chceme zistiť, kde sa bude hráč nachádzať za X sekúnd, k vektoru pozície hráča pripočítame vektor rýchlosti vynásobený X.

Počítanie predikcie lopty je na podobnom princípe, akurát použijeme zložitejšiu vzorec, ktorý sme získali analýzou diplomovej práce P. Ertla, ktorý sme už uviedli vyššie.
Predikcia sa teda počíta každú sekundu počas zápasu, pre každého hráča a loptu, pričom hodnoty získame metódami world.getBall().getPrediction, teamPlayer.getPrediction() a opponentPlayer.getPrediction.

3.14.1. Testovanie

Implementované časti sme testovali na modelových situáciách:

- hráč sa pozerá na loptu pri kopnutí do nej,
- hráč sa pozerá na loptu, keď do nej kope druhý hráč,
- hráč sa pozerá na hráča,
- dvaja hráči sa na seba navzájom pozerajú,
- štyria hráči sa na seba navzájom pozerajú,
- štyria hráči sa na seba navzájom pozerajú a jeden z nich kope do lopty.

Pri všetkých situáciách boli odchýlky približne rovnaké, pod 0,3 sekundy. Táto odchýlka je pravdepodobne spôsobená odchýlkami vnášanými serverom. Je dobré spomenúť, že testovanie a odlaďovanie drobných chýb zabralo 70% času, pretože bolo veľmi náročné nastavovanie zložitejších modelových situácii. Hneď po dokončení bola predikcia použitá v iných úlohách, čo pomohlo odstrániť ešte jej drobné nedostatky.

3.15. Kráčanie za loptou

Implementovali sme metódu notSeenLongTime() pre loptu, ktorá nám vracia čas v sekundách, ako dlho loptu nevidíme. V prvej fáze sme implementovali do plánu vyhodnocovanie tejto metódy. Ak loptu nevidíme dlhšie ako päť sekúnd, tak sa nastaví premenná see_ball na true, inak je nastavená na false. V pláne sa potom vyhodnocuje táto premenná a ak je true, tak hráč začne hľadať loptu. Hľadanie spočíva v otáčaní sa na mieste

o dvadsať stupňov, čiže okolo si sa otočí na 18 krát. Keďže hráč vidí v až 120 stupňovom uhle, dokáže loptu nájsť relatívne veľmi rýchlo, avšak čim rýchlejšie by to dokázal, tým skôr môže začať kráčať k lopte.

V druhej fáze sme túto funkcionalitu vyčlenili do samostatného high skillu – Localize, čiže ak loptu nevidíme dlhšie ako päť sekúnd, spustí sa tento high skill, a hráč začne loptu hľadať. Pri implementácii tohto high skillu sme sa inšpirovali minuloročnou prácou diplomantov. Je tu implementovaná metóda pickLowSkill, v ktorej je okrem low skillu otáčania zabezpečené taktiež aj postavenie sa, v prípade ak hráč pri hľadaní lopty padne na zem. V tretej fáze sme ešte vylepšili otáčanie sa za loptou tak, aby sa otáčal za loptou na tú stranu, kde loptu videl naposledy. Bola vytvorená metóda, ktorá vracia, z ktorej strany hráč vidí loptu(zľava, sprava), a ak ju prestane vidieť tak vracia stále hodnotu, z kade ju videl, a podľa toho sa vyberie otáčanie doľava alebo doprava. Nakoniec bolo pridané otáčanie o deväťdesiat stupňov.

3.16. Optimalizácia pohybov

V prípade otočenia o 5° a 20° je riešenie relatívne rýchle i stabilné nakoľko sa robot iba „šúcha“, no stálo by sa pokus skúsiť vytvoriť pohyb otočenia o malé uhly iným spôsobom a porovnať stabilitu a rýchlosť s existujúcimi. Pri otočení o 90° sa dbalo na stabilitu hráča, no do budúcnosti by sa mohlo vytvoriť rýchlejšie otočenie o 90° možno i trochu na úkor stability.

