Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 3, 812 19 Bratislava

RoboCup 3D

Tímový projekt

Dokumentácia k projektu

Autori: Bc. Zdenko Capík
Bc. Peter Ertl
Bc. Michal Fojtík
Bc. Róbert Godány
Bc. Miroslav Hetteš
Bc. Marek Hruška
Bc. Ján Kováč

Tím: Robokopy (tím č. 15)

Študijný odbor: Softvérové inžinierstvo

Akademický rok: 2009/2010

Vedúci tímu: Ing. Marián Lekavý, PhD.

Obsah

1 ÚVOD

RoboCup je medzinárodná súťaž v robotickom futbale. Do tohto projektu sa môžu zapojiť tímy nadšencov pre umelú inteligenciu, informatiku, robotiku a iné príbuzné oblasti, ktorí si chcú medzi sebou zmerať sily vo vytvorení čo najlepšieho tímu humanoidných robotov schopných odohrať medzi sebou zápas.

V súčasnosti existuje viacero typov líg RoboCup 3D. Okrem líg so skutočnými robotmi vznikli aj ligy simulovaného futbalu. Simulovaný robotický futbal nie je finančne náročný do takej miery ako futbal so skutočnými robotmi. To je aj jeden z dôvodov prečo je simulovaná liga stále viac populárna. V tejto lige sa zúčastnené tímy snažia vytvoriť tím hráčov v rámci obmedzení simulačného prostredia a taktiež musia dodržiavať pravidlá hry. Táto liga prebiehala najprv iba v rámci 2D prostredia. Dnes už RoboCup existuje aj v 3D prostredí, ktoré rozširuje možnosti hry a taktiež zvyšuje náročnosť riešenia niektorých úloh. Na rozdiel od 2D prostredia je v 3D prostredí väčší problém s pohybovaním sa hráča. Vznikajú tak nové výzvy pre súťažiace tímy.

Tento dokument bol vytvorený v rámci predmetu Tímový projekt. Cieľom je popísať danú problematiku a taktiež priblížiť riešenia niektorých problémov simulovaného robotického futbalu s ktorými sme sa stretli v našom tíme. V závere tejto práce sú zhrnuté naše vylepšenia ktoré sme pridali v rámci tohto projektu.

2 ŠPRINTY

2.1 Šprint č. 1

ZAČIATOK ŠPRINTU:	7.10. 2009
KONIEC ŠPRINTU:	21.10. 2009
INFORMÁCIE:	PRVÝ ŠPRINT SLÚŽIL NA OBOZNÁMENIE SA S PROJEKTOM, ZALOŽENIE DOKUMENTÁCIE, WEBOVEJ STRÁNKY. DÔLEŽITÉ BOLI TAKTIEŽ ÚLOHY SPOJENÉ S ANALÝZAMI ZDROJOVÝCH KÓDOV, POHYBOV, PRAVIDIEL ROBOCUP 3D A POD. VŠETKY TIETO ANALÝZY NÁM MALI POMÔCŤ LEPŠIE POROZUMIEŤ NIEKTORÝM ZÁKLADNÝM ČASTIAM ROBOCUP 3D.
POČET PRÍBEHOV:	6

Analýza

2.1.1 Analýza štruktúry zdrojových kódov

Keďže v projekte RoboCup náš tím pokračuje v mieste, kde skončil predošlí tím, je potrebné analyzovať kompletne všetky zdrojové kódy. Tieto analýzy uľahčujú a urýchľujú lokalizáciu niektorých dôležitých častí kódu. Výsledkom tejto analýzy je pre nás prehľadné členenie častí kódov s ich popisom, čo robí konkrétna časť kódu, a ktoré funkcie obsahuje.

Analýza na úrovni modulov

Nasleduje popis tried v jednotlivých moduloch/adresároch. Ku každej triede sme pridali stručný popis, vymenovali hlavné údaje, ktoré uchováva a popísali funkcionalitu. Niektoré nevýznamné veci sme vynechali.

Domovský projektový adresár obsahuje *main* a základnú runnable triedu – *Agent*.

Trieda/súbor	*Popis*	*Údaje*	*Funkcie*
Agent	Predstavuje hráča v štýle Java Runnable	model, behaviour, communication,	Metóda start vykonáva hlavný cyklus: načítanie správ zo servera, rozhodnutie, čo ďalej, zaslanie akcií na server až kým

	objektu	príznak, či treba skončiť cyklus	program nie je ukončený signálom interrupt
main	Vstupný bod programu	IP servera, ID hráča, meno tímu, príznak behu hráča	Parsovanie vstupných argumentov, prípadne výpis usage, inicializácia inštancie Behaviour/ConfigRepository, v prípade, že ide o test, spustiť Tests/TestRunnera, inak vytvorenie a spustenie agenta so zadanými vstupmi. Pri zachytení signálu interrupt ukončenie hráča, na konci zavretie Logging/Loggera.

2.1.1.1 *Communication*

Komunikačná vrstva je v adresári Communication. Obsahuje sokety, wrapper, parser a builder S-výrazov. Tieto triedy sú priamo využité v Agentovi, ktorý v cykle volá Communication.receive(), SExpressionParser.parse(), SExpressionBuilder.build(), Communication.send().

Trieda/súbor	*Popis*	*Údaje*	*Funkcie*
Communication	Wrapper okolo TCP alebo UDP soketu (enkapsuluje komunikačný protokol), to závisí od konfigurácie	TCPSocket, UDPSocket, adresa servera	Deleguje send a receive na príslušný soket
Sockets/TCPSocket	Soket komunikujúci protokolom TCP	Deskriptor soketu	Send a receive s použitím protokolu TCP
Sockets/UDPSocket	Soket komunikujúci protokolom UDP	Deskriptor soketu	Send a receive s použitím protokolu UDP
Sexpressions/ SExpressionParser	Parsuje správy prijaté od servera a priebežne aktualizuje pozície objektov na ihrisku		Parsuje komunikáciu (funkcia parse) zo serveru v podobe s-výrazov a zapisuje ju do modelu hráča. Pre každý prvok sveta má samostatnú private funkciu.
Sexpressions/ SExpressionBuilder	Vytvára správy pre server		Na základe modelu poskladá (funkcia build) string s príkazmi, ktorý sa posiela na server. Pre každý typ akcie má samostatnú private funkciu.

2.1.1.2 *Model*

Zjednotenie všetkých hráčovi dostupných údajov sprostredkuje trieda Model. Detailnejšie v jednotlivých sub-moduloch/pod-adresároch. K väčšine údajov majú priamy prístup SexpressionParser a SexpressionBuilder, aby sa dalo efektívne aktualizovať videnie sveta.

Trieda/súbor	*Popis*	*Údaje*	*Funkcie*
InitialConfiguration	Obsahuje kompletnú metrickú konfiguráciu hráča	Metrické údaje ihriska, metrické údaje tela robota	
Model	Štruktúra zahŕňajúca perceptory, efektory a všetky údaje o svete, tele a interakcii, v každej iterácii hlavného cyklu sú údaje doplnené zo servera	Body, World, Interaction	Compute – dopočíta ostatné údaje, ktoré neprišli zo servera
ModelComputable	Interface pre všetky časti modelu, v ktorých sa dopočítavajú hodnoty.		Compute – dopočíta ostatné údaje, ktoré neprišli zo servera, je zavolaná aj na všetkých ModelComputable vnútorných členoch

2.1.1.3*Model/World*

Tu sa nachádzajú triedy uchovávajúce súradnice a iné údaje objektov na ihrisku, absolútna poloha aj poloha pozorovaná z perspektívy hráča. Niektoré objekty majú konštantnú pozíciu (konštanty) sú deklarované v Model/InitialConfiguration, iné objekty sú pribežne aktualizované Communitaction/ SExpressions/SExpressionParserom a ostatné údaje sú dopočítavané (pozri ModelComputable).

Trieda/súbor	*Popis*	*Údaje*	*Funkcie*
World	Model sveta a všetkých jeho objektov, je to iba štruktúra s linkami na konkrétne objekty	Left/right GoalPost, SV, SZ, JV, JZ Flag, hráči, lopta	Compute – dopočíta potrebné hodnoty pre hráčov a loptu
GoalPost	Uchováva pozíciu bránky na ihrisku. Inicializácia z konštánt v Model/InitialConfiguration	Horná tyč, dolná tyč (Flags)	
Flag	Uchováva pozíciu vlajky na ihrisku (napr. rohy). Inicializácia z konštánt v Model/InitialConfiguration	Point3D – pozícia na ihrisku, seenLocation – kde ju vidím	
PlaygroundMovable	ModelComputable objekt	Point3D – pozícia na ihrisku, seenLocation – kde ho vidím	Compute – dopočíta potrebné hodnoty
Player	PlaygroundMovable updatovaná v každej iterácii	Zdedené od PlaygroundMovable, ID, názov tímu	Zdedené od PlaygroundMovable

	SexpressionParserom, predstavuje futbalovú spoluhráča alebo protihráča		
Ball	PlaygroundMovable updatovaná v každej iterácii SexpressionParserom, predstavuje futbalovú loptu	Zdedené od PlaygroundMovable	Zdedené od PlaygroundMovable

2.1.1.4 *Model/Body*

Tu sa nachádzajú triedy popisujúce telo hráča, stav jeho kĺbov, gyroskop.

Trieda/súbor	*Popis*	*Údaje*	*Funkcie*
Body	Telo hráča poskladané z častí.	Gyroscope, BodyPart head, neck, torso, left/right arm, left/right leg, hmotnosťm ťažisko, oporné body (StrongPoints), lokácia	Výpočet ťažiska, oporných bodov, pozície na ihrisku (chýba)
BodyPart	Všeobecná časť tela, z nich je poskladaný celý hráč	Názov perceptora, efektora, časti tela, rodičovská časť tela, hmotnosť, uhol, posun, lokácia	Relativizácia zadaného bodu vzhľadom k časti tela, výpočet pozície vzhľadom k torzu (zatiaľ nie vzhľadom k zemi)
StrongPoint	oporný bod, slúži hlavne (zatiaľ) na sprehľadnenie a debugovacie účely	Point3D lokácia, názov BodyPartu	
Arm	Ruka poskladaná z BodyParts, inicializácia z konštánt v InitialConfiguration	shoulder, upperarm, elbow, lowerarm	
Leg	Noha poskladaná z BodyParts, inicializácia z konštánt v InitialConfiguration	hip1, hip2, thigh, shank, ankle, foot	
Gyroscope	Zotrvačník, hodnoty uhlová rýchlosť je aktualizovaná SExpressionParserom	Uhlová rýchlosť, absolútny uhol	Aktualizácia absolútneho uhla v časových krokoch

2.1.1.5 *Model/Interaction*

V týchto triedach sa nachádzajú údaje, ktorými hráč interaguje s prostredím (beaming, talking) s výnimkou pohybu.

Trieda/súbor	*Popis*	*Údaje*	*Funkcie*
Interaction	Enkapsuluje interakciu hráča s prostredím: uchováva zistený stav hry, beam premiestnenie a správy na komunikáciu s inými hráčmi	GameState, Beam, message	
GameState	Stav hry a hráča v nej	ID hráča, názov tímu, čas, hrací mód	
Beam	Požadované umiestnenie hráča na hracej ploche	x,y, rotácia (vzhľadom ku kladnej osi x)	

2.1.1.6 *Behaviour*

Obsahuje triedy zodpovedné za pohyb, rozhodovanie. Modul má túto hierarchiu:

- Behaviour -> Action[];
- Action -> State[];
- State -> Move[];
- Move -> JointCondition[].

Trieda/súbor	*Popis*	*Údaje*	*Funkcie*
ConfigRepository	Singleton uchovávajúci command-line argumenty. Je inicializovaný na začiatku v main (funkcia parseArgs).	cmd args	Operácie dátovej štruktúry mapa<Option,string>. Option je enum definované v tomto súbore.
Behaviour	Správanie, číta model a mení ho podľa toho, ako sa chce agent správať. Ide o singleton.	ActionRepository, bool firstRun	Vykonanie kroku pomocou nextStep(model), ak ide o prvý krok, vykoná sa beam, nextAction vyberie ďalšiu akciu z ActionRepository (zatiaľ sa robot snaží kráčať)
Action/ ActionRepository	Množina akcií načítaných zo súborov v adresári akcií	actionQueue, lastAction	Pridávanie akcií, nastavenie current akcie
Action/Action	Postupnosť stavov, ktorými sa prechádza, Dá sa načítať zo súboru	States, aktuálny state	execute(model) – vykonanie nasledujúceho kroku aktuálneho stavu, setNextState – posun na ďalší stav
Action/State	Zahŕňa postupnosť pohybov	MoveQueue, Moves, materská akcia	execute(model) – vykonanie nasledujúceho kroku všetkých pohybov

			(podľa času), pokiaľ sú všetky dokončené, materská akcia je presunutá na ďalší stav
Action/ JointCondition	Predstavuje podmienku na želanú pozíciu kĺbu, pred vykonaním pohybu	Názov kĺbu, pozícia, príznak, či je smer pozitívny	Vyhodnotenie (funkcia evaluate), či daná situácia nastala
Action/Move	Predstavuje pohyb jedným kĺbom	BodyPart, kĺb (joint), JointConditions, štartovací uhol, koncový uhol, posledný uhol, dĺžka trvania, štartovací čas, príznak, či skončil	execute(model) – vykonanie kroku v rámci tohto pohybu, aby sa mohol pohyb vykonať, musia byť splnené JointConditions
Action/ MoveQueue	Usporiadaný rad pohybov (Move) na vykonanie	Zoznam pohybov	Ide o front (queue), kde sa dajú pridávať pohyby, dá sa usporiadať podľa času, pohyby sa dajú synchronizovať tak, že kde sa časovo pretínajú, pridá sa JointCondition

2.1.1.7 *Primitives*

Obsahuje základné štruktúry a algoritmy na matematické operácie.

Trieda/súbor	*Popis*	*Údaje*	*Funkcie*
Point3D	Bod 3D priestore vyjadrený karteziánskymi súradnicami	x,y,z, názov BodyPartu, ku ktorému je relatívny	Posun o vektor, otočenie podľa vektora a uhlu, quaternionu, matice
SphericalCoordinate	Bod v 3D priestore vyjadrený v sférických súradniciach	Vzdialenosť, zenit, azimut	Transformácia na kartézske súradnice
Vector3D	Posunutie v karteziánskom 3D priestore	x,y,z, statické jednotkové vektory	Vektorové +,-, skalárne *,/, skalárny, vektorový súčin, dĺžka, normalizácia, negácia, porovnanie smeru
Quaternion	Vyjadruje otočenie voči nejakému vektoru	x,y,z, w, viacero konštruktorov	*, konjugát, inverzia, normalizácia, exponent, logaritmus, matica
Rotation	Vyjadruje rotáciu okolo 3 základných osí	Uhly pravotočivej rotácie yaw (z), pitch (y), roll(x)	
MathHelper	Poskytuje pomocné matematické	PI, floating point tolerancia	Konverzia medzi stupňami a radiánmi, zaokrúhlenie,

	operácie		signum, konverzia medzi Rotation a Quaternion, výpočet uhla a rotácie medzi 2 vektormi

2.1.1.8 *Logging*

Obsahuje triedy využívané pri logovaní správ do súboru a konzoly.

Trieda/súbor	*Popis*	*Údaje*	*Funkcie*
Loggable	Abstraktná trieda definujúca spôsob zápisu stavu sveta do stringu		Transformácia stavu sveta do stringu
Logger	Priebežne loguje stavy sveta, chyby, debug a iné výpisy	Path, errPath, logFile, errorLog, isOn	spájanie stringov, logovanie do súboru a konzoly, transformáciu stavu sveta do stringu vykoná Loggable, na konci musí byť explicitne zatvorený (v main)

2.1.1.9 *Others*

Obsahuje dátové štruktúry na iteráciu cez zoznamy mien súborov. Využíva knižnicu dirent.lib. Kvalita kódu ale nie je dobrá.

Trieda/súbor	*Popis*	*Údaje*	*Funkcie*
FileList	Zoznam názvov súborov	Názov súboru, odkaz na ďalšiu položku zoznamu	
DirectoryList	Zbytočná trieda, funkcionalita mohla byť presunutá priamo do FileList		Naplní FileList názvami súborov v zadanom adresári

2.1.1.10 *Tests*

Jednoduché testovacie prostredie pre niektoré algoritmy a funkcie, zatiaľ iba tie matematické.

Trieda/súbor	*Popis*	*Údaje*	*Funkcie*
TestRunner	Spúšťa testy a vracia TestResults	Výsledky testov	Zatiaľ iba testy matematických funkcií
TestResults	Výsledky testovania	Bool výsledky namapované na string názvy sub-testov	

Analýza na úrovni dátových štruktúr

Z pohľadu tried ako dátových štruktúr je nižšie znázornená kompozičná hierarchia objektov. V rámci použitej notácie zátvorky *{}* určujú vnútorné položky objektu a zátvorky *[]* predstavujú pole.

Agent = {Model, Behaviour, Communication}
    Model = {Body, World, Interaction}
        Body = {Gyroscope, BodyPart[] }
        World = {GoalPost[], Flag[], Player[], Ball}
        Interaction = {GameState, Beam, message}
    Behaviour = {ActionRepository, Action}
        ActionRepository = {Action[] }
            Action = {State[] }
                State = {Move[] }
    Communication = {Socket[] }

Analýza na úrovni control-flow

Hlavný algoritmus pozostáva z nasledujúcich krokov:

a. Parsovanie vstupných argumentov (main.cpp, Behaviour/ConfigRepository.cpp)
b. Prípadný Test (Tests/*)
c. Vytvorenie agenta (Agent.cpp), otvorenie soketu (Communication/Sockets/*) a zaslanie úvodných správ serveru
d. Prijatie správy zo servera (Communication/*)
e. Parsovanie správy (Communication/SExpressions/SexpressionParser.cpp)
f. Model.compute() – dopočítanie potrebných hodnôt (Model/*, Primitives/*)
g. Behaviour.nextStep() – krokovanie pohybu (Behaviour/*)
h. Vytvor správu pre server (Communication/SExpressions/SexpressionBuilder.cpp)
i. Pošli správu serveru (Communication/*)
j. Späť na krok d.

Záver

Po revízii kódu sme dospeli k záveru, že projekt má mnohé nedostatky. Sú to jednak zlé praktiky písania C++ kódu, chyby, miestami nešikovný návrh a hlavne nevyužívanie knižnice STL a iných základných API pre C++. Preto jednou z našich prvých úloh potrebných pre efektívne

rozširovanie projektu bude jeho reštrukturalizácia (angl. refactoring) a oprava návrhových a implementačných chýb.

2.1.2 Nástroje pre podporu projektu

V tejto úlohe je cieľom nájsť nástroje, ktoré slúžia na podporu projektu. Ide hlavne o nástroje umožňujúce lepšie a efektívnejšie komunikovať v rámci tímu a nástroje na verziovanie softvéru.

Uvažovali sme nad viacerými nástrojmi keďže je dostupné veľké množstvo nástrojov pre podporu projektu. Na verziovanie sme sa po dlhšej analýze dostupných riešení rozhodovali medzi SVN, CVS a TortoiseHG. Nakoniec bol použitý TortoiseHG. Návod na použitie sa nachádza v prílohe A.
Čo sa týka nástroja na komunikáciu v tíme, bol uvažovaný len 1 nástroj – Google groups. V rámci predmetu Tímový projekt nám boli prezentovane taktiež niektoré nástroje pre podporu manažovania projektu. Jednalo sa o platené ale taktiež o voľne dostupné riešenia. Keďže sa jedná o školský predmet do úvahy pripadali hlavne voľne dostupné riešenia. Jedným zo zaujímavých nástrojov bol Redmine, ktorý má webové rozhranie.

2.1.3 Oživenie hráča, inštalácia v tíme

Jeden z prvých predpokladov v úspešnom zapojení sa do projektu je, aby každému členovi tímu fungoval RoboCup z predošlej verzie projektu. Táto úloha zahŕňala správne spustenie serveru a taktiež spustenie agenta.

Analýza riešenia danej úlohy spočívala v nájdení spôsobu, ako spustiť server a následne pripojiť naň hráča. Výsledkom by mal byť taktiež manuál pre všetkých členov tímu s podrobnými informáciami a v prípade problému by mal byť problém komplexne riešený u konkrétneho člena tímu.

2.1.4 Oživenie testovacieho frameworku

Po analýze predchádzajúceho riešenia tzn. riešenia tímu Agenty 007 sme prišli ku skutočnostiam, že v projekte nie je doteraz zakomponovaná žiadna podpora pre komplexné testovanie hráča. Súčasťou projektu sú len unit testy určitých matematických operácii ako napr.:

- Vektorový súčet

- Vektorový súčin
- Uhol medzi 2 vektormi

Návrh

2.1.5 Založenie dokumentácie

Dokumentácia bude v elektronickej forme vo formáte .doc. Pre vymieňanie dokumentov, ktoré sa bude uskutočňovať interne medzi členmi tímu, uvažujeme taktiež o formáte .docx. Návrh bude taktiež spĺňať predpísané názvy kapitol, ktoré sú odporučené v rámci predpísaných kapitol. Celá dokumentácia bude písaná písmom Cambria.

2.1.6 Nástroje pre podporu projektu

Na internú komunikáciu bude použitý systém Google groups. Ak nepôjde o komunikáciu ktorá vyžaduje zainteresovanosť viacerých členov tímu, komunikácia bude prebiehať pomocou elektronickej pošty. Celkové manažovanie projektu bude riešené za pomoci systému Redmine, ktoré poskytuje všetko potrebné k manažovaniu projektu vytváraného pomocou metodiky SCRUM. Systém na verziovanie sa použije TortoiseHG, ktorý je voľne dostupný a spĺňa všetky požiadavky kladené našim tímom.

2.1.7 Založenie webovej stránky tímu

Stránka nášho tímu bude statická, nebude obsahovať žiadne prvky ktoré nie sú statické. Použijeme technológie HTML, PHP a CSS. Vzhľad bude jednoduchý a prehľadný. V časti dokumenty by mali byť všetky dostupné dokumenty ako napr. zápisnice zo stretnutí, dokumentácie, plagát tímu, ponuka pre projekt.

Implementácia

2.1.8 Nástroje pre podporu projektu

Komunikácia v tíme

Vytvorili sme na Google groups skupinu s názvom Robokopy do ktorej sa následne prihlásili všetci členovia tímu.

Manažovanie projektu

Použili sme systém Redmine do ktorého sa prihlásili všetci členovia tímu a následne im boli vedúcim tímu pridelené práva a boli zapojený do projektu Robocup.

Verziovanie

Každý člen tímu si nainštaloval podľa návodu v prílohe A nástroj TortoiseHG. Okrem toho sme začali používať systém Bitbucket, ktorý je voľne dostupný pomocou webového rozhrania.

2.1.9 Vytvorenie webovej stránky tímu

Webová stránka bola vytvorená pomocou HTML, CSS a PHP a je umiestnená na školskom servery LABSS2, jej presná adresa je http://labss2.fiit.stuba.sk/TeamProject/2009/team15is-si/. Obsahuje nasledujúce sekcie:

- Novinky – čo pribudlo na stránke
- Náš tím – predstavenie jednotlivých členov tímu
- Plán – plán projektu v letnom a zimnom semestri, je členený podľa šprintov
- Kalendár – prehľad o udalostiach týkajúcich sa tímu resp. povinností tímu
- Dokumenty – obsahuje dokumentácie, plán projektu, ponuku atď.

Testovanie

2.1.10 Založenie dokumentácie

Dokumentácia bola verejne dostupná a jednotlivé požiadavky na zmenu resp. vylepšenia boli do nej zahrnuté. Taktiež bola ukázaná vedúcemu projektu. Syntaktické chyby v dokumentoch od členov tímu boli pred integráciou do dokumentácie upravené a opravené pomocou spell checkera.

2.1.11 Nástroje pre podporu projektu

Google groups bol otestovaní všetkými členmi tímu zaslaním prvej správy. Táto testovacia správa slúžila ako test funkčnosti aby sa nevyskytovali žiadne problémy s komunikáciou. Google groups je externý program takže za jeho správnosť nezodpovedajú priamo členovia tímu. Preto bola ako záloha uvažovaná komunikácia elektronickou poštou, tá bola taktiež overená zaslaním prvej správy, čím sa overili napr. aj nesprávne adresy jednotlivých členov tímu. Taktiež bolo otestované pridávanie súborov – skončilo úspešne.
Redmine bol otestovaní tak, že jednotlivý členovia tímu pridali svoje tasky, resp. počas šprintu upravovali zostávajúci čas do splnenia úlohy.
TortoiseHG bol testovaní vyskúšaním si niektorých činností a následným reportovaním členovi tímu starajúcemu sa o daný nástroj.

2.1.12 Oživenie hráča, inštalácia v tíme

Testovanie funkčnosti projektu RoboCup u jednotlivých členov prebiehalo spustením serveru a hráča u každého člena tímu. Testovanie resp. celá úloha bola ukončené okamžikom, keď RoboCup fungoval každému členovi tímu.

2.1.13 Vytvorenie webovej stránky

Stránka bola spustená a odskúšaná každým členom tímu. Jednotlivé pripomienky resp. chyby boli opravené správcom našej webovej stránky. Stránka je funkčná, beží rýchlo a všetci členovia tímu hlásili bezproblémové spustenie stránky nášho projektu. Taktiež vedúci tímu videl stránku a všetko bežalo podľa očakávaní.

2.2 Šprint č. 2

ZAČIATOK ŠPRINTU:	21.10. 2009
KONIEC ŠPRINTU:	4.11. 2009
INFORMÁCIE:	V TOMTO ŠPRINTE SME SA ZAMERALI NA DOKONČENIE PRIESTOROVEJ ORIENTÁCIE HRÁČA, KTORÁ PREDTÝM NEBOLA IMPLEMENTOVANÁ, A VYHODNOTENIE AKTUÁLNEHO STAVU ROBOTA. TAKTIEŽ BOLO TREBA VYTVORIŤ TESTOVACÍ FRAMEWORK. JEDNA Z HLAVNÝCH ČASTÍ TOHTO ŠPRINTU BOLA, ABY SI KAŽDÝ ČLEN TÍMU VYSKÚŠAL VYTVÁRAŤ NOVÉ POHYBY PRE AGENTOV A TAK VYTVORIL KAŽDÝ ČLEN ASPOŇ JEDEN NOVÝ POHYB POMOCOU EDITORU POHYBOV.
POČET PRÍBEHOV:	12

Analýza

2.2.1 Analýza prístupov k pohybom humanoidného robota

Aby bolo možné vytvoriť čo najlepšie pohyby robotov, je výhodné pozrieť sa na rôzne prístupy k pohybom humanoidných robotov resp. k pohybom ľudí, a následne vyhodnotiť niektoré zaujímavé myšlienky vyplývajúce z tejto analýzy.

Cieľom analýzy prístupov k pohybom humanoidného robota je opísať existujúce poznatky a realizované experimenty s robotmi s dvoma opornými bodmi. Vývoj takýchto robotov, ktoré sa snažia napodobňovať ľudí, je už dlho cieľom viacerých disciplín ako napríklad robotika. Je to však dosť náročná úloha a roboti majú stále problémy s niektorými primitívnymi úkonmi. Základnými pohybmi relevantnými v oblasti robotického futbalu sú jednoduché chodenie, chodenie k lopte, otáčanie sa, bočný pohyb, kopanie do lopty, vstávanie zo zeme, pokusy o robotický beh, prípadne iné špecifické pohyby. Opísané sú jednak metódy založené na pozorovaniach a digitálnych meraniach ľudskej chôdze, ale aj iné prístupy ktoré vychádzajú z oblasti umelej inteligencie, robotiky a elektrotechniky.

V kapitole sú opísané najskôr niektoré iné druhy robotov, ktoré sú vo vývoji pohybov a správania najďalej. Následne sú hlbšie popísaný proces chôdze robota, jednotlivé druhy chôdze, keďže tento je najdôležitejšou časťou pohybového aparátu – bez spoľahlivého chodenia je humanoidný robot nepoužiteľný. V ďalšej časti sú opísané hlavné V závere kapitoly je opísaná

jedna z možných metód optimalizácie pohybu, ktorá uvádza aj všeobecné možnosti optimalizovania pohybu a necháva priestor aj pre iné optimalizačné techniky. Na konci kapitoly je zhrnutie a zhodnotenie.

2.2.1.1 Iné humanoidné roboty - Sony QRIO, Honda ASIMO

Najznámejšími humanoidnými robotmi sú Honda ASIMO a Sony QRIO. Oba dokážu spoľahlivo chodiť, vykonávať základné úkony aj niektoré pokročilejšie, ako napríklad držanie predmetov, chodenie po schodoch, rozoznávanie osôb a podobne. ASIMO má 130cm a váži 54kg a snaží sa čo najviac napodobňovať človeka s cieľom byť čo najviac autonómny. Preto väčšina jeho pohybov vychádza z pozorovaní ľudskej chôdze, behu a ostatného správania. Najnovšou výhodou jeho chodenia je optimalizácia plynulosti pohybov tak, že nezačínajú prudko jeden cez druhý, ale robot hladko prechádza z jedného pohybu do iného. Takže jeho pohyby vyzerajú realistickejšie a viac ľudsky, čo je úspech, keďže robot je vyvíjaný s tým aby bol čo najlepšie použiteľný na interakciu s ľuďmi. [19][20]

Sony QRIO je jedným z humanoidných robotov vyvíjaných spoločnosťou Sony. Robot váži asi 7 kilogramov a je vysoký 57 centimetrov. Pôvodným cieľom spoločnosti bolo uviesť robota na trh, ak by dokázal vykonávať užitočné činnosti, jeho vývoj bol však ukončený v roku 2006. Hlavným úspechom robota bolo, že je prvým dvojnohým humanoidným robotom schopným behu. Prvý beh sa uskutočnil v roku 2003. Robot dokáže bežať rýchlosťou 14 metrov za minútu, čo je asi 23 centimetrov za sekundu. Táto rýchlosť je oproti podobným predchádzajúcim robotom , ktorý dokázali len chodiť, takmer dvojnásobná. Podľa meraní zaznamenaných pri realizovaných experimentoch s behom mal robot vo vzduchu obidve dolné končatiny približne 20 sekúnd z každých 1000 minút behu, teda asi 4 stotiny sekundy pri každom kroku. V najvyššom bode dosahovali jeho robotické chodidlá výšku 0.5 centimetra nad povrchom. K reálnemu atletickému behu však tento beh má ešte veľmi ďaleko, napriek tomu je to veľmi cenný prínos do oblasti simulácie behu robotov. Robot je schopný okrem behu aj skákania. QRIO bol tiež zapísaný do Guinnesovej knihy rekordov ako prvý humanoidný robot schopný behu. [18]

Na rozdiel od ostatných predchádzajúcich robotov sa nepohybuje klasickým statickým pohybom, ale dynamicky. Tu sa využíva metóda ZMP – Zero Moment Point, čiže robot je podobne ako človek pri behu a stále v nerovnováhe, ťažisko nie je v zóne stability. Nerovnováha spôsobená príťažlivosťou zeme (padanie robota dopredu) je vyrovnávaná silami vyvolanými pohybmi dolných a horných končatín robota, takže robot nespadne. Podobne ako pri ľudskom behu je prenášaná kinetická energia pri výskoku robota (0.5 cm) na kinetickú energiu pohybu. Pri dopade na niektoré z chodidiel stlmujú špeciálne kĺby náraz. Sony preto vyvinula špeciálne servomotory ISA (Intelligent Servo Actuator), ktoré dokážu vyvinúť dostatočnú silu na odrazenie sa od zeme a dokážu tlmiť nárazy. Zároveň dokážu citlivo zmerať svoju aktuálnu

polohu, teda uhol otočenia, čo je veľmi dôležité na stabilizáciu robota po dopade. Robot tiež vie zistiť externé sily naňho, ako napríklad náraz objektu alebo snahu o jeho zhodenie. V takom prípade sa vďaka tlakovým senzorom pokúsi udržať rovnováhu, prípadne zabrániť pádu pomocou rúk.

2.2.1.2 Chôdza robota

Robotická chôdza sa stala predmetom záujmu viacerých výskumov. A to najmä z dôvodu, čo raz väčšieho vsadenia robotov do mnohých oblastí ľudského života cez domácnosť, priemysel až po armádu. Je určite neľahkou úlohou a pre mnohých veľkou výzvou. Táto časť dokumentu obsahuje rôzne prístupy chôdze humanoidných robotov.

2.2.1.3 Humanoidná chôdza

Bipedickí roboti sú o mnoho viac mobilnejší v porovnaní s klasickými, ktorý sa pohybujú prostredníctvom kolies. Sú schopnejší sa pohybovať v rôznorodých prostrediach stabilne (napr. po schodoch). Pričom pri pohybe je snaha napodobniť chôdzu ľudí. Chôdzu vo všeobecnosti môžeme považovať za presun dvoch nôh, pri ktorej dochádza striedavo ku zdvihu chodidiel, pričom v každom čase sa aspoň jedná z nich dotýka zeme. Ak dočasne nastane pri chôdzi pohyb, keď ani jedno chodidlo sa nenachádza na zemi vravíme o behu. Dvojnohú chôdzu môžeme rozdeliť na *statickú* alebo *dynamickú:*

Statická chôdza – ide o pomalú stabilnejšiu, ale neprirodzenú chôdzu. Návrh ťažiska robota k ploche po ktorej kráča. Ak zabezpečíme, že ťažisko robota sa bude stále nachádzať v oblasti nôh a pohyb bude dostatočne pomalý, môžeme dostať celkom stabilnú chôdzu. Robot sa takto dokáže pohybovať bez straty rovnováhy, čoho výsledkom by bol pád. V tomto prípade je možné aplikovať všetky možné techniky, pri ktorých dochádza mechanicky, periodicky k zmene polohy kĺbov robota.

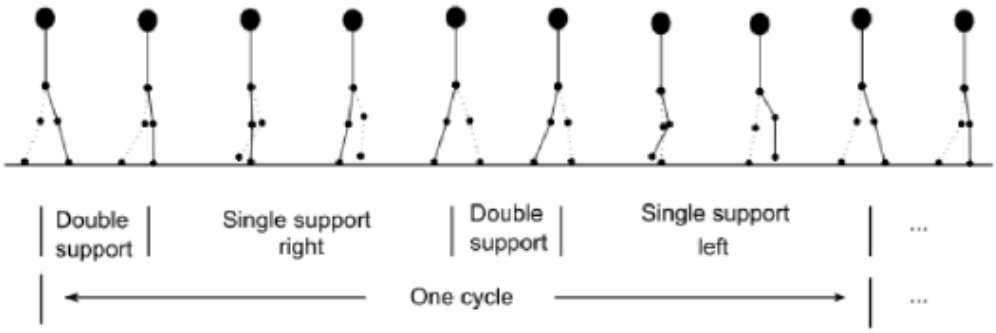
Dynamická chôdza – robot sa dostáva do situácií, kedy sa nachádza často krát na hranici stability. Chôdza je krajšia a prirodzenejšia. Pri potrebe dosiahnuť rýchlejšiu chôdzu, ktorá sa viac podobá chôdzi ľudí sa aplikuje metóda ZMP (angl. Zero Moment Point). Ide o metódu kedy sa všetky sily pôsobiace na robota (všetky aktívne sily) navzájom rušia. Bližší popis ZMP sa nachádza v inej časti dokumentu.

Statická vs. dynamická chôdza

V súčasnosti mnoho humanoidných robotov využíva statickú chôdzu. Aj keď je pomalšia oproti dynamickej, ktorá sa viac podobá chôdzi ľudí, je menej náročná na spotrebu elektrickej energie [15]. Čo je jeden z dôležitých faktorov, ak robot disponuje s energiou obmedzeného množstva. Teda to čo je prirodzenejšie pre človeka nemusí byť prirodzené aj pre robota. Ľudia pri svojom pohybe využívajú nohy a špecifické kĺby, pre ktoré sú iné energetické nároky ako v prípade robotov. Aj keď je snaha napodobňovať chôdzu ľudí je potrebné zvážiť aj na tento aspekt.

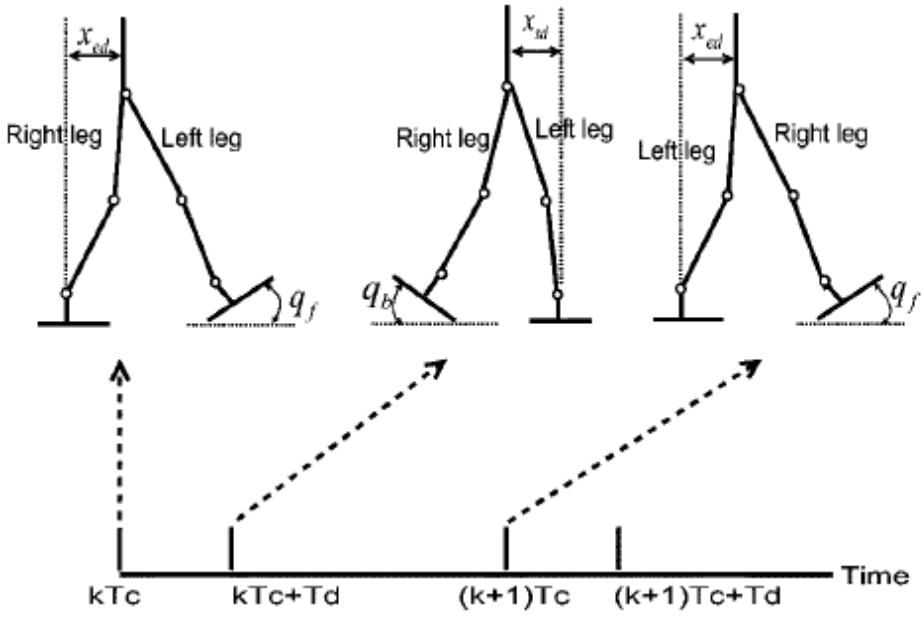
Jednotlivé fázy chôdze

Vzor podľa ktorého sa vychádza pri chôdzi dvojnohých(angl. biped) robotov má dve fázy. A to podľa toho, či je robot v kontakte so zemou s oboma alebo iba s jednou nohou. Na obrázku 1 môžeme vidieť jednotlivé fázy [14].



