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

RoboCup – tretí rozmer
(Dokumentácia k projektu)

Tím č. 17: Neurotics
Členovia tímu: Aleš Katona, Gabriel Braniša, Peter Čimo,
Tomáš Labuda, Juraj Šimon, Ján Kolesár
Študijný program: IS/SI
Ročník prvý, inžinierske štúdium
Email: team17.robocup@gmail.com
Web: http://www2.dcs.elf.stuba.sk/TeamProject/2007/team17is-si/

Vedúci projektu: Ing. Marián Lekavý
Ak. rok: 2007/2008

Obsah

Zoznam obrázkov

1. Úvod

1.1 Účel dokumentu

Tento dokument bol vytvorený v rámci predmetu Tímový projekt v akademickom roku 2007-2008. Venuje sa virtuálnej simulácii 3D robotického futbalu pod názvom RoboCup 3D. Dokument je členený na jednotlivé kapitoly nasledovne. V prvej kapitole si určujeme cieľ nášho projektu, opisujeme zadanie a ponúkame stručný úvod do prostredia RoboCup-u. Druhou kapitolou je analýza. V nej sa najprv pozrieme na kategórie do ktorých sa delí simulačná liga RoboCup-u. Ďalej si povieme niečo o pravidlách a samotnom simulačnom prostredí. Nakoniec sa pozrieme na niekoľko tímov resp. hráčov robotického futbalu a podrobnejšie si ich opíšeme. Tretia kapitola predstavuje našu špecifikáciu požiadaviek. V štvrtej kapitole opisujeme náš hrubý návrh humanoidného hráča. Využijeme pri ňom znalosti z analýzy z kapitoly tri a prispievame svojou vlastnou ideou ako vylepšiť pohyby humanoida na základe evolučného učenia sa. Piata kapitola obsahuje podrobný návrh predovšetkým evolučných algoritmov, v šiestej je opísaný náš prototyp.

1.2 Cieľ projektu

Cieľom projektu je vytvoriť humanoidného hráča so základnými orientačnými a pohybovými schopnosťami (chôdza, státie, postavenie sa). Ako ideologický základ je vhodné využiť prácu a poznatky minuloročného týmu. Nový humanoidný hráč bude pracovať na novej verzií serveru vo verzií 0.5.6. Je potrebná vyhľadať iných humanoidných hráčov, jedného z nich si vybrať a na jeho základe zostaviť hráča vlastného. Implementovať mu základné schopnosti a poskytnúť tak určitý základ pre prácu tímov v budúcich rokoch. Hráč má byť modulárny a schopný rozšírenia v budúcich rokoch.

1.3 Zadanie

Téme RoboCup, presnejšie lige simulovaného robotického futbalu sa naši študenti venujú už osem rokov. Tímy študentov, či už v rámci umelej inteligencie alebo tímového projektu, sa snažia vytvárať a vylepšovať programy, ktoré simulujú správanie sa futbalového hráča. Každý tím sa v rámci obmedzení, určených pravidlami hry futbal a špecifikami simulačného prostredia, snaží vytvoriť čo najlepšieho hráča. Mužstvo, vytvorené z takýchto hráčov, by malo vyhrať nad mužstvom súpera. O súťaži a doterajšej činnosti je dosť popísané aj na stránke STU [20].

Simulácia futbalu pôvodne prebiehala iba v dvoch rozmeroch. Pre zvýšenie reálnosti simulácie bolo vytvorené 3D simulačné prostredie, ktoré rozširuje možnosti hry. 3D simulačné prostredie sa pomerne výrazne líši od doposiaľ používaného 2D prostredia, a to jednak spôsobom simulácie, ale hlavne možnosťami ktoré poskytuje hráčom.

Hlavným cieľom projektu bude vytvoriť hráča pre 3D simuláciu, ktorý dokáže plnohodnotne využívať možnosti poskytované simulačným prostredím. Úlohou teda bude prevziať jednoduchého hráča vytvoreného na našej fakulte v minulom roku a doplniť do neho komplexnejšie typy správania a rozhodovania. Keďže sa predpokladá ďalšie rozširovanie hráča v ďalších rokoch, dôležitými požiadavkami sú prehľadnosť a ďalšia rozširovateľnosť, a to na úrovni návrhu aj implementácie. Dôraz pri vytváraní hráča by mal byť kladený najmä na dobre prepracované a

odladené nižšie schopnosti hráča, ktoré umožnia hráčovi spracovávať vnemy z prostredia a efektívne konať v prostredí (pohybovať sa, pracovať s loptou). Pri návrhu a implementácii bude tiež možné (a žiadané) čerpať z veľkého množstva prístupov existujúcich v 2D simulácii.

Zimný semester je vyhradený na oboznámenie sa s celým simulačným prostredím a hráčom ktorý sa bude rozširovať, a takisto s existujúcimi hráčmi (2D aj 3D), ďalej návrhu a prototypovej realizácii hráča. Dôležitou súčasťou bude vytvorenie plánu implementácie a overovania prístupu v nasledovnom semestri. V letnom semestri nás čaká dokončenie realizácie návrhu a jeho overovanie. Nemenej podstatnou časťou projektu bude vytvorenie dokumentácie, ktorá poskytne tímom v ďalších rokoch odrazový mostík pri použití vytvoreného hráča.

1.4 RoboCup

RoboCup je medzinárodná súťaž autonómnych robotov softvérových agentov. Súťaž je určená na podporu výskumu a výučby umelej inteligencie, robotiky a ďalších príbuzných oblastí. *Robot World Cup Soccer Games and Conferences*, ide o neziskovú organizáciu, ktorá každoročne usporadúva turnaje v robotickom futbale. Snahou tejto spoločnosti je podporovať vývoj rôznych technologických oblastí pomocou riešenia štandardnou metódou, pri ktorej je možné využiť obrovské množstvo technológií. Na tento účel programátori zvolili futbal. Základnou ideou je vytvoriť čo najviac dokonalých fyzických, ale i syntetických, čiže programovo realizovaných agentov, ktorí dokážu hrať futbal na vysokej úrovni. Míľnikom, o ktorý RoboCup-u ide, je zostrojenie tímu humanoidných robotov, ktorí by hrali proti najlepšiemu "ľudskému" tímu podľa oficiálnych pravidiel FIFA. Predpovede organizácie RoboCup hovoria o uskutočnení tohto zápasu okolo polovice tohto storočia.

RoboCup Federation je medzinárodná nezisková organizácia zaregistrovaná v Ženeve za účelom podpory, propagácie a posilňovania výskumu RoboCup-u. Túto iniciatívu v súčasnosti nasleduje približne 1500 výskumníkov v 17 rôznych krajinách. Do súťaže sa môže zapojiť každý tím (resp. jednotlivec), bez ohľadu na to, či je alebo nie je zameraný na samotný výskum. Niektoré tímy sa zúčastňujú súťaže iba zo zábavy (najmä v simulačnej lige), iné sú zamerané na seriózny výskum, pričom svoje výsledky pravidelne publikujú.

Existuje liga simulovaného futbalu pre 2D, kde sú agenti v tvare kruhu v počte jedenásť na hracej ploche. V 3D existujú dva typy simulovaného futbalu a to reprezentácia agenta guličkou, čo je pokračovanie, nadstavba 2D simulovaného futbalu. Tento typ postupne zaniká a nahradila ho simulácia 3D humanoidov v počte troch hráčov na jeden tím. Simulačné prostredie disponuje realistickým simulovaním fyzikálnych zákonov pomocou ODE knižnice.

2. Analýza

V nasledujúcej kapitole si postupne predstavíme najprv základné kategórie súťaže RoboCup a ich pravidlá. Ďalej sa oboznámime so samotným simulačným prostredím a modelom nového humanoidného robota. Následne si predstavíme niekoľko tímov pôsobiacich v oblasti simulovaného robotického futbalu.

2.1 Pravidlá RoboCup-u

Bohužiaľ na oficiálnych stránkach RoboCup-u [1] sme nenašli žiadne užitočné a aktuálne informácie. Kategória 3D simulovaného futbalu sa od roku 2005 kedy boli tieto informácie naposledy aktualizované, dosť zmenila (predovšetkým zmena modelu hráča). Na stránke ligy sa však nachádzajú aktuálnejšie informácie, konkrétne pravidlá z turnaju RoboCup 2007 v Atlante [3]. V Atlante sa použil server rcssserver3d-0.5.6 spustený pod operačným systémom SuSE Linux 10.1. Server a monitor bežali na samostatných počítačoch, a každý tím mal k dispozícii jeden počítač pre svojich hráčov.

2.1.1 Kategória Soccerbot

Jedná sa o kategóriu v ktorej proti sebe súťažia dva tímy hráčov.
Tímy môžu použiť jedného z dvoch typov hráča:

1. *soccerbot055.rsg* – Je to verzia bota, ktorá prišla so serverom rcssserver3d-0.5.5.
   - jednodielny trup
   - žiadne obmedzenia kĺbov – môžu sa otáčať aj do neprirodzených polôh
   - TCH senzor – poskytuje len informáciu o tom že sa niečo niečoho dotýka
   - nemá detekciu kolízií rúk
2. *soccerbot056.rsg* – Tento bot je súčasťou servera rcssserver3d-0.5.6.
   - dvojdielny trup
   - obmedzenia kĺbov – nemôžu sa otáčať do neprirodzených polôh
   - FRP senzor – poskytuje informáciu o sile, ktorá pôsobí pri dotyku a o mieste jej pôsobenia.
   - detekcia kolízií rúk

Pozn.: Obidvaja boti majú rovnakú hmotnosť.

Štruktúra zápasov je nasledovná:

- Jeden tým je zložený z 3 hráčov (v pravidlách nie je napísané či je presne určený brankár, ale pravdepodobne to bude hráč s číslom jeden).
- Hra je rozdelená na dva polčasy. Každý trvá 7,5 min.
- Počas súťaže tímy môžu zmeniť alebo nahradiť svoje zdrojové kódy. Tieto zmeny budú však vykonané na vlastné riziko. Zmena je možná len medzi jednotlivými zápasmi, nie počas hry.
- Ak hráč nebude fungovať organizátori ho nebudú rozbehávať. (Tímy totiž nemusia prísť na súťaž osobne, ale môžu svojich hráčov poslať.)

Na hru dohliada ľudský rozhodca, pričom niektoré fauly sú automaticky detekované simulátorom, no ten nemôže detekovať všetky. Rozhodca má právo udeliť voľný kop poškodenému tímu. Za faul sa pokladá napríklad (no nie len):

- Ak jeden z hráčov úmyselne chytí, alebo sa dotkne lopty rukou, okrem brankára.
- Ak niektorý hráč leží na lopte viac ako 10 sekúnd.
- Ak hráč úmyselne udrie alebo postrčí súperovho hráča.
- Ak jeden tým obkľúči loptu tak, že druhý tým nemá na loptu dosah.
- Ak hráči blokujú bránku tak že sa lopta nemôže dostať dnu (stena z hráčov v bránke).
- Ak tým úmyselne blokuje pohyb hráčov.
- Čokoľvek, čo sa javí ako porušenie fair play, môže byť tiež považované za faul. Záleží na posúdení rozhodcu.

Cieľom hry je hrať futbal podľa obvyklého vnímania fér hry a obmedzení daných virtuálnym simulovaným svetom 3D simulátora. Obchádzanie týchto pravidiel je považované za porušenie povinnosti hrať fér hru a počas turnajových zápasov je to prísne zakázané.

Za porušenie fair play sa považuje napríklad:

- Používanie hráčov iného tímu.
- Zahlcovanie servera posielaním neúmerného počtu správ od klienta.
- Priama komunikácia hráčov použitím iných komunikačných spôsobov, ako napríklad medzi procesová komunikácia.
- Práca so súťažnými počítačmi alebo ich úmyselné reštartovanie.

Čokoľvek z uvedeného je prísne zakázané. Iné stratégie môžu byť označené za porušenie fair play, po konzultácií s komisiou zaoberajúcou sa pravidlami. Obzvlášť deštruktívne narušenie operácií súperových agentov, alebo získanie výhody spôsobom iným, ako explicitne ponúkaným simulátorom, je považovaný za porušenie fair play. Ak si účastníci súťaže nie sú istý ohľadne použitia niektorej metódy, konzultujú to s komisiou zaoberajúcou sa pravidlami pred začiatkom turnaja. Ak sa počas turnaja zistí, že niektorý tým používa ne fér praktiky, bude tento tým okamžite vylúčený z turnaja.