Tabuľka č.2: Optimalizácia pohybov otáčania

Pohyb	Pôvodný čas [s]	Nový čas [s]
turn_left_cont_5, turn_right_cont_5	2,1	1,6
turn_left_cont_20, turn_right_cont_20	2,22	2
turnleft90, turnright90	5,7	4,2

Ďalšia úloha v sebe zahŕňala optimalizáciu vytvorených pohybov otáčania sa hráča. Týka sa to všetkých otočení, a teda o 5°, 20°, 45° a 90°. Podmienkou splnenia a akceptovania úlohy bolo zníženie času otočenia minimálne o 10%.

Všetky dané otočenia a ich časy sa mi podarilo znížiť o viac ako 10%, i keď niektoré na úkor presnosti otočenia. Približné hodnoty starých a nových otočení sú v nasledujúcej tabuľke:

Tabuľka č.3: Približné časy otočení pred a po ich zrýchlení

Pohyb	Pôvodný čas [s]	Pôvodný čas znížený o 10% [s]	Nový čas [s]
turn_right_cont_5_faster/ turn_left_cont_5_faster	1,82	1,64	1,42
turn_right_cont_20_faster/ turn_left_cont_20_faster	2,62	2,36	2,3
turnright45/ turnleft45	7,06	6,35	5,18
turnright90_faster/ turnleft90_faster	4,1	3,69	3,19

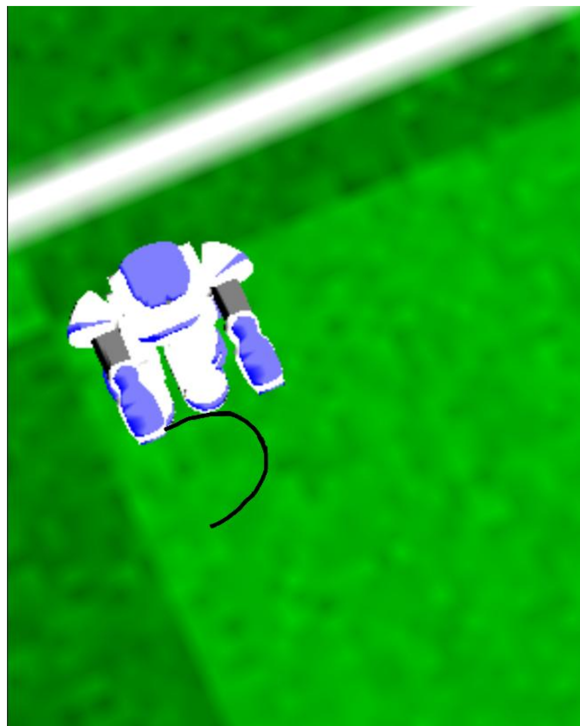
Pri tejto úlohe je možné pohyby ešte zrýchliť a prípadne zachovať rýchlosť a zlepšiť presnosť otočenia o konkrétny počet stupňov.

3.17. Vytvorenie nových pohybov

Úlohou bolo vytvoriť pohyby otočenia o 45°, akéhosi otáčania sa okolo lopty a zatáčajúcej chôdze. Najmä otočenie o 45° bolo potrebné spraviť nakoľko medzi pôvodnými otočeniami o 20° a 90° je pomerne veľké rozmedzie. Posledným pohybom bol kop na diaľku.