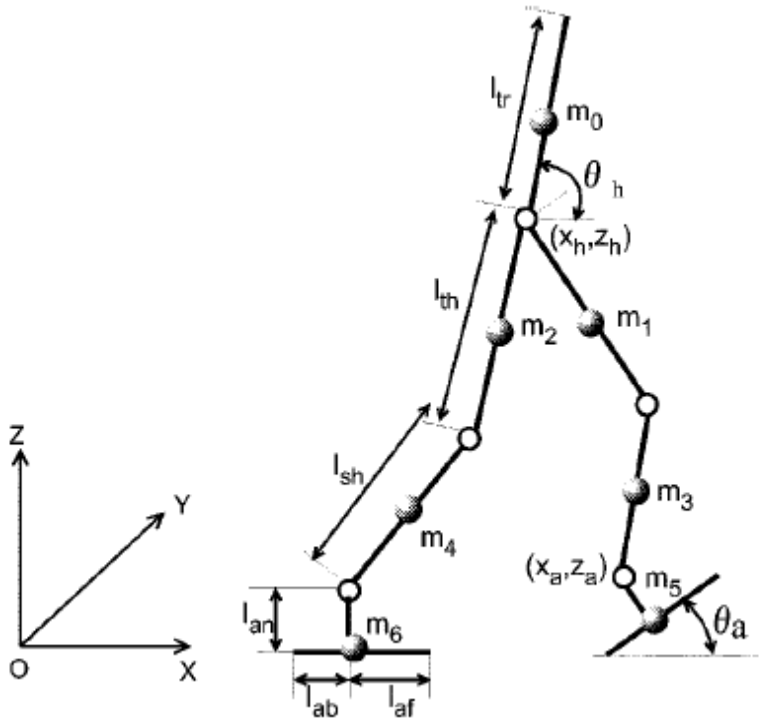
Obr. 1.: Fázy chôdze[14]

Na obrázku 2 môžeme vidieť jednotlivé fázy chôdze z iného uhla pohľadu spolu s vzdialenosťami medzi jednotlivými kĺbmi.



Obr. 2.: Jednotlivé fázy chôdze [14]

Pre lepšie pochopenie aké sily môžu pôsobiť počas chôdze v dôsledku rôznych váh jednotlivých častí tela, na obrázku 3 môžeme vidieť jednotlivé časti nôh a torza dvojnohého robota spolu s ich hmotnosťami.



Obr. 3.: Dolná časť tela humanoidného robota [14]

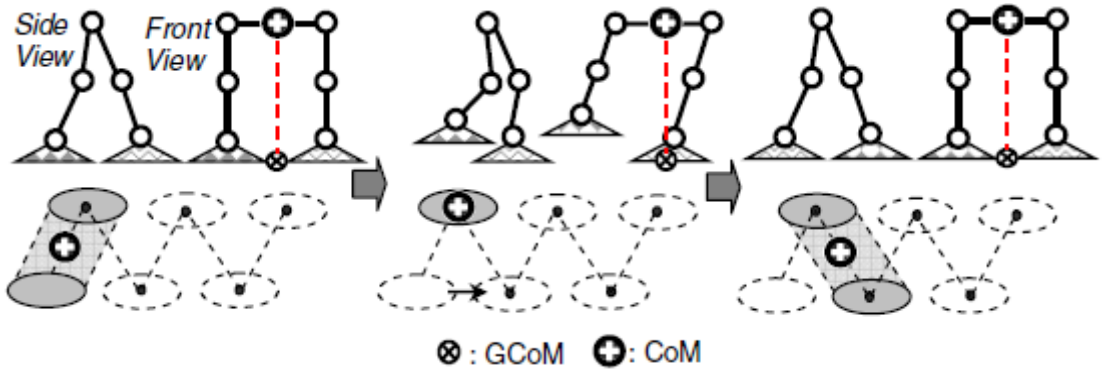
Zvýšenie rýchlosti chôdze

V oblasti napodobňovania ľudskej chôdze pri humanoidných robotoch a generovaní vzorov chôdze sa viedlo viacero štúdií. Napr. [16] sa zaoberala analýzou chôdze dvojnohých robotov založenej na chôdzi ľudí s cieľom zvýšiť rýchlosť pohybu. Keďže v reálnom svete je nutné aby robot prispôsoboval rýchlosť chôdze podľa situácie, je nutné aby analýzy metód generovania brali v úvahu rôzne parametre. Konkrétne štúdia [16] sa zamerala a brala v úvahu tri parametre a to: dĺžku kroku, výšku od bedrového kĺbu po zem a pomer času kedy sa robot dotýka zeme jednou, alebo oboma nohami. Pri výskume a testovaní bol použitý robot *Bonte-Maru 2*, ktorého technická realizácia napodobňovala človeka.

Vzor pre statickú realizáciu chôdze

V tejto časti sa nachádza vzor pre generovanie statickej chôdze. Na obrázku 4 môžeme vidieť natáčanie jednotlivých kĺbov pri chôdzi [16]. Dôležité je poznamenať rôznu polohu

ťažiska(CoM) s cieľom dosiahnuť stabilitu, pri prenášaní rovnováhy z jednej strany na druhú stranu. Na obrázku 4 vidíme premietnutie ťažiska robota na plochu po ktorej sa pohybuje. Pričom pre zachovanie stability je nutné aby sa projekcia ťažiska(GCoM) vždy nachádzala v oblasti chodidla na nohách. V opčnom prípade robot stráca stabilitu a padá.



Obr. 4.: Vzor pre statickú chôdzu humanoidného robota [16]

2.2.1.4 *Referenčné body pre pohyby humanoidného robota*

Na určovanie aktuálneho stavu robota boli vypočítané viaceré takzvané referenčné body. Dajú vypočítať podľa aktuálneho stavu robota a na základe ich vzájomných polôh vieme určiť jeho dynamický stav a overiť či poloha resp. pohyb robota spĺňa stabilizačné očakávania. Medzi ne zahŕňa Zero Moment Point, Foot Rotation Index a Centroidal Moment Pivot.

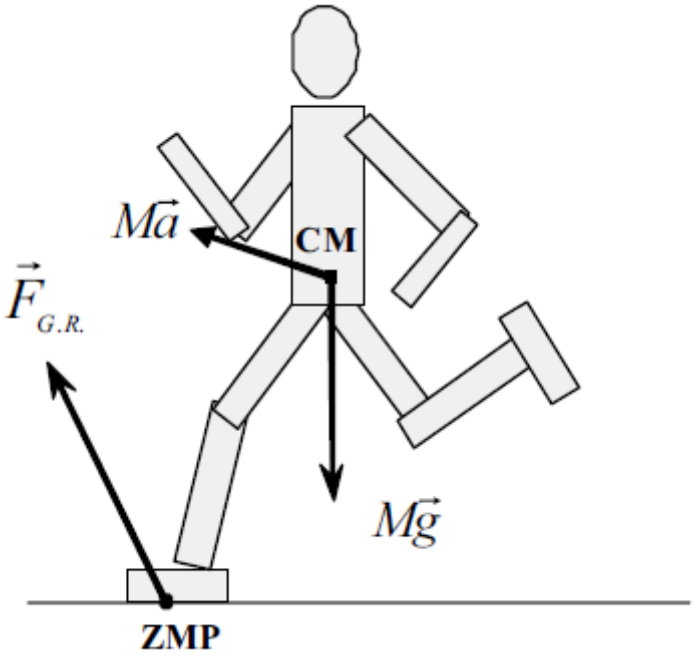
Zero Moment Point (ZMP)

Zero Moment Point je fyzikálna teória, ktorá je veľmi často využívaná pri dynamických pohyboch humanoidných robotov, najmä na plánovanie pohybov robota pri behu a skákaní. Okrem iných ju využívajú aj roboty Asimo, QRIO a TOPIO. Jej autorom je Miomir Vukobratović. Pri aplikácií v oblasti humanoidných robotov je užitočná na vypočítavanie bodov, v ktorých ak by sa do nich robot dokázal dostať, nepôsobila by naňho žiadna výsledná zotrvačná sila. Tým pádom je možné zabezpečiť stabilitu robota po dopade v behu, s ohľadom na veľkú zotrvačnosť jeho hornej časti tela, ktorá zaberá väčšinu jeho váhy a teda má veľkú zotrvačnosť. [17]

Pre prostredie robotického futbalu je využitie tejto metódy pre generovaní chôdze priam ideálne, keďže robot sa pohybuje neustále po rovnom povrchu. Výpočet ZMP očakáva, že zem a chodidlo majú plošnú stretávaciu plochu, a že odpor zemi je dostatočne veľký aby sa chodidlo nešmýkalo.

ZMP sa vypočítava z parametrov robota – pozícií jeho častí, ich hmotností, rýchlosti pohybu každej časti a umiestnenia dolných končatín na zemi. Výsledný bod je taký, že gravitačná sila, pôvodná sila zotrvačnosti a opačne pôsobiaca stabilizujúca sila z podstavy (nôh) robota sa v ňom navzájom rušia. Metóda ZMP je vhodná aj na určovanie aktuálneho stavu robota, a to tak že po výpočte aktuálneho ZMP je možné určiť nejakú oblasť okolo neho, v ktorej ešte teleso pokladáme za relatívne stabilné. Ak sa robot nachádza mimo neho, môžeme ho vyhlásiť za nestabilného a vykonať kroky so snahou o stabilizáciu.

Takto je možné aj odhadnúť budúce pôsobenia externých síl, ale aj pohybov robota na udržateľnosť jeho rovnováhy. Nie vždy je možné robota úplne stabilizovať, takže je vhodné aspoň optimalizovať jeho dynamiku čo najviac tak, aby sa príliš nevychýlil z rovnovážneho stavu a aby ho vždy bolo možné vrátiť do stacionárneho stavu bez toho aby spadol na zem.



Obr. 5.: ZMP sa nachádza v mieste, kde pôsobí spätná sila dotyku zeme, CM (center of mass) je miesto kde na robota pôsobí gravitačná sila a zotrvačnosť – ťažisko.

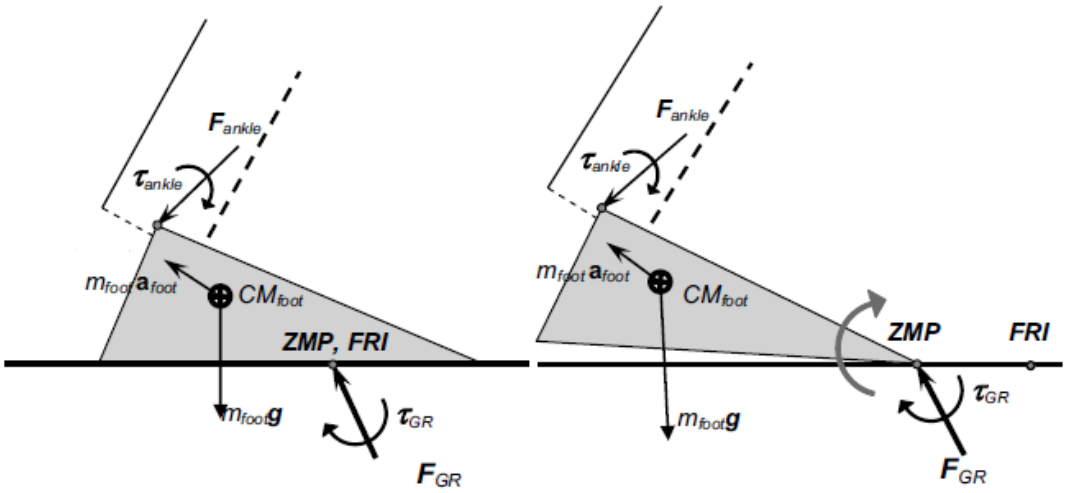
Foot Rotation Index (FRI)

Foot Rotation Index je rozšírením ZMP, je to teória ktorá udáva ako veľmi je zrýchľované otáčanie chodidla robota. V prípade robotov s dvoma nohami je pri chodení nutné v niektorých

fázach stáť na jednej nohe. V prípade že v čase státia na jednej nohe nie je udržaná rovnováha otáčania, je potrebné označiť takýto stav ako nestabilný, keďže dochádza k rotácií a posunu ZMP na okraj chodidla. V takomto momente sa nedá využiť ZMP na stabilizáciu. Preto je potrebné zaviesť koncept FRI, ktorý sa snaží pomôcť určiť zrýchľovanie otáčania podpornej nohy robota.

FRI je podľa [17] definovaný ako bod na stretávacej ploche chodidla a zeme, kde by mala pôsobiť spätná sila dotyku zeme, aby na chodidlo nepôsobil žiadny otáčavý moment. FRI je na rovnakom mieste ako ZMP v prípade, že je chodidlo v nehybnom stave, a odchyľuje sa od neho v prípade otáčavého zrýchľovania chodidla.

Význam FRI je aj v tom, že udáva informácie o tom, aká veľká otáčavá sila pôsobí na chodidlo robota keď stojí na jednej nohe. V prípade, že je FRI na rovnakom mieste v priestore ako ZMP, môžeme o chodidle robota vyhlásiť že je nehybná. V prípade, že sa nachádza FRI v určitej vzdialenosti od ZMP, dá sa z tejto vzdialenosti určiť veľkosť otáčavej sily pôsobiacej na jediné podporné chodidlo robota.



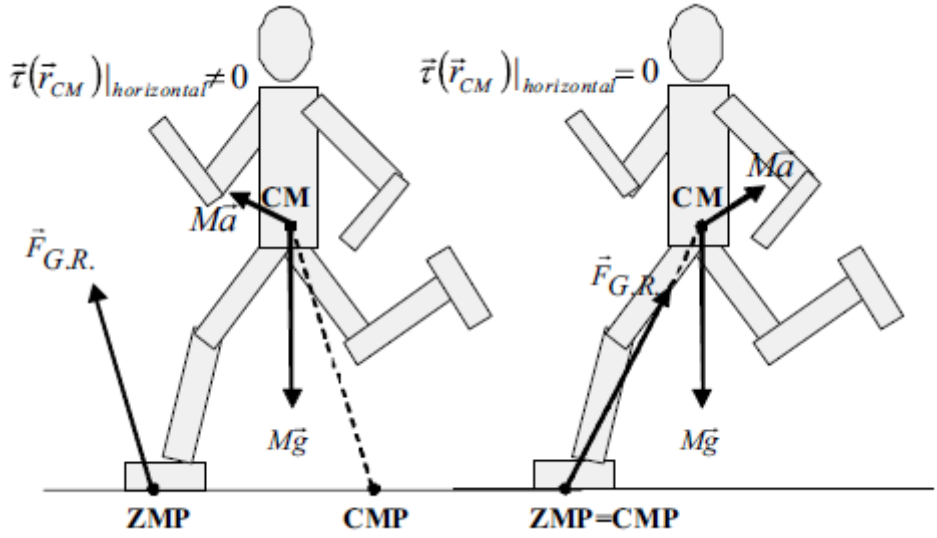
Obr. 6.: FRI je v mieste kde by mala pôsobiť spätná sila dotyku zeme tak, aby sa noha nepohla. Na ľavej časti obrázku je noha bez pohybu a bez pôsobenia otáčavých síl, takže FRI je v mieste ZMP. V pravej časti obrázku sa noha otáča, takže FRI je mimo plochy chodidla.

Centroidal Moment Pivot

Výskumy ľudského pohybu, otáčania, chodenia a behu ukázali, že uhlová hybnosť okolo ťažiska človeka je pri presúvaní po rovnej ploche veľmi malá v každej fáze pohybu. Čiže ľudské telo sa pri chodení natáča len veľmi málo, najmä vďaka tomu, že človek otáčanie spôsobené

vychyľovaním nôh koriguje pomocou hýbania rukami do opačného smeru. Počas behu sa ľudské telo okolo ťažiska otáča len vo fáze keď má niektorú z nôh na zemi, keďže pri „lete“ naň nepôsobia žiadne externé sily. V prípade humanoidných robotov s dvoma nohami je potrebné korigovať otáčanie tela robota okolo ťažiska tak, aby nebolo príliš veľká a robot by stratil stabilitu. Preto bola navrhnutá teória CMP, ktorá udáva otáčanie okolo ťažiska vzhľadom na zem.

Centroidal Moment Point je podľa [17] definovaný ako bod, kde sa priamka rovnobežná s vektorom spätnej sily dotyku zeme, prechádzajúca cez ťažisko, pretína so zemou, resp. podložkou na ktorej robot stojí. V prípade, že je CMP na rovnakom mieste ako ZMP, je otáčavý moment okolo ťažiska nulový. Ak je mimo neho, dá sa na základe ich vzájomných polôh určiť dynamické otáčanie celého tela robota.



Obr. 7.: CMP je miesto kde by spätná sila dotyku zeme mala pôsobiť, aby sa zachovala konštantná horizontálna uhlová hybnosť tela. V pravej časti obrázka je moment sily okolo ťažiska nulový, takže je CMP na rovnakom mieste ako ZMP. V ľavej časti je nenulový, udáva vzdialenosť medzi CMP a ZMP veľkosť momentu sily okolo ťažiska.

2.2.1.5 Padanie a vstávanie zo zeme

Aj napriek najväčším snahám o dokonalý pohyb robota sa môže stať, že sa robot pri vykonávaní pohybov začne vychyľovať a stráca stabilitu. V takomto prípade je vhodné vykonať stabilizujúce úkony, ktoré mu môžu pomôcť vrátiť sa do stabilného stavu. Väčšina prác, konštruktérov a iných tímov RoboCup3D ako stabilizujúci pohyb vykonáva pohyby podobné ľudským – vyvažovanie rukami, znižovanie ťažiska pokrčením nôh v kolenách a zrušením aktuálne

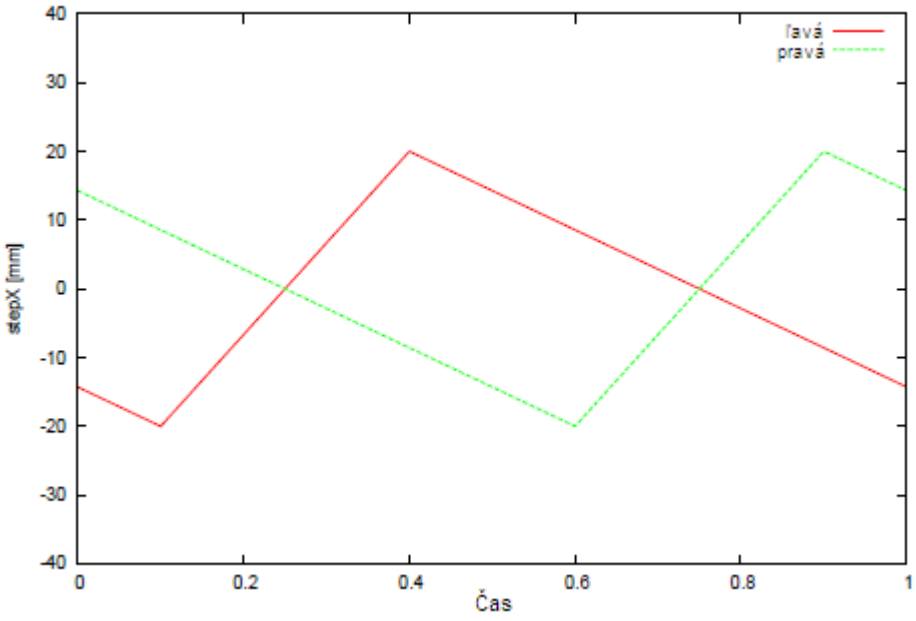
vykonávanej akcie. Udržanie stability je dôležitejšie, keďže pád výrazne oddiali vykonávanie iných akcií. Druhou možnosťou, v prípade hrania robotického futbalu očakávanou, ako môže stratiť robot stabilitu, sú externé vplyvy prostredia, lopty alebo iných hráčov na robota. Tieto často spôsobujú, že robot napriek snahe o stabilizáciu stratí rovnováhu a spadne na zem. V prípade brankára je dokonca padanie na zem očakávanou akciou pri chytaní lopty. Po páde sa však robot musí vedieť postaviť a dostať sa do stavu, z ktorého môže ďalej pokračovať v hre. [5]

Najčastejším spôsobom vstávania robotov po páde je, že má robot preddefinovanú určitú polohu, z ktorej sa dokáže najrýchlejšie postaviť. Takže sa robot najskôr musí dostať do tohto stavu, a to tak že po zaznamenaní pádu zanalyzuje svoj stav, teda ako leží. Následne sa na základe týchto údajov musí vedieť dostať do počiatočného stavu vstávania. Potom už zostáva len vykonať samotné vstanie. Iné riešenia prevažne začínajú z pozície ležania na bruchu, pokračujú opretím sa na ruky, postupne sa prevracia na chodidlá nôh a na záver sa vystrie.

2.2.1.6 Optimalizácia rojom častíc

Pohyb robota, ako napríklad chodenie alebo beh, je možné opísať ako postupnosť fáz pohybu, kde v každej fáze sa kĺby robota nachádzajú v zadaných uhloch. Takto sa dá k samotnému pohybu vytvoriť určitý model, ktorého simuláciou sa odsimuluje vytvorený pohyb. Takéto simulácie sú vhodné na otestovanie a zistenie aký „kvalitný“ daný pohyb je, ale aj na optimalizáciou namodelovaného systému pohybov. Tento prístup zvolili aj v [21], kde rozoberajú optimalizáciu chodenia robota metódou roja častíc. Optimalizácia rojom častíc je vytvorená na základe pozoravní správania biologických spoločenstiev, napríklad vtákov alebo hmyzu.

Základnou časťou každého systému je jedinec, ktorý sa nachádza v rámci prostredia systému a má v ňom svoju polohu a smer pohybu. Tento sa snaží svoju polohu optimalizovať podľa nejakej vyhodnocovacej funkcie. V každom optimalizačnom cykle sa jedinec prispôsobuje prostrediu na nájdenie najvhodnejšieho miesta v systéme, do úvahy berie známe informácie aj históriu svojho pohybu. Po ukončení si jedince navzájom môžu vymeniť svoje vedomosti, buď s niekoľkými náhodnými alebo so všetkými jedincami v prostredí.



Obr. 8.: Ukážka vývoja parametra stepX v závislosti od časovej fázy chodenia

V [21] namodelovali pohyb robota na túto optimalizačnú techniku pomocou určitých parametrov chôdze, ktoré by mali ovplyvňovať jej kvalitu, čiže rýchlosť a stabilitu. Ako parametre zvolili dĺžku a výšku kroku, ohýbanie tela dopredu a doboku (odstránenie váhy na nohe, ktorá sa dvíha) a trajektórie pohybov rúk na vytváranie rovnováhy pri kráčaní. Ako vyhodnocovaciu funkciu zvolili vzdialenosť, ktorú robot prejde za určitý čas. Následne spustili simuláciu vo virtuálnom prostredí na počítačoch, aby pomocou nej zistili čo najideálnejšie parametre optimalizačného algoritmu. Výsledkom bolo, že najlepšie výsledky vychádzali pri simulácií s 12 jedincami, kde si každý vymieňal informácie so všetkými ostatnými.

Následne s týmito parametrami začali optimalizovať reálneho robota, jeho chodenie po rovnej ploche. Robot mal 20 sekúnd na chodenie, začínal s malou dĺžkou kroku a postupne ju zvyšoval. Test realizovali v rôznych smeroch pohybu, v každom smere 25krát, pomocou klasického metra merali vzdialenosť, ktorú robot prešiel. Výsledky každej iterácie manuálne zadávali do laptopu, ktorý vykonával optimalizáciu. Výsledkom bolo chodenie rýchlosťou 17 cm/s smerom vpred, bez toho aby stratil rovnováhu. V samotnom článku [21] je možné nájsť konkrétne výsledné parametre pohybu.

2.2.1.7 Zhodnotenie

V tejto kapitole sme sa snažili opísať prístupy vývojárov, vedcov a niektorých iných tímov RoboCup3D k pohybom humanoidných robotov. Väčšina prístupov si ako hlavný podklad pre vytváranie chôdze berie pozorovanie chôdze človeka, s tým že sa ju snaží prispôsobiť na fyzikálne možnosti dané robotom. Chôdza robotov je viac-menej dobre vyvinutá, čo sa týka aplikácie na RoboCup, tu sa chodí len po rovných plochách, takže chôdza by nemala predstavovať väčší problém. Čo sa týka behu, je situácia horšia, máloktoré reálne roboty vedia bežať väčšou rýchlosťou ako 5-7 km/h. Problémom je aj stabilita pri behu a zastavenie, resp. zmena smeru pohybu. Tieto behajúce roboty sú však len prvými prototypmi takže sa dá v budúcnosti očakávať ich väčší rozvoj.

Veľmi dôležitou časťou pohybovania sa robota je jeho stabilizácia. Tejto téme sa venujú rôzne teórie, z ktorých vychádzajú výpočty bodov ako ZMP, CMP a FRI. Tieto teoretické body by nám mohli veľmi pomôcť pri vylepšovaní stability nášho hráča, keďže je to oblasť ktorá je v ňom dosť málo rozvinutá. Tu by sa dala aplikovať aj optimalizácia spomenutá na konci kapitoly, ktorá by nám umožnila vytvoriť kvalitnejší, teda rýchlejší spôsob chodenia. Súčasná situácia robota je taká, že má implementovanú len statickú chôdzu. Na základe informácií spomínaných v časti o dynamickej chôdzi môžeme vyhlásiť, že by sa mohla implementovať dynamická chôdza ktorá by mala byť rýchlejšia a efektívnejšia. Túto by sme následne zoptimalizovali. Využiť sa dajú aj informácie o spôsoboch pohybov robotov iných RoboCup3D tímov, ktoré sú opísane v inej kapitole.

2.2.2 Analýza prístupov iných tímov RoboCup3D

Nasleduje niekoľko nami vybraných tímov, ktoré boli zapojene do RoboCup 3D. Zamerali sme sa hlavne na tie tímy, ktoré boli úspešné.

Analýza tímu SEU-3D

Táto kapitola analyzuje tím SEU-3D z Číny, ktorý zvíťazil na mnohých významných súťažiach v Robocupe 3D ako napríklad RoboCup 2008. Pri analýze tímu využívame Team Description Paper tímu SEU-3D[12].

Architektúra agenta

Tím SEU-3D zvolil architektúru, ktorá nie je rozdelená do vrstiev, ale postavená na moduloch. Táto architektúra umožňuje zmeniť len jeden modul pri napr. zmenách na serveri a to jej dodáva veľkú flexibilitu.

Model sveta

Pre agenta je nevyhnutné, aby si stále pamätal model sveta, aby sa mohol správať inteligentne.

Lokalizácia agenta v poli

Tím SEU-3D pri lokalizácií agenta v poli využíval 360° pohľad agenta, no súčasný server podporuje len aktuálne zorné pole agenta, čím je ich spôsob lokalizácie pre nás nezaujímavý.

Ťažisko

Tento tím využíva ťažisko pri stabilizácií hráča. Ťažisko sa vyráta ako:

$$P_{COM} = \sum P_i \,.\, m_i$$

Kde P_i je pozícia každej časti tela a m_i jej hmotnosť.

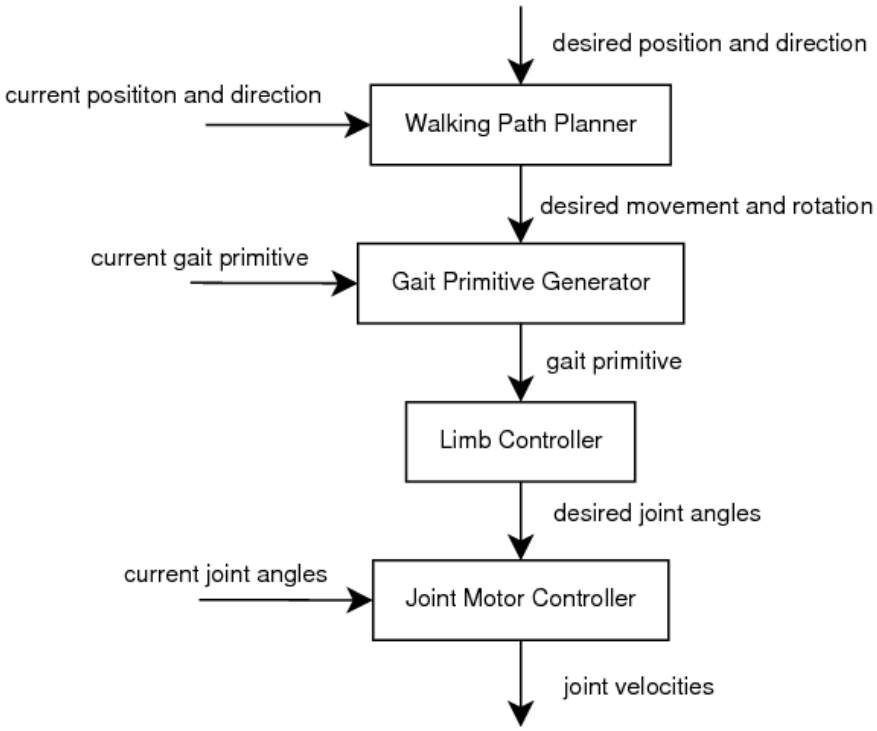
Komunikácia medzi agentmi

Agenti využívajú komunikáciu pre účely oboznámenia ostaných agentov o stave aktuálneho sveta. Táto komunikácia je veľmi dôležitá pre prípadné straty správ serverom. Žiaľ, jedna správa je limitovaná 20 bytmi zo strany servera [22].

Chôdza do rôznych smerov

Tím SEU-3D vytvoril chôdzu, ktorá je navrhnutá a rozdelená do 4 vrstiev.

Vrstva „walking path planner“ prijíma požadovanú pozíciu a smer a ďalej posúva potrebný pohyb a rotáciu vrstve „gait primitive generator“. Táto vrstva ďalej generuje ďalšie základné časti chôdze. V ďalšej vrstve „limb controller“ zisťuje požadované uhly kĺbov. Jednotlivé pohyby kĺbov už zabezpečuje vrstva „joint motor controller“. Tento spôsob riešenia umožňuje plynulú chôdzu bez zastávok potrebných na otáčanie agenta.



Obr. 9.: Architektúra „controller-u" chôdze[22]

Záver

Tento tím vytvoril veľmi zaujímavý koncept chôdze, ktorá je rýchla a nevyžaduje si zastavenia pre účely otáčania. Pri stabilizácií hráča využívajú polohu jeho ťažiska.

Analýza tímu UI-AI3D

V tejto kapitole sa budeme zaoberať analýzou tímu UI-AI3D, ktorý sa zúčastňuje na popredných súťažiach v RoboCup 3D. Pri analýze tímu budeme vychádzať z dokumentu Team Description Paper tímu UI-AI3D [10].

Architektúra agenta

Tím UI-AI3D zvolil viacvrstvovú architektúru, ktorá sa skladá z troch základných vrstiev:

- Komunikačná vrstva
- Vrstva manažmentu akcií
- Rozhodujúca vrstva

Komunikačná vrstva

Táto vrstva má na starosti komunikáciu so serverom a synchronizáciu procesov s časovaním server. Komunikačná vrstva sa ďalej stará o aktualizáciu sveta agenta po príchode nových

informácií zo servera. Ihneď ako príde správa zo server, komunikačná vrstva oznámi vrstve manažmentu akcií že už začal nový cyklus.

Vrstva manažmentu akcií

Táto vrstva má na starosti základné pohyby agenta ako chôdza, otáčanie, kopanie atď. Pre každý jeden základný pohyb dokáže vrstva rozhodnúť ako sa má vykonať pri rôznych situáciach.

Rozhodujúca vrstva

Táto vrstva sa stará o správanie agenta na základe situácie vo vonkajšom svete. Takisto je táto vrstva zodpovedná za stredno úrovňové pohyby, ktoré sú zložené zo základných pohybov.

Schopnosti agenta

Vstávanie

Tím UI-AI3D využíva pri vstávaní humanoidnú metódu, ktorá je vhodná pre ľubovoľné náhodne zvolené situácie. Pri vstávaní tím využíva veľké množstvo efektorov a perceptorov, aby sa agent dokázal prispôsobiť ľubovoľnej situácií.

Chôdza

Pri chôdzi je využitý koncept statickej stabilizácie , kde ťažisko je stále udržiavané nad agentovým podporným priestorom. Tento koncept je zložený z 3 častí:

1. Náklon vbok – v tomto stave agent premiestni ťažisko na podpornú nohu tlačením členkovým kĺbom proti zemi
2. Presun druhej nohy vpred – druhá noha sa presunie vpred a postaví sa na požadovanú pozíciu
3. Zmena polohy tela – po druhej časti pohybu je ťažisko umiestnené pri zadnej nohe, preto je potrebné presunúť telo agenta smerom vpred.

Tento spôsob chôdze je stabilný, no pomalý. Jeho potenciálne zlepšenie by spočívalo v dynamickej stabilizácií.

Lokalizácia agenta v poli

Tím UI-AI3D rieši lokalizáciu pomocou konverzie relatívnych vizuálnych vnemov agenta na globálne súradnice. Táto konverzia je uskutočňovaná prostredníctvom rotačných matíc.

Záver

Architektúra agenta je rozdelená na tri vrstvy, čo uľahčuje modifikáciu jednotlivých častí pri zmenách agenta. Tím UI-AI3D má dobre spracované základné pohyby a lokalizáciu agenta v poli, ktorá je veľmi dobrým základom pre budovanie vyššej logiky. Počas ich práce nevyužili dynamickú stabilizáciu, len statickú, čo spôsobilo pomalšiu chôdzu.

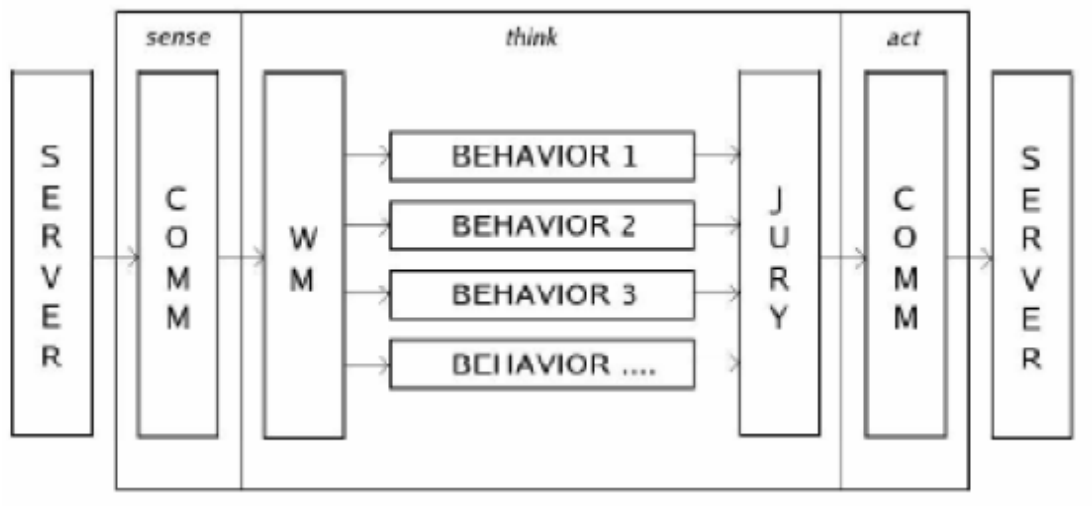
Analýza tímu Little green bats

Little green bats pozostáva zo študentov umelej inteligencie na univerzite v Groningene. Ide o pomerne úspešný holandský tím, ktorý dosiahol viacero úspechov na rôznych podujatiach v simulovanom 3D robotickom futbale. Medzi ich úspechy patrí druhé miesto z majstrovstiev v Atlante(2007), tretie miesto z majstrovstiev v Číne(2008). Boli kvalifikovaný aj na majstrovstvá 2009 v Gratzi, bohužiaľ tu sa im nepodarilo zasiahnuť do bojov o popredné miesta. V nasledujúcej Časti sa nachádza popis architektúry riešenia hráča. Nakoľko little green bats nezverejnili v súčasnosti žiaden oficiálny dokument, budeme sa opierať o analýzu [5].

Litlle green bats pri návrhu svojho hráča vychádzali z nasledovných princípov[5]:

- agent musí byť schopný dokončiť svoju úlohu,
- agent sa musí vedieť rozhodovať na základe situácie,
- rôzne typy správania musia byť rozdelené do skupín tak, aby si navzájom neprekážali,
- niektoré typy správania sú použiteľné iba ak sú súčasťou skupiny, alebo sú použité až po vykonaní iných typov.

Takto špecifikované požiadavky umožňujú programátorom hráča flexibilne zoskupovať správanie do rôznych skupín. Samotný hráč sa už len rozhodne, ktorá skupina je najviac aplikovateľná v danej situácii. Každý typ správania sa môže skladať z podtypov, výsledkom čoho je hierarchia správania.
Na obrázku 10 sa nachádza architektúra hráča, ktorá pozostáva z troch hlavných častí a komunikačného modulu. Obrázok znázorňuje spôsob komunikácie medzi jednotlivými komponentmi.



Obr. 10.: Little green bats architektúra hráča [11].

Model sveta

World model predstavuje modul, ktorého úlohou je reprezentovať stav sveta, ako napríklad aktuálne pozície spoluhráčov a lopty. Nachádzajú sa tu informácie, ktoré hráč obdŕžal zo strany servera.

Skupina správaní

Časť obsahujúca neusporiadanú skupinu rôznych typov správania [11] je založená na nasledovných princípoch:

- Analógia vrstevného systému. Jednotlivé správania komunikujú s modelom sveta alebo s ďalším rozhodovacím modulom
- Informácie prenášané medzi modelom sveta, správaniami a rozhodovacím modulom obsahujú aktuálnu a želanú pozíciu hráča. A taktiež interval <-1, 1>, ktorý vraví o vhodnosti sa dostať do cieľovej pozície. Pričom hodnota väčšia ako, 0 predstavuje vhodnosť. Hodnota rovná 0 nevhodnosť. Číslo menšie ako 0 hovorí o inverznom správaní k cieľovému.
- Možnosť implementovať rôzne separované typy správania sa.
- Možnosť vytvoriť konfiguračný súbor s hierarchickým usporiadaním jednotlivých správaní.

Rozhodovanie

Ide o modul označený ako JURY(porota). Jej úlohou je na základe vstupných informácií a situácie na ihrisku rozhodnúť sa pre vhodného správania sa. Poprípade určité typy správania spolu skombinovať.

Zhodnotenie

Aj keď tím little green bats je relatívne mladým tímom, ponúkajú kvalitné riešenie hráča v prospech ktorého hovoria výsledky, ktoré sa im podarilo dosiahnuť. Princípy z ktorých vychádzali pri definovaní hráča, samotný spôsob rozhodovania nám môžu poslúžiť ako určitý druh inšpirácie pri návrhu nášho vlastného enginu správania.