V pravidlách sa nepíše nič o koučovi alebo trénerovi. Predpokladáme teda že ešte nie je implementovaný v servery a teda ani povolený (použiteľný).

2.1.2 Kategória 3D development

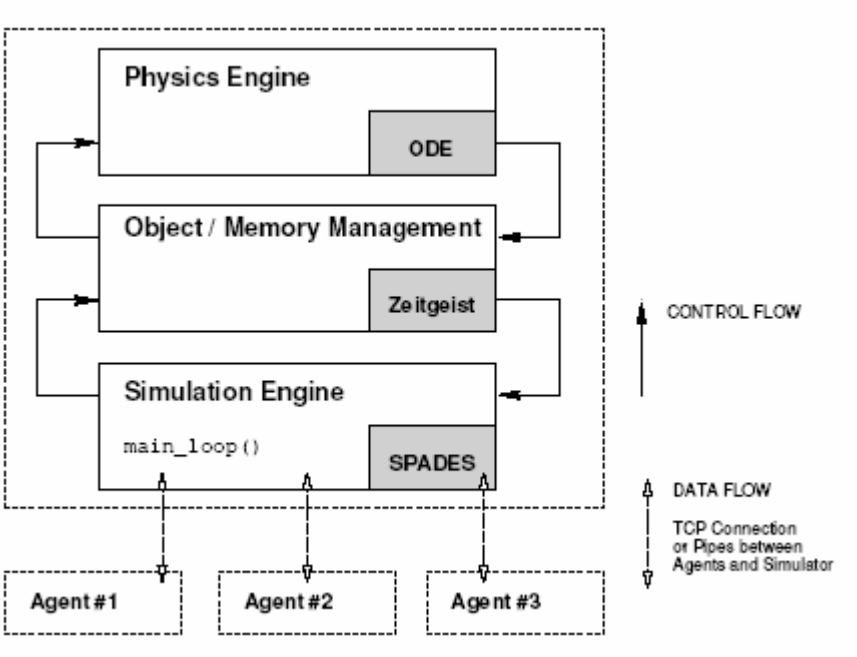
Kategória 3D development je zameraná na urýchlenie vývoja nízko úrovňových schopností hráča, aby tými mohli čo najrýchlejšie prejsť na vývoj vyšších schopností hráčov. Presné pravidlá ešte nie sú definované v použiteľnom rozsahu, ale na oficiálnej stránke ligy [2] sú spomenuté niektoré súťaže, z tejto kategórie:
Technické súťaže - môžu obsahovať kráčanie, kopanie, vstávanie, penalty, ...
Futbalové zápasy 5-vs-5 - Keďže zápas 5-vs-5 je komplexná úloha, boli pravidlá pre rok 2007 upravené. Hráči mohli narábať s loptou akoukoľvek časťou tela (vrátane rúk). Tiež sa mohli pohybovať kráčaním, behaním, plazením, ...

2.2 Simulačné prostredie

Poznámka: Táto kapitola bola revidovaná, revízia sa nachádza v kapitole Revízia č.1 v podkapitole Simulačné prostredie

Ako simulačné prostredie sa využíva SPARK [14], multi-agentový simulačný systém hmotných agentov v troj-dimenzionálnom prostredí. Nasledujúci obrázok (obr. 1.) znázorňuje hlavné časti simulačného systému:

- *Prostriedok pre fyzikálnu simuláciu (Physics Engine)* – implementovaný pomocou knižnice ODE. ODE slúži na fyzikálnu simuláciu komplexných objektov zložených z pevných častí prepojených kĺbmi.
- *Prostriedok na správu objektov (Object / Memory Management)* –implementovaný pomocou frameworku Zeitgeist. Prostriedok umožňuje flexibilný a jednotný prístup k jednotlivým komponentom simulátora:
  - objektom reprezentujúcim simulovanú scénu (sprístupnené ODE objekty),
  - objektom zabezpečujúcim základnú funkcionalitu simulátora.
- *Simulačný prostriedok (Simulation Engine)* – zabezpečuje hlavný cyklus vykonávania a interakciu medzi agentmi a simulátorom. Simulácie väčšieho počtu agentov rozmiestnených na viacerých počítačoch sú ovplyvňované faktormi ako aktuálne zaťaženie siete, rôzne typy latencií a rôzne rýchlosti CPU. Simulácie sú tak často nereprodukovateľné. Z tohto hľadiska je prostriedok implementovaný dvoma spôsobmi:
  - Implementácia pomocou SPADES – berie do úvahy vyššie spomenuté faktory a umožňuje reprodukovateľné simulovanie.
  - Jednoduchá implementácia – vyššie uvedené faktory jednoducho ignoruje, nezaručuje reprodukovateľné simulovanie, ale poskytuje vyšší výkon.



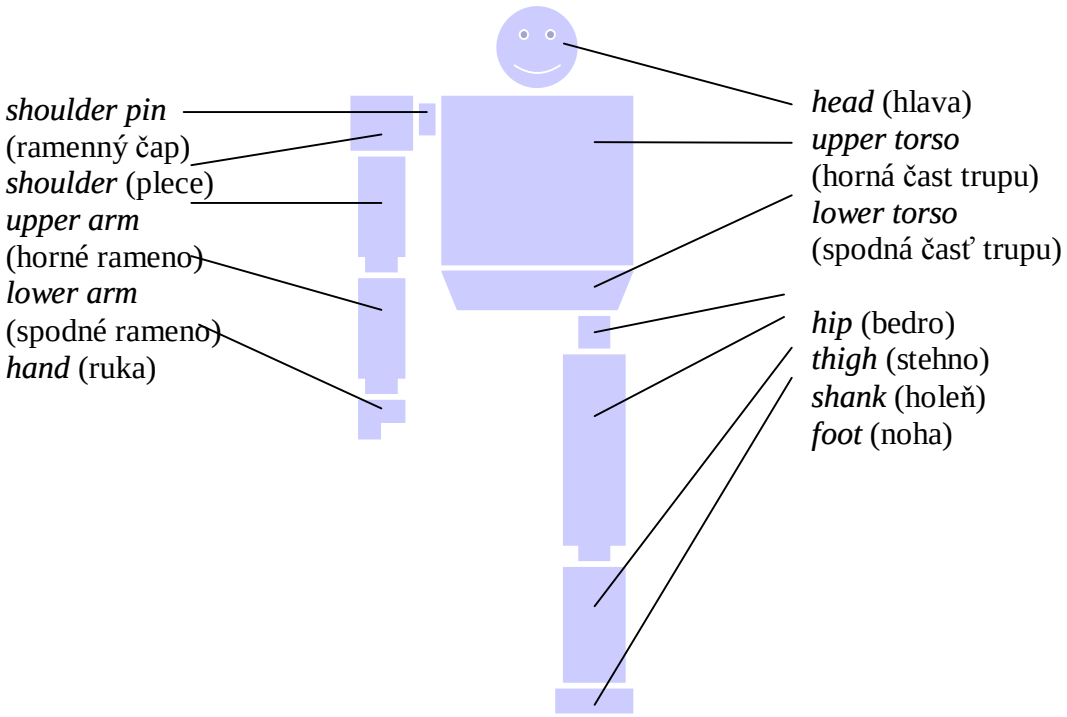
Obr. 1 Architektúra simulátora [14]

Aktuálny server zabezpečujúci simuláciu robotického futbalu je implementovaný práve

systémom SPARK bez použitia SPADES. V súčasnosti je dostupný server vo verzii *0.5.6* a ďalší popis bude preto aktuálny pre túto verziu.

2.2.1 Štruktúra robota

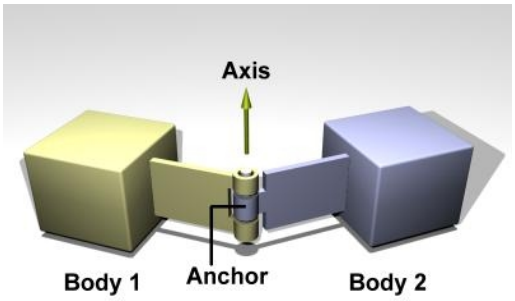
Ako už bolo spomenuté v predchádzajúcej kapitole, simulačný prostriedok SPARK je univerzálny a umožňuje simuláciu rôznych typov hráčov (s rôznymi tvarmi, kĺbmi, efektormi, perceptormi). V súčasnosti sa používa typ robota "soccerbot056" a práve jeho štruktúra bude ďalej popísaná. Nasledujúci obrázok (obr. 2.) znázorňuje jednotlivé časti robota.



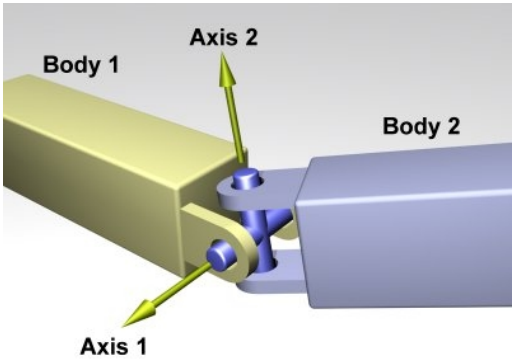
Obr. 2. Štruktúra robota "soccerbot056"

Robot je pre potreby pohybu vybavený niekoľkými kĺbmi. V popisovanom type robota sa využívajú dva typy pohyblivých kĺbov definovaných knižnicou ODE [15]:

- *Pántový kĺb (Hinge joint, HJ)* – poskytuje jeden stupeň voľnosti otáčania (obr. 3.),
- *Univerzálny kĺb (Universal joint, UJ)* – poskytuje dva stupne voľnosti otáčania (obr. 4.).



Obr. 3 Pántový kĺb (Hinge joint)



Obr. 4 Univerzálny kĺb (Universal joint)

Kĺby robota sú vybavené perceptormi, ktoré poskytujú informácie o stave kĺbov a efektormi, pomocou ktorých je možné stav kĺbov meniť. Prepojenie jednotlivých častí robota kĺbmi, názvy perceptorov a efektorov jednotlivých kĺbov popisuje nasledujúca tabuľka (tab. 1.).

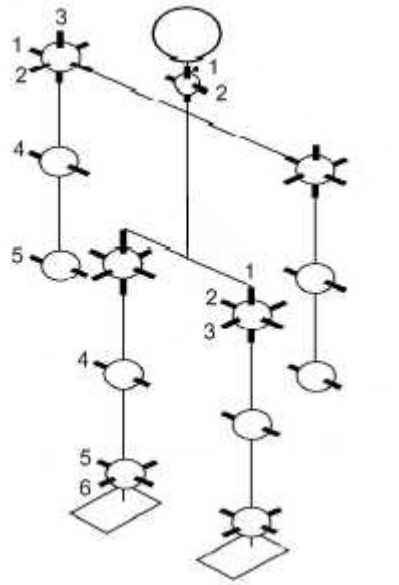
prepojené časti tela robota		typ kĺbu	perceptor*	efektor*
head	upper torso	FJ		
lower torso	upper torso	HJ		
shoulder	upper torso	UJ	laj1_2	lae1_2
upper arm	shoulder	HJ	laj3	lae3
lower arm	upper arm	HJ	laj4	lae4
hand	lower arm	FJ		
hip	lower torso	HJ	llj1	lle1
thigh	hip	UJ	llj2_3	lle2_3
shank	thigh	HJ	llj4	lle4
foot	shank	UJ	llj5_6	lle5_6

FJ – Fixed joint, nepohyblivý kĺb
HJ – Hinge joint
UJ – Universal joint
** – názvy perceptorov a efektorov sú uvedené uvažujúc ľavé končatiny (pre pravé by sa názvy začínali písmenom "r")*

Tab. 1. Prepojenie častí tela kĺbmi

Orientácia osí jednotlivých kĺbov je zrejmá z nasledujúceho obrázku (obr. 5.) (pohyblivé kĺby na krku a rukách sa v súčasnej verzii robota nevyskytujú).

.