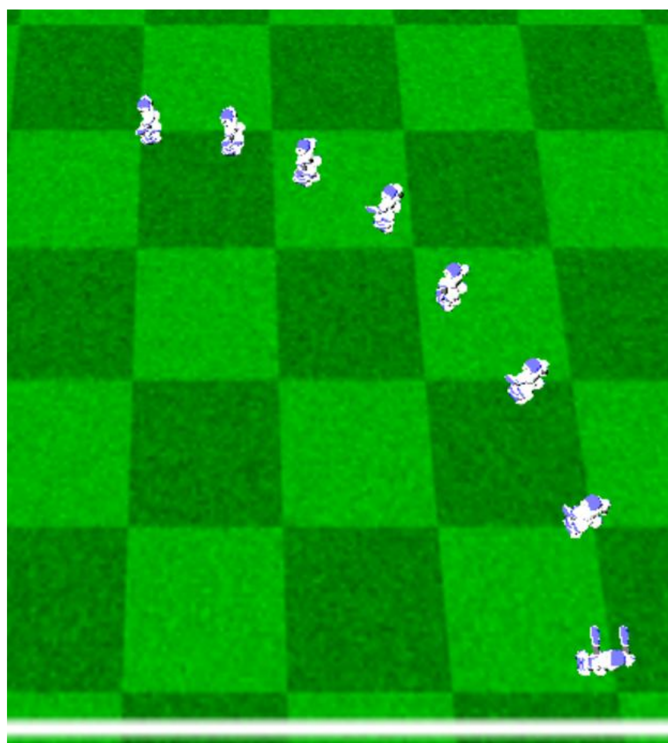
Pohyb otočenia o 45° vychádzal z pohybu otočenia o 90°. Celkový čas otočenia trvá približne 3,3 sekundy, čo je ešte kratšie ako dvojnásobné otočenie o 20°.

V prípade pohybu otáčania sa okolo lopty sa hráč posúva po akejsi kružnici (viď. Obrázok č.x). Tento pohyb vznikol upravením pohybu otočenia o 5 stupňov.



Obrázok č.7: Ukážka pohybu turning_left

Ďalším vytvoreným pohybom bol pohyb chôdze spojenej s otáčaním. Ukážka tohto pohybu je zobrazená na obrázku č. x. Táto chôdza vychádzala z chôdze walk_fine_fast1.



Obrázok č.8: Ukážka pohybu walk_turning_left

Posledným pohybom bol kop na diaľku, ktorý vychádzal z pohybu kick_right_fast. Tento pohyb som upravil tak, aby noha robota mala pri dotyku s loptou vyššiu rýchlosť. Oproti pôvodnému kopu tam je isté zlepšenie, no stále nie je dokonalý.

3.17.1. Možné vylepšenia

Všetkým novým pohybom môže byť vylepšená rýchlosť. Okrem toho pohyby otáčania sa okolo lopty a tak isto aj pohyby chôdze môžu byť doplnené o rôzne polomery otáčania. Kop do diaľky sa ešte určite dá vylepšiť iným prístup k celkovému pohybu.