Analýza tímu Naito-Strikers

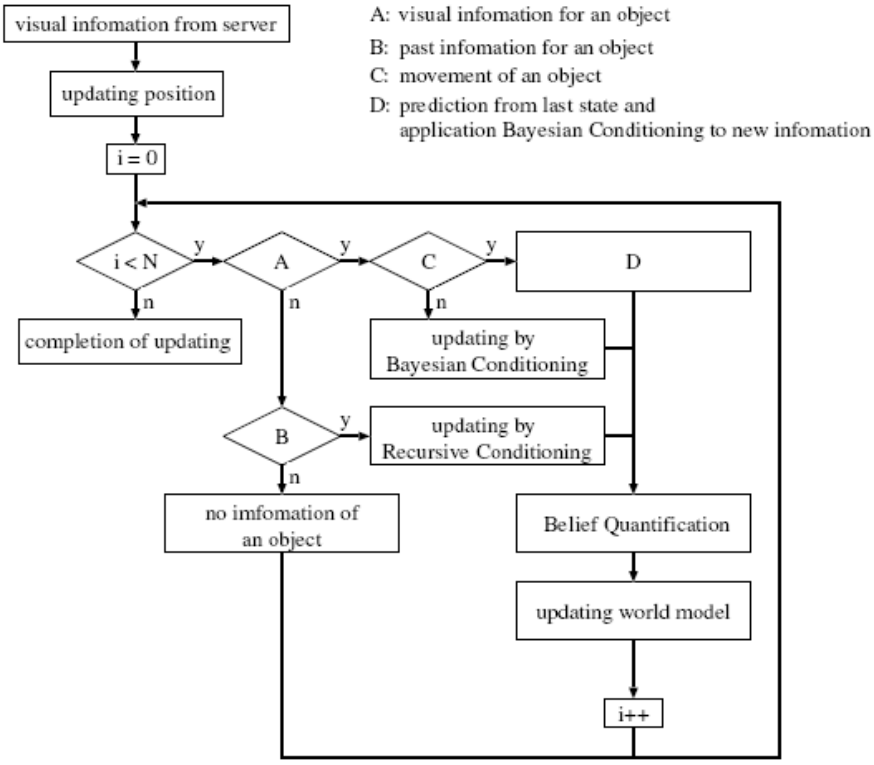
Tento japonský tím pri návrhu svojho hráča vychádzali z myšlienky, ako čo najvhodnejšie reprezentovať informácie zo strany servera. Keďže tie predstavujú základ pri rozhodovaní sa agenta. Výsledkom ich snaženia bol model sveta(*world model*), akýsi druh databázy obsahujúci informácie o pozíciách iných agentoch a lopte.
Pri reprezentácií sveta vychádzali z modelu M.Saxena a T. Guputa. Keďže tento model nebol úplne ideálny pre modelovanie reálneho sveta vylepšili ho [6]. Na obrázku 11 môžeme vidieť architektúru modelovania sveta.

Naito-Strikers obohatili pôvodný model o tieto zlepšenia:

- *Zmena informácií o pohybujúcich sa objektoch.* V prípade pohybov nejakých objektov, agent dokáže predpovedať svoju stav vo vzťahu k predchádzajúcemu stavu. Takto si môže doplniť svoje model sveta o nové informácie pomocou Bayesianových podmienok.
- *Zmena informácií o statických objektoch* pre zmenu informácií o súčasnom stave sveta sa použije klasický Saxenov a Guptov model.

Zhodnotenie

Naito-Strikers pri návrhu svojho hráča sa sústredili na čo najvhodnejšiu reprezentáciu sveta. S cieľom uchovávať informácie o pozíciách ostatných hráčov. Tieto informácie sú rozhodujúce a od nich sa odvíja samotné rozhodovanie agenta. Do architektúry zakomponovali Saxenov a Guputov model, ktorý sa využíva na modelovanie sveta. Pri výpočte pozícií ostatných hráčov využívajú triedu funkcií, výsledkom ktorých je v podstate pravdepodobnosť výskytu hráča v určitom čase na určitej pozícií. Aj tento spôsob riešenia problému pri našom hráčovi zvážime a budeme sa ním zaoberať.



Obr. 11.: Naito-Strikers model sveta [6].

Analýza tímu FC Portugal

Tím FC Portugal sa stál víťazom európskeho šampionátu v RoboCup s nastrieľanými 94 gólmi bez inkasovaného gólu v roku 2000. Tým priniesol niekoľko noviniek do RoboCup a to: flexibilnú stratégiu, taktiku, formácie a rôzne typy hráčov ktoré sa správajú odlišne v rôznych situáciách. Rozlíšenie pohybov v prípade ak hráč ma loptu a v prípade keď loptu nemá. Bližší popis pohybov je dostupný v článku[7].

Tímová stratégia

Spočíva v dynamickom prepínaní formácií. Ako vstup vstupujú entity ako aktuálny stav a zostatok času do konca zápasu. Tento základný koncept dokonca rozšírili o rôzne typy hráčov a vytvorili formácie. Ich prínosom bolo riešenie situácie založenej na strategickej pozícií (SBSP). SBSP je vytvorená pre strategické pozície v ktorých sa predpokladá, že agenti nevstúpia do aktívnej hry v blízkej dobe. Na výpočet strategickej polohy agent analyzuje taktiku, súperovu formáciu a vlastné umiestnenie na ihrisku. Táto pozícia je následné upravená v závislosti od pozícií a rýchlosti lopty ako aj od pozície protihráčov. Bližší popis SBSP je opísaný v článku [8]. Stratégia tímu určovala špecifické role v tíme s rôznymi schopnosťami. Využitie tejto stratégie

v našom tíme nebude pravdepodobne možné s časového hľadiska implementovať. Keďže obyčajný pohyb predstavuje dosť závažný problém. Ako inšpirácia pre vyššiu logiku hráča nám môže poslúžiť predkladaný model a architektúra agenta.

Architektúra agenta

Dynamická zmena pozícií a typov hráčov

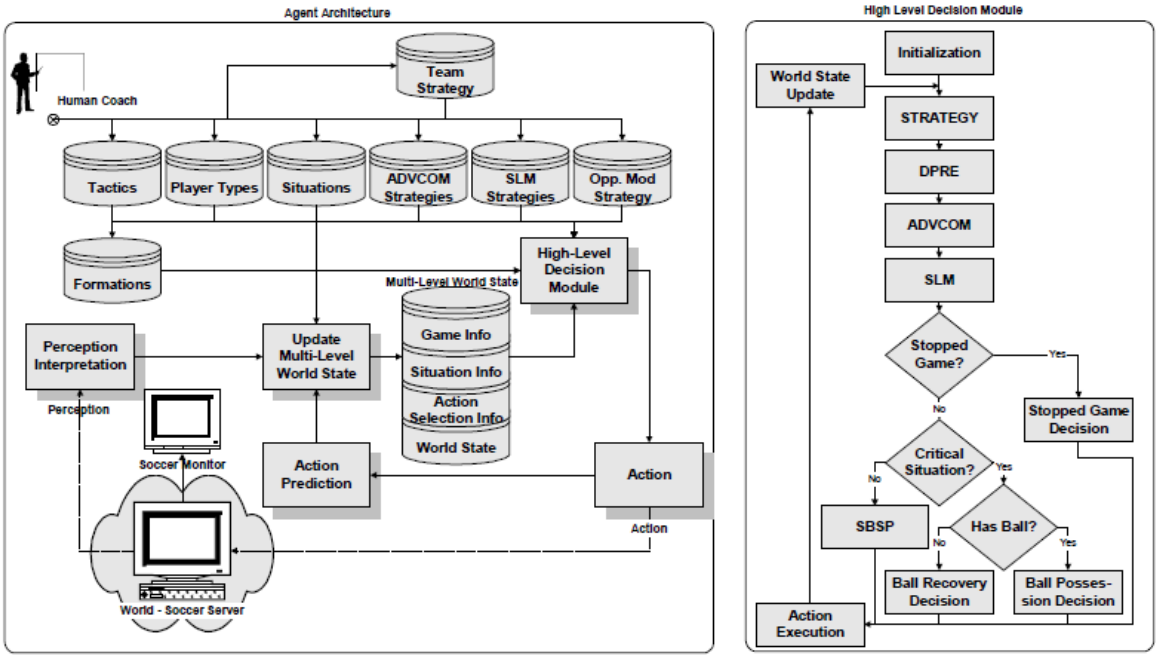
V závislosti od pozície lopty umiestnenia súperov, aktuálneho skóre a iných parametrov vedia hráči simulovať rôzne chovanie. Jeden typ hráčov je agresívnejší a pohybuje sa v súperovej oblasti. Iný typ hráča stabilizuje vlastnú pozíciu a vytvára zálohu v prípade neúspešného risku. Samostatnou kapitolou je chovanie brankára, ktorému pridali vlastnosti ako chytanie lopty a rozhodovanie sa vzhľadom na situáciu. Okrem iného používali heuristické metódy.

Defenzívna stratégia brankára

Podľa článku [8], identifikovali najslabšie chovanie v správaní brankára. Vytvorili model v ktorom sa brankár pohybuje po obdĺžniku blízko pokutového územia, tak aby v každom cykle si brankár všíma len loptu. Keď sa lopta pomaly približuje, brankárova strategická pozícia je určená prienikom línie ktorou prechádza lopta a stredom bránky a jeho aktuálnou pozíciou. V prípade, keď sa lopta pohybuje rýchlo musí predikovať ďalší krok bez náročného vyhodnotenia. V prípade ak brankár vyhodnotí, že aktívnym ani pasívnym spôsobom nedokáže dobehnúť k lopte skôr ako súper, snaží sa loptu zachytiť na bránkovej čiare, čo je zaujímavý spôsob. Zaujímavo je riešené aj rozhodovanie o tom, či má brankár chytiť loptu, alebo loptu má len vykopnúť čo závisí od vzdialenosti a postavenia súpera.

Optimalizovaný kop

Rovnako ako strategickému chovaniu brankára venovali pozornosť optimalizovanému kopu v článku [8]. Kopanie je založené na dynamike rýchlosti hráča a rýchlosti lopty pred kopom a po kope. Využíva sa pri ňom zákon zachovania hybnosti. K získaniu optimalizovaného kopu postupovali výpočtom a rovnako aj experimentálne. Pri kope zohráva úlohu šum, ktorý produkuje server na zanesenie náhody. Fc Portugal vytvorili model ktorý dokáže loptu kopnúť v ktoromkoľvek smere. Kopy boli testované na sade testov s ktorých sa podarilo dosiahnuť rýchlosť 2,47 ms-1. Rýchlosť kopnutia lopty a rovnako aj vzdialenosť kam sa lopta dostane závisí od celkovej hybnosti.



Obr. 12.: Architektúra agenta FC portugal a kontrola toku rozhodovania podľa [8].

Záver

Tento tím vytvoril veľmi zaujímavý koncept stratégie, ktorá je rýchla a dokáže sa dynamický meniť v závislosti od situácie. Závisí od množstva faktorov ktoré sú vyhodnocované. Ako najslabší pohyb a chovanie detekovali správanie a pohyby brankára, a vytvorili zaujímavý model jeho riešenia. Ďalšou vylepšenou vlastnosťou, bolo správanie a pohyby pri kope do lopty čoho výsledkom bolo vytvorenie optimalizovanej techniky kopania. Ako najväčší prínos sa mi zdá rozvinutie ktorému sa venovali a tvorilo jadro ich práce a to situácií založenej na strategickej pozícií hráčov.

Analýza tímu Agenty007

V tejto časti budeme analyzovať tím Agenty 007, ktorý bol na FIIT v roku 2008/09. A vyhral TP Cup v roku 2009. V tomto dokumente budeme vychádzať z finálnej dokumentácie tímu [24].

Rovnovážny modul

Tento modul má riešiť problémy pri pohybe robota. Robot sa napriek presným správam zo servera môže dostať do stavu, kedy sa nachádza v inej polohe ako sa v skutočnosti mal nachádzať. Môže sa tak stať napríklad pritom, keď sa zrazí robot so svojim protihráčom.

Zamerali sa hlavne na výpočet ťažiska robota. Tím sa v návrhu rozhodol používať goniometrické funkcie.

[Xn, Yn, Zn] ziskajNovuPolohuBodu(Xos,Yos, Zos, Xp, Yp, Zp, alfa, beta, gama)

Vstupy:

Xos,Yos, Zos – súradnice bodu, cez ktorý prechádzajú osi otáčania

Xp, Yp, Zp – počiatočné súradnice bodu

Alfa (os Z), beta(os Y), gama(os X) – uhly natočenia bodu okolo daných osí

Výstup:

Xn, Yn, Zn – koncové súradnice bodu po aplikovaní natočení okolo jednotlivých osí

Implementácia ale zahŕňala veľa problémov. Hlavný bol v tom, že samotný gyroskop, ktorý robot obsahuje nepracuje tak ako by sa dalo podľa názvu očakávať. Tieto problémy sa snažili rôznymi spôsobmi ošetriť. Výsledný postup ale nezaručuje vždy správne výsledky, keďže spôsob výpočtu používa predikciu.

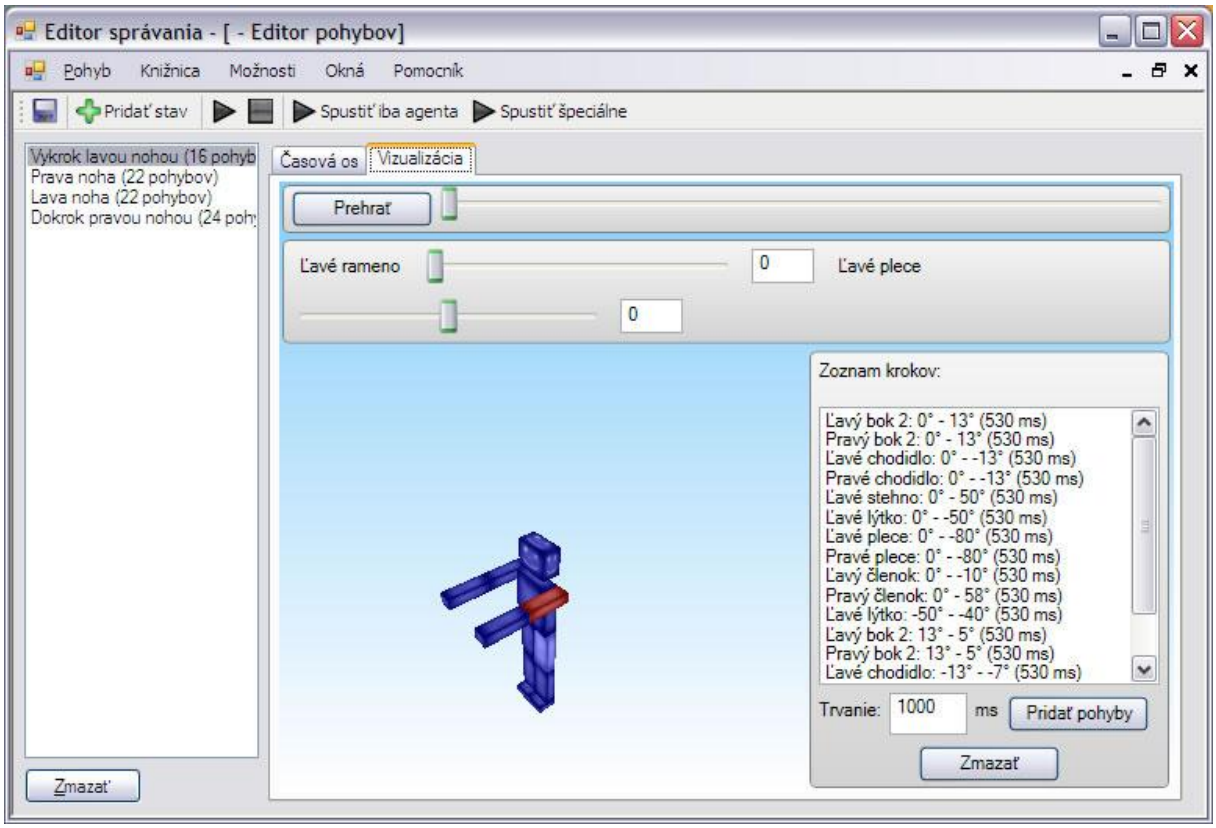
Editor pohybov

Idea tohto konceptu je jednoduchá. Tím sa rozhodol uľahčiť vytváranie pohybov pomocou jednoduchého editora pohybov. Najväčšou výhodou je, že samotné pohyby robota dokáže vytvoriť aj niekto, kto nemá žiadne skúsenosti s programovaním a ani so samotným robotom. Editor ponúka viacero možností na tvorbu pohybov. Pohyby sú reprezentované otočením potrebných kĺbov a tak nie je potrebná žiadna ďalšia implementácia (na rozdiel od DreamTeam-u).

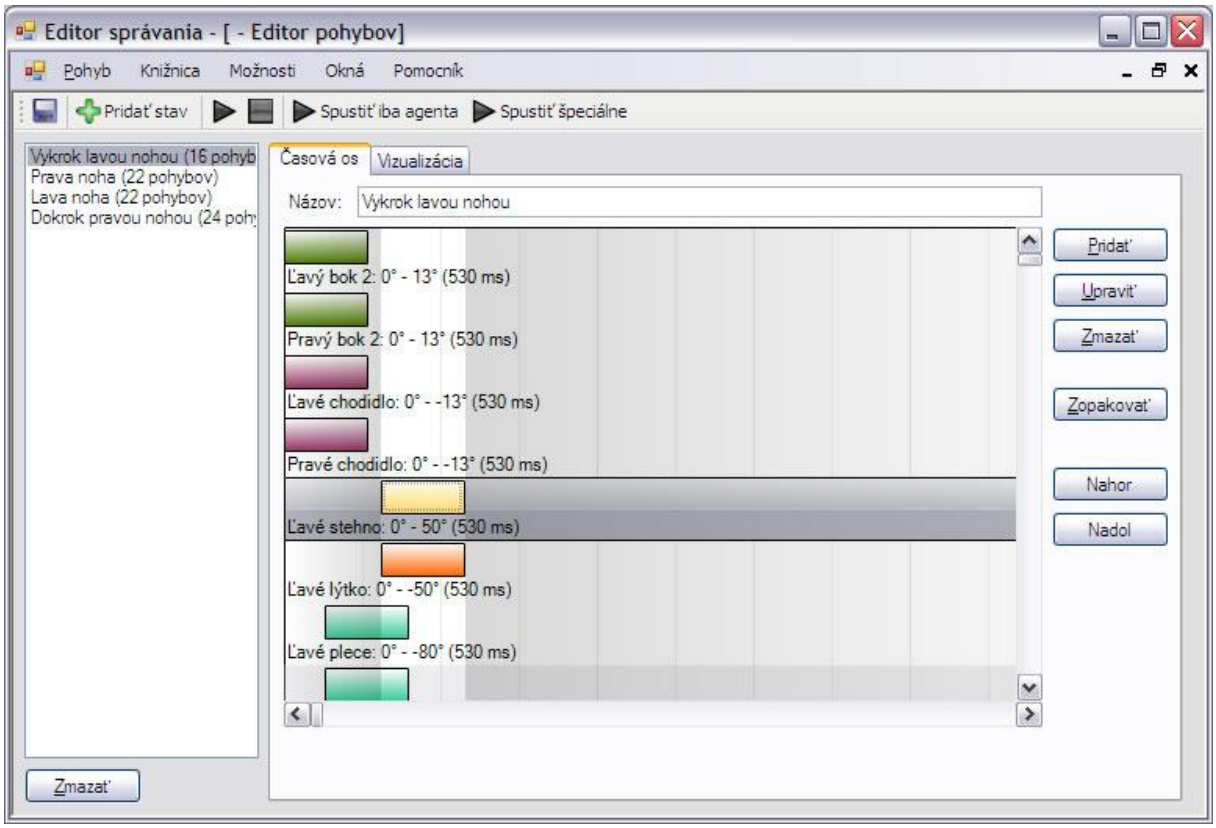
Samotné pohyby sa dajú exportovať do RMO súborov, ktoré využívajú INI zápis údajov. Parsovanie tohto súboru je tak veľmi jednoduché a zároveň samotný súbor je čitateľný a editovateľný v dostatočnej miere aj pre používateľa.

Vizualizácia pohybov je na veľmi dobrej úrovni. V tejto časti urobil tím podľa nás najväčší progres. Tím pomocou editora vytvoril všetky základné pohyby robota ako napríklad vstávanie, chôdza, otáčanie a iné. Samotné pohyby je možné rozdeliť do rôznych stavov, čo ešte viac zjednodušuje vytváranie zložitejších pohybov.

Nasleduje niekoľko obrázkov a príklad RMO súboru.



Obr. 13.: Grafický editor pohybov



Obr. 14.: Editor pohybov s využitím časovej osy

```
[MOTION]
NAME=PohybRukov
DESCRIPTION=
[STATE]
INDEX=0
NAME=Pohyb
SYNCH=false
[JOINTMOTION]
INDEX=1
JOINT=lae2
TIME=0
DUR=1000
STARTPOS=0
ENDPOS=93
```

Obr. 15.: Príklad jednoduchého RMO súboru na pohyb rukou

Záver

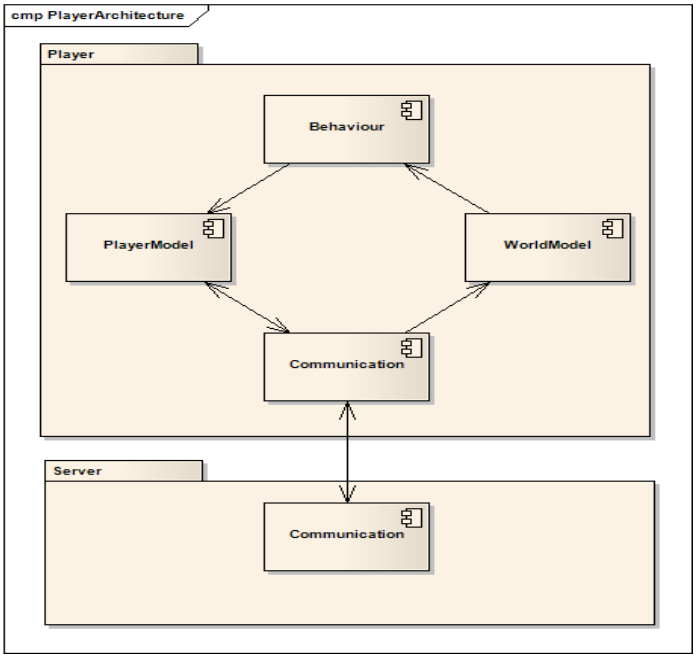
Náš projekt bude stavať na výsledok tohto tímu. Veľmi dobre je spracovaný editor pohybov. Pracuje sa s ním veľmi dobre, je ale potrebné aby sa niektoré veci doladili. Tím vytvoril veľmi dobré podmienky nato, aby sme sa mohli začať zaoberať pohybmi na trochu vyššej abstrakcií.

Analýza tímu DreamTeam

V tejto časti dokumentu budeme analyzovať slovenský tím DreamTeam, ktorý bol v roku 2008/09 na FIIT. V tomto dokumente vychádzame z finálnej dokumentácie tímu [25].

Architektúra Hráča

Tím sa rozhodol použiť architektúru pozostávajúcu z nasledujúcich komponentov



Obr. 16.: Architektúra hráča

Komunikácia (Communication)
Slúži na výmenu správ medzi hráčom a serverom. Súčasťou komunikačného komponentu je aj parser, ktorý zabezpečuje spracovanie prijatej správy zo servera a vytvorenie správy pre server.

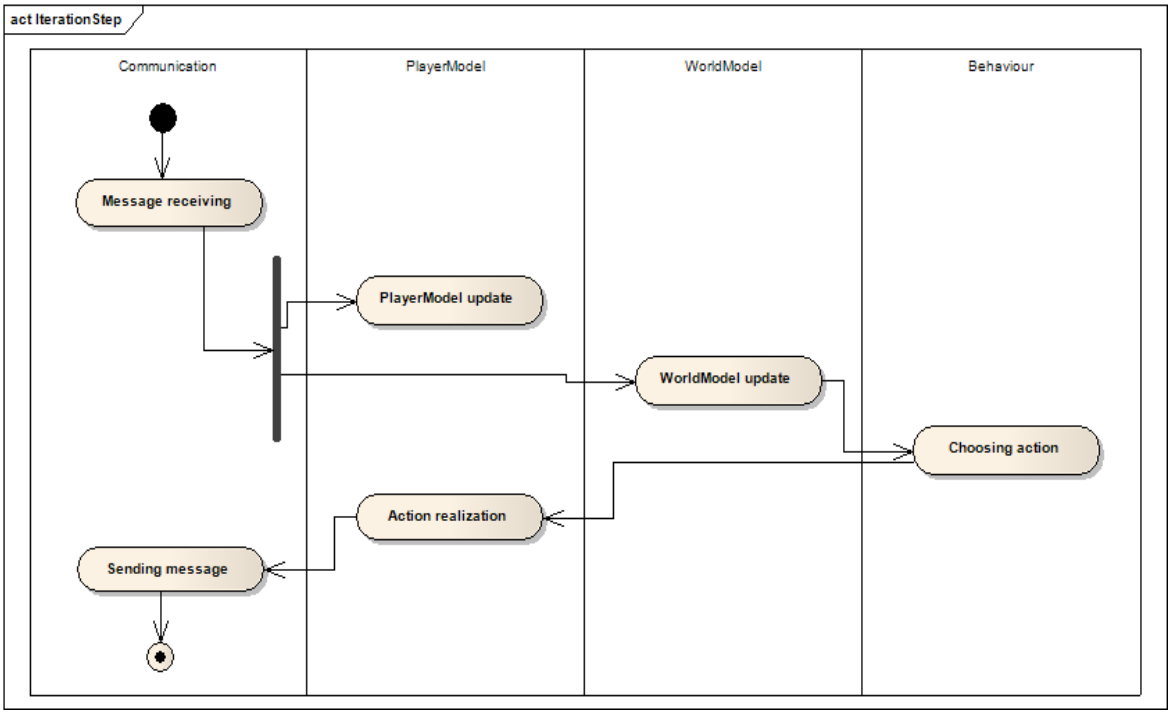
Model sveta (WorldModel)
Ide o komponent, ktorý v sebe nesie všetky informácie o okolitom svete, respektíve o objektoch v ňom. Tieto informácie získava z prijatých správ zo servera. Ide napríklad o polohu lopty, bránok, spoluhráčov ale aj protihráčov.

Model hráča (PlayerModel)
Obsahuje informácie o samotnom hráčovi. A to presne o polohách, natočeniach a rýchlostiach jeho jednotlivých kĺbov.

Správanie (Behaviour)
V tomto komponente sa vykonáva výber vhodnej akcie na realizáciu na základe modelu sveta.

Funkcionalita jednotlivých komponentov sa vykonáva pri cykle. Cyklus začína prijatím správy zo servera, nasleduje aktualizácia modelu hráča a sveta. Na základe zmeny modelov prichádza

na radu výber akcie, ktorá sa vykoná a následne sa odosiela znova na server. Nasledujúci obrázok demonštruje postupnosť pri jednej iterácii cyklu.



Obr. 17.: Krok cyklu

Dynamické vykonávanie činností

Prioritou tohto tímu bolo implementovanie dynamickej časti chôdzi. Na základe modelu sa má detekovať nekonzistentnosť a riziko pádu robota. Ak nastane taká situácia je potrebné, aby komponent správania vybral potrebné akciu na to, aby robot nespadol.

Schopnosti robota pomocou XML

Veľmi zaujímavým konceptom, ktorý implementoval tento tím je, vytváranie pohybov a schopností robota na základe XML súboru. Samotné schopnosti robota sú rozdelené do dvoch kategórií. Na „low skill“ (nízkoúrovňová schopnosť) a „high skill“ (schopnosť vyššej úrovne). Schopnosti vyššej úrovne v sebe zahŕňajú nízkoúrovňové schopnosti. Tie v sebe zahŕňajú fázy. Žiaľ samotný XML súbor na vytváranie pohybov nestačí. V samotnom kóde je potrebné pre každú novú s schopnosť implementovať špeciálnu triedu, ktorá bude implementovať jednotlivé metódy. Na nasledujúcom obrázku je znázornený príklad XML súboru.

```
<robot>
  <high_skills>
    <high_skill name="walk_to_ball">
      <use_low_skill skill="walking"/>
    </high_skill>
  </high_skills>

  <low_skills>
    <low_skill name="walking">
      <initial_phase name="chodzapriprava"/>
    </low_skill>
  </low_skills>

  <phases>
  <phase name="chodzapriprava" next="wageRight">
    <efectors>
      <efector name="lle2">
        <start>0</start>
        <end>-5</end>
      </efector>
      <efector name="rle2">
        <start>0</start>
        <end>5</end>
      </efector>
    </efectors>
    <finalization_phase>ROLLBACK</finalization_phase>
    <rescue_movement>PROCEED</rescue_movement>
    <speed_constant>1</speed_constant>
  </phase>
  </phases>
</robot>
```

Obr. 18.: XML súbor so schopnosťami robota

Záver

Najväčším prínosom tohto tímu bolo vytvorenie dynamickej chôdze. Je to jedna z vecí, ktorú by sme mohli v našom projekte využiť. Zaujímavou ale podľa nás nie moc použiteľnou vecou sú vyššie spomínané schopnosti robota zapísane do XML súboru. Najväčšou nevýhodou je podľa nás už to, že ide o XML súbor a jeho parsovanie môže zaberať zbytočne veľa času. Ďalšou nevýhodou je podľa nás aj to, že samotný XML súbor nestačí a je potrebné vykonať aj implementáciu objektov a metód.

2.2.3 Analýza pravidiel RoboCup

Keďže má RoboCup špecifické pravidlá, ktoré musí každý tím dodržiavať, musí sa ich aj tím naučiť. K analyzovaným pravidlám patria pravidlá hry, určité špecifiká hracieho prostredia, technické možnosti serveru a spôsob komunikácie medzi hráčom a serverom.

Analýza servera SimSpark

Server SimSpark pracuje sekvenčne a v každom cykle zbiera údaje so senzorov všetkých agentov a vyhodnocuje akcie vykonané efektormi agentov. Komunikácia medzi agentom a serverom prebieha pomocou S – výrazov, ktoré majú dĺžku vždy 32 bitov. V tejto kapitole sme vychádzali z opisu jednotlivých efektorov a perceptorov v používateľskom manuále [3]. Podrobnejší opis serveru a jednotlivých efektorov a perceptorov sa nachádza v prílohe.

Perceptory

Perceptory sú určené pre vnímanie okolia agentov.

Základné perceptory

GyroRate perceptor

Tento perceptor slúži na opísanie orientácie tela hráča.

Formát správy: (GYR (n <name>) (rt <x> <y> <z>))

Príklad: (GYR (n torso) (rt 0.01 0.07 0.46))

HingeJoint Perceptor

Tento perceptor slúži na určenie ohnutia kĺbov robota.

Formát: (HJ (n <name>) (ax <ax>))

Príklad: (HJ (n laj3) (ax -1.02))

UniversalJoint Perceptor

Tento perceptor bol v novšej verzií nahradený dvoma HingeJoint perceptormi.

Formát: (UJ (n <name>) (ax1 <ax1>) (ax2 <ax2>))

Príklad: (UJ (n laj1 2) (ax1 -1.32) (ax2 2.00))

Touch Perceptor

Tento perceptor slúži na detekciu kolízie hráčov.

Formát: (TCH n <name> val 0|1)

Príklad: (TCH n bumper val 1)

ForceResistance Perceptor

Tento perceptor slúži na detekciu pôsobenia sily a jej vektora.

Formát: (FRP (n <name>) (c <px> <py> <pz>) (f <fx> <fy> <fz>))

Príklad: (FRP (n lf) (c -0.14 0.08 -0.05) (f 1.12 -0.26 13.07))

Perceptory špecifické pre futbal

Vision Perceptor

Tento perceptor slúži agentovi na orientáciu v poli. Vision Perceptor zachytáva uhol 90°. S každým zachytením objektom je obsiahnutá aj informácia o:

- Vzdialenosť medzi hráčom a objektom
- Uhol v horizontálnej rovine
- Šírkový uhol

Formát: (See (<name> (pol <distance> <angle1> <angle2>)) (P (team <teamname>) (id <playerID>) (pol <distance> <angle1> <angle2>)))

Príklad: (See (F1L (pol 19.11 111.69 -9.57)) (F2L (pol 16.41 -115.88 -11.15))

(F1R (pol 46.53 22.04 -3.92)) (F2R (pol 45.49 -18.74 -4.00)) (G1L (pol 9.88 139.29 -21.07))

(G2L (pol 8.40 -156.91 -25.00)) (G1R (pol 43.56 7.84 -4.68))(G2R (pol 43.25 -4.10 -4.71))

(B (pol 18.34 4.66 -9.90))(P (team RoboLog) (id 1)(pol 37.50 16.15 -0.00)))

GameState Perceptor

Tento perceptor slúži na zistenie veľkosti ihriska a lopty.

Formát: (GS (t <time>) (pm <playmode>))

Príklad: (GS (t 0.00) (pm BeforeKickOff))

AgentState Perceptor

Tento perceptor slúži na zobrazenie aktuálneho stavu batérie a teploty.

Formát: (AgentState (temp <degree>) (battery <percentile>))

Príklad: (AgentState (temp 48) (battery 75))

Hear Perceptor

Tento perceptor slúži na komunikáciu medzi hráčmi.

Formát: (hear <time> 'self'|<direction> <message>)

Príklad: (hear 12.3 self ``helloworld'')

Efektory

Základné efektory

Základné efektory slúžia k definícií základného správania agenta.

Create Effector

Create Effector slúži na odovzdanie názvu súboru agentovi, ktorý opisuje hráča.

Formát: (scene <filename>)

Príklad: (scene rsg/agent/soccerbot056.rsg)

HingeJoint Effector

Tento efektor slúži na ohyb kĺbov o príslušný uhol.

Formát: (<name> <ax>)

Príklad: (lae3 5.3)

UniversalJoint Effector

Tento efektor už nie je podporovaný novým serverom. Slúžil na pohyb kĺbu v smere dvoch osí.

Formát: (<name> <ax1> <ax2>)

Príklad: (lae1 2 -2.3 1.2)

Efektory špecifické pre futbal

Tieto efektory slúžia na ovládanie špecifických vlastností.

Init Effector

Init Effector sa spúšťa po Create Effectore a priraďuje hráča k vybranému tímu.

Formát: (init (unum <playernumber>)(teamname <yourteamname>))

Príklad: (init (unum 1)(teamname FHO))

Beam Effector

Umiesťnuje hráča na hraciu plochu pred začiatkom hry.

Formát: (beam <x> <y> <rot>)

Príklad: (beam 10.0 -10.0 0.0)

Say Effector

Say Effector sa využíva k odosielaniu správ iným hráčom.

Formát: (say <message>)

Príklad: (say ''helloworld'')

Pravidlá simulačnej 3D ligy 2009

Pravidlá sú platné pre ligu simulovaných robotov rok 2009.

Hra pozostáva z dvoch polčasov, z ktorého každý trvá 5 minút.
- Proti sebe hrajú dva tímy. Každý tím pozostáva z 3 hráčov.
- Za výhru sú pridelené 3 body.
- Za remízu je pridelený 1 bod.
- Pri kontumácií je nastavené skóre na 3: 0.

Celkové pravidlá a ľudský rozhodca:

1. OC testuje všetky tímy pred prvým kolom v prípravnom dni. OC nie je zodpovedná za testovanie tímov počas hry.
2. Medzi kolami majú tímy povolené vymeniť verziu vlastných agentov. Výmena je na vlastné riziko. Tímy nemajú prístup do svojho domovského adresára počas kola.
3. Hra je automatický spúšťaná skriptom. Preto je potrebné, aby sa dal hráč spustiť automatickým skriptom.
4. Na každú hru dohliada rozhodca. V mnohých prípadoch je rozhodca vybraný z OC členov. Rozhodca môže byť vybraný aj z dobrovoľníkov, ale potrebuje ovládať dobré pravidlá.
5. Rozhodcom nemôže rozhodovať zápas v ktorom hrá jeho vlastný tím

6. Rozhodca rozhoduje v situáciách ktoré nemôžu byť detekované simulátorom. Vrátane faulov a situácií, keď hra uviazne.
7. Góly sú taktiež posudzované rozhodcom, v prípadoch keď nie sú správne uznané simulátorom.
8. V nepredvídaných situáciách sa rozhodca rozhoduje podľa jeho najlepšieho rozumu.
9. Počas hry môže zasahovať len jeden člen tímu. Zvyčajne je to tím líder.
10. Rozhodnutia rozhodcu sú záväzné.
11. Reklamácie proti rozhodnutiu rozhodcu sú riešené po skončení aktuálneho kola.
12. V prípade ak v situácií rozhodca nemôže rozhodnúť, rozhodca a jeden člen z každého tímu sa obrátia na OC

Pravidlá správania sa agentov

Ležanie brankára pred bránkou

Ak brankár alebo iný agent leží pred bránkou viac ako 30s družstvo je potrestané stratou lopty a lopta je umiestnená na roh pokutového územia. Ak agent pokračuje so svojím správaním viac ako štvrtinu z hry bude vylúčený z hry. Rovnako je vylúčený ak je správanie vyhodnotené ako zámerné.

Dotyk rukou

Brankársky agent nemá právo sa dotýkať lopty rukou, okrem prípadu ak sa nachádza vo svojom pokutovom území. Aby bolo možné identifikovať jedného z agentov ako brankára, mal by mať na drese číslo jedna.

Držanie lopty

Ak hráč drží loptu viac ako 5s jeho tím je potrestaný stratou lopty.

Výkop

Je neplatné dať gól z výkopu. Gól musí byť z akcie, nie z priamočiareho výkopu. V prípade že súper dá gól z výkopu, ten sa neuzná.

Normálna chôdza

Z dôvodu dodržania realistických pohybov, je povinné chodiť biologický realistickým spôsobom. Pri nedodržaní tohto pravidla je súper potrestaný stratou lopty a odkopom. Ak je takéto správanie posúdené ako úmyselné, alebo trvá viac ako štvrtinu z hry zápas je kontumovaný s výsledkom 3:0 s výhodou pre súpera.

Blokovanie Lopty

Je zakázané úmyselne blokovať loptu v snahe zabrániť sa súperovmu agentovi sa dostať k lopte. Napríklad ležaním, alebo blokovaním všetkých trás. Táto situácia je potrestaná stratou lopty a kopom pre súpera.

Tlačenie a ťahanie

Nie je dovolené úmyselne ťahanie alebo tlačenie agenta. Rozhodca by mal požívať vlastný rozum, aby odhalil takéto správanie. Ak súper úmyselne tlačí, alebo ťahá súpera bude potrestaný stratou lopty a kopom pre súpera.

Dotýkanie lopty

Nie je dovolené dotýkať sa lopty viac ako 20 sekúnd nepretržite. Ak takéto správanie nastane. Tím je potrestaný stratou lopty a je udelený voľný kop pre súpera.

Pravidlá v rámci fair play

Cieľom hry je hrať futbal bez faulov s použitím zdravého rozumu a braní ohľadov na obmedzenia simulácie vo virtuálnom svete 3D futbalu.

Za porušenie fair play je sú považované

1. Použitie binárky iného tímu.
2. Zahltenie simulátora posielaním nadbytočných príkazov na klienta.
3. Priama komunikácia hráčov inými, ako povolenými metódami.
4. Práca s konkurenčným strojom a úmyselné reštartovanie.
5. Odchytávanie komunikácie súpera a následné podhodenie správ.
6. Deštruktívne porušenie súpera.

V prípade pochybnosti o porušení pravidiel je sa potrebné opýtať organizačného výboru pred turnajom. Ak sú zistené nekalé praktiky počas turnaja. Tím je automatický diskvalifikovaný.

Ak sa agent odpojí 30 sekúnd pred začatím od simulátora hra je reštartovaná najviac však 3 krát. Ak problém pretrváva po dohode oboch tímov je možné urobiť zmeny v kóde, alebo vymeniť binárku. Zásah do kódu možne byť vykonaný iba ak oprava bude trvať menej ako 2 minúty. To je posledná šanca pre tím. V opačnom prípade je zápas kontumovaný a končí s výsledkom 3:0.
V prípade zlyhania simulátora rozhodca rozhodne o pokračovaní v hre, alebo reštartovaní hry. Rozhodnutie výboru je konečné a nemôže byť uznaná dohoda.