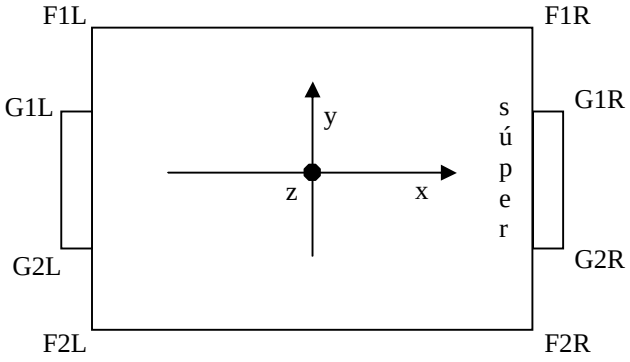


Obr. 5 Orientácia osí jednotlivých kĺbov robota [16]

2.2.2 Štruktúra prostredia robota

Prostredie robota pri simulácii pozostáva z:

- spoluhráčov,
- protihráčov,
- lopty,
- ihriska (obr. 6.):
  - 4 rohové zástavky (*F1L*, *F2L*, *F1R*, *F2R*),
  - 4 okraje bránok (*G1L*, *G2L*, *G1R*, *G2R*).



Obr. 6 Štruktúra ihriska

2.2.3 Komunikačný protokol

Každý hráč (robot) reprezentuje v architektúre klient-server jedného klienta. Komunikácia so serverom prebieha prostredníctvom protokolov TCP a UDP (prednastavený je TCP protokol). Správy vymieňané medzi klientom a serverom majú tvar tzv. S-výyrazu. Ide o konvenciu na reprezentáciu semištruktúrovaných dát[1]. Nasleduje popis komunikačné protokolu medzi klientom a serverom [17][18].

Server->klient
Klient dostáva od servera informácie z perceptorov robota. Správy sú posielané v diskrétnych časových intervaloch 0.02s. Nasleduje popis správ prijímaných z jednotlivých perceptorov z hľadiska syntaxe a sémantiky.

Príklad kompletnej správy prijatej klientom sa nachádza v prílohe A.

2.2.3.1 TimePerceptor

Perceptor poskytuje informácie o aktuálnom simulačnom čase (čas od začiatku simulácie, nie hry).
Vzor: `(time (now <simulačný_čas>))`
Príklad: (príloha A)

2.2.3.2 GameState perceptor

Perceptor poskytuje informácie o aktuálnom stave hry (hrací čas a mód hry).
Vzor:
```
(GS
      (time <hrací_čas>)
      (pm <mód_hry>))
```

- <mód_hry> – môže nadobúdať jednu z hodnôt: *BeforeKickOff, KickOff_Left, KickOff_Right, PlayOn, KickIn_Left, KickIn_Right, corner_kick_left, corner_kick_right, goal_kick_left, goal_kick_right, offside_left, offside_right, GameOver, Goal_Left, Goal_Right, free_kick_left, free_kick_right, unknown.*

Príklad: (príloha A)

2.2.3.3 Vision preceptor

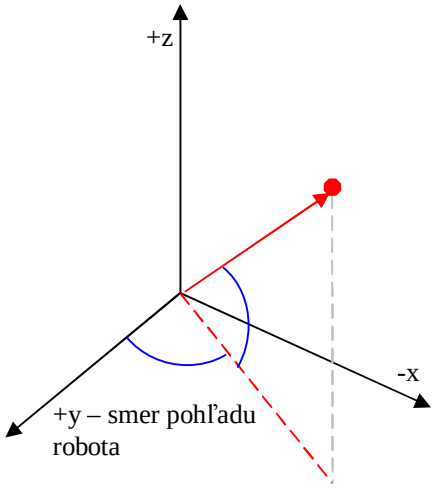
Perceptor poskytuje informácie o robotom viditeľných objektoch. V súčasnosti má robot 360 stupňové videnie (vidí všetky objekty) a objekty majú podobu hmotných bodov. Pre každý objekt poskytuje perceptor polohu objektu relatívne od robota vo sférických súradniciach. Stredom sférickej sústavy je ťažisko horného trupu robota (upper torso). Pre hráčov okrem polohy poskytuje informáciu o tíme a čísle dresu hráča.
Vzor:
```
(See
      (<typ_objektu>
            (pol <vzdialenosť> <uhol_1> <uhol_2>))
      ...
      (P
```

[1]Viac v: S-expression, Wikipedia, http://en.wikipedia.org/wiki/S-expression

```
(team <názov_tímu>)
(id <číslo_hráča>)
(pol <vzdialenosť> <uhol_1> <uhol_2>)))
```

- *<typ_objektu>*
  - 4 rohové zástavky (*F1L*, *F2L*, *F1R*, *F2R*) (obr. 6.),
  - 4 okraje bránok (*G1L*, *G2L*, *G1R*, *G2R*) (obr. 6.),
  - 1 lopta (*B*),
  - spoluhráči a súperi (*P*),
- *<vzdialenosť>* – vzdialenosť objektu od ťažiska hornej časti trupu (stred sférickej sústavy),
- *<uhol_1>* – uhol v rovine *xy* v stupňoch (obr. 7),
- *<uhol_2>* – uhol s rovinou *xy* v stupňoch (obr. 7).

Príklad: (príloha A)



Obr. 7 Znázornenie uhlov poskytovaných perceptorom pre polohu červenej bodky

2.2.3.4 GyroRate perceptor

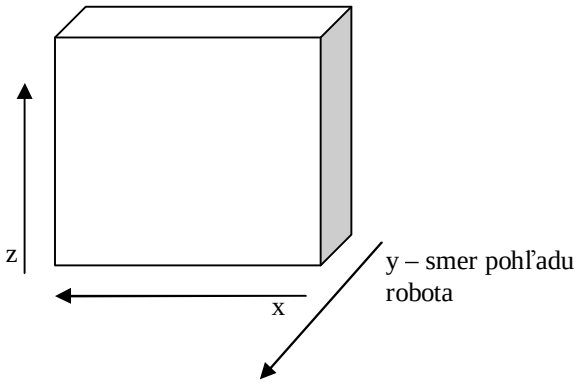
Perceptor je umiestnený v hornom trupe robota (upper torso) a poskytuje informácie o jeho uhlovom zrýchlení okolo jednotlivých osí.

Vzor:

```
(GYR
      (name torso)
      (rt <x> <y> <z>))
```

- *<x> <y> <z>* – uhlové zrýchlenia horného trupu robota okolo osí *x*, *y* a *z* v rad.s^{-2} (obr. 8.).

Príklad: (príloha A)



Obr. 8 Orientácia súradnicových osí robota

2.2.3.5 Touch perceptor

(Súčasná verzia robota nemá tento perceptor aktívny) Perceptor poskytuje binárnu informáciu, či bola určitá z častí tela účastníkom kolízie.

Vzor: `(TCH (name <názov_časti_tela>) (val <hodnota>))`

- <*hodnota*> – *1* v prípade kolízie, inak *0*.

Príklad: `(TCH (name footleft) (val 1))`

2.2.3.6 ForceResistance perceptor

Perceptor poskytuje informáciu o kolíznej sile a priemerný bod pôsobenia kolíznej sily. V prípade, že sa dva povrchy kompletne dotýkajú a existuje množstvo kontaktných bodov, informácia obsahuje len priemerný bod a celkovú silu ako súčet síl pôsobiacu vo všetkých bodoch dotyku.

Vzor: `(FRP (n <názov_časti_tela>) (c <px> <py> <pz>) (f <fx> <fy> <fz>))`

- <*názov_časti_tela*> – v súčasnosti poskytuje perceptor informácie z ľavej (*ls*) a pravej (*rf*) nohy,
- <*px*> <*py*> <*pz*> – lokálne súradnice bodu pôsobenia kolíznej sily,
- <*fx*> <*fy*> <*fz*> – kolízny vektor.

Príklad: (príloha A)

2.2.3.7 Joint perceptors (kĺbové perceptory)

Perceptory poskytujú informácie o stave jednotlivých kĺbov robota (aktuálnom otočení podľa osí otáčania).

Hinge joint

Vzor:
```
(HJ
      (n <názov>)
      (ax <uhol>))
```

- <*názov*> – názov kĺbového perceptora (tab. 1.),
- <*uhol*> – uhol otočenia okolo osi otáčania v stupňoch (obr. 5).

Príklad: (príloha A)

2.2.3.8 Universal joint

Vzor:
```
(HJ
      (n <názov>)
      (ax <uhol_1>)
      (ax <uhol_2>))
```

- <*názov*> – názov kĺbového perceptora (tab. 1.),
- <*uhol_x*> – uhol otočenia okolo osi *x* otáčania v stupňoch (obr. 5).

Príklad: (príloha A)

2.2.3.9 Create effector

Efektor slúži na vytvorenie hráča. Simulačný prostriedok je univerzálny a umožňuje simuláciu rôznych hráčov (s rôznymi tvarmi, kĺbmi, efektormi, perceptormi). Aktuálne sa používa hráč typu "soccerbot056".

Vzor: `(scene <cesta_k_definícii_typu_hráča>)`
Príklad: `(scene rsg/agent/soccerbot056.rsg)`

2.2.3.10 Init effector

Efektor slúži na nastavenie čísla dresu a názvu tímu hráča.

Vzor: `(init (unum <číslo_dresu>) (teamname <názov_tímu>))`
Príklad: `(init (unum 7) (teamname RoboLog))`

2.2.3.11 Beam effector

Efektor slúži na nastavenie pozície hráča na ihrisku pred začatím zápasu.

Vzor: `(beam <x> <y> <uhol>)`

- <*x,y*> – pozícia na ihrisku (obr. 6.),
- <*uhol*> – natočenie hráča v stupňoch (0 stupňov = x+, 90 stupňov = y+).

Príklad: `(beam <10.0> <-10.0> <0.0>)`

2.2.3.12 Joint effectors (kĺbové efektory)

Efektory slúžia na obsluhu motorčekov v kĺboch robota. Pre každý kĺb je možné určiť rýchlosť a orientáciu (určuje znamienko rýchlosti) otáčania podľa osí v kĺbe. Zmysel otáčania podľa osí na obr. 5. je pre kladné hodnoty rýchlosti možné určiť pravidlom pravej ruky (palec ukazuje v kladnom smere príslušnej osi (obr. 8.) a zahnuté prsty v smere otáčania).

Hinge joint
Vzor: `(<názov> <rýchlosť_otáčania>)`
- <*názov*> – názov kĺbového efektora (tab. 1.),
- <*rýchlosť_otáčania*> – rýchlosť otáčania okolo osi otáčania v rad.s^{-1} (obr. 5).

Príklad: `(lae3 1.0)`

Universal joint
Vzor: `(<názov> <rýchlosť_otáčania_1> <rýchlosť_otáčania_2>)`
- <*názov*> – názov kĺbového efektora (tab. 1.),
- <*rýchlosť_otáčania_1*> – rýchlosť otáčania okolo osi otáčania 1 v rad.s^{-1} (obr. 5).
- <*rýchlosť_otáčania_2*> – rýchlosť otáčania okolo osi otáčania 2 v rad.s^{-1} (obr. 5).

Príklad: `(lae1_2 1.0 0.0)`

2.2.4 Zhodnotenie

Kapitola poskytla prehľad simulačného prostredia, štruktúry robota a komunikačného protokolu. Vzhľadom na relatívne nedávne uvedenie humanoidného typu robota a príslušného servera, bolo (je) najväčším problémom hľadanie relevantných zdrojov. Do dnešného dňa totiž nie je dostupná kompletná dokumentácia k serveru pre humanoidný typ robota. Ako zdroj informácií pre túto kapitolu bol okrem uvedenej literatúry použitý aj triviálny prototyp na testovanie komunikačného protokolu zasielaním jednoduchých správ. Empirický pôvod majú najmä informácie o funkčnosti *Vision perceptora* a *GyroRate perceptora*. Funkčnosť niektorých perceptorov nebola vôbec overená (*Touch perceptor*, *ForceResistance perceptor*). Jednou z úlohou prototypu bude preto overiť aj ich funkcionalitu.

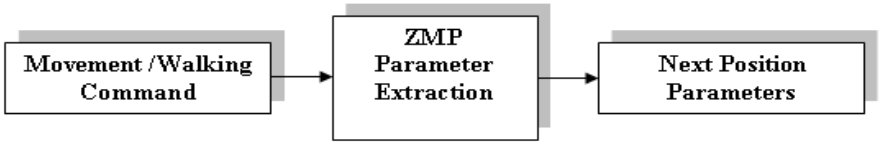
2.3 Analýza rôznych hráčov a tímov

Pred vytvorením našej špecifikácie sme trochu podrobnejšie preštudovali viacero iných tímov a ich hráčov najmä tímov účastniacich sa turnaja v roku 2007 v Atlante.