3.18. Grafické zobrazenie v Test frameworku

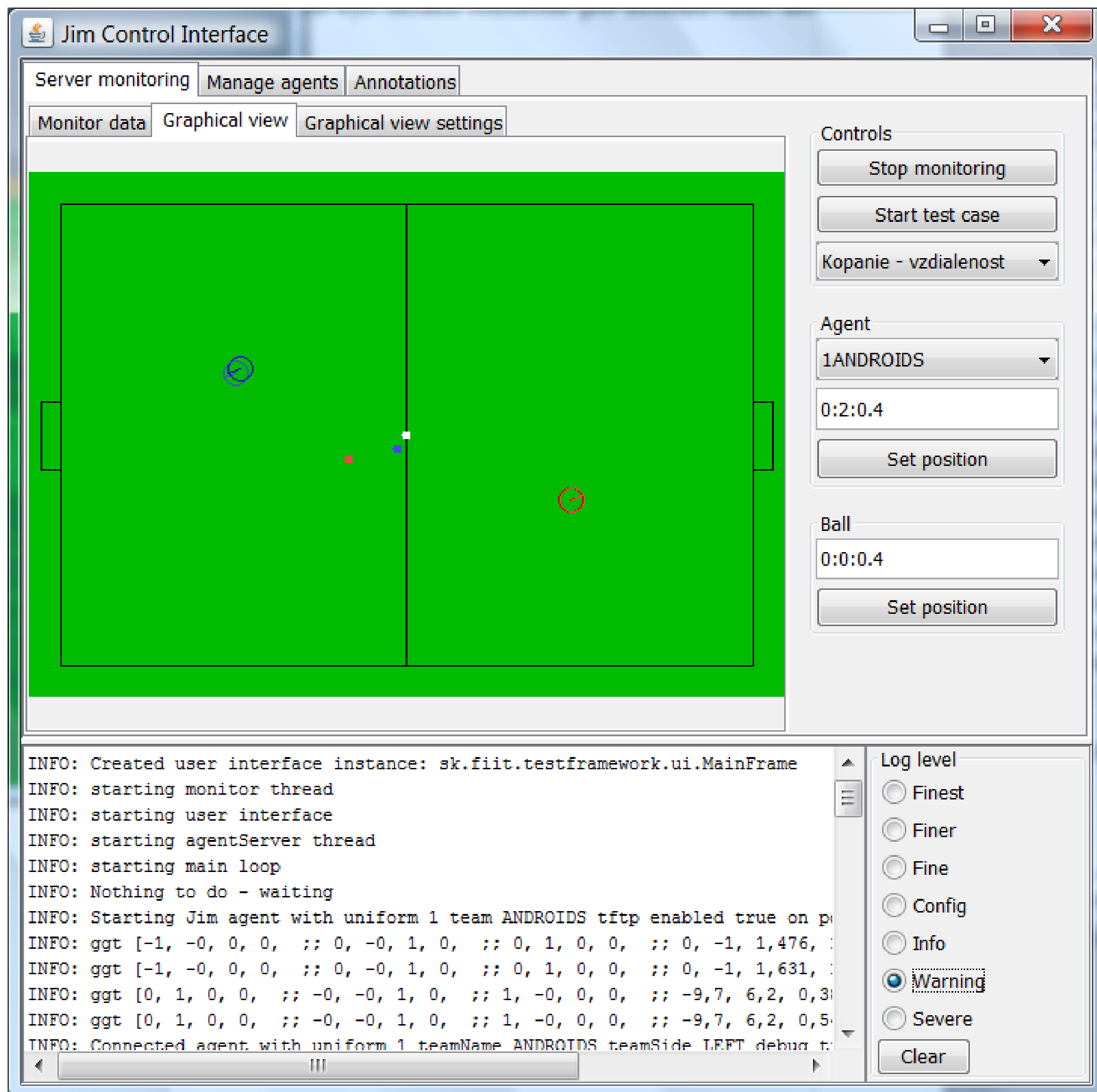
Do test frameworku bolo pridané grafické zobrazenie hry. V súčasnosti sa zobrazujú dáta z monitora pre všetkých agentov a loptu. Ďalej sa môžu zobraziť dáta z modelu sveta jedného alebo všetkých agentov, čo zahŕňa polohu a rotáciu agenta a polohu lopty. Je tak vidieť rozdiely medzi skutočnou polohou agenta a polohou, ktorú má agent vo svojich modeloch. Častokrát je napríklad absolútna pozícia chybná, no relatívna poloha k lopte ostáva správna.

3.18.1. Monitorovanie agentov

Okrem zobrazenia dát graficky bolo do test frameworku pridané aj okno umožňujúce zobraziť dáta z monitora pre jedného konkrétneho agenta. Týchto okien sa dá zobraziť ľubovolný počet, dajú sa tak teda monitorovať všetci agenti naraz. Okno môže byť neskôr rozšírené pre zobrazovanie dát poslaných test frameworku samotným agentom.

3.18.2. Možné dodatočné úpravy

Je možné doplniť dodatočné zobrazované dáta a rôzne filtre určujúce čo sa pre ktorého hráča zobrazí. Odporúčame však s vývojom tohto zobrazenia úplne prestať, a namiesto toho implementovať podporu monitora RoboViz (https://sites.google.com/site/umroboviz/), ktorý podporuje vykresľovanie vlastných dát v 3D prostredí, jednoduché ovládanie agentov a obsahuje mnoho ďalších užitočných funkcií. Jeho nevýhodou ale môže byť vyššia náročnosť na výkon, čo sa môže prejaviť hlavne pri spustení viacerých agentov na slabšom PC.



Obrázok č.9: Ukážka Test frameworku s grafickým zobrazením

3.19. *Opravy v Test fraweworku*

Popri celistvejších úlohách boli vykonávané rôzne drobné opravy a vylepšenia test frameworku. Niektoré sa týkali chýb, ktoré sa tam vyskytovali pomerne dlhú dobu.