Analýza fyzikálneho modelu hráča a prostredia

Táto časť sa zaoberá analýze fyziky hráča, fyzikálnych pravidiel v simulátore a fyzikálnym vzťahom. Snaha tvorcov simulátora je priblížiť sa čo najvernejšie fyzikálnemu modelu reálneho sveta. Napriek všetkej snahe sú niektoré sily a fyzikálne zákony nezapracované v modeli. Komponent simulovaného systému, soket server automaticky vyvoláva dva druhy agentov, ktorí okamžite začínajú cyklus vnímania, premýšľanie a vykonanie akcie. Monitorovacie nástroje sú použite na grafické zobrazenie hry. Prostredie futbalového simulátora je akási skrinka, ktorá obsahuje virtuálne futbalové polia. Rešpektuje FIFA špecifikáciu pre medzinárodne futbalové zápasy.

Fyzika hráča

3D model reprezentujúci hráča (agenta) hrajúceho simulovaný futbal RoboCup 3D je vytvorený podľa reálneho robota NAO. Tento robot bol vytvorený firmou Aldebaran Robotics. Výška robota NAO je 57 cm a jeho hmotnosť je 4,5 kg.

Počítačový 3D model (Obr. 19) pozostáva z 13 častí, pričom spojenie medzi týmito časťami zabezpečujú kĺby.

a) b) c)

Obr. 19.: Hráč RoboCupu (agent). a) spredu b) z boku c) zozadu

Po inicializácii agenta sa jeho jednotlivé časti nastavia na prednastavené hodnoty, ktoré môžeme vidieť v prehľadnej tabuľke na Obr. 20.

Name	Parent	Translation	Mass	Geometry	Name	Anchor	Axis	Min	Max
neck	torso	0, 0, 0.09	0.05	Cylinder L: 0.08 R: 0.015	HJ1	0, 0, 0	0,0,1	-120	120
head	neck	0, 0, 0.065	0.35	Sphere 0.065	HJ2	0, 0,-0.005	1,0,0	-45	45
shoulder	torso	0.098, 0, 0.075(r) -0.098, 0, 0.075(l)	0.07	Sphere 0.01	AJ1	0, 0, 0	1,0,0	-120	120
upperarm	shoulder	0.01, 0.02, 0(r) -0.01, 0.02, 0(l)	0.150	Box 0.07, 0.08, 0.06	AJ2	-Translation	0,0,1	-95(r) -1(l)	1(r) 95(l)
elbow	upperarm	-0.01, 0.07, 0.009(r) 0.01, 0.07, 0.009(l)	0.035	Sphere 0.01	AJ3	0, 0, 0	0,1,0	-120	120
lowerarm	elbow	0, 0.05, 0	0.2	Box 0.05, 0.11, 0.05	AJ4	-Translation	0,0,1	-1(r) -90(l)	90(r) 1(l)
hip1	torso	0.055, -0.01,-0.115(r) -0.055, -0.01,-0.115(l)	0.09	Sphere 0.01	LJ1	0, 0, 0	-0.7071,0,0.7071 (r) -0.7071,0,-0.7071(l)	-90	1
hip2	hip1	0, 0, 0	0.125	Sphere 0.01	LJ2	0, 0, 0	0,1,0	-45(r) -25(l)	25(r) 45(l)
thigh	hip2	0, 0.01, -0.04	0.275	Box 0.07, 0.07, 0.14	LJ3	-Translation	1,0,0	-25	100
shank	thigh	0,0.005,-0.125	0.225	Box 0.08, 0.07, 0.11	LJ4	0,-0.01, 0.045	1,0,0	-130	1
ankle	shank	0, -0.01,-0.055	0.125	Sphere 0.01	LJ5	0, 0, 0	1,0,0	-45	75
foot	ankle	0, 0.03,-0.035	0.2	Box 0.08, 0.16, 0.03	LJ6	0,-0.03, 0.035	0,1,0	-25(r) -45(l)	45(r) 25(l)
Torso			1.2171	Box 0.1, 0.1, 0.18					

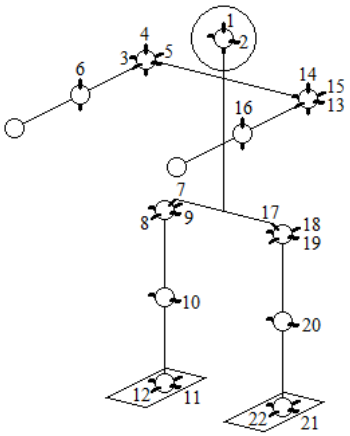
Obr. 20.: Časti robota a ich parametre [3]

Vysvetlenie jednotlivých hodnôt:

- Name – názov konkrétnej časti tela
- Parent – časť tela, na ktorú je naviazaná daná časť tela
- Translation – posunutie bodu pre danú časť tela od rodičovskej
- Mass – hmotnosť časti tela
- Geometry – tvar časti tela
- Name – názov kĺbu v danej časti tela
- Anchor - miesto umiestnenia kĺbu v rámci danej časti tela
- Axis – os otáčania kĺbu
- Min – minimálna hodnota uhlu natočenia kĺbu
- Max – maximálna hodnota uhlu natočenia kĺbu

Vzájomná poloha jednotlivých častí agenta sa mení zmenou uhlov medzi kĺbmi. Poznáme 2 druhy týchto kĺbov:

- **jednoduché kĺby** – obsahujú 1 os otáčania
- **zložité kĺby** – obsahujú 2 alebo 3 osi otáčania



Číslo osi	Názov indexu	umiestnenie	min. hodnota [°]	max. hodnota [°]
1	HEAD1	krk	-120	120
2	HEAD2	krk	-45	45
3	RIGHTARM1	pravé rameno	-120	120
4	RIGHTARM2	pravé rameno	-95	1
5	RIGHTARM3	pravé rameno	-120	120
6	RIGHTARM4	pravý lakeť	-1	90
7	RIGHTLEG1	pravý bok (45°)	-90	1
8	RIGHTLEG2	pravý bok	-45	25
9	RIGHTLEG3	pravý bok	-25	100
10	RIGHTLEG4	pravé koleno	-130	1
11	RIGHTLEG5	pravý členok	-45	75
12	RIGHTLEG6	pravý členok	-25	45
13	LEFTARM1	ľavé rameno	-120	120
14	LEFTARM2	ľavé rameno	-1	95
15	LEFTARM3	ľavé rameno	-120	120
16	LEFTARM4	ľavý lakeť	-90	1
17	LEFTLEG1	ľavý bok (45°)	-90	1
18	LEFTLEG2	ľavý bok	-25	45
19	LEFTLEG3	ľavý bok	-25	100
20	LEFTLEG4	ľavé koleno	-130	1
21	LEFTLEG5	ľavý členok	-45	75
22	LEFTLEG6	ľavý členok	-45	25

Obr. 21.: Kĺby agenta

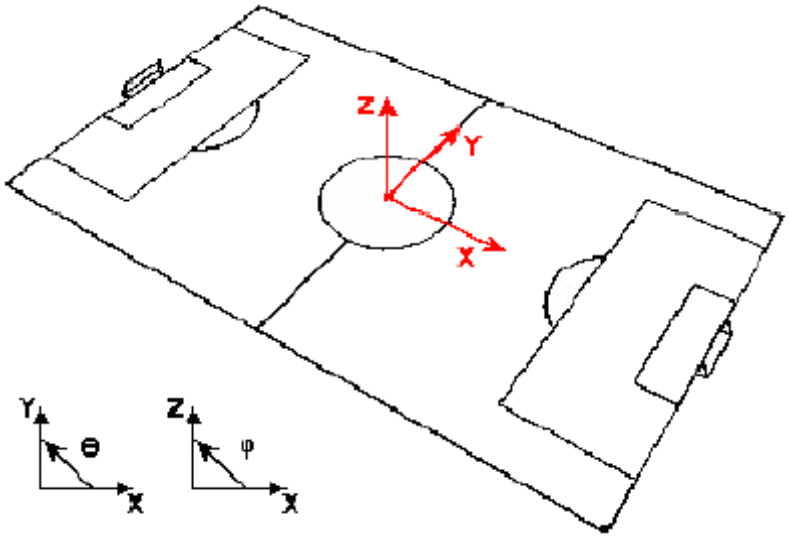
Obr. 21 ukazuje rozmiestnenie jednotlivých kĺbov agenta a všetkých osí, ktoré sa v nich nachádzajú. Kĺby celkovo obsahujú 22 osí, ktoré sú umiestnené na 11 miestach. 2 kĺby sú jednoduché a ostatné sú zložité.
Obrázok 21 bol stiahnutý z http://student.fiit.stuba.sk/~kvetan04/Robocup3D/agent/Klby hraca.png

Fyzika prostredia

RoboCup 3D simulátor je softvér, ktorý poskytuje dva elementy a to jadro systému, nazývané futbalový server a monitorovacie nástroje. Cieľom simulátora sú tri úlohy:

- umožniť hráčom vnímať a následne konať v prostredí
- rozhodovať v niektorých situáciách vo futbalovom zápase
- simulovať fyzikálne zákony platné v reálnom svete.

Globálne súradnice systému sú nasledovne:
X-ová os rozširuje celu horizontálnu líniu, ktorá ide zľava doprava.
Y-ová os je kolmá na x-ovú os a predstavuje šírku ihriska.
Os Z ide dohora a dotvára tak priestorovú líniu. Celkový uhol 0 stupňov je priamo na x-ovej osi a rastie v smere hodinových ručičiek. Grafická reprezentácia jednotlivých osi je na obrázku 1.



Obr. 22.: Grafická reprezentácia súradníc.

Agenti v simulovanom futbale majú množinu efektorov na vykonávanie činnosti v prostredí a množinu perceptorov na vnímanie prostredia. Obidva sú reprezentované reťazcovou formou ako S-výraz.

Pohybový efektor: používa sa na pohyb agenta v poli. Simuluje motor, ktorý berie kartézky 3D vektor ako vstup s maximálne povolenou dĺžkou 100 jednotiek. S-výraz pohybového efektora je (drive <Dx><Dy><Dz>).

Kopací efektor: Urýchľuje loptu radiálne preč od agentovho poľa. Agent musí byť vo vzdialenosti najviac 0.07m. Od lopty aby ju mohol odkopnúť. Jej vstup je šírkový uhol a sila medzi 0 až 100. Výraz je (kick <angle> <power>).

Fyzikálny model

Modely boli odvodené z formálnych matematických analýz. Pre získanie presných a prípustných parametrov pre pohybový model bola použitá štatistická analýza a nástroje na počítanie kriviek. Parametre boli získané za ideálnych podmienok, šum produkovaný simulátorom v efektoroch a perceptoroch sa nebral do úvahy.

Definicia pohybového modelu:

Pohybový model dynamických objektov je opísaný formálne ako funkcia určitej dvojice (pt,vt) v čase t a dvojica (pt+i,vt+i) v čase t+i, kde p je vektor polohy a v je vektor rýchlosti a i > 0 to znamená M: (pt, vt) --> (pt+i, vt+i)

Pohybový model agenta:

Pohyb agenta je ovplyvnený hnacou silou (používa z efektora) a odporom vzduchu. Hnacia sila je definovaná ako Fd = <Fdx,Fdy> vektor sily nemá Z-ovú zložku pretože agent nemôže skákať v implementácií servera (verzia 0.5.2). Odpor vzduchu je definovaný ako Fvzd = - ξv. Konštanta ξ reprezentuje koeficient, ktorý ukladá odpor vzduchu telu a v je rýchlosť tela. Rovnica modelu sily pre každú os.

$$\begin{aligned} \sum F_x &= F_d cos(\theta) - \xi_A v_x = m_A a_x \\ \sum F_y &= F_d sin(\theta) - \xi_A v_y = m_A a_y \end{aligned}$$

m_A je hmotnosť agenta, a je zrýchlenie, Fd je hnacia sila, ξ je koeficient odporu vzduchu a θ je celkový horizontálny uhol v x-y rovine. Nech D je z [0,100] a je hnacia sila v percentách, zaslaná pohybovému efektoru. Vzťah medzi D a Fd je daný:

$$F_d = \frac{D}{100} F_{d_{max}}$$

Diferenciálna rovnosť pohybového modelu v x-ovej osi je vyjadrená ako diferenciálna rovnica v tvare

$$m_A \frac{dv}{dt} + \xi_A v = F_{d_x}$$

Model rýchlosti agenta

Riešením diferenciálnej rovnice pohybu dostávame vzťah:

$$v(t) = \frac{F_d}{\xi_A} + Ae^{-\frac{\xi_A}{m_A}t}$$

$$A = v_i - \frac{F_{d_x}}{\xi_A}$$

$$v(t) = \frac{F_{d_x}}{\xi_A} + (v_i - \frac{F_{d_x}}{\xi_A})e^{-\frac{\xi_A}{m_A}t}$$

Konštantný výraz A môže byť nahradený inicializačnou podmienkou v(0) = v_i ako substitúciou dostávame výraz

nakoniec pre jednoduchosť počítame s konštantnými podmienkami

kde Va_T je konečná rýchlosť agenta. Konečná rýchlosť modelu agenta je vyjadrená ako

$$v(t) = v_{A_T} + (v_i - v_{A_T})e^{-t/\tau_A}$$

ζ_A ζ_A

Pozičný model agenta

je odvodený integrovaním vzťahu konečnej rýchlosti modelu od 0 po t a teda kde pi je

$$p(t) = p_i + v_{A_T}t + \tau_A(v_i - v_{A_T})(1 - e^{-t/\tau_A})$$

integračná konštanta a reprezentuje počiatočnú pozíciu.

Model pohybu lopty

Na rozdiel od agenta pohyb lopty je rozdelený na dve fázy. V prvej fáze pôsobí sila kopu na loptu. Let lopty je ovplyvňovaný gravitačnou silou, odporom vzduchu a silou kopu. V druhej fáze lopta spomaľuje a je ovplyvnená odporom vzduchu a gravitáciou.

Prvá fáza pohybu lopty

V prvej fáze sa lopta správa ako agent, s pôsobením konštantnej sily. Vektor sily je definovaný ako

$$\vec{F_k} = \langle F_{k_x}, F_{k_y}, F_{k_z} \rangle$$

Nech θ_k je celkový horizontálny uhol v rovine x-y medzi agentom a loptu a Φ_k je uhol sklonu poslaný efektorom kopu. Rovnice modelu sily pre každú os sú:

$$\begin{aligned} \sum F_x &= F_k cos(\phi_k) cos(\theta_k) - \xi_B v_x = m_B a_x \\ \sum F_y &= F_k cos(\phi_k) sin(\theta_k) - \xi_B v_y = m_B a_y \\ \sum F_z &= F_k sin(\phi_k) - m_B g - \xi_B v_z = m_B a_z \end{aligned}$$

kde mB je hmotnosť lopty, a je zrýchlenie, Fk je sila kopu. A ξ je koeficient odporu vzduchu. Nech K je z [0,100] a je hnacia sila v percentách, zaslaná kopaciemu efektoru. Vzťah medzi K a Fd je daný:

$$F_k = \frac{K}{100} F_{k_{max}}$$

Rovnako ako u agenta, môžeme zovšeobecniť jedna rovnicu pre obidve osi X a aj Y, ale musíme definovať inú rovnicu pre Z-os. Diferenciálna rovnica, pre os x a y je vyjadrená ako diferenciálna rovnica pohybu pre Z os je vyjadrená ako

$$m_B \frac{dv}{dt} + \xi_B v = F_k$$

$$m_B \frac{dv_z}{dt} + \xi_B v_z = F_{k_z} - m_B g$$

Druhá fáza pohybu lopty

$$\begin{aligned} \sum F_x &= -\xi_B v_x = m_B a_x \\ \sum F_y &= -\xi_B v_y = m_B a_y \\ \sum F_z &= -m_B g - \xi_B v_z = m_B a_z \end{aligned}$$

V druhej fáze, lopta spomaľuje, kým sa nezastaví v pohybe v X -ovej a Y -ovej osi, a to až kým sa nezastaví pohyb v osi Z. Silu môžeme vyjadriť nasledovnými vzťahmi.

Diferenciálnu rovnicu pohybu pre os X a Y môžeme vyjadriť vzťahom

$$m_B \frac{dv}{dt} + \xi_B v = 0$$

a pre os Z vzťahom

$$m_B \frac{dv_z}{dt} + \xi_B v_z = -m_B g$$

Iné fyzikálne vlastnosti

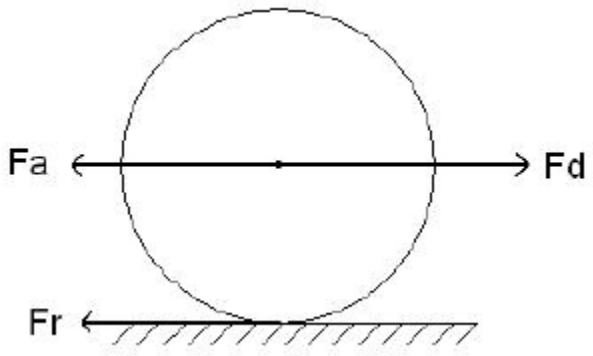
Pri odvodzovaní pohybového modelu neboli brané do úvahy niektoré fyzikálne vlastnosti. Jednou z nich je zachovanie hybnosti, ktoré udáva, že hybnosť je konštantná v uzavretom systéme častíc. Rovnica hybnosti je p=m*v. Tento prípad sa v RoboCup 3D objavuje v troch odlišných situáciách.

1. Keď sa agent zrazí z inými agentmi.
2. Keď sa agent zrazí s loptou.
3. Keď agent kopne do lopty.

Pri kolízii dvoch agentov sa musí moment pred kolíziou rovnať súčtu momentov po kolízii.

Inak povedané musí platiť rovnosť: mA1uA1 +mA2uA2 = mA1vA1 +mA2vA2. Kde u reprezentuje rýchlosť pred kolíziou a v reprezentuje rýchlosť po kolízií. Rovnosť je významná len pri detekcií kolízií. Nanešťastie agenti nemajú senzory na detekciu kolízie. I keď kolízia môže byť odvodená pri detekovaní náhlej zmeny rýchlosti. Rovnica o zachovaní hybnosti je stále k ničomu, pretože to vyvolá dve zmeny va1 a va2 a rovnosť zachovania energie nemôže byť použitá, lebo kolízia je nepružná. Podobný prípad nastáva pri kolízií agenta s loptou.

Pri kope do lopty môže byť moment vypočítaný v jednoduchom prípade, keď agent stojí vedľa lopty v kľude. Zachovanie momentu pred kopnutím do lopty a po kopnutí je dané vzťahom 0 = mAvA+mBvB. Keď zoberieme priemerné hodnoty a to mA = 75kg a mB = 0,43kg a vB je približne 13m/s ak je kopnutá z pokojového stavu a dosadíme do vzorca dostávame vA sa približne rovná -0,0745m/s. Hodnota vA je relatívne malá voči maximálnej rýchlosti agenta 1.616m/s. S toho dôvodu sa neberie rýchlosť do úvahy analýzy modelu pohybov. Ďalším aspektom je krútiaci moment agenta a trecej sily ktorá by mohla vzniknúť. Obrázok 23 ukazuje sily, ktoré pôsobia na agenta. V tomto diagrame je Fd je sila ktorá je konštantná, ak nie je výslovne agent zmení. Fa je trenie vzduchu, ktoré závisí na rýchlostí agenta. A Fr je trenie s podlahou. Príručka futbalového servera nestanovuje nič, o sférických agentoch. Sila, ktorá by mohla vyvolať zmeny na zrýchlenie agenta je Fr. Sila Fr nemá veľkú hodnotu na prekonanie zotrvačnej sily vzhľadom na hmotnosť agenta.



Obr. 23.: Sily pôsobiace na agenta.

Implementácia základných pohybov

2.2.4 Základný pohyb – chôdza

2.2.4.1*Analýza*

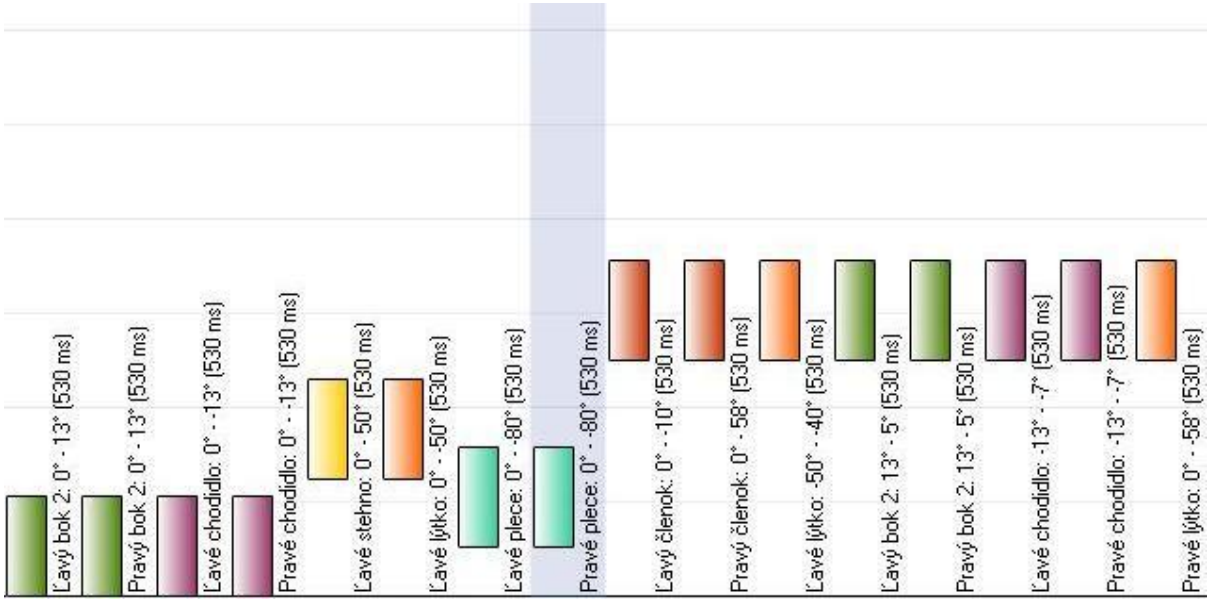
Pri simulácii pohybu hráča sme vychádzali z chôdze človeka. Pri vytváraní jednotlivých pohybov sme sa zamerali najmä na stabilitu hráča nie na rýchlosť presunu. Pri natáčaní jednotlivých kĺbov sme vychádzali z analýzy chôdze humanoidných robotov. Inšpirovali sme sa vzormi pre generovanie ľudskej chôdze. Keďže v súčasnej verzii hráča nie je implementovaná fyzická stabilizácia, nasimulovaný pohyb je pomalý, ale relatívne stabilný. Pri samotnej realizácii pohybu použijeme editor tímu Agenty007.

2.2.4.2*Návrh*

Pre realizáciu chôdze bolo potrebné zapojiť veľké množstvo kĺbov, najmä v dolnej časti tela agenta. Do pohybu bol zapojený bedrový kĺb, lýtka, stehná a chodidlá. S cieľom vytvoriť chôdzu, ktorá sa bude najviac podobať tej ľudskej do pohybu sme zapojili aj ruky.

2.2.4.3*Implementácia*

Pre realizáciu pohybu sme použili editor pohybov tímu Agenty007. V tejto časti sa nachádza fyzická realizácia pohybu. Teda kĺby pri ktorých dochádza k natáčaniu, uhol o aký sa majú otočiť a dĺžka trvania pohybu. Samotný pohyb je súčasťou editora pohybov. Chôdza je rozdelená na 24 väčšie pohyby- Výkrok ľavou nohou, pohyb pravou a ľavou nohou a dokrok pravou nohou. Konkrétne na obrázku 17 sa nachádza výkrok ľavou nohou.



Obr. 24.: Postup krokov pre výkrok ľavou nohou.

2.2.4.4 *Testovanie*

Test sa uskutočnil na dvoch počítačoch s rôznymi technickými parametrami. **PC1:** Procesor: 1,8GHz, RAM: 1,5GB, OS: Win XP (32bit). **PC2**: Procesor: 2,24GHz, RAM:3GB, OS: Win Vista (32bit). V tabuľke 1 a 2 sa nachádzajú testy pre jednotlivé konfigurácie počítačov.

Počítač	**Očákavaný výsledok**	
PC1	Agent vykonáva chôdzu a dostane sa do polovice ihriska.	
Poradové číslo spustenia	**Výsledok testu**	**Skutočný výsledok**
1.	Zlyhal	Agent spadol hneď pri pokuse spraviť krok
2.	Zlyhal	Agent spravil iba jeden krok a spadol
3.	Zlyhal	Agent spadol hneď pri pokuse spraviť krok
4.	Zlyhal	Agent spadol hneď pri pokuse spraviť krok

Tabuľka 1 – test vykonaný na PC1

Počítač	**Očákavaný výsledok**	
PC2	Agent vykonáva chôdzu a dostane sa do polovice ihriska.	
Poradové číslo spustenia	**Výsledok testu**	**Skutočný výsledok**
1.	Zlyhal	Agent spravil iba jeden krok a spadol
2.	OK	Agent sa dostal do polovice ihriska
3.	OK	Agent sa dostal do polovice ihriska
4.	OK	Agent sa dostal do polovice ihriska

Tabuľka 2 – test vykonaný na PC2

2.2.4.5 *Zhodnotenie*

Z výsledkov testu je možné vidieť, že úspech pohybu do značnej miery záleží od konfigurácie počítača. Viditeľné nedostatky môžeme zaznamenať pri rýchlosti vykonávania pohybu. Tento problém je možne odstrániť fyzickou stabilizáciou hráča. Nasimulovaný pohyb predstavuje základ od ktorého sa môžeme odraziť pri realizácií pohybov, ktoré sa budeme snažiť do určitej miery parametrizovať.

2.2.5 Základný pohyb – otáčanie

2.2.5.1*Analýza*

Pri simulácii otáčania hráča sme vychádzali z pozorovania otáčania sa človeka. Pri vytváraní jednotlivých pohybov sme sa zamerali najmä na stabilitu hráča nie na rýchlosť otáčania. Pri natáčaní jednotlivých kĺbov sme vychádzali z humanoidných robotov. Keďže v súčasnej verzii hráča nie je implementovaná fyzická stabilizácia, nasimulovaný pohyb je pomalý a trochu nestabilný. Pri samotnej realizácii pohybu použijeme editor tímu Agenty007.

2.2.5.2*Návrh*

Pre realizáciu otáčania bude potrebné zapojiť veľké množstvo kĺbov, najmä v dolnej časti tela agenta. Do pohybu bol zapojený bedrový kĺb, lýtka, stehná, torzo a chodidlá. Robot sa bude otáčať tak, že pohyb začne naklonením na stranu, aby udržal rovnováhu pri zdvihnutí nohy. Následne zdvihne jednu nohu a otočí sa na druhej do strany, na ktorú sa chce otočiť. Počas pohybu hýbe ostatnými časťami tela tak aby udržiaval rovnováhu. Na záver sa postaví naspä na zdvihnutú prvú nohu, zdvihne druhú a priloží ju k otočenej nohe.

2.2.5.3*Implementácia*

Pre realizáciu pohybu sme použili editor pohybov tímu Agenty007. V tejto časti sa nachádza fyzická realizácia pohybu. Teda kĺby pri ktorých dochádza k natáčaniu, uhol o aký sa majú otočiť a dĺžka trvania pohybu. Samotný pohyb je súčasťou editora pohybov. Otáčanie je rozdelené do fáz opísaných v návrhu, každá z nich sa vykonáva v inom časovom intervale.

2.2.5.4*Testovanie*

Test som urobil na počítačoch s takýmito parametrami: Procesor: 2,24GHz, RAM:3GB, OS: Win Vista (32bit). V tabuľke 3 sa nachádzajú výsledky testu.

Počítač	**Očákavaný výsledok**	
PC1	Agent sa otočí o približne 90° v danom smere.	
Poradové číslo spustenia	**Výsledok testu**	**Skutočný výsledok**
1.	OK	Agent sa otočil o 90°
2.	OK	Agent sa otočil o 90°
3.	OK	Agent sa otočil o 90°
4.	OK	Agent sa otočil o 90°

Tabuľka 3 – výsledky testu otáčania robota

2.2.5.5*Zhodnotenie*

Hráč sa vie otáčať do oboch strán postupne a bez pádu. Otáčanie však je vždy konštantné, nedá sa zmeniť uhol, o ktorý sa má hráč otočit. Avšak zmena uhlu je len záležitosťou zmeny otočenia kĺbu medzi torzom a otáčanou nohou tak, aby sa otočila o očakávaný uhol. Bolo by však asi potrebné aj zmeniť stabilizačnú časť pohybu tak, aby sa robot neprevážil na niektorú stranu.

V prípade otáčania o uhol väčší ako 90° treba doimplementovať prechodnú pozíciu s oboma nohami na zemi, a odtiaľ začať ďalšie otáčanie.

2.2.6 Základný pohyb – vstávanie

2.2.6.1*Analýza*

Vstávanie je jeden z najdôležitejších pohybov, je to jediné zotavenie po páde, preto treba zaistiť, aby hráč bol schopný vstať. Pôvodný pohyb a editor sme prevzali od tímu Agenty007.

2.2.6.2*Návrh*

Aby sa agent mohol postaviť, musí najprv spadnúť. Súčasťou pohybu bude spadnutie na brucho a nasledovné postavenie sa zo zapojením rôznych častí tela. Agent musí byť schopný sa nakoniec vyrovnať.

2.2.6.3*Implementácia*

Pohyb bol pôvodne vytvorený v editore pohybov tímu Agenty007 a tam bol aj upravovaný. Pozostáva z troch častí:

1. Pád dopredu (používa členky a plecia)
2. Vstanie (používa boky, lýtka, plecia, lakte, stehná)
3. Vyrovnanie (používa plecia, stehná, boky, členky)

Pôvodný pohyb tímu Agenty007 už bol funkčný, v rámci realizácie úlohy boli niektoré uhly a časy jemne modifikované.

2.2.6.4*Testovanie*

Test sa uskutočnil na dvoch počítačoch s rôznymi technickými parametrami. **PC1:** Procesor: 1,8GHz, RAM: 1,5GB, OS: Win XP (32bit). **PC2**: Procesor: 2,24GHz, RAM:3GB, OS: Win Vista (32bit). V tabuľke 4 a 5 sa nachádzajú testy pre jednotlivé konfigurácie počítačov.

Počítač	**Očákavaný výsledok**	
PC1	Agent spadne na brucho a potom sa postaví.	

Poradové číslo spustenia	**Výsledok testu**	**Skutočný výsledok**
1.	Zlyhal	Agent stratil rovnováhu uprostred vstávania
2.	Zlyhal	Agent stratil rovnováhu uprostred vstávania
3.	OK	Agent sa úspešne postavil
4.	Zlyhal	Agent stratil rovnováhu uprostred vstávania

Tabuľka 4 – test vykonaný na PC1

Počítač	Očákavaný výsledok	
PC2	Agent spadne na brucho a potom sa postaví.	

Poradové číslo spustenia	Výsledok testu	Skutočný výsledok
1.	OK	Agent sa úspešne postavil
2.	OK	Agent sa úspešne postavil
3.	Zlyhal	Agent stratil rovnováhu uprostred vstávania
4.	OK	Agent sa úspešne postavil

Tabuľka 5 – test vykonaný na PC2

2.2.6.5*Zhodnotenie*

Na prvom počítači bol agent menej stabilný, čo môže byť zapríčinené aj nestabilnou komunikáciou zo serverom. V druhom teste sme dosiahli lepšie výsledky, môžeme predpokladať, že agent je schopný vstať z brucha.

2.2.7 Základný pohyb – kop do lopty

2.2.7.1*Analýza*

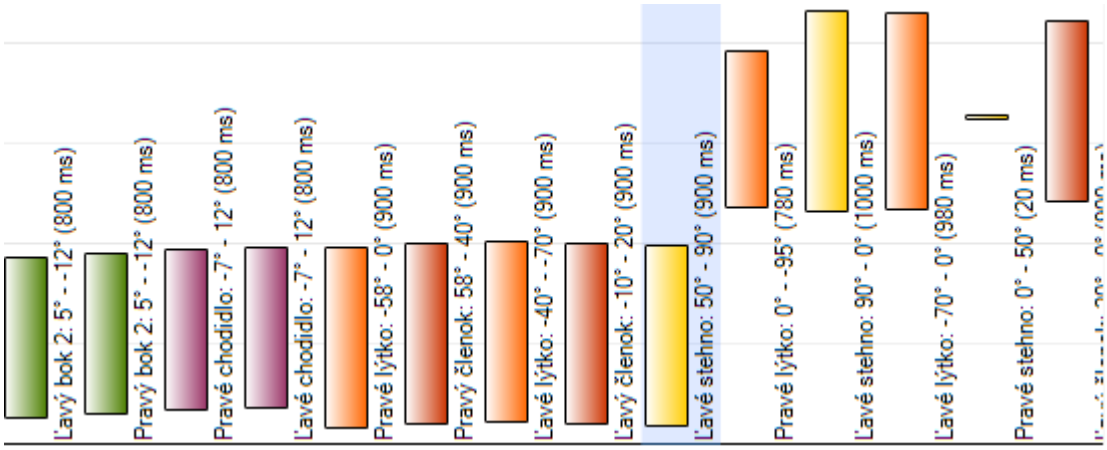
Technika pohybu vychádza z chôdze a je upravená tak, aby bola lopta zasiahnutá s čo najväčšou hybnosťou. Vytvorený pohyb je najefektívnejší v prípade, ak sa hráč nachádza od lopty, ktorá ma súradnice 0,0 na súradniciach 0.18, 0.05. Pohyb je možné vykonať v móde kickOff Left, pretože hráč pri kope do lopty prechádza na súperovu stranu a v móde BeforeKickOff by išlo o porušenie pravidiel na čo server reaguje tak, že umiestni hráča na náhodnú pozíciu. Dôležité je pri kope udržať rovnováhu. Hlavným cieľom je aby hráč vykonal pohyb následne kopol do lopty a zostal stáť.

2.2.7.2*Návrh*

Pri pohybe bolo potrebné zapojiť množstvo kĺbov aby agent ostal v rovnováhe a nedošlo k jeho pádu. Vytvorený pohyb bude spúšťaný na PC s nasledovnou konfiguráciu.
Procesor: Genuine Intel(R) T2250 @1.73GHz
RAM: 2,5GB
Typ systému: 32-bitový operačný systém W7

2.2.7.3*Implementácia*

Pohyb bol vytvorený v editore pohybov. Na obrázku 25 je zobrazená druha fáza pri ktorej dochádza k samotnému kopu. Na druhej strane oproti rovnovážnemu kopu bol vyvinutý kop pri ktorom agent nezostal v rovnováhe. Úspešnosť tohto kopu bola nízka a rovnako razancia strely sa neprejavovala významnejším prírastkom.



Obr. 25.: Druhá fáza pri kope do lopty zo statickej pozície.

2.2.7.4 *Testovanie*

Je možné k pohybu vytvoriť automatické testy ktoré by vyzerali nasledovne. Na začiatku musí platiť podmienka že lopta sa nachádza na pozícií 0,0 a hráč je umiestnený na pozícií -0.18, 0.05 po vykonaní pohybu sa hráč nachádza na inej pozícií a ostane stáť. Lopta zmení polohu z pozície 0,0 na inú pozíciu. Overovanie vytvoreného pohybu sme vykonali len vizuálne. Výsledok je pre prehľadnosť uvádzaný v tabuľke 6.

situácia	Hráč	Lopta	Umiestnenie lopty
1	Ostal stáť	Bola zasiahnutá	Za stredovým kruhom
2	Ostal stáť	Bola zasiahnutá	Za stredovým kruhom
3	Ostal stáť	Bola zasiahnutá	Za stredovým kruhom
4	Spadol	Bola zasiahnutá	Za stredovým kruhom
5	Ostal stáť	Bola zasiahnutá	Za stredovým kruhom
6	Ostal stáť	Bola zasiahnutá	V stredovom kruhu
7	Ostal stáť	Bola zasiahnutá	Za stredovým kruhom
8	Ostal stáť	Bola zasiahnutá	Za stredovým kruhom
9	Ostal stáť	Bola zasiahnutá	Za stredovým kruhom
10	Ostal stáť	Bola zasiahnutá	Za stredovým kruhom

Tabuľka 6 – vykonaný test

2.2.7.5 *Zhodnotenie*

Navrhnutý pohyb funguje podľa očakávaní, čo je možné pozorovať aj výsledkom testu. Možné nedostatky sú cítiteľné pri rýchlosti vykonania pohybu. Rovnako by bolo možné zefektívniť využitie hybnej sily. Navrhnutý pohyb predstavuje pilotnú verziu od ktorej sa dá odvíjať pri nasledovnom vylepšovaní a ladení pohybu- kop do lopty.

2.2.8 Exhibičné pohyby – krok bokom

2.2.8.1 *Analýza*

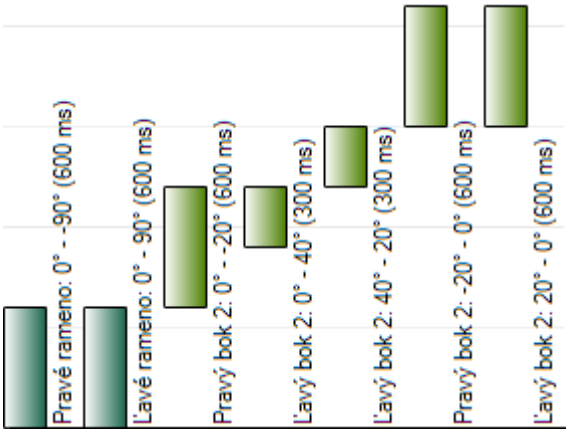
Pri chôdzi bokom bude potrebné využiť humanoidné pohyby akým je tzv. presun vbok. Agent zatiaľ žial nedokáže reagovať na svoju nestabilitu, z čoho vyplýva, že bude potrebné vytvoriť pozíciu agenta, v ktorej je agent pri danej chôdzi najstabilnejší. Pri realizácií pohybu využijeme editor pohybov od tímu Agenty 007.

2.2.8.2 *Návrh*

Pri realizácií pohybu budeme využívať predovšetkým nohy, a to konkrétne ľavý a pravý bok. Po pridaní agenta musí zaujať stabilnú polohu vzhľadom na krok bokom.

2.2.8.3 *Implementácia*

Pre realizáciu pohybu sme použili editor pohybov tímu Agenty007. Na začiatku agent upaží ruky vbok, čím si zabezpečí stabilnú polohu pre krok vbok. Následný pohyb vbok bol vytvorený pomocou dvoch základných pohybov. Prvý je pomalý krok do jednej strany. V strede tohto pohybu agent štvornásobnou rýchlosťou vykročí do druhej strany, čím sa pohne vbok.



Obr. 26.: Postupnosť krokov pre kráčanie vbok.

2.2.8.4 *Testovanie*

Test sa uskutočnil na počítači s nasledovnými technickými parametrami:

Procesor: 2,24GHz, RAM: 4GB, OS: Win 7 (64bit)

V tabuľke 7 sa nachádzajú testy kráčania vbok.