2.3.1 Sama3D [4]

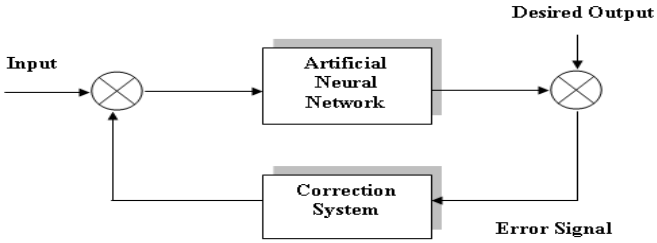
Jedná sa o humanoidného robota ktorý dokáže vykonávať jednoduché pohybové úkony. Na chôdzu využíva neurónovú sieť. Pôvodne robot využíval tzv. ZMP (Zero Moment Point) metódu. Pri ZMP metóde je cieľom udržanie bodu v ktorom je súčet všetkých momentov síl rovné nule, vo vnútri konvexného n-uholníka všetkých dotykových bodov robota s podlahou počas pohybu. Metóda vychádza z matematického modelu kinematiky každej časti schopnej pohybu. Pre každý pohyb je odvodená 3 dimenzionálna rotačná matica a pre každý kĺb je určený translačný vektor. Využitie tejto metódy však vedie k obrovskému zaťaženiu procesora pre každý krok resp. pohyb. Toto zaťaženie autori robota však dokázali znížiť učiacou procedúrou. Na tento účel bola vytvorená neurónová sieť. Sieť bola trénovaná na hodnotách extrahovaných pomocou metódy ZMP. Naučená sieť bola potom schopná určiť ďalší krok (súradnice a ohyby kĺbov) bez potreby zložitých kalkulácií za pomoci ZMP metódy. Robot bol zostrojený a otestovaný v prostredí Webots.

Implementácia robota prebiehala v troch krokoch. V prvom kroku boli určené základné možné pohyby robota v hernom prostredí. Následne boli prepočítané metódou ZMP (v prostredí Matlabu). Z týchto výpočtov boli extrahované údaje pre pohyby kĺbov pre zvolený typ pohybu ako je znázornené na obrázku č. 9.



Obr. 9 Extrahovanie ZMP parametrov pre určenie výslednej pozície [4]

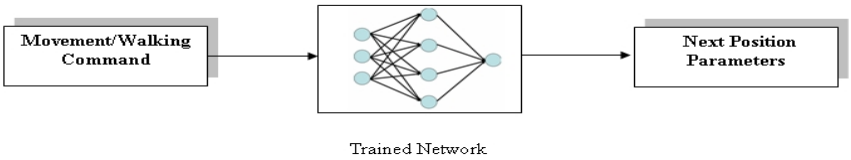
Ďalším krokom bolo vytvorenie a učenie neurónovej siete so vstupmi, ktorými boli dvojice príkaz pre pohyb a k nemu údaje získané ZMP metódou (polohy a rýchlosti kĺbov). Schéma učenia sa je znázornená na obrázku č. 10.



Obr. 10 Učenie neurálnej siete

Sieti je poskytnutý vstup (príkaz pohybu a ZMP údaje). Z výstupu je vypočítaná chyba ktorá je zavedená do korekčného systému. Aplikuje sa korekcia vstupu a sieť sa trénuje ďalej až dokým chyba nie je zanedbateľná.

Po tom čo sa sieť naučí je možné jej priamo vložiť príkaz pohybu a na jej výstupoch dostaneme priamo hodnoty ktoré by poskytla ZMP metóda avšak bez nutnosti toľkých výpočtov, ako ukazuje obrázok.



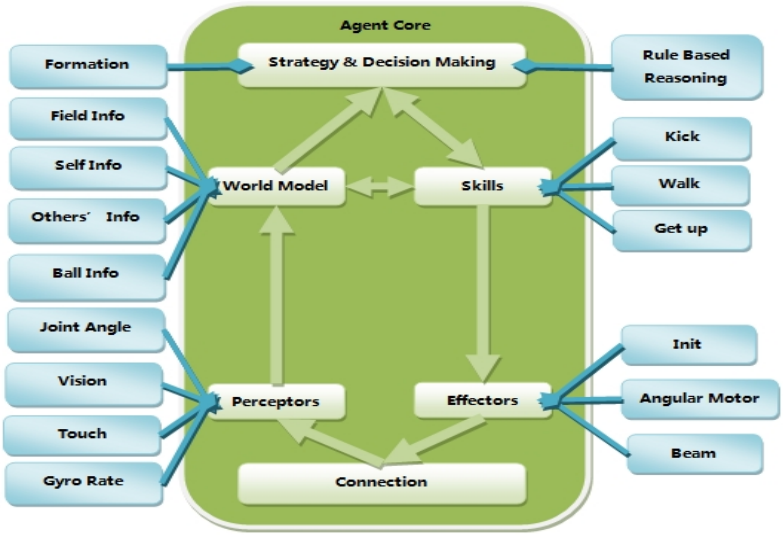
Obr. 11 Určovanie parametrov za pomoci neurónovej siete

Robot tohto týmu disponoval len základnými pohybovými funkciami na najnižšej úrovni. Robot nebol schopný žiadnych vyšších funkcií ako je práca s loptou či plánovanie a pod.

2.3.2 SEU-3D [5]

Tím vznikol v roku 2005 a úspešne sa zúčastnil viacerých súťaží s robotmi v podobe gule čo bola staršia verzia serveru pre RoboCup 3d. Po prechode na nov3iu verziu serveru a humanoidného hráča sa tím sústredil na vývoj základných schopností, vlastného vývojového prostredia a robustného kódu. V roku 2007 sa umiestnili na prvom mieste na RoboCup Iran Open a na treťom mieste v svetovom robocup3d.

Architektúra agenta je modulárna, formou zásuvných modulov a tzv. singletonov. Teda napríklad pri zmene verzie servera, nie je potrebné meniť celý kód stačí len zmeniť príslušný modul. Na obrázku je znázornená schéma agenta.



Obr. 12 Vnútorné usporiadanie agenta

Zelený box predstavuje jadro robota. Moduly vo vnútri štvorca sú implementované ako singletony pre jednoduchšiu komunikáciu. Moduly mimo štvorca sú implementované ako zásuvné moduly.

World model

Slúži na uchovanie si stavu sveta. Z tohto stavu môže robot odvodiť svoju pozíciu P_v a Rotáciu R_v na hracej ploche. Pozícia je odvodzovaná z troch pevných bodov ktoré predstavujú okraje ihriska. Napríklad z horného a dolného ľavého rohu a vrchného pravého rohu p1l, p2l a p1r. Môžeme zostaviť nasledujúce rovnice.

$$\begin{cases} |\boldsymbol{P_v} - \boldsymbol{P_{1l}}| = \rho_{1l_d} \\ |\boldsymbol{P_v} - \boldsymbol{P_{2l}}| = \rho_{2l_d} \\ |\boldsymbol{P_v} - \boldsymbol{P_{1r}}| = \rho_{1r_d} \\ -P_{1l_x} = -P_{2l_x} = P_{1r_x} = 0.5 \cdot field\_length \\ P_{1l_y} = -P_{2l_y} = -P_{1r_x} = 0.5 \cdot field\_width \\ P_{1_z} = P_{2_z} = P_{3_z} = 0 \end{cases} \quad (1)$$

Z ktorých pozícia robota P_v môže byť vypočítaná takto:

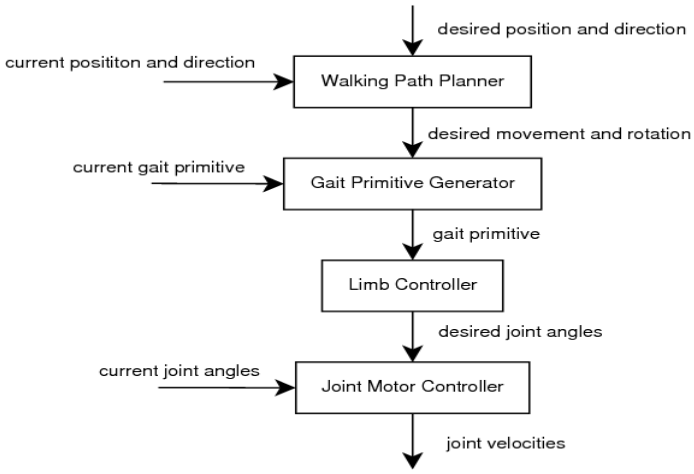
$$\boldsymbol{P_v} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} \frac{{\rho_{1l_d}}^2 - {\rho_{1r_d}}^2}{2 \cdot field\_length} \\ \frac{{\rho_{2l_d}}^2 - {\rho_{1l_d}}^2}{2 \cdot field\_width} \\ \sqrt{{\rho_{1l_d}}^2 - (P_x - P_{1l_x})^2 - (P_y - P_{1l_y})^2} \end{bmatrix} \quad (2)$$

Určenie ťažiska je taktiež dôležité z hľadiska udržania stability robota. Ťažisko môže byť vypočítané vzorcom

$$\boldsymbol{P_{CoM}} = \sum \boldsymbol{P}_i \cdot m_i$$

kde P_i je pozícia časti tela a mi je jeho objem. Ťažisko sa počíta v reálnom čase. Pohľad robota je posunutý na jeho torzo (a nie v hlave).

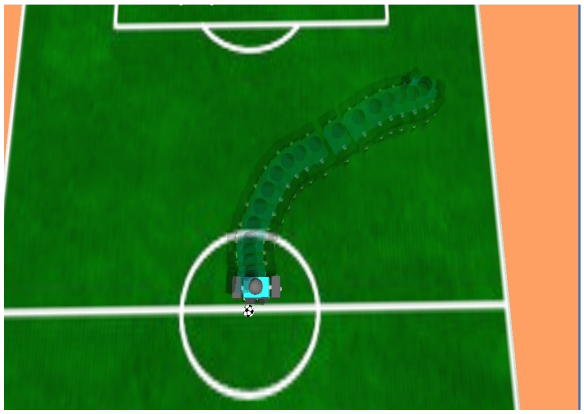
Ovládač vše smerového chodenia agenta má nasledujúcu štruktúru.



Obr. 13 Model ovládača chôdze

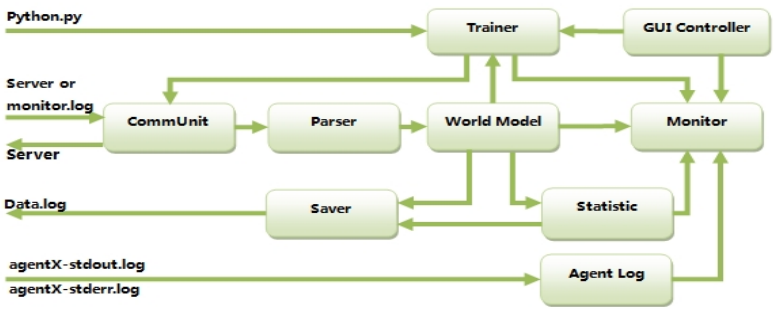
Model ovládača dvojnohej chôdze - Obsahuje viacero vrstiev. *Walking Path Planner* obdrží súradnice a smer chôdze resp. cieľu chôdze. Naplánuje pohyb a odošle ho do *Gait Primitive Generator*. Tento vygeneruje ďalší primitívny pohyb. Tento sa odošle do *Limb Controller* ktorý určí potrebné ohyby kĺbov a potrebné akcie odošle do *Joint Motor Controler* ktorý pohyb kĺbov vykoná.

Výsledkom je hladký neprerušovaný pohyb do všetkých smerov. Robot sa tak nepotrebuje zastaviť keď chce zmeniť smer.



Obr. 14 Kontinuálny pohyb robota

Ako vývojový nástroj bolo použité prostredie *seu-3d-toolkit*. Slúži ako server i nástroj na simuláciu a trénovanie robotov. Z popisu robota nebolo zrejmé či si toto prostredie navrhli sami členovia týmu alebo či sa jedná o určité všeobecné testovacie a vývojové prostredie.



Obr. 15 štruktúra seu-3d-toolkit.