3.19.1. Rotácia v dátach z monitora

V pôvodnej verzii grafického zobrazenia sa nám nezobrazovala správne rotácia hráča získaná z dát z monitora. Problém s týmito dátami nám hlásili aj členovia druhého tímu pri tvorbe testov. Po krátkom skúmaní som zistil, že dáta sú uložené v kartézskom vektore, ktorého hodnoty ale nevyjadrujú skutočný vektor v priestore, ale otočenie podľa jednotlivých osí v radiánoch.

Toto otočenie je navyše podľa niektorých osí vyjadrené v opačnom smere než dáta, ktoré používa agent. Tento fakt nebol nikde v dokumentácii ani v komentároch popísaný. Bola teda do triedy Player pridaná metóda getNormalizedRotation, ktorá vráti hodnotu rotácie v takej forme, ktorá zodpovedá forme dát používaných samotným agentom. Okrem toho bola táto časť príslušne zdokumentovaná.

3.19.2. Správa testov

Členovia druhého tímu sa sťažovali na nefunkčnosť testov dostupných v test frameworku. Po krátkom skúmaní vyšlo najavo, že ich testy závisia na prítomnosti vopred pripojeného agenta v tíme s názvom „Left“. Rozhodli sme sa preto pridať do triedy AgentManager metódy, ktoré pripravia prostredie na vykonávanie testov a vrátia objekt agenta použiteľného pre testovanie.

3.20. Kop do lopty

3.20.1. Silný priamy kop do lopty

Kop ktorý sme prebrali po predchádzajúcom tíme bol veľmi slabý a preto ho bolo nutné vylepšiť. Pôvodný kop nepracoval so všetkými kĺbmi na nohách, alebo iba veľmi obmedzene, čo bolo príčinou tomu, že agent nedokázal kopnúť d lopty dostatočnou silou. Kopy boli veľmi pomalé a pri hre by sa nedali používať, keďže loptu posunú len o vzdialenosť približne 2 metre.

Problémom bolo aj to, že agent neprikračoval k lopte, čím sa pripravoval o možnosť využiť väčšiu silu.

Kop do lopty bolo potrebné nakoniec úplne prerobiť. Do pohybu boli zakomponované ďalšie časti. Momentálne sa skladá z nasledujúcej súslednosti:

- Naváženie sa na jednu nohu. Agent sa naváži na nohu, ktorou bude kopať, aby sa druhou nohou mohol postaviť vedľa lopty.
- Prikročenie k lopte. Agent prikročí k lopte nohou, ktorou nebude kopať a pripraví si tak lepšiu pozíciu na strelu.
- Zodvihnutie nohy. V tejto fáze sa agent nakloní dopredu a súčasne dvíha nohu, ktorou bude kopať a ohýba koleno.
- Kop do lopty. Tu sa agent súčasne vystiera a kope do lopty, pričom využíva kĺby oboch bedier, koleno a členky na oboch nohách na lepšiu stabilitu.

- Poslednou fázou je stabilizovanie polohy po vykonaní kopu, kedy sa agent vracia do počiatočnej polohy.

3.20.2. Overenie

Kop sa na prvý pohľad javí oveľa dynamickejšie oproti pôvodnému. Agent pri pôvodnom kope kopal v priemere 2,5 metra. Novovytvorený kop dosiahol z desiatich úspešných kopnutí priemer 5,6 metra, pričom maximálna dĺžka, ktorú sa ním podarilo dosiahnuť bola až 6,7 metra. Toto znamená, že kop má ešte potenciál na budúce zlepšenie, či už vylepšením samotného pohybu, alebo zlepšenou počiatočnou pozíciou, z ktorej bude agent do lopty kopať.

3.21. High Skill – Hranie futbalu

Na nájdenie spoluhráčov sme implementovali metódu findTeamMates, ktorá vracia false, ak nevidí všetkých spoluhráčov, a vracia true, ak vidí všetkých spoluhráčov. Samotné hľadanie lopty je zakomponované v pláne, a ak hráčov nevidí, tak sa naplánuje otáčanie hráča.