Počítač	**Očakávaný výsledok**	
PC1	Agent kráča vbok a nespadne	
Poradové číslo spustenia	**Výsledok testu**	**Skutočný výsledok**

1.	OK	Agent úspešne zvládol celý pohyb a nespadol
2.	OK	Agent úspešne zvládol celý pohyb a nespadol
3.	OK	Agent úspešne zvládol celý pohyb a nespadol
4.	OK	Agent úspešne zvládol celý pohyb a nespadol
5.	OK	Agent úspešne zvládol celý pohyb a nespadol

Tabuľka 7 – test kráčania vbok

2.2.8.5*Zhodnotenie*

Z výsledkov testu je viditeľné že agent zvládla pohyb bez problémov. Pohyb do strany je pomalý, no napriek nefunkčnej stabilizácií agenta je agent stabilný. Pohyb by bolo ešte možné vylepšiť, resp. zrýchliť znížením časov jednotlivých pohybov.

2.2.9 Exhibičné pohyby – kývanie rukami

2.2.9.1*Analýza*

Tento pohyb by mal slúžiť do prípadnej exhibície. Z tohto dôvodu nie sú naň kladené niektoré špecifické požiadavky. Hlavným cieľom je vytvorenie nejakého zaujímavého pohybu, ktorý sa nepoužíva priamo v hre proti iným tímom. Robot po niekoľkých úvodných pohyboch zakýva do oboch strán.

2.2.9.2*Návrh*

Pri pohybe rukami nie je nutné používať všetky dostupné kĺby. Postačujú na to iba kĺby ktoré ovládajú horné končatiny robota, krk a hlavu. Pohyb, ktorý vznikne bude spustený a otestovaný na PC tejto konfigurácie:

Procesor: Intel (R) Core (TM) 2 Duo T5870 @2.0GHz
RAM: 2.0GB
Typ systému: 32-bitový operačný systém Windows XP

2.2.9.3*Implementácia*

Tento pohyb bol vytvorený v editore pohybov. Keďže pri pohybe rukami neboli využívané dolné končatiny, bola stabilita robota bezproblémová a robot nepadal.

2.2.9.4*Testovanie*

Testovanie prebiehalo počas spustenia robota v simulačnom prostredí. Pričom sledované bolo či robot padá na zem alebo udrží stabilitu. Test bol uskutočnený 10x a vždy bolo testovanie úspešné – robot nepadal. Tento stav bol zapríčinený aj z toho dôvodu, že pohyb rukami nemá takmer žiadny vplyv na udržanie stability robota.

2.2.10 Exhibičné pohyby – kliky

2.2.10.1 *Analýza*

Je potrebné vytvoriť pohyb, pri ktorom robot urobí kliky. Nebude nijakým spôsobom kontrolovaná jeho nestabilita. Po vykonaní klikov je potrebné aby sa robot postavil. Pri realizácií pohybu využijeme editor pohybov od tímu Agenty 007.

2.2.10.2 *Návrh*

Nato aby agent vykonal kliky, je potrebné aby bol rukami opretý o zem. Je teda nutné by spadol s vystretými rukami, o ktoré sa bude na zemi opierať. Následne pohybmi rúk bude vykonávať kliky. Toto zopakuje 3x. Potom sa musí postaviť do vzpriamenej polohy a vyrovnať všetky kĺby do začiatočného stavu.

2.2.10.3 *Implementácia*

Celkový pohyb sa skladá z troch stavov.

4. Pád (používa členky, ktoré po páde vyrovná do začiatočnej polohy)
5. Drepy (používa ramená a predlaktia)
6. Vstávanie (používa stehná, boky, lýtka a členky)

2.2.10.4 *Testovanie*

Test sa uskutočnil na dvoch počítačoch s rôznymi technickými parametrami. **PC1:** Procesor: 1,66GHz Core 2 Duo, RAM: 3GB, OS: Win XP (32bit).

Počítač	**Očakávaný výsledok**	
PC1	Agent spadne na ruky, urobí 3 kliky a potom sa postaví.	

Poradové číslo spustenia	**Výsledok testu**	**Skutočný výsledok**
1.	OK	Agent sa úspešne vykonal kliky a postavil sa
2.	OK	Agent sa úspešne vykonal kliky a postavil sa
3.	OK	Agent sa úspešne vykonal kliky a postavil sa
4.	OK	Agent sa úspešne vykonal kliky a postavil sa
5.	OK	Agent sa úspešne vykonal kliky a postavil sa

Tabuľka 8 – test vykonaný na PC

2.2.10.5 *Zhodnotenie*

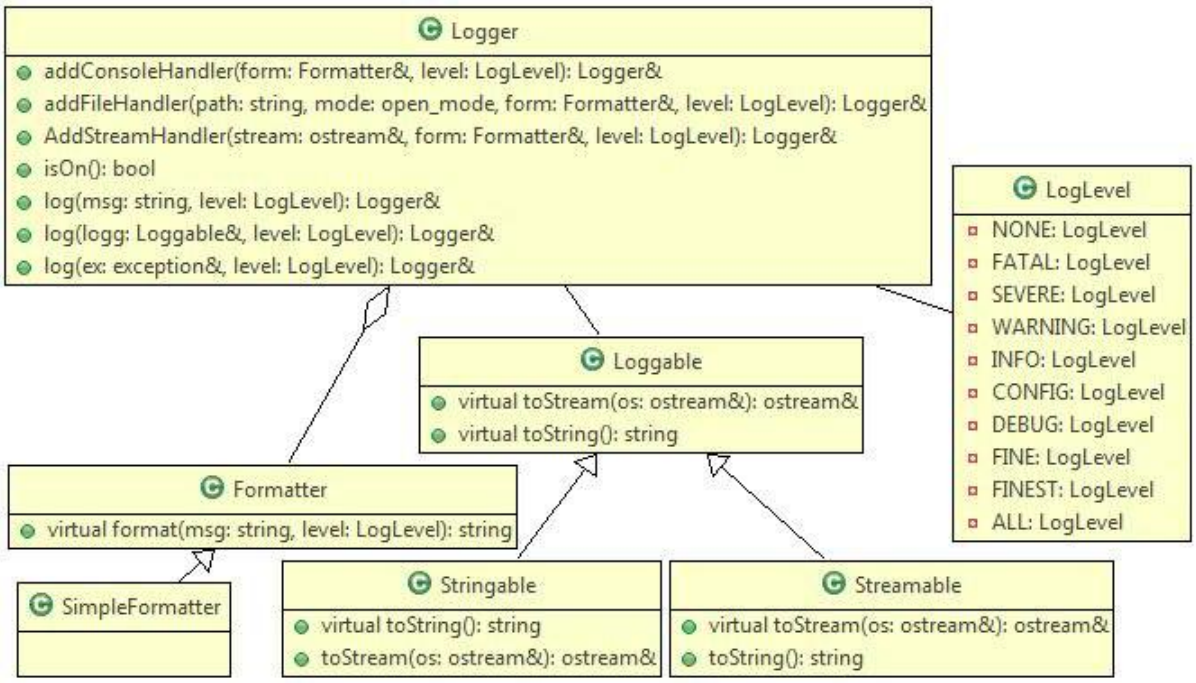
Podľa výsledkov testov je jasné, že požadovaný pohyb sa podarilo dosiahnuť. Robot je dostatočne stabilný napriek absencií kontroly nestability. Pohyby pri vstávaní by však mohli byť viac koordinované a rýchlejšie aby sa dostalo absolútne optimálneho výsledku.

2.2.11 Úprava prevzatého projektu do flexibilnejšej podoby

Po analýze prevzatého zdrojového kódu sme dospeli k záveru, že obsahuje mnohé návrhové a implementačné nedostatky a chyby. Na mnohých miestach autori namiesto štandardných knižníc používali vlastné naprogramované v C a zvyčajne nie veľmi kvalitné. Cieľom tejto úlohy bolo reštrukturalizovať niektoré časti projektu do flexibilnejšej podoby a hlavne odstrániť chyby a neefektívnosti s využitím možností, ktoré C++ ponúka.

Logging API

C++ nemá stromovú objektovú hierarchiu, t.j. nemôže poskytovať funkciu toString pre každý objekt ako napr. v Jave, preto vznikla trieda Loggable poskytujúca metódy toString a toStream. Kvôli prívetivému používaniu vznikli jej dve podtriedy Stringable a Streamable, každá z nich poskytuje jednu metódu s využitím tej druhej.



Obr. 27.: Diagram tried pre logovacie API vytvorený v nástroji AmaterasUML

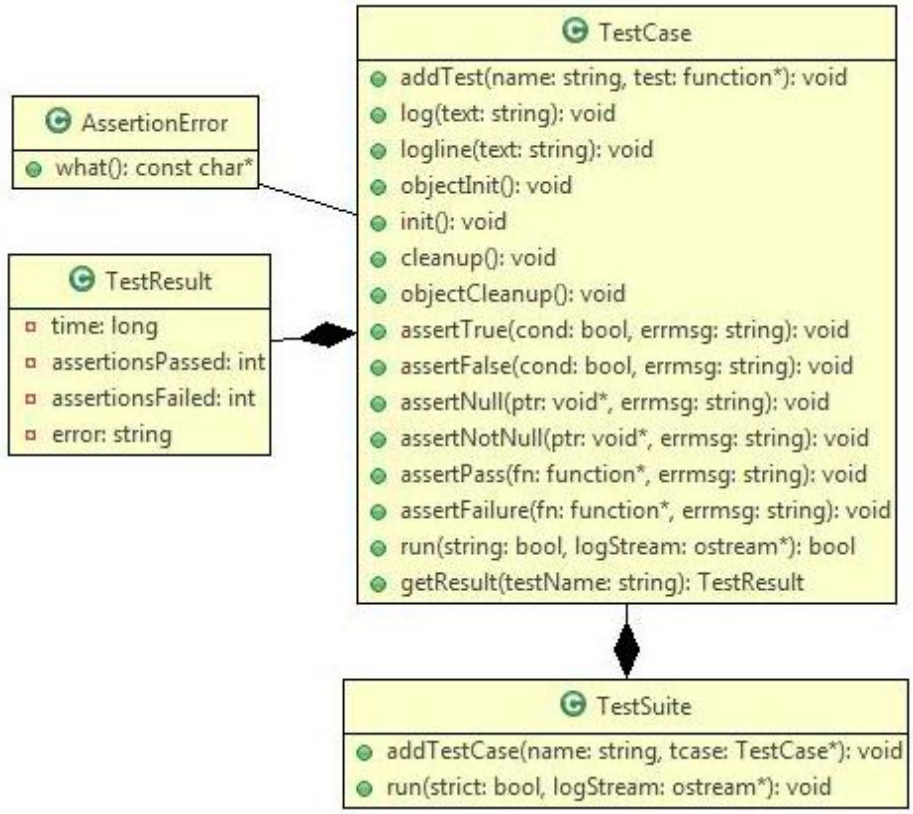
Logger je postavený podobne ako java.util.logging.Logger. Dajú sa uňho zaregistrovať Handlers pre rôzne prúdy bajtov - streams (konzola, súbory, stringstream). Každý Handler má priradený objekt typu Formatter na formátovanie správ a LogLevel na filtráciu správ podľa dôležitosti. Pôvodný Formatter (SimpleFormatter) iba vypíše správu a odriadkuje, pokiaľ je LogLevel správy nižší ako INFO, vypíše aj ten. Preťažené funkcie Logger::log sú prístupné aj cez operátor << s implicitným použitím úrovne SEVERE pre výnimky a INFO pre všetko ostatné. Pokiaľ Logger zapisuje do súboru, ten je v jeho deštruktore zatvorený. Aby objekty nejakej triedy boli logovateľné, stačí dediť danú triedu od triedy Stringable alebo Streamable a implementovať potrebnú funkciu.

Príklad použitia:

```
class Person: public Streamable {
	...
public:
	ostream& toStream(ostream& os) const {
		return os << name << „[“ << birthYear << „]“;
	}
	...
};
...
void testLogger() {
	Logger logger;
	Logger.addConsoleHandler();
	Logger.addFileHandler(“log.txt”, ios::app); // log.txt opened
	logger << “prvy log” << Person(“Jozko Mrkvicka”, 1985);
	logger << runtime_error(“dummy error”);
} // log.txt closed in destructor
```

Testovacie API

Pôvodné testovacie API bolo prepísané štýlom TestCase/TestSuite. TestCases sa nachádzajú v adresári Tests, deklarácie Test API v súbore Utils/test.h, používa sa namespace test.



Obr. 18.: Diagram tried pre testovacie API vytvorený v nástroji AmaterasUML

C++ nemá anotácie ani reflektívne funkcie, testy (funkcie) preto treba registrovať priamo v kóde volaním funkcie TestCase::addTest, ktorá berie smerník na členskú funkciu. Pre jednoduchšiu registráciu bolo vytvorené makro ADD_TEST(trieda, funkcia), napr.

```
ADD_TEST(MathTest, testCoordConversions);
```

sa rozvinie do

```
addTest(„testCoordConversions“, fcast(&MathTest::testCoordConversions));
```

Podobne bolo vytvorené makro ADD_TEST_CASE pre TestSuite, napr.

```
ADD_TEST_CASE(MathTest);
```

sa rozvinie do

```
addTestCase(„MathTest“, new MathTest);
```

Pre každú aserčnú funkciu je k dispozícii makro, ktoré automaticky vytvára chybovú správu, napr.

```
ASSERT_TRUE(cart == spher);
```

sa rozvinie do

```
assertTrue((cart == spher), string(„cart == spher“) + " is not true, " +
utils::source(__FILE__, __LINE__));
```

Funkcia TestCase::run berie ako argumenty príznak strict a odkaz na prúd bajtov (stream), do ktorého má vypisovať výsledky. Pôvodné hodnoty sú true a std::cout. Ak strict je true, pri zlyhaní asercie bude vyhodená výnimka (AssertionError) a daná testovacia funkcia (nie celý

TestCase) nebude ukončená, v tomto prípade bude TestResult pre funkciu obsahovať najviac jednu zlyhanú aserciu. V opačnom prípade, keď strict je false, test pokračuje ďalej aj pri nesplnenej asercii, t.j. nebude vyhodená výnimka. Funkcie TestCase::log a TestCase::logline umožňujú zápis do logovacieho prúdu pre daný test.

Príklad použitia:

```
class MathTest: public test::TestCase {
public:
      MathTest() {
            ADD_TEST(MathTest, testCoordConversions);
      }
      void testCoordConversions();
};

...

void MathTest::testCoordConversions() {
      Cartesian cart(1,2,3);
      Spherical spher(cart);
      ASSERT_TRUE(cart == spher);
      ASSERT_TRUE(spher == cart);
      ASSERT_TRUE(cart.length() == spher.length());
      ASSERT_TRUE(~cart == ~spher);
      ASSERT_TRUE(-cart == -spher);
      ASSERT_TRUE((cart + spher) == (cart * 2));
      ASSERT_TRUE((cart - spher) == Cartesian());
      ASSERT_TRUE((cart - spher) == Spherical());
      ASSERT_TRUE((cart * 2).equal(spher * 2, 0.00001f));
      ASSERT_TRUE((spher * 2).equal(cart * 2, 0.00001f));
      cart = spher;
      spher = cart;
      ASSERT_TRUE(cart == spher);
      ASSERT_TRUE(spher == cart);
}

...

class MainTestSuite: public test::TestSuite {
public:
      Tests(): test::TestSuite("RoboCup3D Main Test Suite") {
            ADD_TEST_CASE(MathTest);
      }
};

...

MainTestSuite().run();
```

Math API

Najvýraznejšou zmenou je použitie šablón a možná parametrizácia typu čísel, napr: float/double/long double a.i. Vektory aj body v 3D priestore sú reprezentované súradnicami

karteziánskymi (trieda Cartesian) alebo sférickými (trieda Spherical). Medzi týmito súradnicovými systémami sú umožnené implicitné konverzie (viď vyššie uvedený MathTest). Tieto triedy poskytujú základné aritmetické vektorové operácie, rovnako aj pokročilejšie, napr. rotácie. Typovo parametrizovaná bola aj trieda Quaternion.

Ďalšie úpravy

Medzi ďalšie úpravy projektu patrí aj oprava chýb, odstránenie zbytočných dynamických alokácií s preferovaním statickej alokácie, používanie referencií, konštantných premenných a funkcií, odstránenie using namespace klauzúl a niektorých zbytočných inkludovaných súborov z hlavičkových súborov, pre efektivitu odstránenie niektorých zbytočných dočasných objektov a inlining mnohých malých funkcií. Tiež došlo k zmenám v použitých knižniciach, kde namiesto vlastných implementácií dátových štruktúr sa využívajú triedy STL, na mnohých miestach bol upravený tok príkazov a kód sprehľadnený definíciou premenných až na mieste použitia.

Overenie

Vytvorené API boli otestované v projekte buď priamo nasadením (logovanie) alebo cez TestCases (testy, matematické triedy a funkcie). Využitie inliningu a odstránenie niektorých dočasných objektov vo výrazoch malo za následok, že mnohé matematické operácie bežali v nárazovom teste o 30 až 40 % rýchlejšie.

2.3 Šprint č. 3

ZAČIATOK ŠPRINTU:	4.11. 2009
KONIEC ŠPRINTU:	18.11. 2009
INFORMÁCIE:	TRETÍ ŠPRINT BOL ZAMERANÝ NA ANALYZOVANIE NÁVRHU A ANALÝZY STREDNEJ LOGIKY NAVRHNUTEJ PREDCHÁDZAJÚCIM TÍMOM. NA ZÁKLADE TÝCHTO ANALÝZ BY SME BOLI SCHOPNÍ POVEDAŤ, AKOU CESTOU SA VYDÁME MY, PRI IMPLEMENTOVANÍ STREDNEJ LOGIKY. EŠTE PRED SAMOTNÝM PRIDANÍM STREDNEJ LOGIKY DO PROJEKTU BOLO NUTNÉ VYTVORIŤ NIEKTORÉ VYHODNOCOVACIE FUNKCIE. TIE NEBOLI DOKONČENÉ V TOMTO ŠPRINTE.
POČET PRÍBEHOV:	2

Analýza

2.3.1Analýza strednej logiky minuloročného tímu v návrhu

Cieľom vyššej logiky hráča je vytvorenie správania tak, aby bol schopný skladaním elementárnych pohybov hrať futbal na nižšej úrovni. Na vyššej úrovni aby bol schopný komunikovať aj s inými hráčmi a tým vytvárať tímovú spoluprácu, ktorá sa vyžaduje v tímovej hre. Nevyhnutná podmienka pre vykonávanie logiky hráča je znalosť elementárnych pohybov, ktoré budú kombinované za účelom vytvorenia požadovaného cieľa. Dosiahnutie cieľa navrhujú riadiť pravidlami. Načítavanie strednej logiky by sa malo uskutočniť zo súboru, rovnako ako samostatné pohyby. Logika nebude vyhodnocovaná pri každom cykle (trvajúcom 20 ms), ale bude volaná v stavoch, kedy je možné zmeniť rozhodnutie. Logická akcia bude sekvenčným výberom pravidiel a im prislúchajúcich pohybov. Podmienky v pravidlách vytvorili ako logické výrazy zostrojené z nasledujúcich predikátov[1]:

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.

Danú množinu predikátov by bolo vhodné rozšíriť o ďalšiu množinu a to:

IsBallMine – zistí, či hráč má loptu prípadne kto má k lopte bližšie.

WhereAmI – zistí kde sa aktuálne nachádza hráč.

Pričom musia platiť nasledujúce vlastnosti:

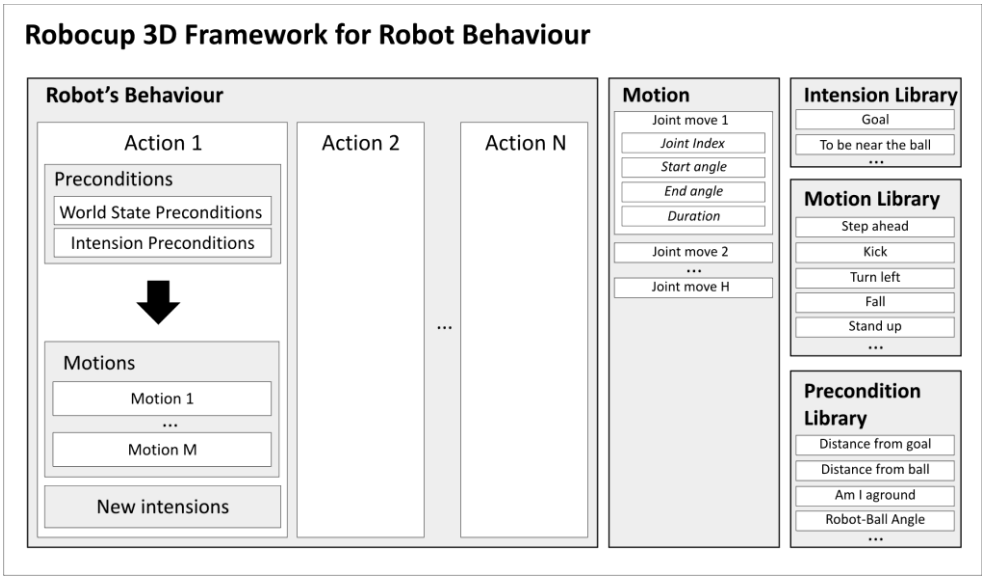
Môže platiť iba jeden predikát z IsBallInDirectView, IsBallOnTheLeft a IsBallOnTheRight.

Môže platiť iba jeden predikát z IsGoalpostInDirectView, IsGoalpostOnTheLeft a IsGoalpostOnTheRight.

Predikáty IsGoalpostInDirectView, IsGoalpostOnTheLeft a IsGoalpostOnTheRight uvažujú stred bránky.

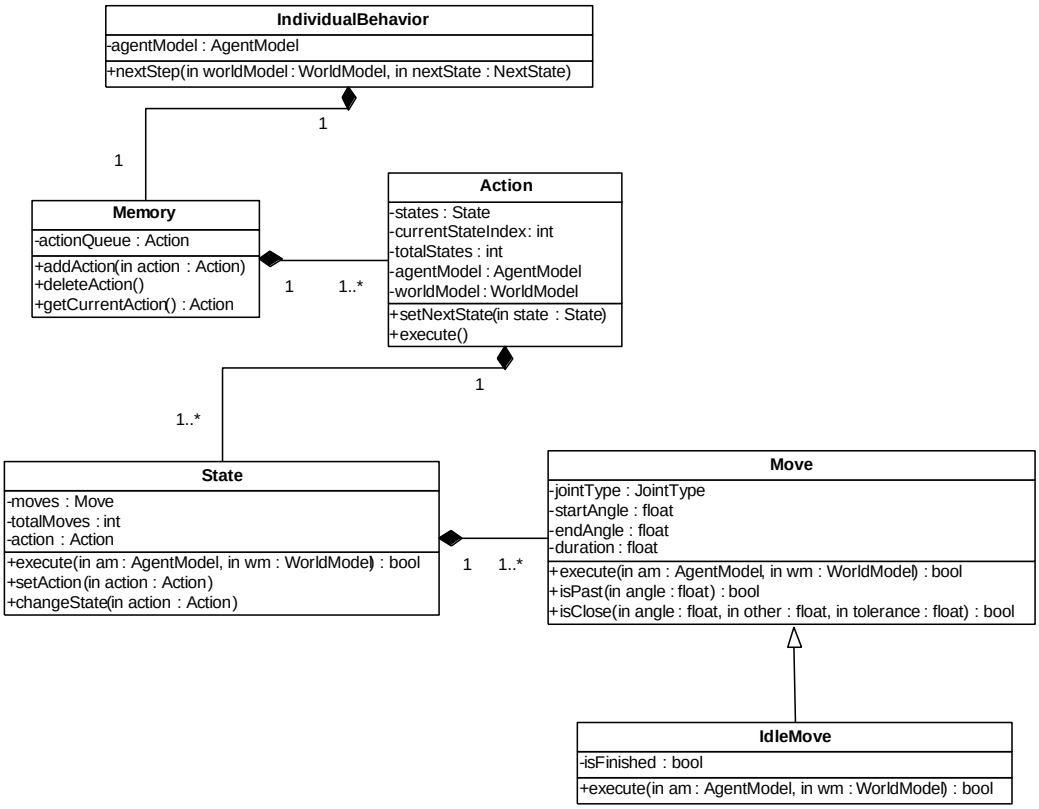
Predikáty IsGoalpostInDirectView, IsGoalpostOnTheLeft a IsGoalpostOnTheRight uvažujú súperovu bránku.

Predikáty IsBallInDirectView, IsBallOnTheLeft, IsBallOnTheRight, IsGoalpostInDirectView, IsGoalpostOnTheLeft a IsGoalpostOnTheRight neuvažujú Z-ovú súradnicu predstavujúcu výšku, pracujú v dvojrozmernej sústave pri pohľade zhora na ihrisko.



Obr. 19.: Návrh frameworku pre správanie robota podľa [1].

Podľa návrhu na Obr. 19 je vidno že podľa aktuálneho stavu sveta sa má hráč rozhodovať na základe podmienok. Neodmysliteľnou časťou pre vykonávanie logických pohybov je zvládnutie základných pohybov a činností, ktoré skladá do výslednej akcie. Výsledný pohyb by sa mal nachádzať v časti Motion. O ďalšej činnosti sa rozhoduje na základe požadovaného cieľa, čo je naznačené v časti Intension Library. Základne, elementárne pohyby sú navrhnuté v časti Motion Library. Pre vykonanie požadovanej akcie overuje platnosť vstupných podmienok. Ak vyhodnotí daní cieľ ako nedosiahnuteľný, dynamicky tento cieľ zmení. Vstupné podmienky a vyhodnocovacie funkcie v návrhu sú logický umiestnené v časti Precondition Library. Tento návrh pôsobí modulárny a ľahko rozšíriteľný. Samotné správanie rozdelili na niekoľko štádií. Pričom každá akcia by mala mať vstupné podmienky. Správanie závisí od aktuálneho stavu sveta v ktorom sa robot nachádza. Na základe neho vyberá akcie. Pre vykonanie akcií je potrebné vykonať pohyby a overovať ich stav. Na Obr. 20 je zachytený diagram tried znázorňujúci vlastné správanie.



Obr. 20.: Diagram tried znázorňujúci vlastné správanie.

Pre vylepšenie stability navrhli rovnovážny modul, ktorý má za úlohu udržať agenta v rovnovážnej polohe a zistiť aktuálne ťažisko. Z možných alternatív pre dosiahnutie rovnováhy analyzovali postupy výpočtu pomocou:

> Rovnovážnej rovnice n-tého rádu. Rovnicu je možné zostaviť z hmotnosti jednotlivých časti tela a ich umiestnenia vzhľadom na torzo tela. vypočítať aké sily pôsobia na ťažisko.
>
> Pomocou diferenciálnej rovnice. Vzhľadom na náročnosť použitia diferenciálnych rovníc sa týmto postupom bližšie nezaoberali.
>
> Podľa vzorcov fyziky. Pomocou platných zákonov fyziky, ktorá simuluje reálny svet (rýchlosť, zrýchlenie, hybnosť alebo silu) vypočítať celkovú silu na neho pôsobiacu, alebo celkovú hybnosť, ktorá by nemala prekročiť určitú hranicu.

Novú polohu bodu počítali pomocou goniometrických funkcií vychádzajúci zo začiatočnej pozície bodu. Túto metódu navrhujú použiť na výpočet ťažiska, keďže je známa poloha a hmotnosť jednotlivých častí. Sledovaním ťažiska a jeho pohybu v čase je možné zistiť či hráč stráca rovnováhu, v prípade ak začne zrýchľovať nad kritickú hodnotu, alebo je v rovnováhe. Po

prepočte hráč uváži, či sa dá pádu zabrániť vyvinutím sily pôsobiacej proti pádu ktorú následne aplikuje, alebo pád je neodvratný. V takomto prípade informuje svojich spoluhráčov o tomto stave čo by malo byť súčasťou vyššej logiky.

2.3.2 Analýza implementácie strednej logiky

Keďže sa minuloročný tím zameral najmä na implementáciu editora pohybov, obsahuje kód agenta hlavne časti, ktoré sú potrebné na správne vykonávanie základných pohybov zo súboru. Implementovaný je jednoduchý systém na načítanie a spracovanie pohybov zo súboru. Agent si zaznamenáva informácie z prostredia, ktoré ale nevyužíva na žiadne rozhodovanie ani udržiavanie rovnováhy. Agent tiež má implementované vypočítavanie niektorých užitočných hodnôt o svojom tele a okolitom svete, na základe ktorých sa dajú vytvoriť rozhodovacie funkcie.

Analýza jednodlivých tried z pohľadu implementácie strednej logiky

Agent – obsahuje základné cykly, ktoré sa dookola opakujú až kým sa agent nevypne. Neobsahuje žiadnu konkrétnu implementáciu logiky ani správania. Využíva iné triedy na zabezpečenie funkcionality pohybu robota.

InitialConfiguration – obsahuje užitočné konštanty na prácu s robotom, ktoré sú použiteľné na rozhodovanie agenta, avšak sa na tento účel nepoužívajú.

Model – celkový model hráča – svet, interakcia s ostatnými hráčmi a samotný hráč, používaná len na zabalenie týchto troch komponentov do jedného.

World – obsahuje údaje o ostatných objektoch na ihrisku – bránky, rohové zástavky, iní hráči a lopta. Implementované dopočítavanie niektorých hodnôt na základe prijatých hodnôt zo servera. Tieto hodnoty sú použiteľné na rozhodovanie agenta, toto však zatiaľ nie je implementované.

Body – podobné informácie ako v triede World, ale o tele hráča. Tiež funguje dopočítavanie, ktoré však nie je využité.

Interaction – interakcia hráča s prostredím, skôr vhodné na vyššiu logiku na základe komunikácie hráčov. Implementované len základné funkcie, zatiaľ nie sú využité nikde.

Behaviour – základná trieda reprezentujúca správanie hráča. Obsahuje zoznam možných akcií (krokov), ktoré sa skladajú zo stavov. Zatiaľ je však správanie pevne dané ako postupnosti krokov, kde je v každom stave priradený jeden krok (nextStep). V tejto triede by malo byť

implementované rozhodovanie, ktorá akcia sa má vykonať. Vstupmi rozhodovania by boli údaje vypočítané a uložené v triede Model.

Action – reprezentácia jednej akcie, ktorá sa skladá zo stavov. Akcie sa načítavajú zo súborov. Nie sú implementované podmienky v rámci načítavania zo súborov ani v kóde agenta. V tejto triede by sa mali doplniť rozhodovanie v rámci jednej akcie, v závislosti od stavu robota a okolia. Vstupmi rozhodovania by boli údaje vypočítané a uložené v triede Model.

ActionRepository – obsahuje úložisko akcií načítaných pri spustení. Dá sa v ňom nastaviť aktuálne vykonávaná akcia, ktorá by sa mala meniť v závislosti od vyšších rozhodnutí. V súčastnosti sa aktuálna akcia nastavuje po poradí bez akéhokoľvek rozhodovania.

State – reprezentácia stavu v rámci akcie. Obsahuje základnú synchronizáciu pohybov kĺbmi na základe podmienok o stave niektorého z kĺbov.

JointCondition – základná podmienka na pozíciu kĺbu v nejakom čase. Môže poslúžiť ako ukážka rozhrania podmienky na určitý stav modelu.

Move – Elementárny pohyb kĺbom, obsahuje JointCondition na základe ktorej sa začína vykonávať.

MoveQueue

Primitives – dátové štruktúry a algoritmy, súvisiace s fyzikou a matematikou simulácie. Už použité na dopočítavanie stavov a v budúcnosti použiteľné vytváranie elementárnych podmienok z ktorých sa spravia zložitejšie podmienky.

Zhodnotenie

Minuloročný tím navrhol a vytvoril základné pohyby, bez ktorých nie je možné implementovať nič vyššie. Navrhol tiež podmienečné vykonávanie pohybov, podmienky by sa mali editovať v editore na pohyby a vykonávať v agentovi. Túto časť však navrhli len veľmi stručne a nestihli ju implementovať. Aktuálne teda agent neobsahuje žiadnu implementovanú vyššiu alebo strednú logiku. Základná architektúra agenta umožňuje takúto funkcionalitu pridať bez väčších zásahov do nej. Sú tiež hotové výpočty niektorých hodnôt potrebných na rozhodovanie.

2.4 Šprint č. 4

ZAČIATOK ŠPRINTU:	18.11. 2009
KONIEC ŠPRINTU:	2.12. 2009
INFORMÁCIE:	V ŠTVRTOM ŠPRINTE BOLO POTREBNÉ DOKONČIŤ VYHODNOCOVACIE FUNKCIE, KTORÉ SA NEDOKONČILI V 3. ŠPRINTE. HLAVNÝM CIEĽOM BOLO PRIDANIE PODPORY STREDNEJ LOGIKY DO EDITORU POHYBOV. ĎALEJ BOLO POTREBNÉ VYTVORIŤ NOVÉ POHYBY V EDITORE, KTORÉ BY SA DALI POUŽIŤ NESKÔR PRI SKLADANÍ POHYBOV.
POČET PRÍBEHOV:	4

Analýza

2.4.1Vizualizácia strednej logiky pomocou editoru pohybov

Editor pohybov bol vytvorený pre vytváranie a editáciu základných pohybov agenta. Pre zobrazenie vytvoreného pohybu slúži v editore 3D model robota, ktorý dokáže zadefinované pohyby odsimulovať.

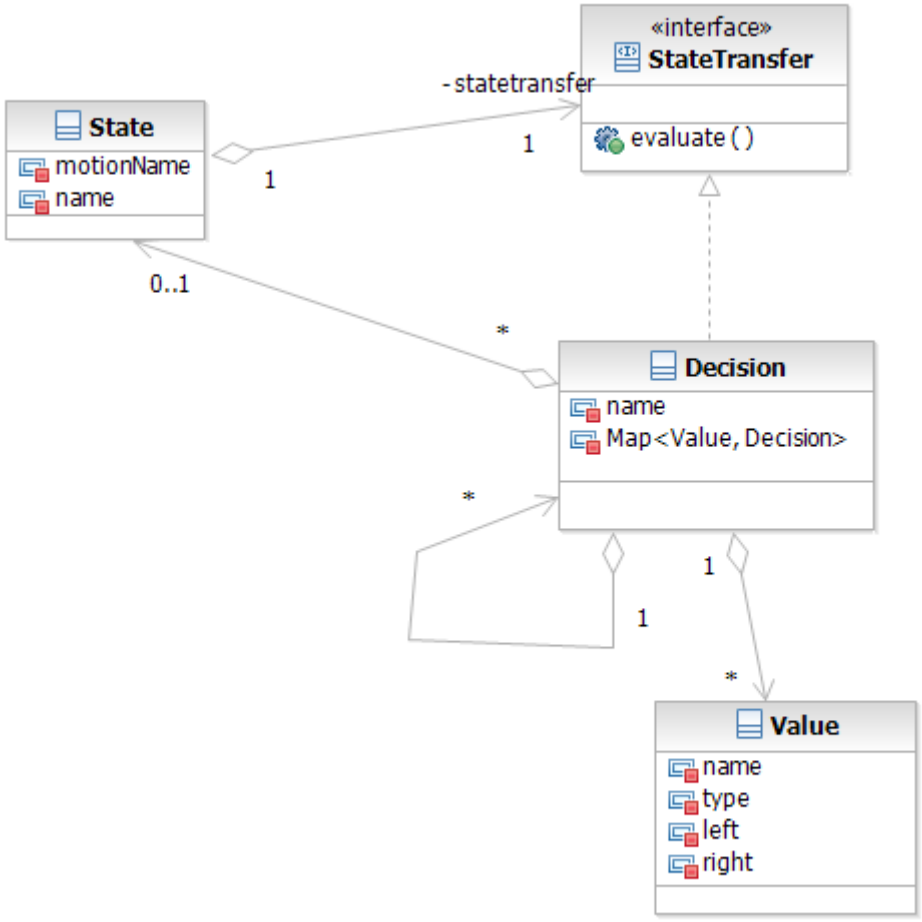
Pri vizualizácií strednej logiky využijeme 3D model robota pri vizualizácií spojení základných pohybov strednej vrstvy.

Návrh

2.4.2Načítavanie a ukladanie strednej logiky hráča

Reprezentácia strednej logiky

Reprezentáciu strednej logiky sme navrhli tak, ako to ukazuje diagram na obrázku 21.



Obr. 21.: Diagram strednej logiky.

2.4.3Vizualizácia strednej logiky pomocou editoru pohybov

Pri implementácií je potrebné využiť už existujúci 3D model agenta, do ktorého je potrebné načítať pohyby skladajúce sa z elementárnych otočení kĺbov.

Model agenta si vyberá nasledujúce pohyby na základe hodnoty podmienky, ktorú mu zadá používateľ manuálne.

Podmienky aj s pohybmi sú získavané z rozhodovacieho stromu strednej logiky.

Implementácia

2.4.4 Výpočet orientácie kamery, pozícií, rýchlostí, zrýchlení objektov a predikcia

Výpočet orientácie kamery z okolitých objektov sme prevzali od tímu LittleGreenBats. Orientáciu kamery uchovávame v rotačnej matici R, transláciu kamery od stredu ihriska vo vektore $\boldsymbol{v}_t$. R a $\boldsymbol{v}_t$ sú inicializované podľa počiatočnej pozície a orientácie (efektor Beam). Ak vidíme aspoň 3 statické objekty (poznáme absolútne pozície), vypočítame z nich osi $\boldsymbol{x}, \boldsymbol{y}$ tak, ako ich vidíme, potom tieto vektory normalizujeme. Os $\boldsymbol{z}$ vypočítame ako ich vektorový súčin. Potom $R = \begin{pmatrix} \boldsymbol{x}_x & \boldsymbol{y}_x & \boldsymbol{z}_x \\ \boldsymbol{x}_y & \boldsymbol{y}_y & \boldsymbol{z}_y \\ \boldsymbol{x}_z & \boldsymbol{y}_z & \boldsymbol{z}_z \end{pmatrix}$. Ak nevidíme dostatočne veľa objektov, použijeme perceptor GyroRate, ktorý udáva zmenu orientácie kamery od minulého kroku v Eulerových uhloch. Vytvoríme rotačné matice pre jednotlivé zložky R_{GRx}, R_{GRy}, R_{GRz}. Potom $R(t+1) = R(t)\left(R_{GRx}R_{GRy}R_{GRz}\right)$.

Keď poznáme orientáciu kamery, na výpočet $\boldsymbol{v}_t$ stačí, aby sme videli aspoň jeden statický objekt, potom od jeho absolútnej pozície odčítame videnú pozíciu (vektor) otočený do absolútnej sústavy ihriska. Ak vidíme viac statických objektov, výsledky spriemerujeme. Keď poznáme R a $\boldsymbol{v}_t$, vieme vypočítať absolútnu pozíciu ľubovoľného objektu pripočítaním otočeného vektora videnej pozície k translácii kamery od stredu ihriska. Rýchlosť je derivácia pozície podľa času a zrýchlenie derivácia rýchlosti podľa času, tieto vieme vypočítať pri znalosti pozícií v za sebou idúcich krokoch.

Predikcia je založená na predpoklade, že zrýchlenie sa v nasledujúcom kroku nezmení. Vypočítame teda nový vektor rýchlosti a novú pozíciu.