Hlavné výhody tohto prostredia:

- záznam zápasu v priamom aj spätnom smere
- prezeranie výstupov z agenta (logy) v danom čase
- kontrola kamery: pevná alebo napojená na objekt
- „motion blur“ pre indikáciu pohybu robota
- podpora tvorby a analýzy modelu robota
- všetky objekty sveta sú dostupné prostredníctvom stromu objektov na obrazovke
- nezávislosť na platforme

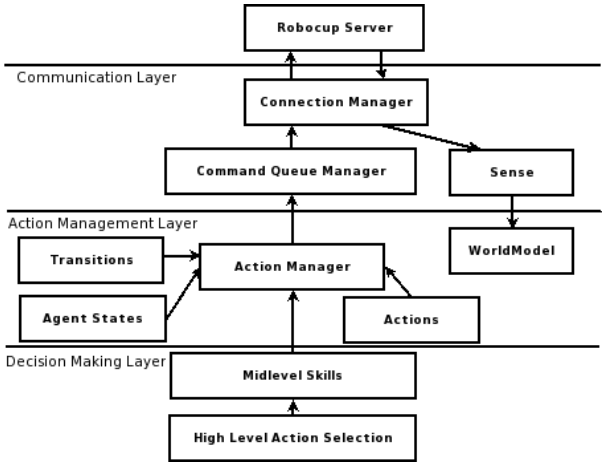
Ako vidno zo schémy agenta tento už pravdepodobne okrem základných pohybov dokázal vykonávať aj vyššie funkcie, o čom nasvedčuje aj fakt že sa tento tým umiestnil v Atlante na treťom mieste.

2.3.3 UI-AI [10]

Pohybový model robota a jeho základné schopností boli definované deterministickým konečným automatom. Jeden stav automatu tvorili polohy jednotlivých častí tela a uhly kĺbov. Tento jednoduchý dizajn v sebe však priniesol niekoľko úskalí:

- akcie robota nebolo možné rozdeliť na menšie podakcie ktoré by boli znovupoužiteľné.
- Robot nebol schopný opustiť vykonávanú akciu uprostred a začať vykonávať inú.
- Agent sa nebol schopný okamžite prispôsobiť aktuálnym údajom zo senzorov.

Architektúra agenta bola rozdelená na 3 vrstvy



Obr. 16 Komunikačná vrstva.

Communication layer - Vrstva sa stará o komunikáciu so serverom. Beží paralelne spolu s ostatným kódom. S každou prichádzajúcou správou aktualizuje svet robota. Ak akčná vrstva vyžaduje vykonanie príkazov táto vrstva zabezpečí ich vhodné usporiadanie v rade príkazov a následné postupné odosielanie serveru na spracovanie.
Akčná vrstva. Táto vrstva zabezpečuje základné zručnosti agenta ako chodenie, otáčanie sa, kopanie a pod. Je rozdelená na štyri časti:

- *Stavy agenta*: Tu sú popísané stavy agenta. Každý stav môže byť opísaný buď veľmi konkrétne ako napr. polohami jednotlivých častí tela, alebo môže byť opísaný aj abstraktne

formou podmienok ohraničujúcich určité hodnoty.

- *Prechody*: Definujú prechody medzi stavmi. Jeden prechod medzi stavmi jeden a dva znamená že robot sa vie dostať zo stavu jeden do stavu dva.
- *Akcie*: Obsahujú základné schopnosti robota ako chodenie , otáčanie sa, postavenie sa a kopanie. Každá akcia je definovaná ako určitá cesta po množine stavov. Tieto stavy musia medzi sebou mať definované prechody. Pre opakujúce sa pohyby sa využíva opakujúci sa cyklus.
- *Manažér akcií*: Stará sa o prepínanie akcií medzi sebou. V súčasnej verzií nie je možné okamžite sa prepnúť na inú akciu ak sa už vykonáva iná akcia.

Ďalšou vrstvou je *Vrstva rozhodnutí*. Implementuje stredné schopnosti agenta a stará sa o radenie akcií za sebou tak aby sa dosiahol vyšší celok. Príkladom sú funkcie kopni-loptu-smerom, choď-na-miesto, chyť-loptu a pod.

Agent disponoval nasledujúcimi schopnosťami:

- *Postavenie sa*. Agent využíva ľudský spôsob postavenia sa. Táto metóda fungovala primerane pre všetky náhodne vygenerované polohy.
- *Chôdza*. Na základe metódy určenia a udržovania ťažiska tím určil tri základné kroky algoritmu pre chôdzu:
    - *nakláňanie sa do strán* – agent prenesie ťažisko na podpornú nohu, odrazením sa od zeme chodidlom druhej nohy.
    - *posunutie nohy dopredu* – druhá noha sa posunie vpred o určenú dĺžku.
    - *posunutie tela* – ťažisko agenta sa po minulom kroku posunie a agent tak musí pohnúť celou vrchnou časťou tela na vyváženie

Táto metóda napriek svojej stabilite je veľmi pomalá.

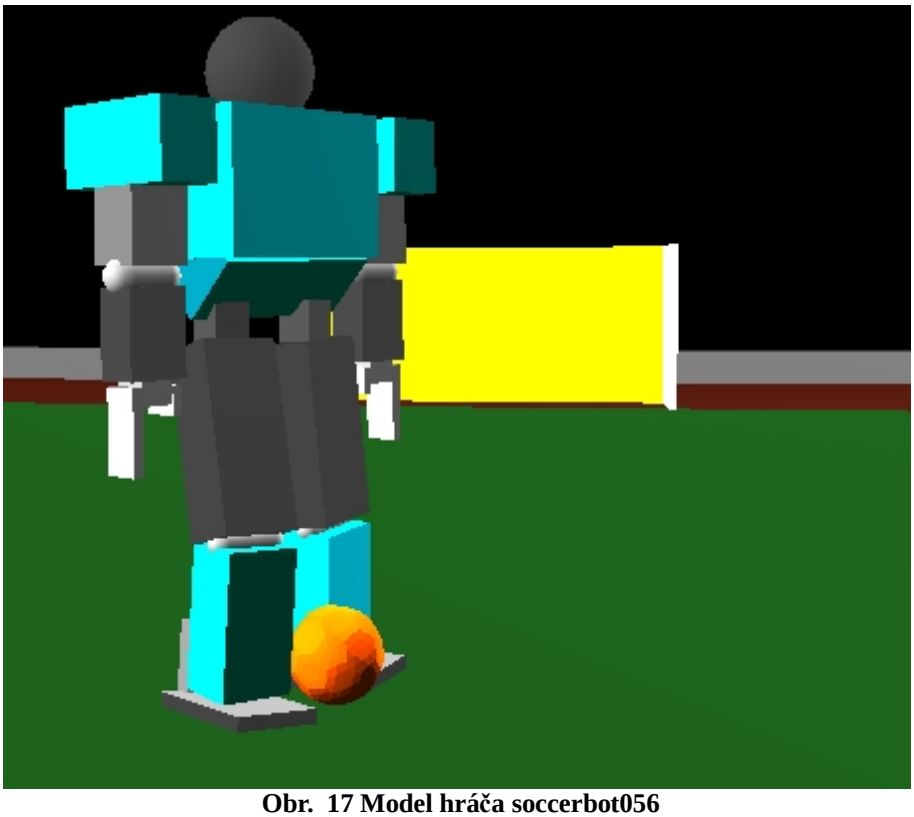
Vývojový nástroj

Tím si vyvinul vlastný nástroj na ovládanie robota za pomoci klávesnice. Bolo možné jednoducho nastaviť ohyby všetkých kĺbov a rýchlosť pohybu. Pomocou tohto nástroja potom vytvorili stavy ktoré boli obsiahnuté v množine stavov v akčnej vrstve agenta ako aj prechodov medzi nimi.

2.3.4 Agent Zigorat [6]

Skupina Zigorat je výsledkom spolupráce viacerých univerzít. Cieľom celého projektu je kooperácia výskumu robotiky medzi univerzitami. Ideou je vytvoriť multi-agentový systém pre výskum širokého spektra algoritmov umelej inteligencií.

Hráč Zigorat je modelom „soccerbot056.rsg“.



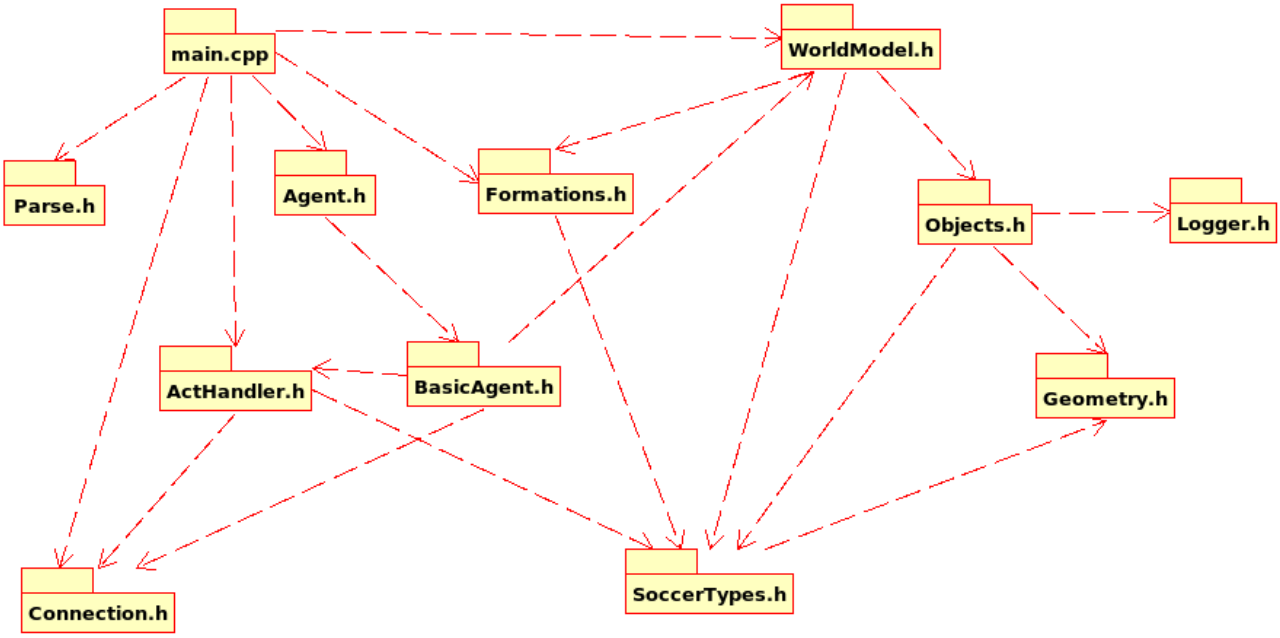
Obr. 17 Model hráča soccerbot056

Na Obr. 18 je znázornené poprepájanie zdrojového kódu na úrovni súborov. Prezeranie začína v súbore main.cpp, ktorý obsahuje main funkciu aplikácie. Tu sa deklarujú všetky potrebné globálne objekty tried:

- *TroboCupConnection [Connection.h]* – poskytuje nadviazanie klienta na server pomocou soketov cez TCP/IP protokol. Prednastavená IP servera je localhost a 3100 port.
- *Formations [Formations.h]* – slúži na určenie rozostavenia hráčov v poli. Rôzne formácie ako inicializačná, ofenzívna, defenzívna apod. sú zaznamenané v súbore *formations.conf*. Každý hráč zastupuje určitú rolu v rámci formácie, ktorá je určená v konfiguračnom súbore formácií a označená číslom (brankár, obranca, útočník).
- *WorldModel [WorldModel.h]* – objekt tejto triedy poskytuje agentovi všetky potrebné informácie o svojom okolí, rozmery ihriska, bránky, aký je aktuálny čas simulácie, pozície rohových zastávok, informácie o kĺbových spojoch agenta, jeho energiu a pod.

- *ActHandler [ActHandler.h]* – slúži na zasielanie úkonov / príkazov na server. Úkon je určitá abstrakcia, zovšeobecnenie. Trieda dané úkony pred zaslaním na server prepíše do syntaxe príkazu. Obsahuje zásobník úkonov, ktorý ich akumuluje a jedným príkazom sa odošlú všetky na server. Je možné zasielať príkazy aj úkony na server osobitne.