Na vyhodnotenie, či je hráč najbližšie k lopte sme implementovali metódu nearestToBall, ktorá vracia true, ak je hráč najbližšie k lopte, a flase ak nie je. Samotné rozhodovanie je taktiež v pláne, a teda ak hráč je najbližšie k lopte, tak ide za ňou, ak nie tak kráča do formácie. Či je hráč druhý najbližší k lopte, sme implementovali v metóde secondToBall, ktorá vracia true ak je a false ak nie.

3.21.1. Testovanie

Implementované časti sme testovali na modelových situáciách 2 vs 2 hráči a 3 vs 3 hráči.

3.22. Zrýchlenie a spomalenie chôdze

Cieľom tejto úlohy bolo vylepšiť stabilitu existujúcej optimalizovanej chôdze na jej začiatku a konci. Chôdza by nemala začínať ani končiť v plnou rýchlosťou. Preto bolo implementované zrýchľovanie chôdze na jej začiatok a spomaľovanie na jej koniec. Tým by sa malo zabrániť nestabilite (rozkývaniu agenta) v týchto fázach chôdze.

Boli vytvorené tri súbory obsahujúce zrýchlenie, normálnu chôdzu a spomalenie. Tieto pohyby vychádzajú z pohybu *walk_fine_fast2*, ktorá predstavuje normálnu chôdzu. Pri spomalení aj pri zrýchlení boli upravené dĺžky jednotlivých fáz pohybu (150 ms). Pri zrýchlení má prvá fáza dva krát takú dĺžku ako pri normálnej chôdzi (300 ms). Dĺžka fáz sa

rovnomerne skracuje na dĺžku fáz normálnej chôdze (260 , 220 a 180 ms). Pri spomalení sa naopak dĺžky fáz zväčšujú a to presne opačným spôsobom ako pri rozbiehaní. Zrýchlenie taktiež obsahuje prvotné „prikrčenie“ agenta do stabilnej polohy pre chôdzu. Toto pri krčenie sa samozrejme už v normálnej chôdzi ani spomalení nenachádza. Výsledkom je veľmi stabilný pohyb, ktorý je vhodný pre ďalšiu optimalizáciu na jeho zrýchlenie.

3.23. Optimalizácia zrýchlenej a spomalenej chôdze

Cieľom tejto úlohy bolo vylepšiť rýchlosť chôdze so zrýchlením a spomalením, tak aby bola stabilná aj pri chôdzi na veľké vzdialenosti.

Pri tejto úlohe sa vychádzalo z chôdze *walk_fine_fast2_optimised*, ktorá bola približne o 20% rýchlejšia ako pôvodná chôdza, ale bola veľmi nestabilná na začiatku chôdze. Boli preto upravené pohyby pre zrýchlenie a spomalenie z minulej úlohy, aby sa táto nestabilita odstránila. Štyri zrýchľujúce fázy majú dĺžky 260, 220, 180 a 140 ms.

V opačnom poradí sú dĺžky spomaľovania. Výsledná chôdza je stabilná na začiatku aj na konci avšak veľmi často agent spadne približne medzi 20-30 krokmi, pričom dovtedy je chôdza veľmi stabilná. Príčinu tejto náhlej nestability nepoznáme a bude predmetom ďalšej optimalizácie.

3.24. Vedenie lopty

Bol vytvorený pohyb podobný chôdzi, ktorý je základom pre vedenie lopty agentom. Pri tomto pohybe sme sa snažili zdvíhať nohy čo najmenej nad zem aby lopta neodskakovala ďaleko od agenta. Pri vytváraní tohto pohybu sme vychádzali z vedenia lopty tímom Austin Villa. Vytvorený pohyb je stabilný ale je potrebné ho ešte urýchliť.

Ďalej sme vytvorili high skill, ktorý zabezpečuje udržanie lopty stále pred agentom. To sa zabezpečí výberom vhodného pohybu, podľa toho kde sa práve lopta vzhľadom k agentovi nachádza.