Výpočty boli testované pri rôznych počiatočných pozíciách a orientáciách (efektor Beam) a pri kráčaní a kopaní do lopty.

2.4.5Načítavanie a ukladanie strednej logiky hráča

Stredná logika hráča je reprezentovaná abstraktným stavovým automatom, ktorý pozostáva zo stavov a prechodov medzi stavmi. Každý stav má svoju implementáciu prechodovej funkcie, svoje meno a meno pohybu, ktorý sa má v tomto stave vykonávať. Rozhodovanie hráča pozostáva zo zistenia svojho stavu, vykonania rozhodnutia o tom do ktorého nového stavu sa má ísť, prechod do nového stavu a vykonanie akcie pre daný nový stav.

Prechodová funkcia medzi stavmi je realizovaná rozhodovacím stromom. Každý stav má teda vlastný rozhodovací strom, resp. jeho koreňový uzol. Zároveň je každý stav listom tohto stromu. Každý uzol stromu má tiež definovanú mapu hodnôt (intervalov) na ďalšie uzly a názov funkcie s ktorej výsledkom sa majú porovnávať intervaly. Pri vyhodnocovaní sa jednotlivé intervaly porovnajú s vrátenou hodnotou funkcie a vyhodnocovanie sa presunie do uzla priradeného k intervalu, do ktorého vrátená hodnota patrí. Vyhodnocovanie končí, keď sa dostane do listu stromu, teda do uzlu ktorý ma definovaný stav. Tento stav sa vráti ako výsledok rozhodovania.

Na simulovanie návratových hodnôt v rámci editora bola vytvorená samostatná trieda so statickou funkciou, ktorá na základe zadaného reťazca (názvu funkcie, ktorej hodnota sa má vrátiť) vráti určitú hodnotu z intervalu <0,1>. Implementácia tejto funkcie závisí od prostredia.

Formát súboru

Súbor je rozčlenený na jednotlivé časti, najskôr musí byť uvedená časť s jednotlivými symbolickými hodnotami VALUES, za ňou zoznam rozhodovacích uzlov DECISION, potom definícia štartovacieho stavu STARTINGSTATE a nakoniec zoznam stavov STATE.

Členenie súboru

VALUES

- Zoznam hodnôt, každá hodnota má svoje meno a interval, ktorý k nej prislúcha.

DECISION

- NAME - názov rozhodovacieho uzla
- FUNCTION – rozhodovacia funkcia
- Zoznam podmienok – jednotlivé intervaly, resp. hodnoty, ktoré majú priradené ďalšie rozhodovacie uzly, resp. stavy do ktorých sa má prejsť v prípade, že hodnota funkcie patrí do interval

STARTINGSTATENAME – názov stavu

- MOTIONNAME – názov pohybu, ktorý sa v tomto stave má vykonávať
- ROOTDECISION – koreňový uzol rozhodovacieho stromu v tomto stave

STATE

- Rovnaké ako STARTINGSTATE.

Ukážka súboru strednej logiky

```
[VALUES]
true=<1,1>
false=<0,0>
…
[DECISION]
NAME=Je spadnuty
FUNCTION=HRAC_STOJI
true=Vidi loptu
false=Vstan
[DECISION]
NAME=Vidi loptu
FUNCTION=HRAC_VIDI_LOPTU
false=Otoc sa za loptou
true=Rozhodni sa podla lopty
…
[STARTINGSTATE]
NAME=Otoc sa za loptou
MOTIONNAME=Otocenie_vlavo
ROOTDECISION=Je spadnuty
[STATE]
NAME=Otoc sa za loptou
MOTIONNAME=Otocenie_vlavo
ROOTDECISION=Je spadnuty
[STATE]
NAME=Beh s loptou
MOTIONNAME=Beh_s_loptou
ROOTDECISION=Je spadnuty
…
```

2.4.6Vizualizácia strednej logiky pomocou editoru pohybov

V rámci implementácie sme vytvorili novú metódu load(String filename), ktorá načíta súbor s pohybom do spájaného zoznamu dátových štruktúr JointMotions. Tento spájaný zoznam slúži

ako vstup pre funkciu start, ktorá následne spustí 3D model robota a vykoná pohyb uložený v spájanom zozname.

Jednotlivé pohyby sú načítavané a vykonávané na základe rozhodovacieho stromu DecisionTree. Po vykonaní každého pohybu má používateľ možnosť zadať hodnotu funkcie v intervale <0,1> pomocou dialógového okna, čím ovplyvní výber ďalšieho pohybu.

Testovanie

2.4.7Načítavanie a ukladanie strednej logiky hráča

Otváranie a ukladanie bolo uskutočnené tak, že boli v programe načítané jednotlivé ukážkové súbory do pamäti, a následne boli zapísané späť na disk. Súbory boli potom porovnané, či obsahujú rovnaké dáta. Takýto test bol vykonaný viackrát po sebe, všetky testy skončili úspešne.

2.4.8Vizualizácia strednej logiky pomocou editoru pohybov

Pri testovaní sme postupovali nasledovne. Vytvorili sme si súbor so strednou logikou, ktorú sme následne načítali v editore a sledovali vykonávanie modelu agenta na základe nami zadaných hodnôt funkcií. Výsledne správanie sme porovnávali s požadovaným. Výsledky testovanie sú znázornené v nasledovnej tabuľke:

Očakávaný výsledok		
Model robota vykoná pohyby na základe načítaného súboru so strednou logikou		
Poradové číslo spustenia	**Výsledok testu**	**Skutočný výsledok**
1.	OK	Model vykonal správne pohyby
2.	OK	Model vykonal správne pohyby
3.	OK	Model vykonal správne pohyby

4.	OK	Model vykonal správne pohyby
5.	OK	Model vykonal správne pohyby

Tabuľka 9. Testovanie načítania súborov so strednou logikou.

2.5 Šprint č. 5

ZAČIATOK ŠPRINTU:	2.12. 2009
KONIEC ŠPRINTU:	16.12. 2009
INFORMÁCIE:	V PIATOM ŠPRINTE SME DOKONČOVALI TASKY, KTORÉ SA NESTIHLI V PREDCHÁDZAJÚCOM ŠPRINTE. OKREM TOHO BOLO POTREBNÉ PRIPRAVIŤ VŠETKY DOKUMENTY TAK, ABY SME ICH MOHLI PREZENTOVAŤ. TAKTIEŽ BOLO POTREBNÉ AKTUALIZOVAŤ WEBOVÚ STRÁNKU.
POČET PRÍBEHOV:	2

Analýza

2.5.1Zložené pohyby pre potreby strednej logiky

Pre potreby strednej logiky je potrebné, aby sme vytvorili určité pohyby, ktoré by sa dali spustiť za sebou, resp. aby sme vytvorili zopár chýbajúcich pohybov. Dohodli sme sa na vytvorení nasledujúcich pohybov v editore:

- Chôdza – väčší a menší krok, vykročenie jednou nohou
- Otáčanie sa – viac pohybov, kde bude veľkosť otočenia odstupňovaná
- Postavenie sa, ak agent padne na chrbát
- Vrátenie nohy do základnej polohy, ak agent nekopne do lopty

Tieto pohyby by mali potrebám projektu zatiaľ stačiť. Pomocou nich by sme vedeli prezentovať niektoré zložené pohyby, ktoré patria do strednej logiky hráča. Dôraz je kladený hlavne na stabilitu hráča pri vykonávaní pohybov a nie na jeho rýchlosť. Do budúcna bude potrebné pridať aj iné pohyby, ale ich implementácia je časovo náročná, takže spomenuté pohyby vyhovujú základnému cieľu – odprezentovaniu strednej logiky v RoboCup 3D.

Návrh

2.5.2Zložené pohyby pre potreby strednej logiky

Pri chôdzi, otáčaní a vrátení nohy agenta do základnej polohy sme sa inšpirovali už hotovými pohybmi, ktoré sme vytvorili my, alebo ktoré už boli vytvorené tímom Agenty007. Následnou úpravou týchto pohybov vzniknú tie očakávané. Pri týchto pohyboch budú použité takmer všetky kĺby agenta. Pohyb, týkajúci sa postavenia agenta z chrbta je nutné vytvoriť úplne od začiatku, keďže sme sa ešte nestretli s takýmto pohybom. Agent sa najprv bude musieť otočiť do jednej strany, či už ľavej alebo pravej, následne sa pomocou kĺbov v dolných končatinách postaví. Agent môže byť v základnej polohe mierne pootočený po ukončení tohto pohybu. Ak bude otočený do neželanej strany, bolo by výhodné otočiť ho tak, aby skončil v tej istej polohe v ktorej spadol.

Implementácia

2.5.3Zložené pohyby pre potreby strednej logiky

Pre realizáciu pohybov sme použili editor pohybov tímu Agenty007. V tejto časti sa nachádza fyzická realizácia pohybu. Teda kĺby pri ktorých dochádza k natáčaniu, uhol o aký sa majú otočiť a dĺžka trvania pohybu. Všetky pohyby sme uložili vo formáte .rmo a sú dostupné v knižnici pohybov.

Testovanie

2.5.4Zložené pohyby pre potreby strednej logiky

Keďže nové pohyby pre potreby strednej logiky vytvárali viacerí členovia tímu, boli aj otestované na viacerých PC. Celkovo testovanie prebiehalo na 2 PC:

PC1:

Procesor: Genuine Intel(R) T2250 @1.73GHz
RAM: 2,5GB
Typ systému: 32-bitový operačný systém W7

PC2:

Procesor: Intel (R) Core (TM) 2 Duo T5870 @2.0GHz
RAM: 2.0GB
Typ systému: 32-bitový operačný systém Windows XP

Nie každý pohyb bol otestovaný na každom z uvedených PC. Jednotlivé pohyby boli testované na týchto PC:

PC1: vrátenie nohy do základnej polohy pri neodkopnutí lopty

PC2: chôdza

Testovanie prebiehalo spustením agenta v prostredí RoboCup 3D, a to celkovo 15x pre každý pohyb. Následne sa zapísal stav, či agent spadol alebo nie. Rýchlosť pohybov nebola testovaná pretože nebola prioritou.

Názov pohybu	# nespadnutí/# spustení	Úspešnosť [%]
Vykročenie ľavou nohou	14/15	93,3
Menší krok	13/15	86,6
Väčší krok	12/15	80
Vrátenie nohy do zákl. p.	15/15	100

Tabuľka 10. Testovanie pohybov.

Celková úspešnosť: 89,97%

Zhodnotenie

Vytvorené pohyby aj napriek faktu, že celková úspešnosť je necelých 90%, spĺňajú očakávania. Nestabilita hráča je často spôsobená tým, že agent nedostane vždy všetky správy od servera.

2.6 Šprint č. 6

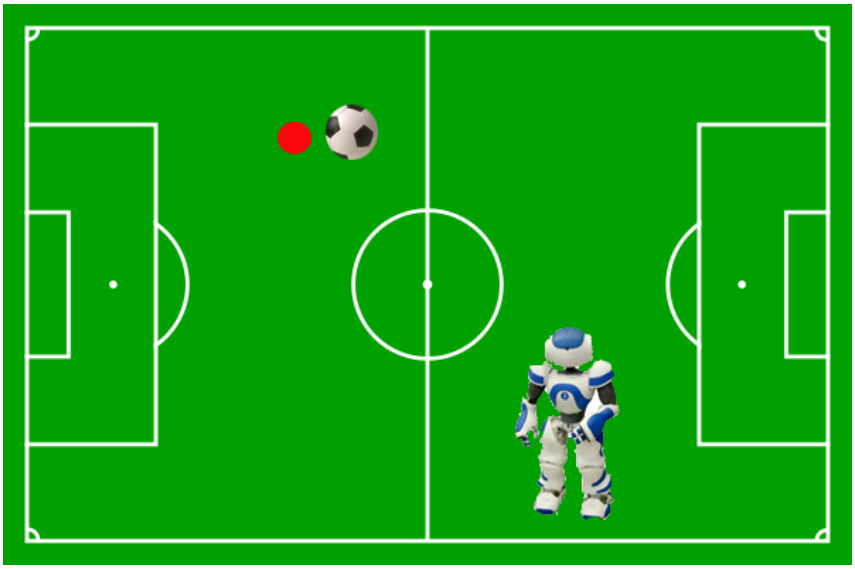
ZAČIATOK ŠPRINTU:	17.02. 2010
KONIEC ŠPRINTU:	03.03. 2010
INFORMÁCIE:	V ŠIESTOM ŠPRINTE SME DOKONČOVALI TASKY, KTORÉ SA NESTIHLI V PREDCHÁDZAJÚCOM ŠPRINTE. OKREM TOHO BOLO POTREBNÉ PRIPRAVIŤ VŠETKY DOKUMENTY TAK, ABY SME ICH MOHLI PREZENTOVAŤ. TAKTIEŽ BOLO POTREBNÉ AKTUALIZOVAŤ WEBOVÚ STRÁNKU.
POČET PRÍBEHOV:	8

2.6.1 Elementárna logika

Táto kapitola obsahuje návrh elementárnej logiky hráča, ktorý zahŕňa základné rozhodovanie pri hre futbal. Návrh zahŕňa získanie lopty, jej vedenie až k bráne a samotnú streľbu na bránu.

2.6.1.1 Návrh

Návrh elementárnej logiky dopĺňa návrh strednej logiky z kapitoly X o podporu subautomatov. Prechodová funkcia, ktorá je reprezentovaná rozhodovacím stromom po vyhodnotení v pôvodnom návrhu vrátila stav, ktorý obsahoval iba elementárny pohyb. Nový návrh dopĺňa stav aj o prechod do subautomatu. Stav môže byť reprezentovaný aj skupinou iných stavov, medzi ktorými sú opäť definované prechodové funkcie. Takto sa rozširuje celé riešenie a vzniká abstraktný stavový automat, kde každý stav môže byť reprezentovaný ďalším abstraktným stavovým automatom. Doplnenie logiky si vyžiadalo aj zmeny v konfiguračnom súbore, kde premenná v stave **motioname** bola nahradená premennou **action**. Na obrázku 1 sa nachádza agent, lopta a cieľová pozícia (vyznačená červeným) kam sa musí dostať agent z akejkoľvek polohy na ihrisku pred zahájením útoku.



Obr. 22. – Bod pred loptou

Elementárna logika predstavuje vhodné spájanie základných pohybov tak, aby agent bol schopný prísť k lopte, viesť ju až k bránke a vystreliť na ňu. Napríklad samotné strieľanie sa skladá zo skupiny elementárnych pohybov ako chôdza, krokovanie v pred a v bok tak, aby zasiahnutie lopty agentom bolo čo najúčinnejšie a zároveň agent nespadol. Pre zabezpečenie logiky bolo nutné doplniť sadu vyhodnocovacích funkcií.

Popis elementárnej logiky v jednotlivých krokoch:

1. V inicializačnom stave je ako prvé vyhodnotené rozhodnutie v rámci prechodovej funkcie, či je hráč schopný hry, teda či stojí alebo leží na zemi. Pomocou vyhodnocovacích funkcií sa zistí, či hráč leží na bruchu, chrbte, boku a na základe toho sa vykoná príslušný elementárny pohybu pre postavenie.

2. Kontrola, či hráč vidí loptu. Cieľom je vykonať čo najväčšie možné otočenie tak, aby hráč bol schopný vidieť loptu a spoluhráčov.

3. Ďalej ak je zabezpečené, že hráč stojí a vidí loptu sa rozhodne podľa toho, kto má loptu o rozmiestnení hráčov na ihrisku.

4. Rozhodovanie jednotlivých hráčov závisí od pozície lopty a protihráčov. Vždy hráč, ktorý sa
   nachádza najbližšie k lopte sa ju snaží získať. Ostatní spoluhráči sa rozhodujú podľa toho, či je tím v ofenzíve resp. defenzíve. Napríklad v útoku sa hráč snaží uvoľniť naopak pri obrane pokryť voľných protihráčov.

5. Hráč, ktorý vlastní loptu, pred streľbou na bránu berie v úvahu vzdialenosť od bránky. V prípade, že v strele bráni protihráč, hráč spraví kľučku alebo prihrá voľnému spoluhráčovi.

6. Ak sa hráč nachádza v dostatočnej blízkosti od brány vystrelí na ňu.

2.6.1.2 Zhodnotenie

V tejto kapitole sme stručne popísali základnú logiku hráča, ktorej cieľom bolo zabezpečiť schopnosť hráča zistiť pozíciu lopty na ihrisku viesť ju až k bráne a následne vystreliť. V prípade, že agent spadne byť schopný sa postaviť a pokračovať ďalej vo vykonávaní. Súčasťou návrhu je aj súbor, ktorý obsahuje samotné stavy a rozhodovacie stromy.

2.6.2 Úprava hráča pre komunikáciu so serverom 0.6.3

2.6.2.1 Analýza

V serveri 0.6.3 pribudol akcelerometer (nový perceptor). Okrem toho v predchádzajúcej verzii hráča sme neparsovali všetky perceptory (See - časti tiel ostatných hráčov, hear, TCH, FRP). To si vyžadovalo zmeny v dátovom modeli a v komunikačnom module.

2.6.2.2 Návrh a implementácia

Keďže už hráč vidí rôzne časti tela ostatných hráčov, inštanciu triedy Body (kolekcia častí tela - BodyPart) obsahuje každý hráč (inštancia triedy Player). Agent (Me) je podtriedou triedy Hráč a obsahuje ďalšie informácie o svojej kamere a správaní. Trieda World je kolekciou statických objektov (Flags) a dynamických (lopta, hráči). Teda sa podstatne zmenila kompozičná a dedičná hierarchia tried. Tiež každý objekt (vrátane častí tela) je identifikovateľný na základe svojho id (implementovaný ako enum). Jednotlivé pohyby teda uchovávajú iba id efektorov a teda sú aplikovateľné na ľubovoľnú inštanciu Body. Pre každý enum pribudli metódy toString a fromString, takže sa zjednodušilo parsovanie správ a konfiguračných súborov.

V predchádzajúcej verzii agenta parser priamo zasahoval do dátových štruktúr v modeli hráča, teraz pre každý perceptor existuje samostatná trieda a konkrétne dátové štruktúry majú metódu update(<Perceptor>&). Interakcia medzi dátovým modelom a komunikačným modulom sa zúžila na triedy Perceptors a Effectors.

2.6.2.3 Testovanie

Nový model hráča bol testovaný pri spustení hráča (chôdza so vstávaním), to je opísané v rámci inej úlohy v tomto dokumente. Funkčnosť väčšiny zmien sa prejavila na korektnom správaní hráča a ostatné sme kontrolovali z logov programu.

2.6.2.4 Zhodnotenie

RoboCup server sa neustále vylepšuje a my sa snažíme nášho hráča týmto zmenám prispôsobovať. Vylepšenia v nových verziách servera sú často pre hráča prospešné (perceptor ACC, väčšie ihrisko, dlhšie správy pre efektor Say, optimalizácia pre väčší počet hráčov, a.i.).

2.7 Šprint č. 7

ZAČIATOK ŠPRINTU:	03.03. 2010
KONIEC ŠPRINTU:	17.03. 2010
INFORMÁCIE:	V SIEDMOM ŠPRINTE SME DOKONČOVALI TASKY, KTORÉ SA NESTIHLI V PREDCHÁDZAJÚCOM ŠPRINTE. OKREM TOHO BOLO POTREBNÉ PRIPRAVIŤ VŠETKY DOKUMENTY TAK, ABY SME ICH MOHLI PREZENTOVAŤ. TAKTIEŽ BOLO POTREBNÉ AKTUALIZOVAŤ WEBOVÚ STRÁNKU.
POČET PRÍBEHOV:	10

2.7.1 Resetovanie a skladanie pohybov

2.7.1.1 Analýza

V predchádzajúcej verzii nebolo možné resetovať ani skladať pohyby. Tým pádom nebolo možné dvakrát po sebe zopakovať ten istý pohyb. Tiež boli problémy s tým, že každý pohyb implicitne začínal so začiatočnej pozície a nedal sa plynulo nadviazať na predchádzajúci pohyb.

2.7.1.2 Návrh

Každý pohyb bude mať metódu reset, ktorá nastaví interné počítadlo na nulu. Okrem toho súčasťou každého pohybu budú počiatočné uhly kĺbov. Potom v prípade, že predchádzajúci pohyb neskončil v štartovacej polohe nasledujúceho pohybu, hráč automaticky konštantnou rýchlosťou ohne kĺby do želaných pozícií ešte predtým, ako začne vykonávať tento pohyb. Medzi nastavením kĺbov a samotným pohybom bude parametrické oneskorenie (pre každý pohyb zvlášť), aby vedľajšie efekty nastavenia kĺbov (zotrvačnosť) veľmi neovplyvnili vykonávaný pohyb.

2.7.1.3 Implementácia

Nastali zmeny v triede Action. Pribudla metóda reset(), atribúty startingPose (zoznam počiatočných uhlov jednotlivých kĺbov) a delay (oneskorenie). Pohyb sa vykonáva nasledovne:

1. reset
2. nastav počiatočné hodnoty kĺbov (startingPose)

3. počkaj (delay)
4. vykonaj pohyb (rovnako ako v starej verzii hráča)

Okrem pohybov vytvorených v editore sú k dispozícii procedurálne pohyby InitialPose (ohne všetky kĺby do polohy 0), StraightPose (vyrovná hráča) a Idle (nehýbe hráčom, iba počká 1 sekundu).

Nové atribúty pohybov vyžadovali rozšírenie syntaxe .rmo súboru:

```
[MOTION]
NAME=postavenie
DESCRIPTION=
[PRECONDITIONS] # pociatocne hodnoty klbov
NAME=lae1       # efektor klbu
ANGLE=0         # velkost uhla v stupnoch
NAME=rae1
ANGLE=0
...
DELAY=0.5       # oneskorenie v sekundach
[STATE]
...
```

2.7.1.4 Testovanie

Resetovanie aj kĺbové podmienky boli otestované na chôdzi so vstávaním, ktorej bola venovaná samostatná úloha a jej opis je súčasťou tohto dokumentu.

2.7.1.5 Zhodnotenie

Táto úloha mala vysokú prioritu, lebo od nej závisí plynulé a opakovateľné vykonávanie sekvencií pohybov podľa hráčovej logiky. Až po jej splnení sme mohli vytvoriť a prezentovať komplexnejšie hráčovo správanie.

2.8 Šprint č. 8

ZAČIATOK ŠPRINTU:	17.03. 2010
KONIEC ŠPRINTU:	31.03. 2010
INFORMÁCIE:	V OSMOM ŠPRINTE SME DOKONČOVALI TASKY, KTORÉ SA NESTIHLI V PREDCHÁDZAJÚCOM ŠPRINTE. OKREM TOHO BOLO POTREBNÉ PRIPRAVIŤ VŠETKY DOKUMENTY TAK, ABY SME ICH MOHLI PREZENTOVAŤ. TAKTIEŽ BOLO POTREBNÉ AKTUALIZOVAŤ WEBOVÚ STRÁNKU.
POČET PRÍBEHOV:	8

2.8.1 Chôdza s postavením

2.8.1.1 Analýza

Cieľom úlohy bolo vytvoriť základ pre bezpečný pohyb hráča. Ak chce hráč vykonať sekvenciu pohybov, musí zareagovať, keď padá a postaviť sa. Výstupom úlohy bol súbor správania, ktorý systematicky opakuje zadaný pohyb (chôdzu) a rieši postavenie v prípade pádu.

2.8.1.2 Návrh

V hlavnom stave bude hráč cyklicky opakovať kráčanie (pohyb „Chodza“). V tomto stave vždy vyhodnocuje, či padá (funkcia „padamZoStoja“). Ak áno, automaticky sa vystrie (pohyb „StraightPose“) a počká (pohyb „Idle“), kým sa ustáli. Potom zistí, na akej strane leží (funkcia „naAkejStraneLezim“) a ak neleží na bruchu, musí sa pretočiť (pohyb „otocenieZchrbta“), toto opakuje, kým sa mu to nepodarí. Keď leží na bruchu, postaví sa (pohyb „postavenie“). Pokiaľ sa postavenie nepodarilo, hráč sa znova vystrie, počká a zistí, ako sa má postaviť. Keď hráč už stojí, pokračuje vo vykonávaní chôdze.

2.8.1.3 Implementácia

```
# VYHODNOCOVACIE FUNKCIE
# padamZoStoja
# naAkejStraneLezim
# bezPohybu

# POHYBY
# Chodza
# StraightPose
# otocenieZchrbta
# postavenie
# Idle

[VALUES]
false=<0,0.5)
true=<0.5,1>

[DECISION]
NAME=PadamZoStoja?
FUNCTION=padamZoStoja
true=VyrovnajSa
false=HlavnaLogika

[DECISION]
NAME=SomVyrovnany?
FUNCTION=bezPohybu
true=PockajChvilu
false=VyrovnajSa

[DECISION]
```

```
NAME=DocakalSom?
FUNCTION=bezPohybu
true=AkoLezim?
false=PockajChvilu

[DECISION]
NAME=AkoLezim?
FUNCTION=naAkejStraneLezim
3=VstanZBrucha
2=PretocSaNaBrucho
<0,1>=VyrovnajSa

[DECISION]
NAME=PretocilSomSa?
FUNCTION=bezPohybu
true=AkoLezim?
false=PretocSaNaBrucho

[DECISION]
NAME=SkonciloVstavanie?
FUNCTION=bezPohybu
true=HlavnaLogika
false=VstanZBrucha

[STARTINGSTATE]
NAME=HlavnaLogika
MOTIONNAME=Chodza
ROOTDECISION=PadamZoStoja?

[STATE]
NAME=VyrovnajSa
MOTIONNAME=StraightPose
ROOTDECISION=SomVyrovnany?

[STATE]
NAME=PockajChvilu
MOTIONNAME=Idle
ROOTDECISION=DocakalSom?

[STATE]
NAME=PretocSaNaBrucho
MOTIONNAME=otocenieZchrbta
ROOTDECISION=PretocilSomSa?

[STATE]
NAME=VstanZBrucha
MOTIONNAME=postavenie
ROOTDECISION=SkonciloVstavanie?
```

Implementácia vyhodnocovacích funkcií je popísaná vyššie.

2.8.1.4 Testovanie

Test sa uskutočnil na počítači PC1 s technickými parametrami: procesor: 1,66GHz Core 2 Duo, RAM: 4GB, OS: Windows 7 (32bit).

Počítač		**Očakávaný výsledok**
PC1		Agent kráča, spadne, postaví sa a kráča ďalej.
Test	**Výsledok**	**Skutočný výsledok**
1.	OK	Agent spadol a cyklicky sa pokúšal postaviť, ale nedarilo sa mu.
2.	OK	Agent spadol a cyklicky sa pokúšal postaviť, ale nedarilo sa mu.
3.	OK	Agent spadol a postavil sa.
4.	OK	Agent spadol a cyklicky sa pokúšal postaviť, ale nedarilo sa mu.
5.	OK	Agent spadol a cyklicky sa pokúšal postaviť, ale nedarilo sa mu.

Tabuľka 11 – test vykonaný na PC1

Napriek tomu, že z 5 testov sa agent iba raz úspešne postavil, všetky testy boli úspešné, lebo cieľom bolo korektné správanie hráča a ten zistil, že sa mu nepodarilo postaviť a pokúšal sa postaviť znova. V budúcnosti treba vylepšiť pohyby „otocenieZchrbta“ a „postavenie“.

2.8.1.5 Zhodnotenie

Toto správanie hráča je dôležitý pokrok hlavne preto, že v budúcnosti môžeme hocijakú sekvenciu pohybov (nielen jeden cyklický pohyb) „obaliť“ do vstávacieho modulu.

2.9 Šprint č. 9

ZAČIATOK ŠPRINTU:	31.03. 2010
KONIEC ŠPRINTU:	14.4. 2010
INFORMÁCIE:	V DEVIATOM ŠPRINTE SME DOKONČOVALI TASKY, KTORÉ SA NESTIHLI V PREDCHÁDZAJÚCOM ŠPRINTE. OKREM TOHO BOLO POTREBNÉ PRIPRAVIŤ VŠETKY DOKUMENTY TAK, ABY SME ICH MOHLI PREZENTOVAŤ. TAKTIEŽ BOLO POTREBNÉ AKTUALIZOVAŤ WEBOVÚ STRÁNKU.
POČET PRÍBEHOV:	4

2.9.1 Editor strednej logiky

2.9.1.1 Analýza

Stredná logika je reprezentovaná v súbore. Tento súbor dokáže agent načítať a následne podľa neho vykonávať jednotlivé pohyby. Súbory so strednou logikou sú textové súbory, ktoré je potrebné v priebehu riešenia projektu meniť prípadne vytvárať. Editácia prostredníctvom klasických textových editorov je veľmi nepraktická a preto je potrebné vytvoriť nástroj, ktorý dokáže túto logiku meniť, prípadne aj vytvárať.

V súčasnosti existuje editor pohybov, ktorý dokáže vizualizovať strednú logiku a preto je vhodné umiestniť editor strednej logiky taktiež do editoru pohybov.

2.9.1.2 Návrh

Návrh editoru strednej logiky bude vychádzať predovšetkým zo štruktúry strednej logiky. Stredná logika sa skladá z automatov. Jednotlivé automaty sa skladajú zo States, ktoré reprezentujú jednotlivé pohyby agenta. Aby agent dokázal vyhodnotiť, ktorý State je preňho najvhodnejší, použije Decisions. Decisions obsahujú vyhodnocovanie funkcie, na základe ktorých je agent schopný vytvárať rozhodnutia. Na záver obsahuje ešte hodnoty Values, ktoré sú len akýmsi mapovaním návratových hodnôt funkcií.

Všetky vytvorené automaty budú ukladané do jedného priečinku. V rámci editoru si bude možné prepínať medzi všetkými existujúcimi automatmi prostredníctvom combo boxu.

Pre zobrazenie automatu vytvoríme strom, ktorý bude rozdelený na States, Decisions a Values. V každej z týchto vetví budú zobrazené jej jednotlivé položky.

Každú položku stromu bude možné editovať prostredníctvom formulára, ktorý bude taktiež súčasťou rovnakého okna. Pri kliknutí používateľa na jednotlivú položku sa zobrazia jej hodnoty v príslušných textových poliach, ktoré budú editovateľné. V prípade, že sa používateľ

rozhodne urobiť zmenu, po kliknutí pravým tlačidlom na strom sa mu zobrazí kontextová ponuka, prostredníctvom ktorej bude môcť vytvárať nové alebo mazať už existujúce položky.

K dispozícií bude aj ukladanie zmenenej strednej logiky, ktorá môže byť uložená ako nový súbor alebo ako už existujúci.

Súčasťou editora bude aj stromové zobrazenie strednej logiky. V rámci tohto stromového zobrazenia bude vidieť nadväznosti jednotlivých Decisions a States.

2.9.1.3 Implementácia

Editor strednej logiky bol vytvorený ako nadstavba editoru pohybov od predchádzajúceho tímu. Bola vytvorená nová trieda DecisionTreeEditor, ktorá obsahuje všetky potrebné metódy, ktoré sú využité pri editácií strednej logiky. Pri inicializácií editoru je načítaný súbor so strednou logikou prostredníctvom už existujúcich metód triedy LoadSave. Tento súbor je načítaný do štruktúry Automat, ktorá už obsahuje všetky metódy a štruktúry pre prácu s automatom.

Pri vytváraní jednotlivých stromových štruktúr automatov sú využité objekty TreeView.Tag, ktoré referencujú na skutočný objekt v štruktúre automat. Takýmto spôsobom je možné zabezpečiť aktualizáciu jednotlivých štruktúr automatu priamo prostredníctvom stromovej štruktúry automatu.

Ukladanie strednej logiky do súboru je riešení prostredníctvom triedy LoadSave, ktorá obsahuje metódu pre uloženie automatu do súboru.

2.9.1.4 Testovanie

Testovanie editoru strednej logiky prebiehalo podľa dopredu vytvorených scenárov, ktoré boli vytvorené na základe návrhu editora strednej logiky. Každý scenár bol otestovaný na viacerých rôznych vstupoch. Testy spolu s výsledkami sú znázornené v nasledujúcich tabuľkách:

Očakávaný výsledok		
Editor načíta súbor so strednou logikou a zobrazí ho v stromových štruktúrach		
Súbor	**Výsledok testu**	**Skutočný výsledok**
automat1.txt	OK	Editor načítal a zobrazil súbor správne
automat2	OK	Editor načítal a zobrazil súbor správne
sample1.txt	OK	Editor načítal a zobrazil súbor správne

Tabuľka 12 – test načítavania súboru so strednou logikou

Očakávaný výsledok

Editor uloží súbor so strednou logikou		
Súbor	**Výsledok testu**	**Skutočný výsledok**
automat1.txt	OK	Editor uložil a následne načítal rovnaké stromové štruktúry automatu
automat2	OK	Editor uložil a následne načítal rovnaké stromové štruktúry automatu
sample1.txt	OK	Editor uložil a následne načítal rovnaké stromové štruktúry automatu

Tabuľka 13 – test ukladania strednej logiky do súboru

Očakávaný výsledok		
Do stromovej štruktúry sa pridá nová položka		
Položka	**Výsledok testu**	**Skutočný výsledok**
Decision	OK	Editor vložil do stromovej štruktúry aj automatu položku decision
States	OK	Editor vložil do stromovej štruktúry aj automatu položku state
Values	OK	Editor vložil do stromovej štruktúry aj automatu položku value

Tabuľka 14 – test pridávania nových položiek do strednej logiky

Očakávaný výsledok		
Zo stromovej štruktúry sa vymaže položka		
Položka	**Výsledok testu**	**Skutočný výsledok**
Decision	OK	Editor vymazal položku decision zo stromovej štruktúry aj automatu
States	OK	Editor vymazal položku state zo stromovej štruktúry aj automatu

Values	OK	Editor vymazal položku value zo stromovej štruktúry aj automatu

Tabuľka 15 – test mazania položiek zo strednej logiky

Očakávaný výsledok		
Editovaná položka strednej logiky bude aktualizovaná v automate aj v stromovej štruktúre		
Položka	**Výsledok testu**	**Skutočný výsledok**
Decision	OK	Položka decision bola aktualizovaná v stromovej štruktúre aj automate
States	OK	Položka state bola aktualizovaná v stromovej štruktúre aj automate
Values	OK	Položka value bola aktualizovaná v stromovej štruktúre aj automate

Tabuľka 16 – test zmeny položky strednej logiky

2.9.1.5 Zhodnotenie

Editor strednej logiky bol vytvorený podľa návrhu a funguje spoľahlivo. Je ním možné meniť a vytvárať nové súbory so strednou logikou. V budúcnosti je ho ešte možné rozšíriť o prívetivejšie používateľské rozhranie.

2.9.2 Výpočet hodnôt nezávisle na hráčovej domácej polovici ihriska

2.9.2.1 Analýza

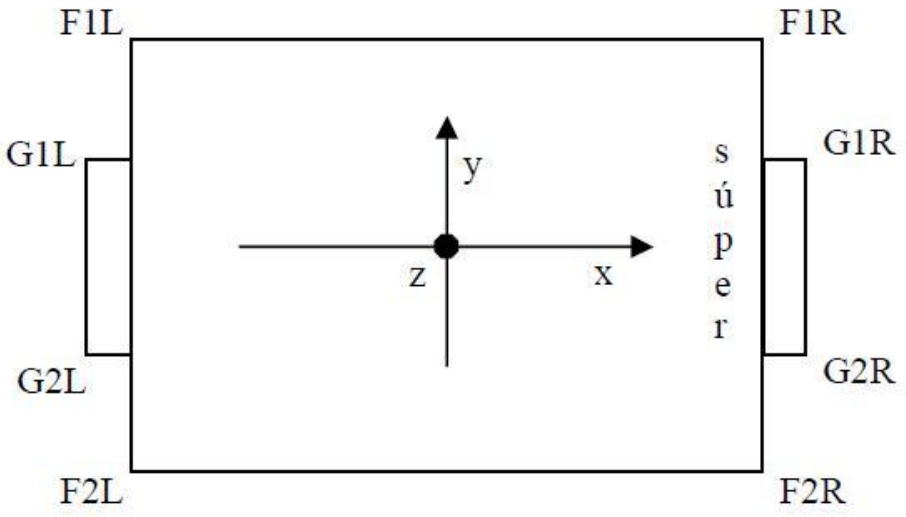
Správanie hráča je riadené tým ako vníma svet – cez jeho vnútorný stav, hodnoty premenných ktoré obsahujú informácie, ktoré získal svojim perceptormi o okolí. Najdôležitejšími sú informácie o polohe hráča vrámci ihriska a relatívnych pozíciách iných objektov (hráčov, lopty, bránok, ...) voči nemu. Tieto sa vypočítavajú na základe perceptora *See* z aspoň troch z ôsmych objektov na ihrisku – štyroch tyčiek bránok a štyroch rohových zástaviek. Absolútna pozícia každého z týchto objektov na ihrisku je dopredu známa.

2.9.2.2 Návrh

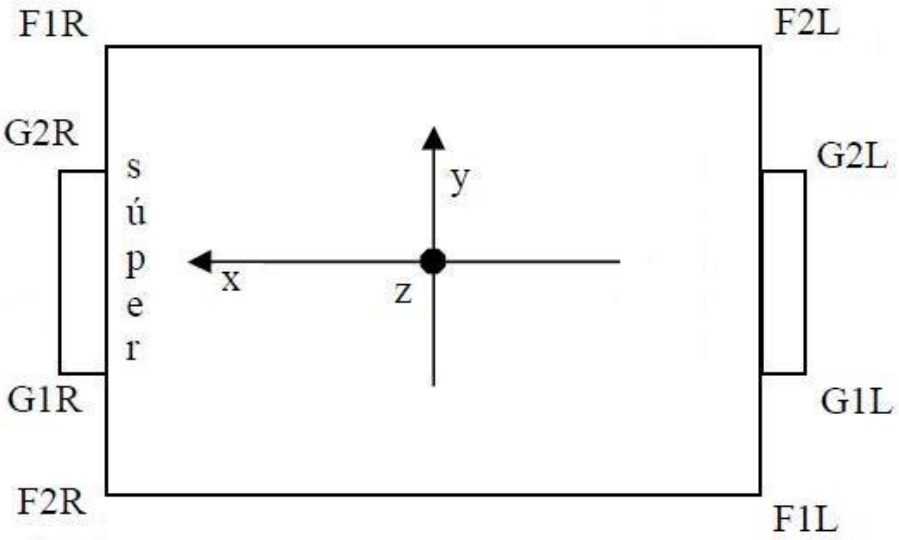
Hráč pôvodne predpokladal, že je vždy na domácej polovici na ľavej strane ihriska, takže všetky výpočty prebiehali relatívne k tejto strane ihriska. Toto však vždy nemusí platiť, preto bolo potrebné vytvoriť prepočítavanie hodnôt tak, aby vyhodnocovacie funkcie, ale aj iné časti agenta mohli pracovať nezávisle na hráčovej domácej polovici ihriska.