Agent [Agent.h] – objekt triedy *Agent* informuje server, aký model agenta využíva, pracuje s telom agenta ovládaním jeho spojov, t.j. vkladá jednotlivé úkony do zásobníka, ktoré sa odošlú naraz na server. Tu sa aj nachádzajú metódy rôznych akcií, jedna z nich je aj chôdza riešená Zigorat agentom. Podstatnou metódou je mainLoop, ktorá vytvára cyklus ukončovaný správou zo servera pri ukončení simulácie. V cykle prebieha rozhodovanie sa agenta na základe prijatých nových informácií prostredia zo servera (ktoré rozparsuje a uloží do svojich lokálnych štruktúr), naplánovanie úkonov do zásobníka a ich odoslanie na server.



Obr. 18 Poprepájanie zdrojového kódu na úrovni súborov

Chôdza Zigorat agenta

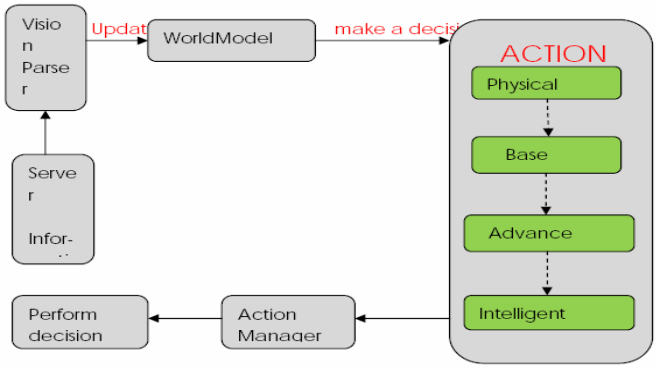
Je definovaná staticky, agent slepo vykonáva úkony bez akejkoľvek dynamickej korekcie. Úkony tela, ako zdvihnutie kolena, posunutie ruky apod., sú plánované podľa relatívneho času simulácie, čo je problém ak príkazy neprijal server v danom potrebnom simulačnom čase. Postupnosť úkonov sa tak naruší a agent padá. Ďalej ak agent narazí na nečakanú prekážku, chôdza sa neprispôsobí na vyrovnanie rovnováhy. Takto určená chôdza je vhodná a úspešná pri bezchybnom spojení a priamej ceste bez prekážok, kde tím agenta Zigorat našli vhodnú postupnosť správne určených úkonov tela pre priamočiary pohyb agenta.

Zigorat sa umiestnil na druhom mieste v robocup3d na turnaji v Atlante a v Teheráne na RoboCup IranOpen 2007 sa dostal až do finále.

2.3.5 DNU Explorer [7]

Hráč sa skladá z niekoľkých častí:

1. *Vision Parser*: Pre transformovanie informácií získaných zo serveru na informácie použiteľné pre "WorldModel".
2. *WorldModel*: Používa informácie z "Vision Parser" pre zistenie aktuálnych informácii o agentovi a hracom poli, a podľa nich robí rozhodnutia.
3. *Action Manager*: Dáva konečné rozhodnutia podľa priority jednotlivých akcií, a či sa majú vykonať alebo nie.
4. *Action Perform*: Predá konečné rozhodnutia do vykonávacej sekvencie na vykonanie.



Obr. 19 Schéma agenta DNU Explorer

Action Manager:

- vytvorí sekvenciu akcií - izoluje každú akciu na detailné kroky a vytvorí z nich sekvenciu
- filtruje akcie

Posúdi, či sa má akcia vykonať alebo nie. Ak áno vyprázdni sa *vykonávacia sekvencia* a vloží sa do nej daná akcia. Rozhodnutie sa vykonáva podľa priority akcie.
ak je výsledok predošlého kroku neúspešný, skontroluje sa *vykonávacia sekvencia*, či je prázdna alebo nie. Ak je prázdna vloží sa do nej daná akcia, v opačnom prípade sa vykonajú všetky akcie z *vykonávacia sekvencia* a potom sa do nej vloží daná akcia.

- vykoná všetky akcie z *vykonávacej sekvencie*

2.3.6 Fantasia [8]

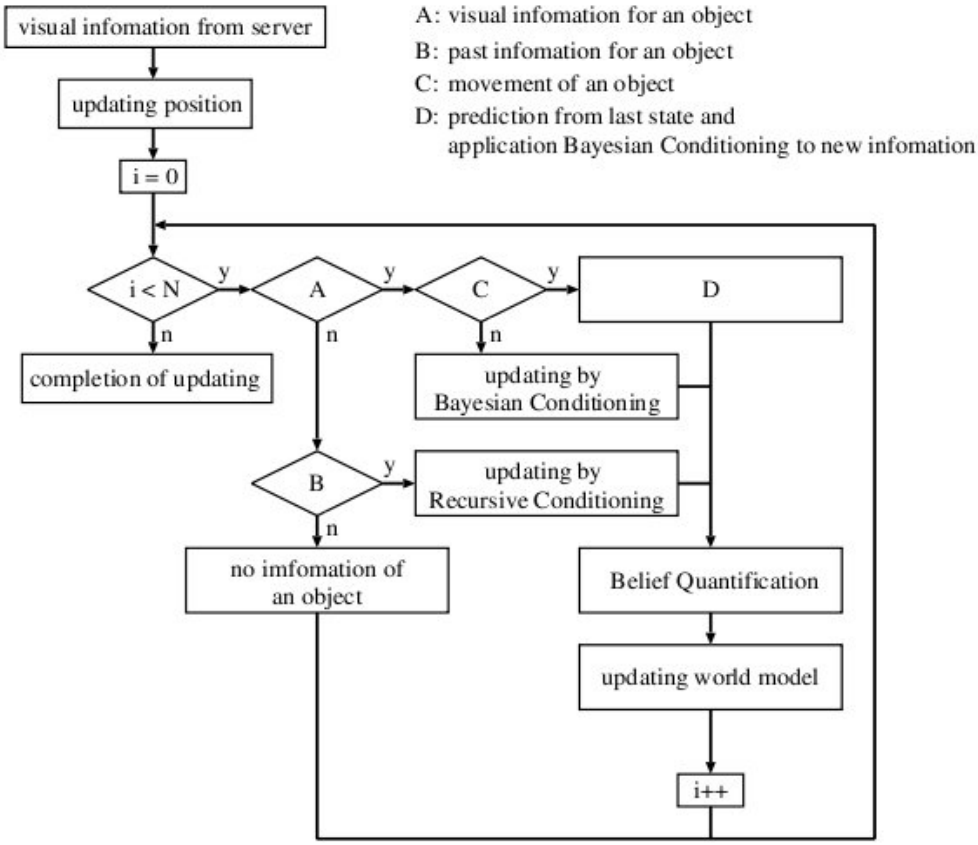
Hráč Fantasia sa zúčastnil RoboCup China Open 2006, kde vyhral prvé miesto v kategórií RoboCup simulation 3D. Využíva koncept *Zero Moment Point* (ZMP) – bod nulového pohybu pre zabezpečenie dynamickej stability dvojnohého robota. ZMP je definovaný ako bod na podlahe, v ktorom je suma všetkých momentov síl, je rovná nule. Ak je ZMP vo vnútri konvexného obalu všetkých dotykových bodov medzi chodidlami a podlahou, je možné aby dvojnohý robot chodil. Ďalej sa tento konvexný obal všetkých kontaktných bodov volaný *stabilná oblasť*.

Robot je ovládaný zmenou uhlovej rýchlosti každého kĺbu. Ovládame ho pomocou kľúčových krokov (key frames):

Kďe ™ znamená čas trvania m-tého kroku, T_{mn} znamená uhol n-tého kĺbu v m-tom kroku. Kroky medzi týmito krokmi sú približne určené pojitou funkciou. Uhly kĺbov a dobu trvania krokov môžu byť ďalej optimalizované pomocou EDA algoritmu (Estimation of Distribution Algorithm).

2.3.7 Naito Striker [9]

Pri tvorbe tohto hráča sa jeho tvorcovia zamerali na vybudovanie akéhosi druhu databázy tak, aby autonómny agent mohol správne vyhodnotiť informácie o ostatných hráčoch a lopte. Používajú vylepšený M. Saxena a T. Guputaov model:

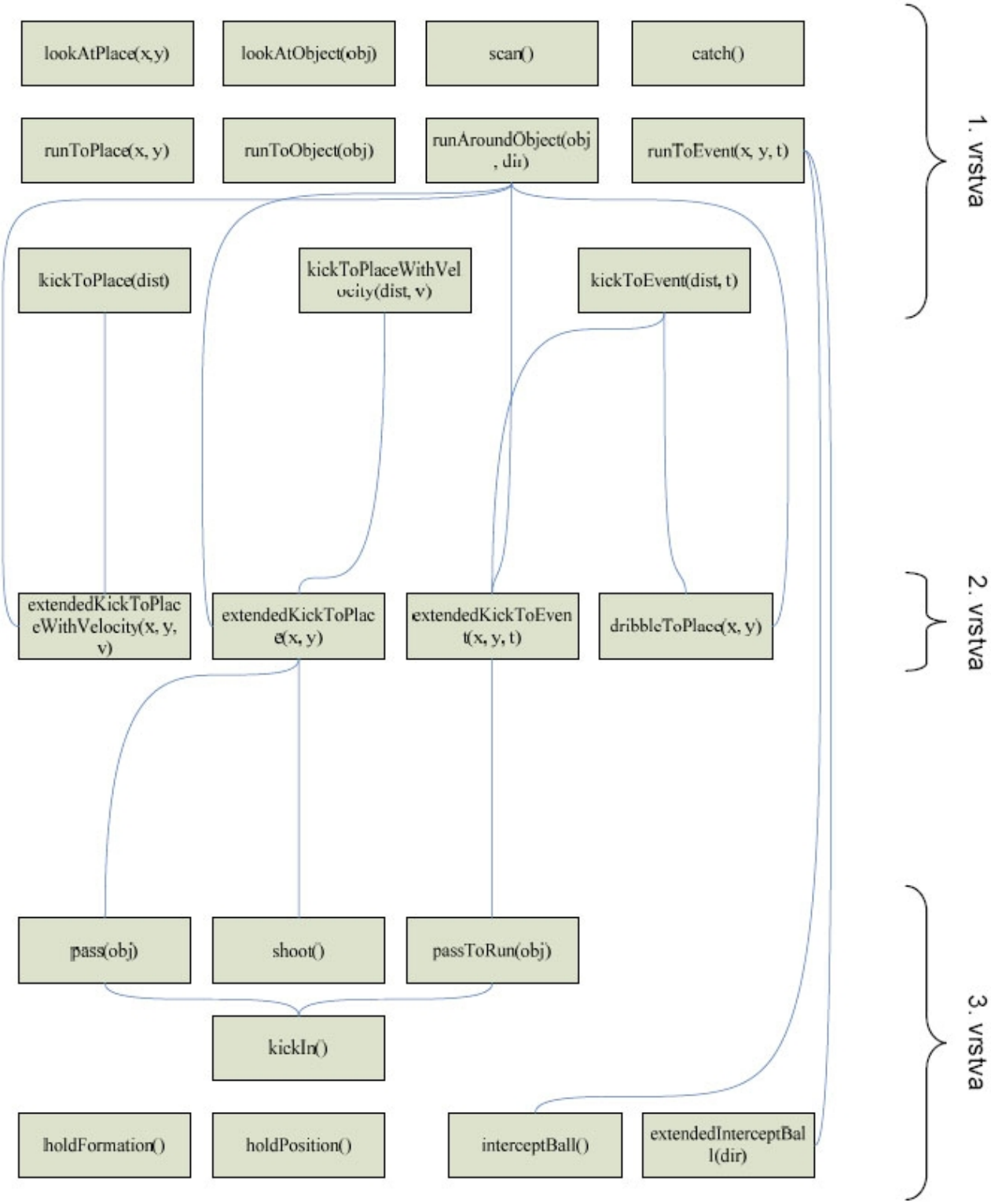


Obr. 20 Schéma agenta Naito Striker

2.3.8 Agent Hazard

Jedná sa o hráča z roku 2006 vytvorenom na predmete Tvorba softvérového systému v tíme. Cieľom bolo vytvoriť hráča na základe práce tímu z roku 2005 s pokročilejšími schopnosťami a vyspelejším rozhodovaním.

Moduly správania
Správanie sa hráča je robené prostredníctvom modulov a rozdelené do troch vrstiev na obr. 21.



Obr. 21 Návrh hierarchie modulov správania

Pre moduly správania vrstvy *n* platí, že využívajú iba moduly správania nižších vrstiev (t.j.*1...n-1*). Takéto rozdelenie je výhodné z hľadiska vyjadrenia závislostí.

Popis jednotlivých modulov:

Moduly správania prvej vrstvy

Medzi základné moduly správania patria:

- moduly pre beh hráča
- moduly pre kopanie
- moduly pre pozorovanie prostredia

runToPlace(x, y)	Beh na miesto (x,y)
runToObject(obj)	Beh k objektu obj.
runAroundObject(obj, dir)	Beh k objektu obj. Pri dobehnutí k objektu má mať spojnica ťažísk objektu a hráča smerový uhol dir.
runToEvent(x, y, t)	Dobehnutie na miesto (x,y) v case t.
kickToPlace(dist)	Kopnutie na vzdialenosť danú parametrom dist. Kope sa smerom v ktorom je hráč natočený.
kickToEvent(dist, t)	Kopnutie na vzdialenosť danú parametrom dist. Kope sa smerom v ktorom je hráč natočený. Lopta má doraziť na cieľové miesto v čase t.
kickToPlaceWithVelocity(dist, v)	Kopnutie na vzdialenosť danú parametrom dist. Kope sa smerom v ktorom je hráč natočený. Lopta má doraziť na cieľové miesto s rýchlosťou v.
lookAtPlace(x,y)	Hráč natočí kameru tak, aby zadané miesto bolo v strede jeho zorného pola.
lookAtObject(obj)	Hráč natočí kameru tak, aby zadaný objekt bol v strede jeho zorného pola.
scan()	Hráč otáča kameru tak, aby postupne zaznamenal celé svoje okolie.

Tab. 2 Moduly správania prvej vrstvy

Moduly správania druhej vrstvy

Medzi rozšírené moduly správania patria rozšírené moduly kopania a modul driblovania s loptou.

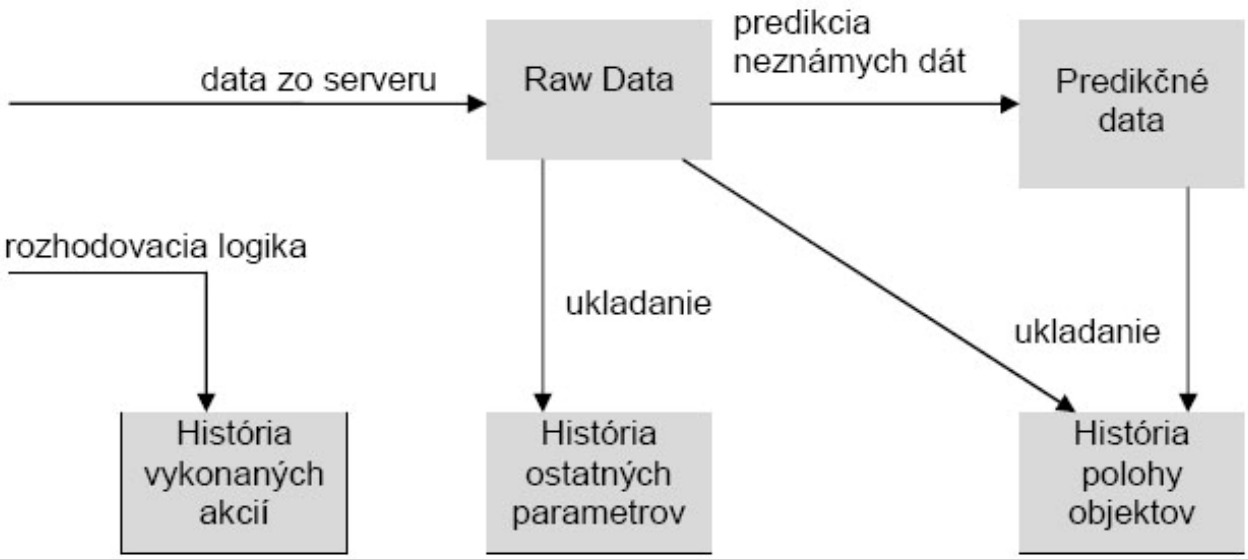
extendedKickToPlace(x, y)	runAroundObject + kickToPlace
extendedKickToEvent(x, y, t)	runAroundObject + kickToEvent
extendedKickToPlaceWithVelocity(x, y, v)	runAroundObject + kickToPlace
dribbleToPlace(x, y)	Beh s loptou na dané miesto. Hrác musí mat pred zaciatkom tohto správania loptu vo svojej blízkosti.

Tab. 3 Moduly správania druhej vrstvy

Moduly správania tretej vrstvy	
pass(obj)	Prihrávka objektu obj. Hráč musí mať pred začiatkom tohto správania loptu vo svojej blízkosti
passToRun(obj)	Prihrávka do pohybu objektu obj. Hráč musí mat pred začiatkom tohto správania loptu vo svojej blízkosti.
shoot()	Strela na bránu. Hráč musí mat pred začiatkom tohto správania loptu vo svojej blízkosti. holdPosition() - Hráč zostáva stať na mieste
holdFormation()	Hráč sa presunie na svoje miesto vo formácii.
kickIn()	Vkopnutie lopty do hry
interceptBall()	Prerušenie pohybu lopty.
extendedInterceptBall(dir)	Prerušenie pohybu lopty a nastavenie sa do polohy, pri ktorej má spojnica lopty a hráča uhol dir.

Tab. 4 Moduly správania tretej vrstvy

Agentov model sveta je reprezentovaný tzv. *information storage* znázornenom na obrázku



Obr. 22 Information Storage

Raw data – je vrstva, ktorá obsahuje a pracuje s informáciami, ktoré prichádzajú zo servera a sú (odhliadnuc od možného šumu) presné.
Predikčné dáta – sú generované prediktorom, sú nevyhnutné na tvorbu stratégie.
História polohy objektov – v tejto časti sú uložené všetky súradnice objektov od začiatku simulácie, sú to informácie o polohe v polárnych súradniciach ako prichádzajú zo servera a tiež prepočítané pravouhlé súradnice. Taktiež je tu história predikovaných súradníc.
História vykonaných akcií – všetky akcie vykonané daným hráčom v chronologickom poradí.
História ostatných parametrov – stav batérie, teplota, uhol pohľadu, mód hry.

Hráč ako taký je schopný samostatnej hry. Implementácia z časových dôvodov však nebola kompletná. Niektoré moduly sú vytvorené len na úrovni prototypu.

2.4 Zhodnotenie analýzy

Podrobne sme si rozobrali simulačné prostredie v ktorom prebiehajú zápasy RoboCup-u. Zistili sme si aktuálny model humanoidného robota. Je to pomerne veľký skok od predošlej verzie servera s ktorou pracoval napríklad tím v minulom roku kedy model robota tvorila len guľa. Aktuálny model má humanoidnú formu a disponuje niekoľkými kĺbmi po celom tele, pričom kĺby môžu byť dvoch rôznych druhov. Zistili sme že komunikácia medzi robotmi a serverom prebieha na protokoloch TCP alebo UDP je teda možné (a na súťaži nutné) spustiť svoj tím na osobitnom počítači a po sieti sa pripojiť na počítač kde beží serverová aplikácia. Analýza rôznych tímov ukázala aké rôznorodé riešenia je možné použiť pri tvorba agentov. Spomedzi analyzovaných agentov sme si pre ďalšiu prácu vybrali dvoch a to konkréte agenta Hazarda z minulého roku od tímu 6th sense, a agenta Zigorata. Agent hazard disponuje výhodným modelom správania sa organizovaným do vrstiev z hľadiska úrovne správania sa a ďalej do modulov. Táto architektúra sa javí ako výhodná, flexibilná a ľahko rozšíriteľná. Bohužiaľ agent hazard je postavený na staršom type servera a preto nemôžeme jeho kód využiť celý. Preto sme hľadali ďalej a ako už bolo spomenuté ako druhého kandidáta sme si vybrali agenta Zigorata. Tento agent je stavaný na nový typ serveru a poskytuje vynikajúci model komunikácie so serverom. Agent taktiež disponuje základnými pohybovými vlastnosťami ako je otáčanie sa a chôdza tie sa však pri našich testoch javili ako pomerne nestabilné. Rozhodli sme sa použiť oboch agentov a z každého zobrať to najlepšie.

3. Špecifikácia požiadaviek

Cieľom projektu bude navrhnúť prototyp hráča resp. časti hráča. Tento prototyp by mal byť schopný fungovať na novom type servera a mal by mať formu humanoidného robota. Opierať sa budeme o idei tímu z minulého roka v predmete TSST, ako aj našej analýzy iných svetových tímov. Budeme sa sústrediť na základné schopnosti hráča. Nový typ serveru ako aj humanoidný hráč majú iné charakteristiky ako ich staršie verzie. Úlohou bude základná orientácia hráča v priestore, jednoduchý pohyb (chôdza) a schopnosť postaviť sa.

4. Hrubý návrh

Z analýzy predošlých agentov sme usúdili, že použitie statických metód na riadenie jednotlivých funkcií agentov nie je vhodné. Z tohto dôvodu sme si zvolili použitie evolučného algoritmu. Keďže táto metóda ešte nebola použitá rozhodli sme sa špecializovať len na jednu oblasť súťaže development a tou je vstávanie agenta z ľubovoľnej polohy.

Z analýzy štruktúr agentov vyplynulo, že najvhodnejšie bude skombinovať komunikačnú vrstvu ľubovoľného agenta a vytvoriť, respektíve prevziať architektúru modelu správania sa z agenta Hazarda. Tento postup sme zvolili z dôvodu veľmi kvalitnej štruktúry modelov správania sa v agentovi Hazardovi. Aj keď je agent vytvorený na starú generáciu RoboCup serverov (guličky), všeobecná štruktúra jeho modelov správania sa dá aplikovať aj na novú generáciu agentov.

Základnou myšlienkou nášho agenta bude využitie evolučného algoritmu na dosiahnutie cieľa. Naším cieľom je postavenie sa do polohy, ktorá bude vhodná na ďalší pohyb. Treba povedať, že táto poloha nebude určená staticky, ale bude sa odvíjať od celkovej situácie a potrieb, ktoré budú určené vyššou plánovacou vrstvou.

Pre príklad: agent stratí rovnováhu a spadne. Pred spadnutím bolo jeho cieľom priblížiť sa k lopte. Agent zahájí model správania ktorý zabezpečí postavenie, pričom však výsledným cieľom tohto kroku nebude statické rozloženie častí tela, ale taká poloha, z ktorej sa dá najefektívnejšie pokračovať v nadradenom pláne, t.j. poloha z ktorej agent najlepšie zahájí pohyb k lopte.

4.1 Štruktúra agenta

Agent neurotik sa bude skladať zo štyroch hlavných častí:

1. Hlavný vykonávací cyklus
2. Komunikačný modul
3. Pohľad na svet (worldview modul)
4. Modely správania sa

Hlavný vykonávací cyklus bude samotné jadro agenta. V tejto časti agenta sa bude vykonávať volanie jednotlivých podmodulov ktoré zaručia danú funkčnosť.

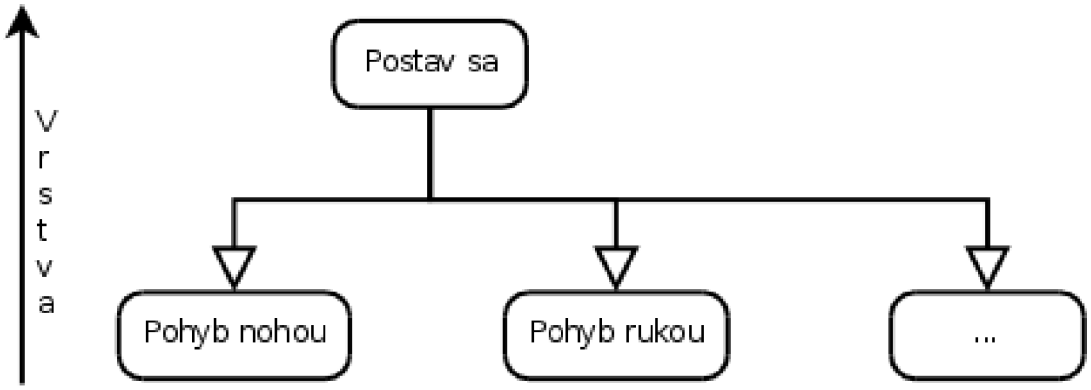
Komunikačný modul agenta bude prevzatý z agenta Zigorata. Modul bude upravený pre naše potreby, hlavne plánujeme zmeny v rozhraní, funkčnosť ostatne nezmenená. Modul sa bude starať o posielanie a prijímanie správ zo servera.