2.9.2.3 Implementácia

Základom bolo vedieť, ktorá je hráčova domáca polovica. Táto informáciá prichádza v správe *GS* (GameState) v časti *team*, a ukladá sa v štruktúre opisujúcej hráčov tím. Následne sa pri spracovávaní perceptora *See* na základe tejto informácie modifikujú vstupy tak, aby zodpovedali pohľadu hráča na ihrisko. Čiže v prípade, že hráč je doma na ľavej strane, nič sa nemení a v prípade že je doma na pravej strane, preklopia sa vstupy o objektoch v stredovej súmernosti podľa stredu ihriska (na obrázkoch).



Obr. 23: Normálny pohľad hráča – keď je doma na ľavej strane ihriska



Obr. 24: Pohľad hráča keď je doma na pravej strane ihriska

2.9.2.4 Testovanie

Výpočet hodnôt opisujúcich stav sveta nezávisle na hráčovej domácej polovici ihriska bol otestovaný pri testovaní všetkých ostatných úloh, najmä vyhodnocovacích funkcií a pohybov, ktoré boli spúštané s hráčom na oboch možných stranách ihriska.

2.9.2.5 Zhodnotenie

Výsledné prepočty sú dôležitou súčasťou logiky agenta, bez ktorej nemôže spoľahlivo fungovať. Využívajú sa vo všetkých pokročilejších schopnostiach agenta, najmä pri vyhodnocovaní stavu sveta a rozhodovaní o pohybe agenta.

2.9.3 Vyhodnocovacie funkcie

2.9.3.1 Analýza

Cieľom vyhodnocovacích funkcií, tak ako boli navrhnuté v rámci návrhu strednej logiky agenta, je poskytnúť modulu strednej logiky podporné prostriedky pre rozhodovanie. Agent má vo vnútornom dátovom modely uložené údaje o svete, ktoré potrebuje sprístupniť strednej logike. Vyhodnocovacie funkcie sú protokolom, ktorý toto zabezpečuje. Niektoré vyhodnocovacie funkcie priamo vracajú hodnoty niektorých premenných, väčšinou však najskôr spracujú dáta a vracajú priamo konkrétnu hodnotu, podľa ktorej dokáže agent rozhodnúť.

2.9.3.2 Návrh

Vyhodnocovacie funkcie sú vlastne množinou funkcií, implementovaných priamo v agentovi v jazyku C++. Využívajú vnútornú reprezentáciu agenta o stave na ihrisku, všetky uložené informácie v pamäti získané z perceptorov (najmä perceptora *See*). Navrhnuté boli spolu so strednou logikou tak, že vracajú akúkoľvek hodnotu typu *float*. Možné reprezentácie skutočných hodnôt (*true, false, nas_tim, superov_tim, klucka, prihravka, ...*) sú namapované na rôzne hodnoty v množine reálnych čísel, v závislosti od konkrétnej vyhodnocovacej funkcie. Každá funkcia je identifikovaná svojim menom v kóde agenta, toto meno sa používa aj v rámci konfiguračných súborov strednej logiky.

2.9.3.3 Implementácia

Nasleduje popis jednotlivých implementovaných vyhodnocovacích funkcií.

akoSomNatoceny

Funkcia predpokladá, že agent je vystretý. V tom prípade buď stojí alebo leží na bruchu, chrbte alebo boku. Najprv sa hráčov vektor orientácie otočí do absolútnych súradníc a potom sa vypočíta uhol medzi otočeným vektorom a osou Z. Ak je uhol blízky 0 alebo PI, hráč leží na chrbte, resp. bruchu. Ak je uhol blízky pravému, hráč buď leží na boku (pravom alebo ľavom) alebo stojí, to sa určí na základe uhla medzi hráčovým ľavým vektorom (otočeným do absolútnej súradnicovej sústavy) a osou Z.

padamZoStoja

Funkcia predpokladá, že agent je v stojí, takže keď je jeho hlava pod určitou výškovou hranicou (závislou od rozmerov tela hráča), funkcia ohlási, že agent padá.

bezPohybu

Skontroluje, či agent nevykonáva nijaký pohyb. Používa sa na zistenie, či práve vykonávaný pohyb už skončil.

akoDalekoSomOdLopty

Vracia plošnú vzdialenosť hráča k lopte na základe relatívnej pozície lopty ku hráčovi.

akoDalekoSomOdNasejBrany

Vracia plošnú vzdialenosť hráča k vlastnej bránke na základe relatívnych pozícií tyčiek bránky ku hráčovi

akoDalekoSomOdIchBrany

Vracia plošnú vzdialenosť hráča k súperovej bránke na základe relatívnych pozícií tyčiek bránky ku hráčovi.

somStabilny

Zistí či je hráč stabilný, hráč je stabilný ak je gyro rate 0,0,0, čiže sa nevychyluje z aktuálnej pozicie.

stojim

Zistí či hráč stojí, hráč stojí, ak sa nevykonáva žiadny pohyb, je stabilný a akcelerometer ma kladnú hodnotu veľkosti aspoň 9 na osi Z a na osiach X a Y absolutnu hodnotu menej ako 1.

naAkejStraneLezim

Zistí aktuálnu polohu hráča. Predpokladá, že hráč je vyrovnaný a leží buď na bruchu alebo na chrbte. Používa gyroskop alebo akcelerometer, podľa toho, čo je prístupné. Hráč buď leží na bruchu (3.0f), na chrbte (2.0f) alebo ani jedno (1.0f) alebo nemá dosť informácií na vyhodnotenie (0.0f).

lezimNaBruchu

Hráč leží na bruchu, ak sa nevykonáva žiadny pohyb, je stabilný a akcelerometer ma zápornú hodnotu veľkosti aspoň 9 na Y.

lezimNaChrbte

Hráč leží na chrbte, ak sa nevykonáva žiadny pohyb, je stabilný a akcelerometer ma kladnú hodnotu veľkosti aspoň 9 na Y.

dokazemDoLoptyKopnut

Na základe vzdialenosti k lopte vráti či hráč dokáže kopnúť do lopty (1.0f) alebo nie (0.0f).

mozemStrielatNaBranu

Vracia hodnoty "ano" (1.0f), "nie" (2.0f), "klucka"(3.0f) a "prihravka" (3.0f) ak mám nejakých spoluhráčov. Ak postavíme do cesty protihráča a hráč nebude mať žiadnych spoluháčov, funkcia by mala vrátiť "klucka", to znamená, že hráč si pred seba kopne loptu vľavo alebo vpravo, nemôže páliž pred seba. Funkcia vráti "nie" ak hráč je dosť ďaleko od brány a strela by neohrozila bránu (napr. sme na našej strane). "ano" vráti ak je hráč blízko brány a môže strielať.

ktoMaLoptu

Na základe toho ako blízko sú jednotliví hráči k lopte vráti toho, ktorý je k nej najbližšie. Potom podľa tejto informácie zistí, do ktorého tímu hráč patrí a vráti tento tím.

2.9.3.4 Testovanie

Jednotlivé vyhodnocovacie funkcie boli otestované v rámci testovania strednej logiky – pri chôdzi so vstávaním a otáčaním z chrbta na brucho. Funkcie, ktoré neboli zahrnuté v tejto strednej logike, boli otestované pomocou výpisov do konzoly - v každom cykle boli vypísané hodnoty všetkých vyhodnocovacích funkcií. Hráč bol umiestňovaný na rôzne pozície na ihrisku, vzhľadom k lopte a vzhľadom na súperovho hráča. Všetky funkcie fungovali spoľahlivo a nijako výrazne nespomaľovali hráča napriek tomu, že boli vykonávané v každom cykle.

2.9.3.5 Zhodnotenie

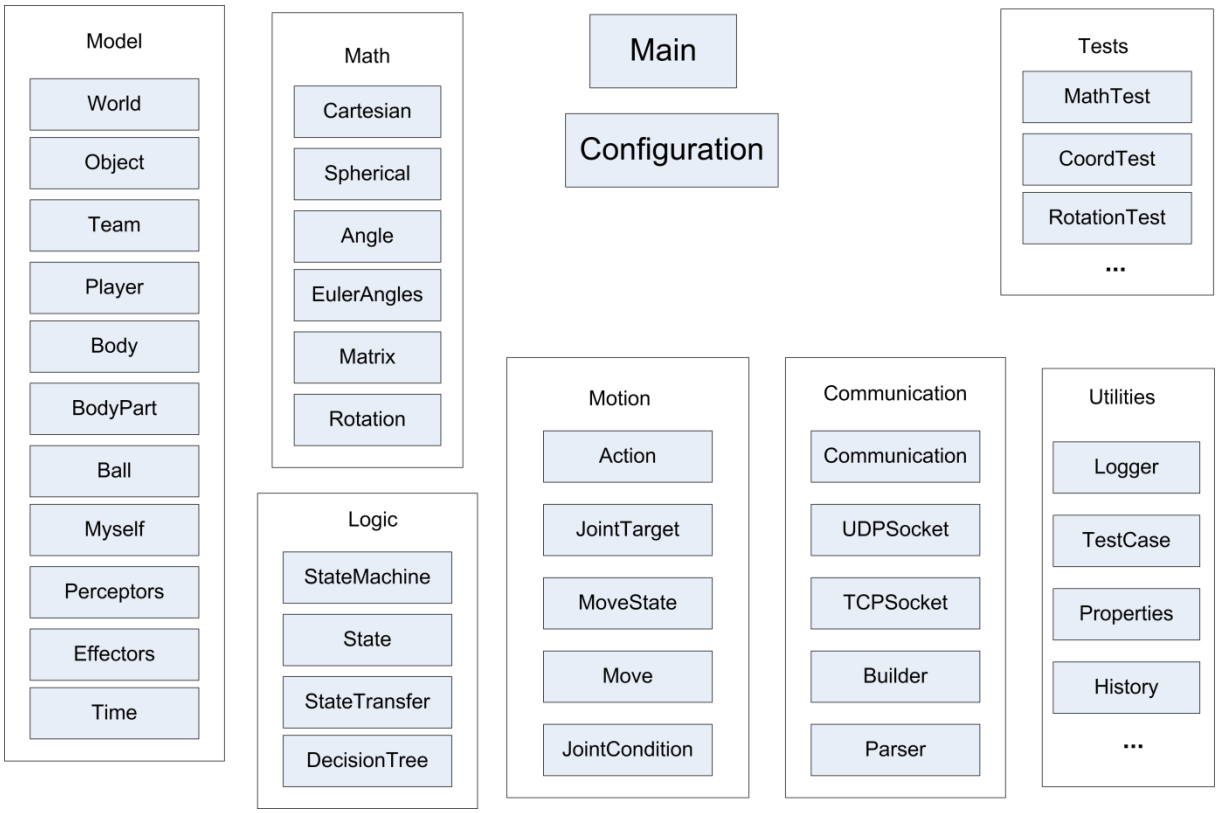
Výsledkom implementácie je funkčný a otestovaný modul obsahujúci najpotrebnejšie vyhodnocovacie funkcie, slúžiace pre potreby strednej logiky. Do budúcnosti je možné ho rozšíriť o ďalšie zložitejšie funkcie, prípadne iné potrebné časti.

2.10 Šprint č. 10

ZAČIATOK ŠPRINTU:	14.04. 2010
KONIEC ŠPRINTU:	28.04. 2010
INFORMÁCIE:	V DESIATOM ŠPRINTE SME DOKONČOVALI TASKY, KTORÉ SA NESTIHLI V PREDCHÁDZAJÚCOM ŠPRINTE. OKREM TOHO SME SA ZÚČASTNILI ŠTUDENTSKEJ VEDECKEJ KONFERENCIE IIT.SRC
POČET PRÍBEHOV:	3

Príspevok na IIT.SRC sa nachádza v prílohe D.

3 ZHRNUTIE



Obr. 25 : Diagram tried

Projekt je rozdelený do niekoľkých modulov (na obrázku):

- Model
  Uchováva informácie o stave sveta a objektov: pozície, rýchlosti, zrýchlenia.
- Math
  Sú tu umiestnené triedy s matematickými štruktúrami a operáciami.
- Logic
  Obsahuje stavový automat a prechodové funkcie reprezentované rozhodovacím stromom. Hierarchické automaty zatiaľ nie sú implementované.
- Motion
  Triedy slúžiace na vykonávanie pohybov.
- Communication

Sprostredkuje komunikáciu so serverom protokolom TCP alebo UDP, stará sa o parsovanie a skladanie správ pomocou s-výrazov.

- Utilities
  Pomocné triedy na vedľajšie činnosti ako parsovanie konfiguračných súborov, logovanie a.i.
- Tests
  Sú tu umiestnené testy niektorých tried a algoritmov vo forme TestCase-ov.

V rámci našej práce sme v agentovi implementovali:

- Matematický model
  Výpočet orientácie a pozície kamery (na základe videných objektov a GyroRate), prevod bodov z kamerovej do globálnej sústavy, výpočet rýchlostí a zrýchlení objektov, súradníc ťažiska a častí tela a jednoduchú predikciu pohybu na niekoľko simulačných krokov
- História kinematických údajov pre každý dynamický objekt
- Správanie pomocou stavového automatu
  Implementácia automatu, jeho načítanie a vykonávanie
- Predpodmienky pre pohyby
  Rozšírenie formátu .rmo, načítanie a procedurálne vykonávanie predpodmienok
- Pomocné funkcie
  Logovanie, testovanie a.i.

Editor strednej logiky

Na základe návrhu zo zimného semestra sme rozšírili existujúci editor. Rozšírenie editora spočívalo v troch bodoch a to:

1. Modul elementárnych pohybov.

2. Modul strednej logiky.

3. Modul modelovania strednej logiky.

V editore pribudla možnosť modelovať strednú logiku hráča čo je umožnené samostatným robustným komplexným modelom editora strednej logiky. Modul modelovania strednej logiky umožňuje jednoduchým a efektívnym spôsobom vytvárať logiku ako aj editovať existujúcu logiku. V rámci jedného súboru je možné meniť ľubovoľné parametre strednej logiky .

Ďalšou časťou ktorá pribudla do editora bol komplexný modul elementárnych pohybov. V tomto module sa nachádzajú pohyby ktoré je možné nadväzovať za sebou a tak vytvoriť komplexný pohyb pozostávajúci z ľubovoľného počtu jednoduchých pohybov. riadený strednou logikou

Posledným rozšírením editora bolo vytvorenie modulu strednej logiky, ktoré umožňuje jednoduchým spôsobom spúšťať strednú logiku namodelovanú pomocou modulu na modelovanie strednej logiky.

POUŽITÁ LITERATÚRA

1. *3D Simulation Rules.* (október 2009). Dostupné na Internete: http://www.robocup2009.org/files/20090609_rules.pdf

2. Bitbucket.org (október 2009), http://www.bitbucket.org

3. Boedecker, J. (október 2009). *SimSpark User's Manual (verzia 1.1),.* Dostupné na Internete: http://simspark.sourceforge.net/wiki/images/a/ad/User-manual.pdf

4. Bustamante, C. F. (október 2009). A phisic model for the RoboCup 3D soccer simulation. *Proceedings of the 2007 spring simulation multiconference.*

5. Little Green Bats. (november 2009). *RoboCup 3D Simulation HowTo.* Dostupné na Internete: http://sourceforge.net/projects/littlegreenbats/files/Documentation/Howtorobocop/littlegreenbats_howtorobocop-0.1.tar.gz/download

6. NAITO-StrikerS. (október 2009). *The World Model for Autonomous Soccer Agents.* Cit. november 2009. Dostupné na Internete: http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/NAITO-StrikerS2007.pdf

7. Reis, L. L. (október 2009). *Flexible Teamwork and Configurable Strategy.* Cit. oktober 2009. Dostupné na Internete: http://www.ieeta.pt/robocup/documents/FCPortugalInteresting.ps.zip

8. Reis, L. L. (október 2009). *Portugal Team Description RoboCup 2000 Simulation League Champion.* Cit. október 2009. Dostupné na Internete: http://www.ieeta.pt/robocup/documents/FCPortugalChampion.ps.zip

9. Reis, L. L. (október 2009). *Reis, L.P., Lau, N., Oliviera, E.C.* Cit. október 2009. Dostupné na Internete: http://www.ieeta.pt/robocup/documents/SBSP.pdf

10. Saeid Akhavan, M. B. (október 2009). Dostupné na Internete: AI3D 2007 Team Description: www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/uiai2007TDP.pdf

11. Tím Hviezdna 11. (november 2007). *Dokumentácia k projektu. FIIT STU.* Cit. november 2009. Dostupné na Internete: http://labss2.fiit.stuba.sk/TeamProject/2007/team11is-si/download/tp_dokumentacia_final.pdf

12. Xu Yuan, T. Y. (október 2009). *SEU-3D 2007 Soccer Simulation Team.* Dostupné na Internete: http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/SEU-3D-TDP07.pdf

13. D.Collien, G. Huyn (október 2009).: From AIBO to Nao The Transition from 4Legged to 2Legged Robot Soccer David Collien

14. Q.Huang (október 2009). *Planning Walking Patterns for a Biped Robot* http://www.cc.gatech.edu/fac/Chris.Atkeson/legs/kuff1a.pdf

15. V. Duindam, S. Stramigioli (november 2009) Modeling and Control for Efficient Bipedal Walking Robots: A Port-Based Approach.: Dostupné na internete: http://books.google.sk/books?id=G4sySMh8wGcC&lpg=PR2&ots=lfUwTNq7Qv&dq=Modeling%20and%20Control%20for%20Efficient%20Bipedal%20Walking%20Robots%3A%20A%20Port-Based%20Approach&pg=PR2#v=onepage&q=&f=false

16. H. Yussof, M. Ohka, M. Yamano, Y. Nasu, (november 2009) Analysis of Human-Inspired Biped Walk Characteristics in a Prototype Humanoid Robot for Improvement of Walking Speed. Dostupné na internete: http://www.computer.org/portal/web/csdl/doi/10.1109/AMS.2008.182

17. Popovic M.B., Goswami A., Herr H. (október 2009). Ground Reference Points in Legged Locomotion:Definitions, Biological Trajectories and Control Implications http://www.ambarish.com/paper/Popovic_Goswami_Herr_IJRR_Dec_2005.pdf

18. Press Release (október 2009). World's First Running Humanoid Robot http://www.sony.net/SonyInfo/News/Press_Archive/200312/03-060E/

19. Honda (október 2009). ASIMO Walking http://world.honda.com/ASIMO/technology/walking.html

20. Honda (október 2009). ASIMO Specificaton http://asimo.honda.com/asimo_specifications.html

21. Niehaus C., Rofer T., Laue T. (október 2009). Gait Optimization on a Humanoid Robot using Particle Swarm Optimization, Universität Bremen, IEEE

22. Xu Yuan, Wang Wei, Zhao Xuqing, Chen Si, Jiang Hong, Tan Yingzi (október 2009), *Research on RoboCup Simulation 3D,* dostupné na internete: http://sites.google.com/site/xuyuancn/research_proposal.pdf

23. Bc. P. Nosko, Bc. D. Rodina, Bc. D. Slamka, Bc. P. Smolinský, Bc. O. Ševce, Bc. I. Tomovič, (október 2009).Robocup 3D-dokumentácia, 2009

24. Finálna dokumentácia tímu(október 2009) Agenty 007 dostupná na internete http://labss2.fiit.stuba.sk/TeamProject/2008/team07is-si/dokumentacia3.pdf

25. Dokumentácia (október 2009) DreamTeam dostupná na internete http://labss2.fiit.stuba.sk/TeamProject/2008/team01is-si/Documents/dokumentacia_letny_semester.doc

PRÍLOHA A: Návod na používanie nástroja TortoiseHG

Inštalácia

1. Stiahnuť z http://bitbucket.org/tortoisehg/stable/wiki/download, najnovší v čase písania http://bitbucket.org/tortoisehg/stable/downloads/TortoiseHg-0.8.3-hg-1.3.1+7cea12e70129.exe.
2. Plugin do Visual Studio 2003/2008, ale nevie nič viac ako TortoiseHG, takže sa dá aj bez neho http://mercurial.selenic.com/wiki/OtherTools, najnovší je dostupný na adrese http://dl.sharesource.org/visualhg/visualhg-1.0.6.msi.

Stiahnutie repozitára

Vytvoriť si niekde adresár kde to chcete mať (c:/workspaces/robocup napr.) a tam kliknúť pravým a *Clone a repository*, zadať *Source Path*: https://ACCOUNT_NAME@bitbucket.org/fojtik/robocup3d-team-project/. Treba v URL nahradiť ACCOUNT_NAME za login z BitBucketu a zadať heslo.

Zmeny

Každý má u seba svoj vlastný repository, s vlastnými zmenami, históriou a všetkým čo k tomu patrí. Vždy po ukončení editovania sa commituje. Commit sa ale vykoná len nad lokálnym repozitárom, takže len klik pravým nad adresár a *HG Commit...* Zobrazí sa zoznam zmenených vecí a hotovo.
Keď je niečo väčšie dokončené, nejaká celá časť, treba to synchronizovať s hlavným repozitárom. Najlepší spôsob je mať lokálne 2 repozitáre, jeden v ktorom sa robia zmeny, a druhý ktorý je vždy synchronizovaný s centrálnym (na BitBuckete). Vždy pred aplikovaním zmien si synchronizovať svoju kópiu centrálneho repozitára. Doňho potom pripojiť spravené zmeny, otestovať a vložiť na centrálny.
Takže napríklad kópia centrálneho (BitBucket) sa nachádza v c:/workspaces/robocupSync, klik pravým na tento folder a *TortioseHG...* -> *Synchronize*, zadať Repo https://ACCOUNT_NAME@bitbucket.org/fojtik/robocup3d-team-project/ a *Pull*. Dá sa použiť aj *Incoming* na otestovanie či sú vôbec nejaké zmeny. Potom tak isto synchronizovať s vlastnými zmenami, *TortioseHG...* -> *Synchronize*, zadať ako Repo lokálny adresár so zmenami (c:/workspaces/robocup napr.) a *Pull*, resp. *Incoming*. Ak sú nejaké konflikty treba ich vyriešiť, to je trocha zložitejšie.
Nakoniec treba otestovať mergnutý workspace a nahrať zmeny na server, *TortioseHG...* -> *Synchronize* a *Push* na https://ACCOUNT_NAME@bitbucket.org/fojtik/robocup3d-team-project/.

PRÍLOHA B: Ako si rozbehať hráča

Táto časť dokumentu obsahuje návod inštalácie hráča pre OS Windows, samotné rozbehanie pozostáva z nasledovných krokov:

Inštalácia prostredia

1. Inštalácia Microsoft Visual C++ 2008 Redistributable Package, dostupné na: http://www.microsoft.com/downloads/details.aspx?FamilyID=9b2da534-3e03-4391-8a4d-074b9f2bc1bf&displaylang=en [online október 2009]

2. Inštalácia simspark a rcssagent3d. Inštaláciu servera verzie 0.6.2 môžeme realizovať pomocou win inštalácie, alebo si inštalačky môžeme skompilovať ručne.
   a. Pre vytvorenie inštalácie ručne je potrebné ísť podľa návodu: http://simspark.sourceforge.net/wiki/index.php/Installation_on_Windows
   b. Inštalácia simspark dostupné na: http://sourceforge.net/projects/simspark/files/simspark/0.1.2/simspark-0.1.2-win32.exe/download [online október 2009]
   c. Inštalácia rcssagent3d dostupné na: http://sourceforge.net/projects/simspark/files/rcssserver3d/0.6.2/rcssserver3d-0.6.2-win32.exe/download [online október 2009]

Spustenie simulácie

1. Spustíme server a monitor:
   Ak bola zachovaná default inštalácia spustíme nasledovné baťáky:
   a. C:\Program Files\rcssserver3d 0.6.2\bin\simspark.cmd
   b. C:\Program Files\rcssserver3d 0.6.2\bin\rcssmonitor3d.cmd
2. Spustenie hráča Agenty007. Dostupné na: http://labss2.fiit.stuba.sk/TeamProject/2008/team07is-si/Produkt.rar [online október 2009]
   Hráča môžeme spustiť buď priamo cez editor (pozri manuál k editoru), alebo cez command line. Pre spustenie hráča z príkazového riadku je potrebné spustiť súbor robocup3d.exe s parametrami –file názov súboru s pohybom. Hráča (RoboCup3D.exe) si môžeme skompilovať aj ručne z workspacesu projektu Agenty007.

PRÍLOHA C: Analýza servera SimSpark

Server SimSpark je simulačné prostredie, v ktorom prebieha simulácia robotického 3D futbalu. Táto príloha vychádza z opisu servera v dokumentácií tímu Agenty 007 [23] a používateľského manuálu servera SimSpark [3].

1 Architektúra servera

Server je postavený na rámci Zeitgeist. Jeho jednotlivé súčasti sú vytvorené v rôznych jazykoch, najmä C++ a Ruby. Server podporuje nahrávanie zásuvných modulov (pluginov) v reálnom čase. Tieto zásuvné moduly musia byť napísané v jazyku Ruby. Vizualizácia scény servera je vykonávaná s využitím knižníc OpenGL a SDL knižnice. Systém je nastaviteľný aj na iné renderovacie knižnice (vďaka rozhraniu Kerosin).

Dôležitou súčasťou servera SimSpark je vrstva **Oxygen**, ktorá okrem iného obsahuje scénu (reprezentovanú grafom). Ďalej sú v tejto vrstve zapuzdrené transformácie, geometria scény a objektov v nej zahrnutých a kolízie medzi nimi. Vrstva Oxygen takisto udržuje pripojenie s agentmi – robotmi 3D futbalu. Po spustení servera je vo vrstve Oxygen vytvorená modifikovateľná cyklická slučka. Jej modifikovateľnosť spočíva v tom, že je možné pridávať zásuvné moduly, cez ktoré slučka v každom svojom behu prejde a vykoná vybrané ich funkcie.

2 Práca servera

Server SimSpark pracuje sekvenčne. V každom simulačnom cykle server zozbiera informácie z každého senzoru každého agenta. V cykle takisto vyhodnotí všetky akcie vykonané efektormi jednotlivých agentov. Server renderuje simuláciu za použitia interného alebo externého monitora (podľa nastavení v konfigurácií). SimSpark monitor dostáva od servera informácie o zmenách na scéne a renderuje ich. Formát dát sa nazýva Monitor formát, obsahuje napríklad informácie o aktuálnom móde hry alebo skóre. SimSpark monitor môže aj čítať renderované dáta zo súboru, čím sa dá prehrať záznam (replay) zaznamenanej hry. V tomto kontexte sa monitoru hovorí logplayer. Monitor sa v tomto móde spúšťa s prepínačom --logfile, ktorého argument je názov súboru obsahujúceho zaznamenanú hru. Aktuálna verzia 0.6.0 simuluje hru humanoidných robotov (na rozdiel od niektorých starších verzii, ktoré simulovali pohyb sfér).

3 Komunikácia medzi serverom a agentom

V komunikácií medzi serverom a agentom sa používajú S - výrazy (S - výraz je buď reťazec, alebo zoznam ďalších S - výrazov). S – výrazy sú používané napríklad v programovacom jazyku Lisp na uloženie kódu aj dát. Správy sú kódované v ASCII (1 znak = 1 byte). Každá správa je prefixovaná svojou dĺžkou (32 bitové bezznamienkové číslo vo formáte Big Endian - najvýznamnejšie bity sú posielané ako prvé).

4 Perceptory

Perceptory slúžia v RoboCup 3D na vnímanie okolia pre jednotlivých hráčov (agentov). Server pomocou nich posiela každému hráčovi špecifickú správu o jeho pozícií v prostredí, pomocou ktorých sa hráč môže rozhodovať. Perceptory sa môžu rozdeliť do dvoch základných skupín a to základné a špecifické pre futbal.

4.1 Základné perceptory

V tejto časti sú popísané základné perceptory, ktoré sa nachádzali na starom type serveru. Popisujú chovanie, ktoré je obvyklé v danom prostredí a nieje spojené priamo s futbalovými schopnosťami hráča.

4.1.1 GyroRate Perceptor

Perceptor GyroRate slúži na opísanie orientácie tela hráča. Údaje sa prenášajú pomocou správy, ktorá obsahuje GYR identifikátor a názov tela, ku ktorému patrí. Ďalej obsahuje tri hodnoty rotačných uhlov. Práve tieto tri uhly určujú celkovú polohu vzhľadom k súradnicovej sústave.

Formát správy pre tento perceptor vyzerá následovne:

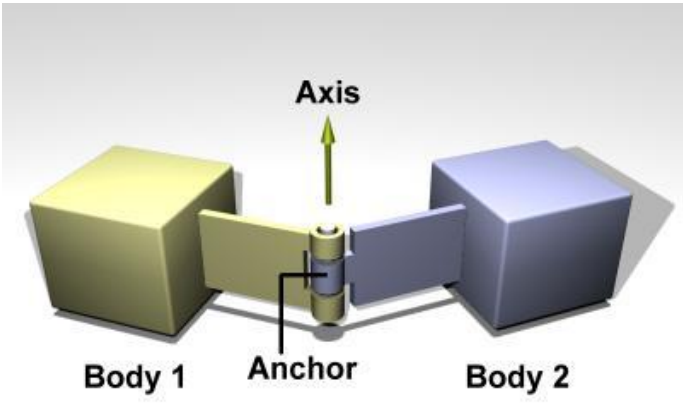
(GYR (n <name>) (rt <x> <y> <z>))

pričom <name> charakterizuje časť tela a hodnoty x, y, z hodnotu o, ktorú je daná časť posunutá.

Príklad: (GYR (n torso) (rt 0.01 0.07 0.46))

4.1.2 HingeJoint Perceptor

HingeJoint perceptor určuje, o koľko stupňov sa ohne daný kĺb robota. Kĺb zobrazený na obrázku (Obr.) je , kde je vidno práva os ax (Axis).



Obr. 1 - Ukážka jednoduchého kĺbu [3]

Formát správy pre tento perceptor vyzerá následovne:

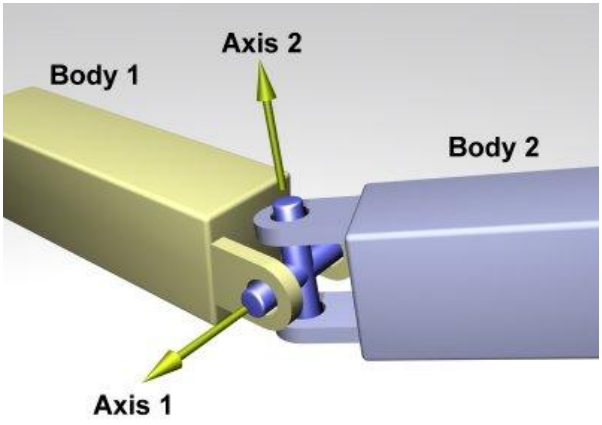
(HJ (n <name>) (ax <ax>))

pričom <name> charakterizuje názov kĺbu a hodnota ax určuje uhol, o ktorý sa daný kĺb ohol. Hodnota ax = 0 označuje, že kĺb je vystretý.

Príklad: (HJ (n laj3) (ax -1.02))

4.1.3 UniversalJoint Perceptor

UniversalJoint perceptor sa už na novom type hráča nevyskytuje. Nahradili ho dva HingeJoint perceptory, pomocou ktorých hráč ohne kĺby do dvoch smerov. Pred tým hráč mohol pohybovať kĺbom pomocou dvoch osí (Obr.), čo nový hráč už nepodporuje.



Obr. 2 - Ukážka zložitého kĺbu [3]

Formát správy pre tento perceptor vyzerá následovne:

(UJ (n <name>) (ax1 <ax1>) (ax2 <ax2>))

E

pričom <name> charakterizuje názov kĺbu a hodnoty ax1 a ax2 určujú uhol ohybu.

Príklad: (UJ (n laj1 2) (ax1 -1.32) (ax2 2.00))

4.1.4 Touch Perceptor

Tento perceptor slúži na oznámenie kolízie jednotlivých hráčov. Toto oznámenie sa vykonáva pomocou binárnych hodnôt 0 a 1. Hodnota 0 označuje, že kolízia nenastala a hodnota 1, značí kolíziu.

Formát správy pre tento perceptor vyzerá následovne:

(TCH n <name> val 0|1)

Príklad: (TCH n bumper val 1)

4.1.5 ForceResistance Perceptor

ForceResistance perceptor slúži na oznámenie pôsobenia sily a jej vektora. Súradnice c určujú bod, na ktorý sila pôsobí a súradnice f zobrazujú práve vektor tejto sily.

Formát správy pre tento perceptor vyzerá následovne:

(FRP (n <name>) (c <px> <py> <pz>) (f <fx> <fy> <fz>))

Príklad: (FRP (n lf) (c -0.14 0.08 -0.05) (f 1.12 -0.26 13.07))

4.2 Perceptory špecifické pre futbal

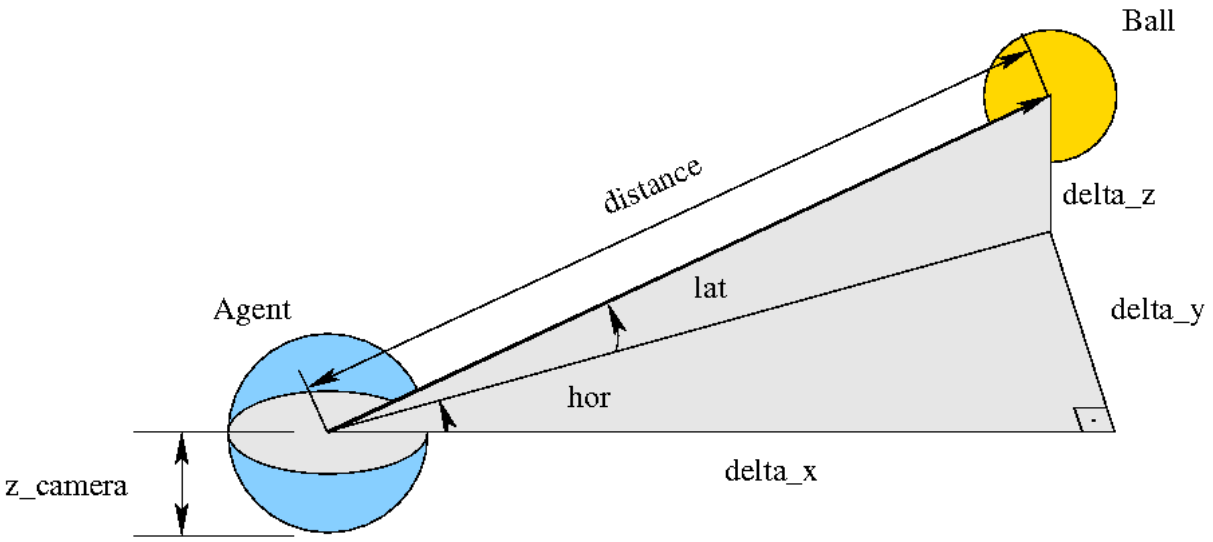
Keďže potrebujeme, aby hráč mal aj futbalové vlastnosti musí mať taktiež aj perceptory, ktoré to umožňujú. Perceptory ďalej opisujem sú spojené priamo s futbalovými schopnosťami hráča.

4.2.1 Vision Perceptor

Na to aby hráč mohol využívať svoje futbalové schopnosti je nutné aby vedel kde sa nachádza a taktiež aby videl ostatných hráčov, loptu a brány. Na to mu slúži práve perceptor Vision, ktorý zachytáva 90° uhol. Na začiatku hry je hráč natočený automaticky na súperovu stranu ihriska, ale musí vedieť aj natočiť sa na opačnú stranu. Tento perceptor doručí zoznam videných objektov, kde objekty môžu byť ostatné roboty, lopta alebo čiary na ihrisku. V súčasnosti je na ihrisku osem orientačných bodov, 4 rohy a 4 kolíky brániek.

S každým zachytením objektom sú pridružené aj:

- Vzdialenosť medzi hráčom a objektom
- Uhol v horizontálnej rovine. Nulový uhol vždy smeruje k súperovej bráne.
- Širkový uhol. Nulový uhol tu znamená horizontálne.



Obr. 3 - Aplikácia polárnych súradníc na 3D Soccer Serveri [2]

Všetky uhly a vzdialenosti sú uvádzané relatívne ku pozícií kamery. Kamera je umiestnená v strede torza robota.

Šum pozostáva z nasledovných častí:

- Malá kalibračná odchýlka je pridávaná k pozícií kamery. Pre každú os, odchýlka je rovnomerne rozdelená v intervale -0.005 a 0.005 m. Odchýlka je vypočítaná raz pre celý zápas.
- Dynamický sum je normálne distribuovaný okolo 0.0 + vzdialenostná odchýlka: sigma = 0.0965 + uhlová odchýlka (x - y): sigma = 0.1225 + uhlová odchýlka (šírková): sigma = 0.1480

Správa začína slovom See, za ktorým nasledujú objekty.

- Rohy sú zapísané ako F1L, F1R, F2L, F2R.
- Tyčky brán ako G1L, G1R, G2L, G2R.
- Lopta ako B.
- Hráči ako P s ďalšími informáciami (team <názov tímu>) (id <playerID>)

Formát správy pre tento perceptor vyzerá nasledovne:

(See (<name> (pol <distance> <angle1> <angle2>)) (P (team <teamname>) (id <playerID>) (pol <distance> <angle1> <angle2>)))

Príklad: (See (F1L (pol 19.11 111.69 -9.57)) (F2L (pol 16.41 -115.88 -11.15))

(F1R (pol 46.53 22.04 -3.92)) (F2R (pol 45.49 -18.74 -4.00)) (G1L (pol 9.88 139.29 -21.07))

(G2L (pol 8.40 -156.91 -25.00)) (G1R (pol 43.56 7.84 -4.68))

(G2R (pol 43.25 -4.10 -4.71)) (B (pol 18.34 4.66 -9.90)) (P (team RoboLog) (id 1)

(pol 37.50 16.15 -0.00)))

4.2.2 GameState Perceptor

Tento perceptor sa využíva hlavne na začiatku, keďže pomocou neho hráč zistí veľkosť ihriska a lopty. Počas hry sa však využíva taktiež, nakoľko hráčovi hovorí aký je čas zápasu a v akom stave sa hra nachádza (napr. stav pred pokutovým kopom.

Formát správy pre tento perceptor vyzerá následovne:

(GS (t <time>) (pm <playmode>))

Príklad: (GS (t 0.00) (pm BeforeKickOff))

4.2.3 AgentState Perceptor

AgentState perceptor ukazuje stav batérie a teplotu agenta. Stav batérie ukazuje v percentách a teplotu v stupňoch.

Formát správy pre tento perceptor vyzerá následovne:

(AgentState (temp <degree>) (battery <percentile>))

Príklad: (AgentState (temp 48) (battery 75))

4.2.4 Hear Perceptor

Hear perceptor ako už sám názov napovedá slúži na komunikáciu medzi hráčmi. Táto komunikácia však neprebieha priamo, ale len cez server. Hráč taktiež nemôže počuť všetko, ale len do vzdialenosti, ktorú určuje server.

Formát správy pre tento perceptor vyzerá následovne:

(hear <time> 'self'|<direction> <message>)

Príklad: (hear 12.3 self ``helloworld'')

5 Efektory

Efektory sa požívajú na všetky činnosti, ktoré chceme s našim hráčom vykonať. Pomocou nich posielame serveru správy na zmeny jednotlivých činností, ktoré následne hráč vykoná. Taktiež aj efektory sa delia do dvoch základných skupín a to základná skupina a skupina špecifická pre futbal.

5.1 Základné efektory

Tak isto ako základné perceptory aj základné efektory slúžia k určeniu základného správania, ktoré je obvyklé v danom prostredí a nieje spojené priamo s futbalovými schopnosťami hráča.

5.1.1 Create Effector

Pomocou Create efektoru sa odovzdá agentovi názov súboru, ktorý obsahuje opis hráča. Je dostupný po pripojení agenta k serveru a po ňom sa očakáva efektor SoccerInit, ktorý hráča priradí k vybranému tímu.

Formát správy pre tento efektor vyzerá následovne:

(scene <filename>)

Príklad: (scene rsg/agent/soccerbot056.rsg)

5.1.2 HingeJoint Effector

K pohybu jednotlivými kĺbmi hráča potrebujeme HingeJoint efektor, ktorý nám umožňuje zadať názov kĺbu, s ktorým chceme hýbať a uhol ohybu, o ktorý chceme daným kĺbom ohnúť.

Formát správy pre tento efektor vyzerá následovne:

(<name> <ax>)

Príklad: (lae3 5.3)

5.1.3 UniversalJoint Effector

Tak isto ako UniversalJoint perceptor aj UniversalJoint efektor je využívaný iba na starom modeli serveru a slúži na pohyb kĺbu v smere dvoch osí. Toto však nový model hráča už nepodporuje.

Formát správy pre tento efektor vyzerá nasledovne:

(<name> <ax1> <ax2>)

Príklad: (lae1 2 -2.3 1.2)

5.2 Špecifické efektory pre futbal

K ovládaniu špecifických vlastností robota slúžia práve špecifické efektory pre futbal. Tieto efektory ovládajú perceptory, ktoré sú taktiež špecifickými pre futbal.

5.2.1 Init Effector

Ako sme už spomínali Init efektor sa spúšťa zvyčajne následne po Create efektore a priradí hráča k vybranému tímu.

Formát správy pre tento efektor vyzerá následovne:

(init (unum <playernumber>)(teamname <yourteamname>))

Príklad: (init (unum 1)(teamname FHO))

5.2.2 Beam Effector

Efektor Beam musí byť zavolaný ešte pred začiatkom hry a určuje umiestnenie hráča na hraciu plochu po jeho inicializácií.

Formát správy pre tento efektor vyzerá následovne:

(beam <x> <y> <rot>)

Príklad: (beam 10.0 -10.0 0.0)

5.2.3 Say Effector

K odoslaniu správy ostatným hráčom sa využíva Say efektor. Správa sa však neposiela priamo, ale cez server. Kódovanie správy je ASCII.

Formát správy pre tento efektor vyzerá následovne:

(say <message>)

Príklad: (say ``helloworld'')

PRÍLOHA D: Príspevok na študentskú vedeckú konferenciu IIT.SRC

Decision-Making in RoboCup 3D

Zdenko CAPÍK, Peter ERTL, Michal FOJTÍK, Robert GODÁNY, Miroslav HETTEŠ, Marek HRUŠKA*

Slovak University of Technology
Faculty of Informatics and Information Technologies
Ilkovičova 3, 842 16 Bratislava, Slovakia
robokopy@googlegroups.com

Extended Abstract

The ambition of our project is to contribute to achieve the main goal of RoboCup initiative - beating world football champions in a football match. In our work we focused on designing robust, generic and extensible AI solution for a player to cover main aspects of the football game. The result of our effort is the player's decision-making system with elements of prediction which is dynamically changed according to the situation on the pitch. A part of our project is an editor allowing the developers to model simple or complex player's decision-making, but is also suitable for people without any deeper knowledge about the technical details of the RoboCup 3D simulation. Everybody who wants to play virtual football can easily create player behaviour simply by defining their own states and actions executed in those states. The architecture of our player is composed of module interpreting elementary actions (e.g. rotation, walking), the core player application interface, decision-making system - intermediate behaviour.

In our work we have adopted a project by team Agenty007 [1]. Agenty007 created a movement editor, a tool that allows modelling fundamental movements, e.g. walking, rising from the ground and kicking the ball. We decided to build our solution on the idea of separating player

---

* Master degree study programme in field: Software Engineering
Supervisor: Marián Lekavý, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

configuration from player implementation to be able to easily model, change and then interpret configured behaviour. Therefore, in our project, we continue to develop intermediate logic by adding functionality to the player and editor of movements.

An action on the lowest level is represented by motions of limbs, i.e. rotations of hinge joints. Each motion contains information about joint and its angle to rotation, start and duration time of the movement. We are currently developing the set of basic movements required to play football.

For intermediate logic support we created module that automatically processes information from the environment and calculates additional data not directly obtainable from the sensors, from simple (position, velocity, momentum, acceleration of objects) to complex (who has the ball etc.). This module provides developers of the player (us and our followers), a powerful tool that greatly simplifies the creation of a higher logic.

Player's intermediate logic is encoded by an abstract state machine, consisting of states and transitions between states. Each state consists of its name, transition condition and an action executed in this state. This action can be a fundamental move or another state machine. This way, it is possible to define hierarchical behaviours. To allow more general behaviours, the states can be parametric. Parameter binding is executed during the transition condition evaluation. Decision making consists of evaluating player's state, making a decision about which state should be the next (i.e. evaluating the transition condition), followed by carrying out actions for the new state.

The transition condition of a state is currently implemented as a decision tree (our design allows other implementations, possible approaches include rule-based systems, probabilistic models and pattern matching). Each state has its own decision tree. Each node of the tree consists of a map of values (intervals) to other nodes and the name of a function providing results to be compared with these values. Evaluation then continues in the node assigned to the interval to which the returned value belongs. The evaluation ends when it reaches a leaf of the tree, which is a reference to a new state. This state is then the result of the evaluated transition condition.

To illustrate a real-world example of the decision tree, we can imagine nodes with functions such as "ball is in direct view", "we have the ball", "agent is lying on the floor" etc. For example, in the case of the evaluation function “am I the closest player to the ball”, fuzzy logic will be applied. The result of this function would be a number in the specified range, indicating distance of the agent to the ball, considering distances of other players. After the evaluation of nodes in the decision tree, the agent executes the action stored in the leaf node. States can be more abstract, like offensive, defensive and shooting, or elementary movements such as walking, rotation and running.

Whole player's intermediate behaviour can be created using the editor, which supports definition of the states, transitions functions - decision trees and simulation of all movements.

As mentioned, we tried to create a user interface suitable not only for people in area of Robocup 3D simulation, but for everybody who wants to create custom player's behaviour.

Our approach in intermediate logic is not the only one. There are some other techniques. Most teams use hard-wheal decision trees. You can also find a fully different way, like using weighting sum that is counted from situation evaluation functions. Very interesting approaches are neural networks and also fuzzy regulation for fitness of individual actions.

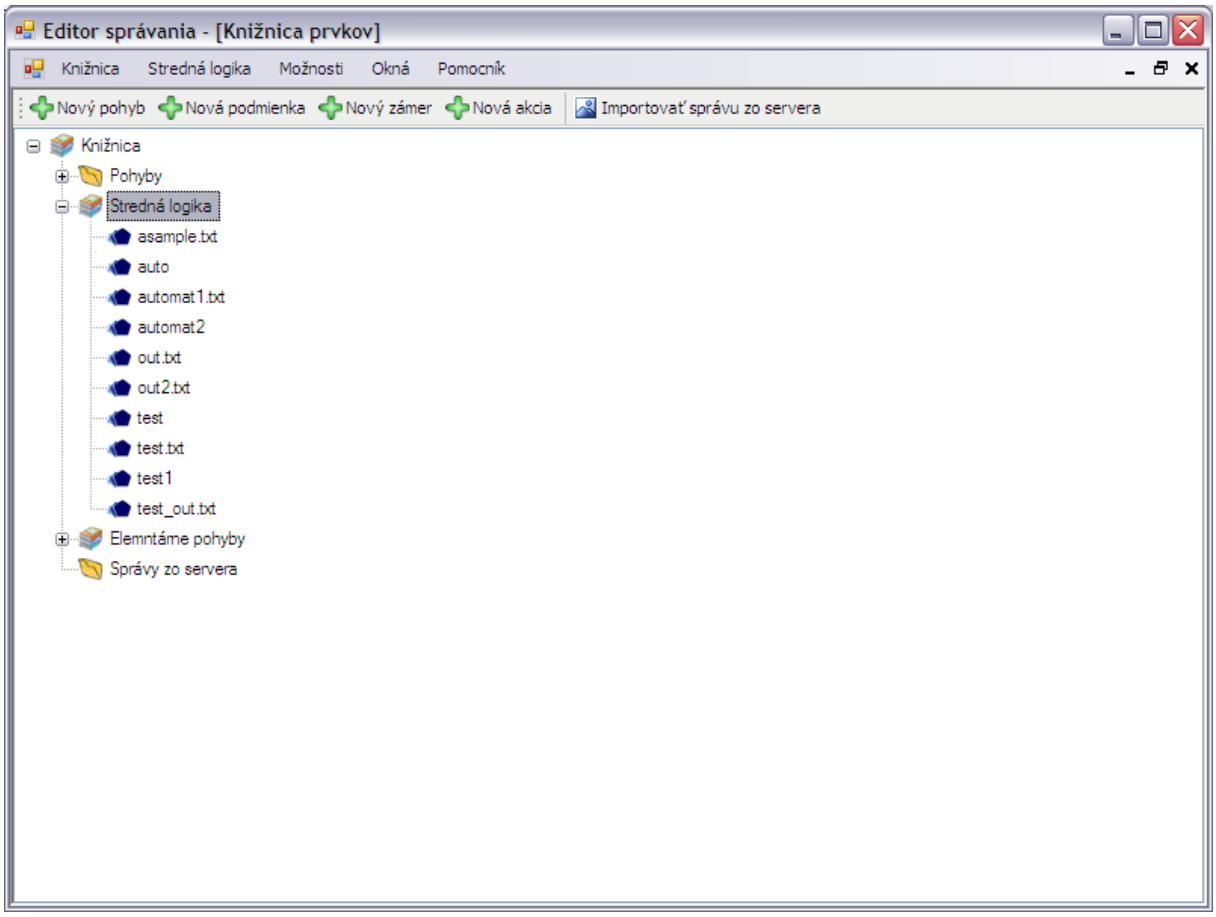
Acknowledgement: This work was partially supported by the Scientific Grant Agency of Slovak Republic, grant No. VG1/0508/09.

References

[1] Team Agenty007, [Online; accessed February 20th, 2010], Available at: http://labss2.fiit.stuba.sk/TeamProject/2008/team07is-si/

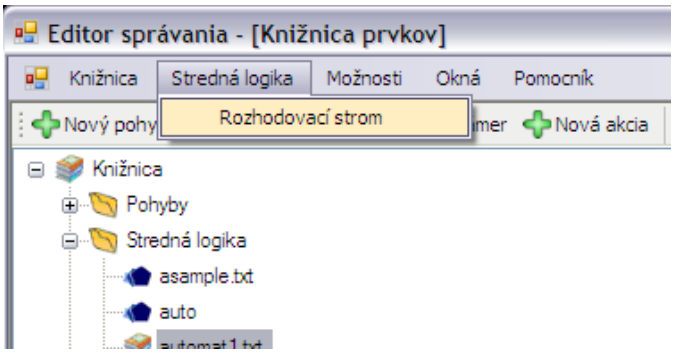
PRÍLOHA E: Používateľská príručka modulu strednej logiky do editoru pohybov

Editor strednej logiky je integrovaný do editora pohybov. Po spustení editora je dostupná možnosť vybrať si priamo niektorý z už vytvorených súborov strednej logiky. (Obrázok 1).

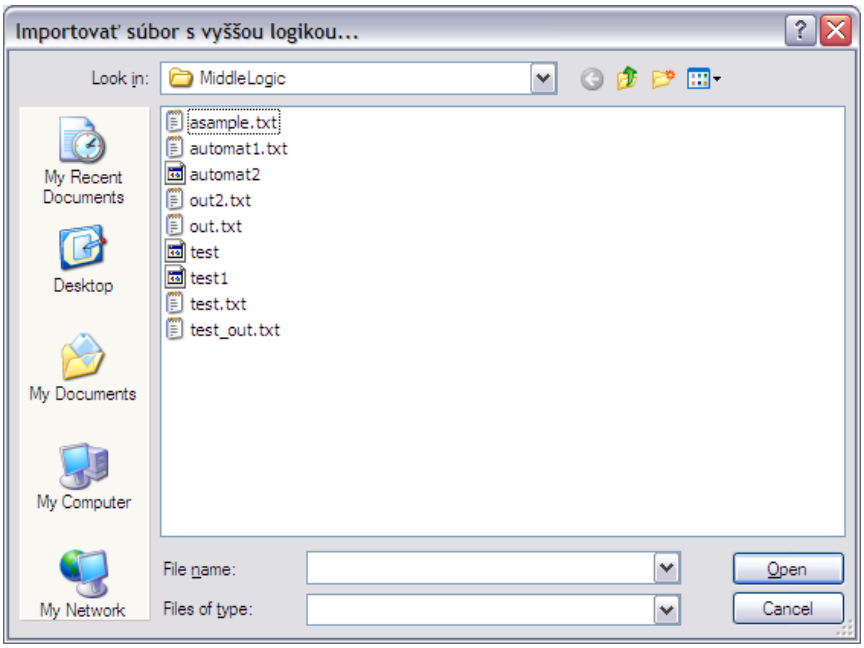


Obr. 1. : Hlavné okno editora

Druhá možnosť ako otvoriť konkrétny súbor so strednou logikou je priamo v menu zakliknúť možnosť Stredná logika ->Rozhodovací strom. (Obrázok 2) a následne zadať cestu k požadovanému súboru. (Obrázok 3)

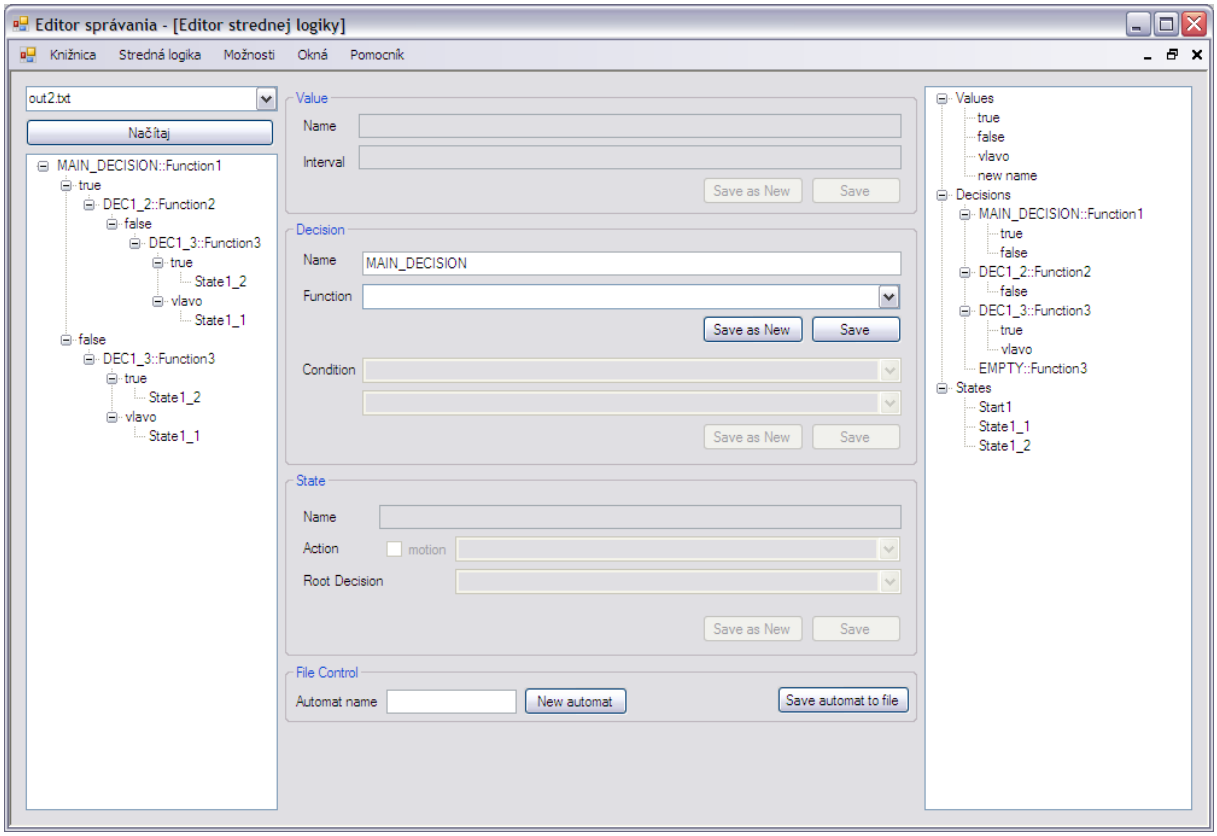


Obr. 2.: Prístup k súboru strednej logiky, ktorý nie je priamo v knižnici



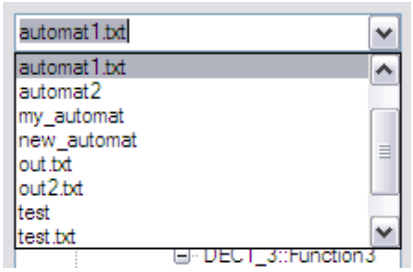
Obr. 3.: Výber súboru strednej logiky, ktorý chceme editovať

Po výbere súboru vidíme samotný editor strednej logiky. (Obrázok 4)

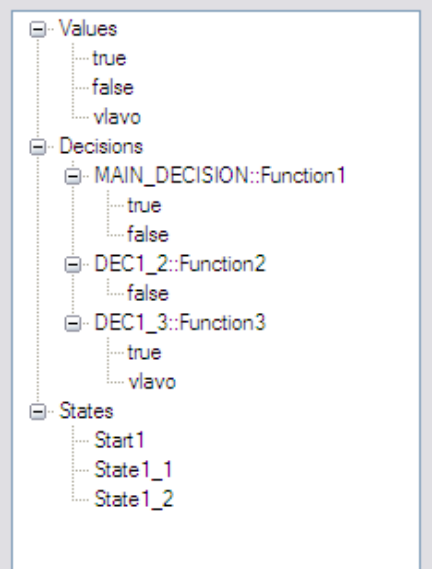


Obr. 4.: Editor strednej logiky

V ľavej časti obrazovky máme z hora zoznam dostupných automaty v zvolenom priečinku (Obrázok 5). Tlačítko na načítanie vybraného súboru. A najdôležitejšiu sekciu – rozhodovací strom. Strednú časť okna tvoria formuláre, ktoré slúžia na editovanie a pridávanie nových častí rozhodovacieho stromu. Zoznam všetkých dostupných komponentov máme uložený v pravej časti okna (Obrázok 6). Uložiť vykonaná zmeny môžeme pomocou tlačítka „Save automat to file“.



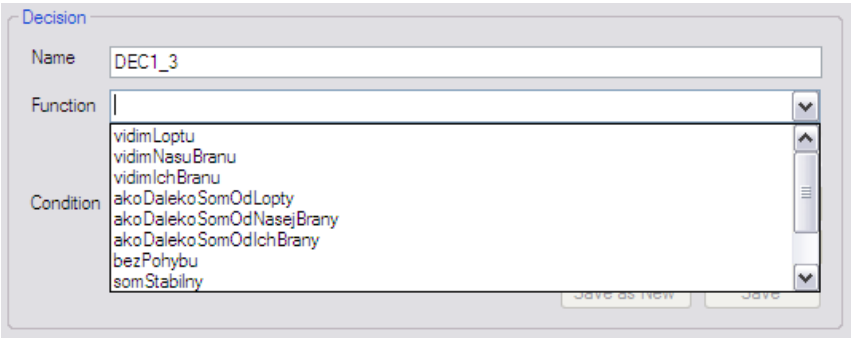
Obr. 5.: Výber automatov, dostupných v aktuálnom priečinku



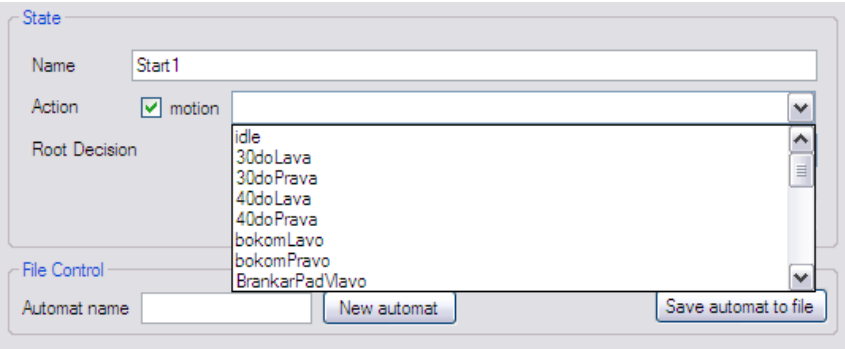
Obr. 6.: Výber automatov, dostupných v aktuálnom priečinku

Ak chceme editovať niektorú vetvu stromu jednoducho klikneme na ňu a v strednej časti obrazovky sa nám naplnia hodnoty buď pre stav (state), rozhodovací uzol (decision), alebo podmienku v rozhodovacom uzle (conditions).

Pri nastavovaní rozhodovacej funkcie nám pomáha zoznam všetkých dostupných funkcií. (Obrázok 7). Podobne je tomu pri výbere pohybu v stave (Obrázok 8)

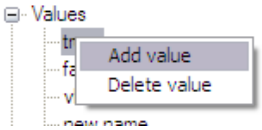


Obr. 7.: Zoznam dostupných rozhodovacích funkcií

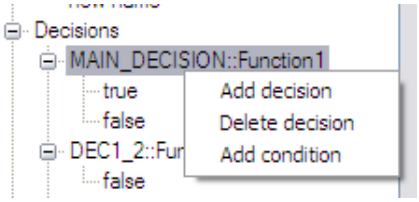


Obr. 8.: Zoznam dostupných pohybov

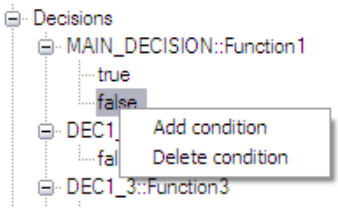
Pridávať nové komponenty je možne v pravej časti okna, potom ako klikneme pravým tlačítkom myši. Môžeme pridávať Values (Obrázok 9), Decision (Obrázok 10), Conditions (Obrázok 11), a State (Obrázok 12). Rovnakým spôsobom je možné vykonať mazanie komponentov. Zmazať je ale možné len tie komponenty, ktoré sa nevyskytujú v iných častiach rozhodovacieho stromu.



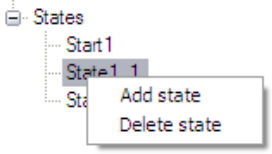
Obr. 9.: Pridávanie Value / mazanie Value



Obr. 10. Pridávanie Decision / mazanie Decision



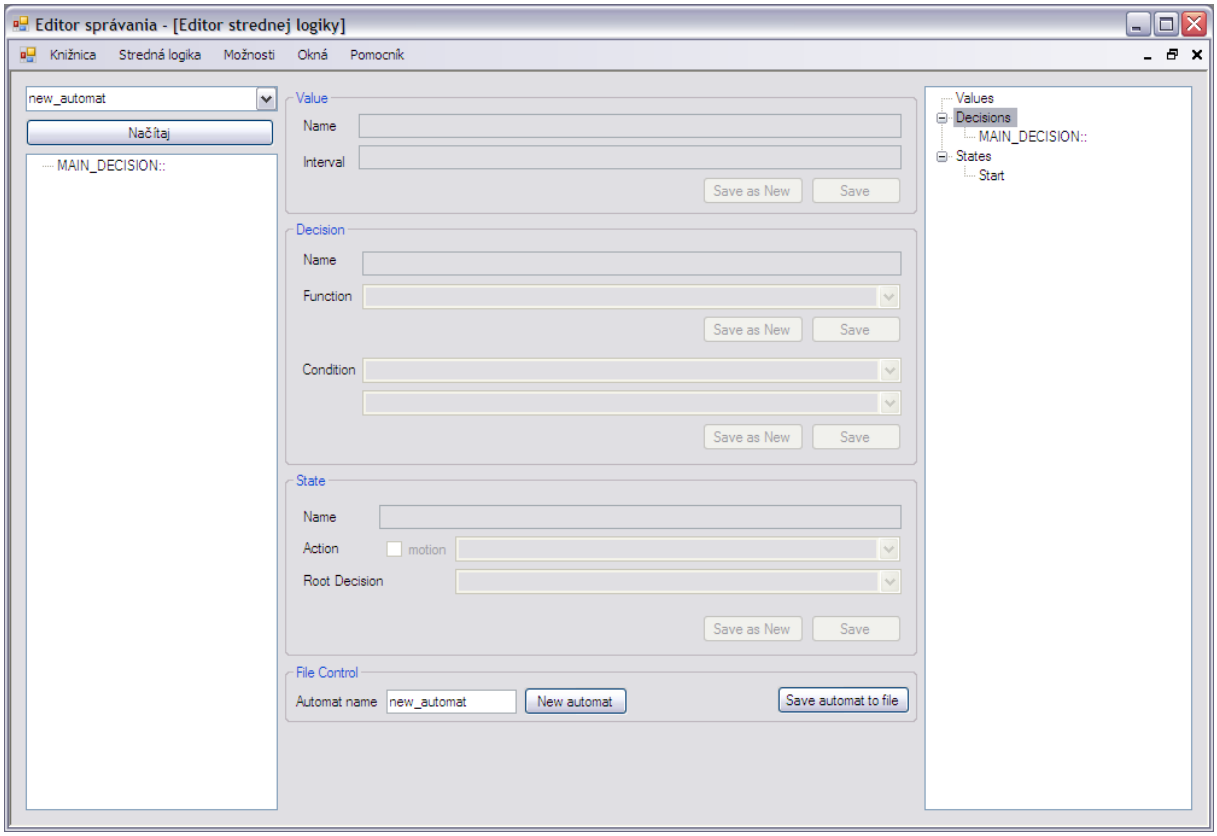
Obr. 11.: Pridávanie Condition / mazanie Condition



R

Obr. 12.: Pridávanie State / mazanie State

Ak chcem vytvoriť nový automat, zadáme meno súboru, a klikne na tlačítko New automat. Po vykonaní tejto akcie sa nám v editore zobrazí načítaný nový súbor so strednou logikou. (Obrázok 13)



Obr. 13.: Novo vytvorený súbor so strednou logikou

PRÍLOHA F: Externé testovanie tímom Critical Error

6 Testovanie

6.1 Pohyby

Boli nám dodané nasledovné pohyby.

- Otáčanie
- Chôdza
- Vstávanie z chrbta
- Vstávanie z brucha
- Kop do lopty

Testovanie prebiehalo vykonaním pohybu viac krát po sebe na tom istom počítači. Hardvérová konfigurácia počítača je nasledovná:

Procesor: 2,66 GHz, Intel Core Duo 2

Operačná pamäť: 4 Gb

Grafická karta: NVidia GeForce 9600 GT

6.1.1 Testovacie tabuľky pre jednotlivé pohyby

6.1.1.1 *Otočenie*

Pokus č.	Pohyb	Cieľ				Vyhodnotenie
1	Otočenie doľava	Otočenie stupňov	hráča	o približne	+30	Neúspešne
2	Otočenie doľava	Otočenie stupňov	hráča	o približne	+30	Úspešne
3	Otočenie doľava	Otočenie stupňov	hráča	o približne	+30	Úspešne
4	Otočenie doľava	Otočenie stupňov	hráča	o približne	+30	Neúspešne
5	Otočenie doľava	Otočenie stupňov	hráča	o približne	+30	Úspešne
6	Otočenie doprava	Otočenie	hráča	o približne	-30	Úspešne

		stupňov				
7	Otočenie doprava	Otočenie stupňov	hráča	o približne	-30	Úspešne
8	Otočenie doprava	Otočenie stupňov	hráča	o približne	-30	Neúspešne
9	Otočenie doprava	Otočenie stupňov	hráča	o približne	-30	Úspešne
10	Otočenie doprava	Otočenie stupňov	hráča	o približne	-30	Úspešne

Tabuľka 1 – Testovanie otočenia

6.1.1.2 *Chôdza*

Pokus č.	Pohyb	Cieľ	Vyhodnotenie
1	Chôdza	Hráč má prejsť dlhší úsek	Neúspešne Hráč spadol pri 3. kroku
2	Chôdza	Hráč má prejsť dlhší úsek	Úspešne
3	Chôdza	Hráč má prejsť dlhší úsek	Úspešne
4	Chôdza	Hráč má prejsť dlhší úsek	Neúspešne Hráč spadol pri 7. kroku
5	Chôdza	Hráč má prejsť dlhší úsek	Úspešne
6	Chôdza	Hráč má prejsť dlhší úsek	Neúspešne Hráč spadol pri 3. kroku
7	Chôdza	Hráč má prejsť dlhší úsek	Neúspešne Hráč spadol pri 4. kroku
8	Chôdza	Hráč má prejsť dlhší úsek	Úspešne
9	Chôdza	Hráč má prejsť dlhší úsek	Úspešne
10	Chôdza	Hráč má prejsť dlhší úsek	Neúspešne Hráč spadol pri 3. kroku

Tabuľka 2 – Testovanie chôdze

6.1.1.3 *Vstávanie z chrbta*

Pokus č.	Pohyb	Cieľ	Vyhodnotenie
1	Vstávanie z chrbta	Hráč sa má spadnúť na chrbát, následne sa postaviť	Úspešne
2	Vstávanie z chrbta	Hráč sa má spadnúť na chrbát, následne sa postaviť	Úspešne
3	Vstávanie z chrbta	Hráč sa má spadnúť na chrbát, následne sa postaviť	Úspešne
4	Vstávanie z chrbta	Hráč sa má spadnúť na chrbát, následne sa postaviť	Úspešne
5	Vstávanie z chrbta	Hráč sa má spadnúť na chrbát, následne sa postaviť	Úspešne
6	Vstávanie z chrbta	Hráč sa má spadnúť na chrbát, následne sa postaviť	Úspešne
7	Vstávanie z chrbta	Hráč sa má spadnúť na chrbát, následne sa postaviť	Úspešne

8	Vstávanie z chrbta	Hráč sa má spadnúť na chrbát, následne sa postaviť	Úspešne
9	Vstávanie z chrbta	Hráč sa má spadnúť na chrbát, následne sa postaviť	Neúspešne Hráč sa nepretočil na brucho
10	Vstávanie z chrbta	Hráč sa má spadnúť na chrbát, následne sa postaviť	Úspešne

Tabuľka 3 – Testovanie vstávania z chrbta

6.1.1.4 *Vstávanie z brucha*

Pokus č.	Pohyb	Cieľ	Vyhodnotenie
1	Vstávanie z brucha	Hráč sa má spadnúť na brucho a následne sa postaviť	Úspešne
2	Vstávanie z brucha	Hráč sa má spadnúť na brucho a následne sa postaviť	Úspešne
3	Vstávanie z brucha	Hráč sa má spadnúť na brucho a následne sa postaviť	Úspešne
4	Vstávanie z brucha	Hráč sa má spadnúť na brucho a následne sa postaviť	Úspešne
5	Vstávanie z brucha	Hráč sa má spadnúť na brucho a následne sa postaviť	Úspešne
6	Vstávanie z brucha	Hráč sa má spadnúť na brucho a následne sa postaviť	Úspešne
7	Vstávanie z brucha	Hráč sa má spadnúť na brucho a následne sa postaviť	Neúspešne Hráč sa nepostavil
8	Vstávanie z brucha	Hráč sa má spadnúť na brucho a následne sa postaviť	Úspešne
9	Vstávanie z brucha	Hráč sa má spadnúť na brucho a následne sa postaviť	Úspešne
10	Vstávanie z brucha	Hráč sa má spadnúť na brucho a následne sa postaviť	Úspešne

Tabuľka 4 – Testovanie vstávania z brucha

6.1.1.5 *Kop do lopty*

Pokus č.	Pohyb	Cieľ	Vyhodnotenie
1	Kop	Hráč sa nastaví k lopte a kopne do nej	Úspešne
2	Kop	Hráč sa nastaví k lopte a kopne do nej	Neúspešne Hráč viac menej minul loptu
3	Kop	Hráč sa nastaví k lopte a kopne do nej	Úspešne
4	Kop	Hráč sa nastaví k lopte a kopne do nej	Čiastočne neúspešne Hráč po vykonaní kopu spadol
5	Kop	Hráč sa nastaví k lopte a kopne do nej	Úspešne
6	Kop	Hráč sa nastaví k lopte a kopne do	Čiastočne neúspešne

		nej	Hráč po vykonaní kopu spadol
7	Kop	Hráč sa nastaví k lopte a kopne do nej	Úspešne
8	Kop	Hráč sa nastaví k lopte a kopne do nej	Úspešne
9	Kop	Hráč sa nastaví k lopte a kopne do nej	Neúspešne Hráč viac menej minul loptu
10	Kop	Hráč sa nastaví k lopte a kopne do nej	Úspešne

Tabuľka 5 – Testovanie kopu

6.1.2 Vyhodnotenie

Každý z dodaných pohybov bol funkčný. Jediné problémy boli s chôdzou. Pri jej testovaní totiž hráč relatívne často padal na zem. Po páde sa však dokázal postaviť a pokračovať vo chôdzi ďalej.

6.2 Agent

Do agenta bol podľa dokumentácie zapracovaný modul strednej logiky. Agenta sme testovali na dvoch dodaných súboroch strednej logiky.

Prvý obsahoval dlhú chôdzu. Ak počas nej hráč spadol na chrbát, na brucho alebo na bok, vedel tento stav detegovať a následne na neho zareagovať (postaviť sa). Testovanie dlhej chôdze prebiehalo tak, že sme hráča nechali vykonávať chôdzu pokiaľ sa nedostal k stredovej čiare. Ak sa k nej dostal, bol test vyhodnotený ako úspešný, bez ohľadu na to koľko razy počas jej vykonávania spadol. Ak sa mu to nepodarilo, test bol vyhodnotený ako neúspešný.

Druhý súbor so strednou logikou zabezpečoval presun hráča k lopte. Test bol vyhodnotený ako úspešný ak sa hráč dostal k lopte, bez ohľadu nato koľko razy počas vykonávania tejto akcie spadol.

6.2.1 Testovacie tabuľky pre modul strednej logiky

6.2.1.1 *Dlhá chôdza*

Pokus č.	Pohyb	Cieľ	Vyhodnotenie
1	Dlhá chôdza	Hráč vykonáva chôdzu, ak spadne postaví sa a pokračuje ďalej	Úspešne
2	Dlhá chôdza	Hráč vykonáva chôdzu, ak spadne postaví sa a pokračuje ďalej	Neúspešne Hráč sa nepostavil po 2. páde
3	Dlhá chôdza	Hráč vykonáva chôdzu, ak spadne postaví sa a pokračuje ďalej	Neúspešne Hráč sa nepostavil po 3. páde otočil na opačnú stranu
4	Dlhá chôdza	Hráč vykonáva chôdzu, ak spadne postaví sa a pokračuje ďalej	Úspešne
5	Dlhá chôdza	Hráč vykonáva chôdzu, ak spadne postaví sa a pokračuje ďalej	Úspešne
6	Dlhá chôdza	Hráč vykonáva chôdzu, ak spadne postaví sa a pokračuje ďalej	Neúspešne Hráč sa nepostavil po 2. páde
7	Dlhá chôdza	Hráč vykonáva chôdzu, ak spadne postaví sa	Úspešne

		a pokračuje ďalej	Hráč kráčal 10s bez pádu
8	Dlhá chôdza	Hráč vykonáva chôdzu, ak spadne postaví sa a pokračuje ďalej	Úspešne
9	Dlhá chôdza	Hráč vykonáva chôdzu, ak spadne postaví sa a pokračuje ďalej	Neúspešne Hráč sa nepostavil po 4. páde
10	Dlhá chôdza	Hráč vykonáva chôdzu, ak spadne postaví sa a pokračuje ďalej	Úspešne

Tabuľka 6 – Testovanie dlhej chôdze

6.2.1.2 *Presun hráča k lopte*

Pokus č.	Pohyb	Cieľ	Vyhodnotenie
1	Presun k lopte	Hráč sa má presunúť k lopte, ak spadne postaví sa a pokračuje ďalej	Úspešne
2	Presun k lopte	Hráč sa má presunúť k lopte, ak spadne postaví sa a pokračuje ďalej	Úspešne
3	Presun k lopte	Hráč sa má presunúť k lopte, ak spadne postaví sa a pokračuje ďalej	Neúspešne Hráč sa nepostavil po 3. páde
4	Presun k lopte	Hráč sa má presunúť k lopte, ak spadne postaví sa a pokračuje ďalej	Čiastočne úspešne Hráč spadol na loptu, ale nevstal
5	Presun k lopte	Hráč sa má presunúť k lopte, ak spadne postaví sa a pokračuje ďalej	Úspešne
6	Presun k lopte	Hráč sa má presunúť k lopte, ak spadne postaví sa a pokračuje ďalej	Neúspešne Hráč sa nepostavil po 4. páde
7	Presun k lopte	Hráč sa má presunúť k lopte, ak spadne postaví sa a pokračuje ďalej	Úspešne
8	Presun k lopte	Hráč sa má presunúť k lopte, ak spadne postaví sa a pokračuje ďalej	Neúspešne Hráč sa nepostavil po 4. páde
9	Presun k lopte	Hráč sa má presunúť k lopte, ak spadne postaví sa a pokračuje ďalej	Úspešne
10	Presun k lopte	Hráč sa má presunúť k lopte, ak spadne postaví sa a pokračuje ďalej	Úspešne

Tabuľka 7 – Testovanie presunu hráča k lopte

6.2.2 Vyhodnotenie

Pri oboch testovacích súboroch sme zaznamenali problémy pri vstávaní po viacnásobnom páde. Samotný pohyb zabezpečujúci chôdzu nie je dobre odladený, čo sa prejavilo už pri testovaní pohybov. Vyhodnocovanie a samotná stredná logika však fungovala veľmi dobre a účinne. Škoda, že niektoré pohyby neboli implementované na lepšej úrovni, zvyšovalo by to pozitívny dojem zo strednej logiky.

6.3 Editor

Pri používaní editora sme využili používateľskú príručku z dokumentácie. Vytvorené súbory v editore sme netestovali v agentovi. Testovali sme všetky popisované funkcie z dokumentácie. Boli to:

- Vytvorenie nového súboru so strednou logikou
- Načítanie existujúceho súboru

- Uloženie načítaného súboru
- Pridanie, Editovanie, Zmazanie Values
- Pridanie, Editovanie, Zmazanie Decision
- Pridanie, Editovanie, Zmazanie Condition k decision
- Pridanie, Editovanie, Zmazanie State

6.3.1 Vyhodnotenie

Vykonanie všetkých akcií prebehlo bez problémov. Editor je teda dostatočne odladený. Čo by sa dalo editoru vytknúť je samotná práca s ním. Ta totiž nie je vždy dostatočne intuitívna. Pri vytváraní stromovej štruktúry stromu by bolo vhodná použiť Drag&Drop.