Pohľad na svet sa prevezme z agenta Zigorata. Pre naše potreby budú zmeny v tejto časti minimálne.

Agent Neurotik bude využívať štruktúru modelov správania sa (behaviors) z agenta Hazarda. Štruktúra je založená na modularite a jednoduchosti jednotlivých modelov.

Každý model je reprezentovaný triedou, ktorá je zdedená z abstraktného rozhrania BaseBehaviorModule. Toto rozhranie obsahuje metódy, ktoré sa v konkrétnych modeloch implementujú. Pomocou makier REGISTER_MODULE(meno) sa jednotlivé moduly zaregistrujú do systému. Makrom USE_MODULE(meno, cesta) sa modul prihlási ako aktívny na použitie.

Samotné moduly sú rozdelené do viacerých vrstiev. Agent Hazard mal 3 vrstvy, pričom platí, že každý model vrstvy *x*, môže používať len moduly vrstvy *x – 1*.



Obr. 23 Vrstvy modelov udalostí

V našom agentovi tento princíp zachováme, avšak počet vrstiev modelov sme obmedzili na dve. Spodná vrstva bude obsahovať základné modely pre pohyb jednotlivých častí tela, a vrchná vrstva bude obsahovať primitívne modely typu „postav sa“. Systém bude navrhnutý tak, aby bol do budúcnosti rozšíriteľný.

4.2 Model vstávania

Hlavnou úlohou agenta Neurotika v tomto projekte je postavenie sa do vhodnej polohy vzhľadom na celkový plán akcie na vyšších úrovniach. Z analýz sme usúdili, že statický prístup k problémom rovnováhy nie je vhodný, zvlášť pre situáciu postavenia kde nie je možné spoliehať na predpoklady určitej začiatočnej polohy.

Z tohto dôvodu sme si vybrali ako spôsob riešenia použitie evolučného algoritmu.

4.3 Využitie evolučných algoritmov

Evolučné algoritmy využívajú, na riešenie optimalizačných problémov, princípy evolúcie živej hmoty. Simulujú darwinov evolučný proces. Ten obsahuje tieto tri zložky [12]:

1. **Prirodzený výber** – Silnejší jedinci majú väčšiu šancu reprodukcie ako slabší jedinci.
2. **Náhodný genetický drift** – Náhodné udalosti v živote jedincov. Napríklad náhodná smrť silného jedinca pred tým, než dostal šancu na reprodukciu.
3. **Reprodukcia** – Genetická informácia potomkov vzniká kombináciou génov rodičov, pričom môže dochádzať k jej poškodeniu (mutácii).

Simulovaná evolúcia prebieha nad určitou množinou jedincov, v ktorej sa cyklicky opakuje výber rodičov a reprodukcia. Po reprodukcii vzniká nová množina jedincov a evolúcia ďalej prebieha nad touto novou množinou.

Takouto simulovanou evolúciou môžeme napríklad naučiť hráča pohybovať sa. Pohyb hráča je v podstate zložený z množstva pohybov jeho motorčekov. Celý pohyb sa teda dá definovať ako chronologická postupnosť rýchlostí otáčania všetkých hráčových motorčekov. Na začiatok by bolo

vhodné otestovať možnosti aplikácie niektorého z evolučných algoritmov na jednoduchšom pohybe. Napríklad na postavení sa.

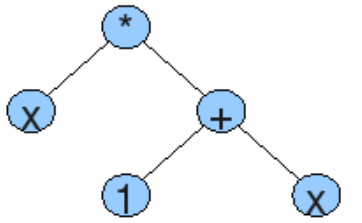
Postavenie sa pomocou evolučného algoritmu
Pre realizáciu simulovanej evolúcie je potrebné zadefinovať nasledovné:

- jedinca a jeho reprezentáciu
- spôsob ohodnocovania schopnosti jedincov prežiť (fitness)
- spôsob výberu rodičov
- operácie kríženia a mutácie
- spôsob výberu jedincov do novej generácie

4.3.1 Reprezentácia jedinca

Jedinec je definovaný genetickou informáciou. Tá musí (pre potreby evolúcie pohybu) definovať pohyb robota. Bude ju teda tvoriť chronologická postupnosť základných pohybov robota (pohybov jednotlivých kĺbov), indexovaných časom, kedy sa tento pohyb začne vykonávať. Dĺžka chromozómu nie je dopredu známa. Preto je vhodné použiť tzv. „Messy“-chromozómy [13] (m-chromozómy). Tie majú premenlivú dĺžku. Samotný chromozóm je tvorený postupnosťou usporiadaných dvojíc, zložených z indexu a hodnoty génu. Ak sa v chromozóme nachádza viac génov s rovnakým indexom, použije sa len jeden z nich, a to prvý v poradí. V tomto konkrétnom prípade by index tvoril čas a hodnotu génu zoznam vstupov pre metódy zabezpečujúce pohyb jednotlivých kĺbov.

Robot má senzory, ktoré mu poskytujú nejaké informácie o okolitom priestore. Bolo by teda vhodné, aby robot riadil svoj pohyb na základe týchto informácií. Pohyby by teda mali byť relatívne vzhľadom na tieto informácie (aspoň na niektoré z nich). Ale ako? Dopredu určiť túto závislosť je nemožné. Mohla by teda tiež byť predmetom evolúcie. Elementy zoznamu hodnôt jednotlivých génov by nemali byť len obyčajné čísla. Mohli by to byť funkcie, ktoré umožnia ich výpočet na základe robotových informácií o okolí. Problémom tohto typu sa zaoberá metóda *Genetického* programovania [13]. V nej je gén tvorený koreňovým stromom, kde sú vrcholy stromu ohodnotené funkciou a listy hodnotami.



Obr. 24 Príklad koreňového stromu. Funkcia F(x)=x*(1+x)

Jedinec teda bude reprezentovaný m-chromozómom, v ktorého génoch bude index reprezentovať čas a hodnotu zoznam koreňových stromov.

4.3.2 Fitness

Fitness označuje schopnosť prežitia a reprodukcie jedincov. Spôsob ohodnocovania fitness teda určuje kam bude evolúcia smerovať. V tomto prípade chceme aby sa robot čo najrýchlejšie postavil tak, aby mohol čo najskôr vykonať inú akciu (musí stáť na zemi) a aby pri tom minul čo najmenej energie. Do výpočtu fitness teda vstupujú nasledovné parametre:

1. Rozdiel maximálnej dosiahnutej výšky hlavy a výšky hlavy vo vzpriamenej polohe
2. Čas za ktorý sa do tejto výšky dostal ako dlho v nej zotrval (či potom nespadol)
3. Množstvo energie potrebnej na postavenie sa.

Parameter číslo jedna je rozdiel z dôvodu zamedzenia vyskakovania do čo najvyššej výšky.

Presné závislosti a podiely jednotlivých parametrov sa nedajú dopredu presne určiť a budú určené pri testovaní a ladení.

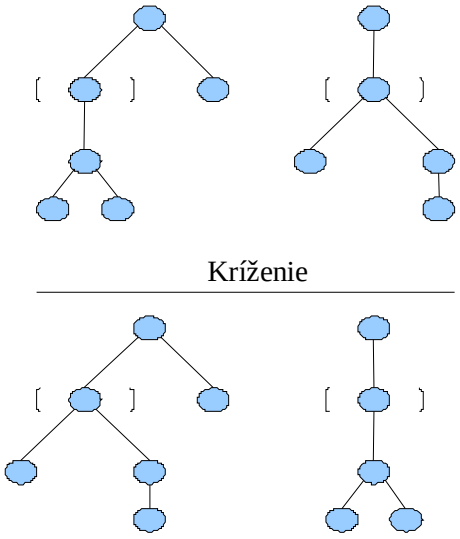
Poznámka: Fitness funkciu sme sme upravili, a jej konečnú podobu je možno nájsť v kapitole Revízia č.1, podkapitole Fitness funkcia

4.3.3 Spôsob výberu rodičov

Po ohodnotení jedincov hodnotou fitness, je nutné vybrať niekoľko jedincov, ktorý pristúpia k reprodukcii. Stratégií ne tento výber je niekoľko. Napríklad výber najlepších jedincov, turnajový výber, pseudonáhodný výber, … No pseudonáhodný výber imituje aj náhodne udalosti v živote jedincov. Preto budeme používať túto stratégiu, konkrétne implementovanú pomocou ruletového výberu.

4.3.4 Operácie kríženia a mutácie

Operácie kríženia musia byť definované tak, aby zasahovali chromozóm aj jednotlivé gény v chromozóme. Začneme teda tieto operácie definovať od najnižších vrstiev.
Kríženie dvoch koreňových stromov spočíva vo výmene podstromov. V obidvoch stromoch sa nezávislé od seba náhodne zvolí bod kríženia. A podstromy začínajúce v týchto bodoch kríženia sa jednoducho vymania.



Obr. 25 Príklad kríženia dvoch stromov. Zátvorky označujú náhodne zvolené body kríženia.

Mutáciou stromu budeme rozumieť nahradenie náhodného podstromu iným, náhodne vygenerovaným, stromom. Kríženie dvoch génov bude spočívať v postupnom krížení všetkých ich stromov na zhodných indexoch (prvý s prvým, druhý s druhým, …) a výberu indexu z jedného z nich. Mutácia génu je náhodná zmena indexu a mutácia stromov. Kríženie dvoch m-chromozómov potom bude prebiehať nasledovne. Náhodne sa zvolí bod kríženia a začnú sa krížiť jednotlivé gény. Až po bod kríženia chromozómov sa budú brať indexy z prvého rodiča a za bodom kríženia sa budú brať indexy z druhého rodiča. Mutácia m-chromozómu je náhodné pridanie alebo odobratie génu.

4.3.5 Výber jedincov do novej generácie

V novej populácii budú určite všetci potomkovia, ktorí vznikli pomocou kríženia alebo mutácie z predchádzajúcej populácie. No treba sa rozhodnúť či do nej zahrnieme aj niekoľko jedincov z predchádzajúcej populácie. Napríklad najlepších jedincov. Evolučné algoritmy imitujú evolúciu v prírode. Dalo by sa teda napríklad uvažovať o nejakom veku jedincov.

Ak do novej generácie zahrniem niekoľko najlepších jedincov, ich gény nebudú zabudnuté pri nevhodnej voľbe rodičovských párov a zlom výsledku reprodukcie. No jedince by sa mohli vyberať aj ruletovým systémom.

Presný systém výberu spresníme pri testovaní.

5. Podrobný návrh

Poznámka: Táto kapitola bola revidovaná, revízia sa nachádza v kapitole Revízia č.1, v podkapitole Podrobný návrh.

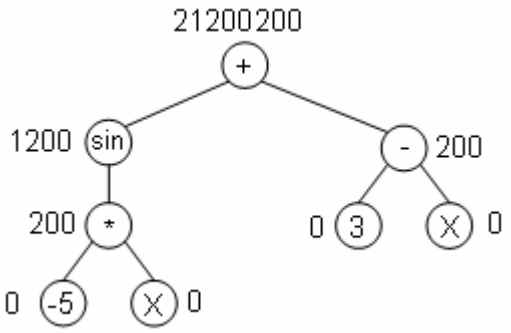
V tejto kapitole podrobnejšie rozoberáme návrh niektorých častí robota. Našou úlohou je preskúmanie evolučných algoritmov a možnosti ich aplikácie na učenie sa robota. Primárne sa teda zameriame na podrobnejší návrh evolučných algoritmov.

5.1 Návrh evolučného algoritmu

Na začiatku sa náhodne vygeneruje populácia jedincov. Veľkosť populácie sa určí experimentálne. Jedinci sa vykonajú a ohodnotia sa hodnotou fitness. Vyberú sa rodičovské páry, prebehne kríženie a mutácia. Potomkovia sa vykonajú a ohodnotia sa hodnotou fitness. Následne sa pridajú do pôvodnej populácie a z nej sa vytvorí nová populácia. Tým, že sa do novej populácie berú aj jedince zo starej populácie, sa zabráni strate dobrých génov pri zlom výbere rodičov. Ďalej sa už len bude opakovať cyklus od výberu rodičov, po vytvorenie novej generácie.
Pri vytvorení novej generácie sa jedinci uložia do súboru, pre prípad vypnutia počítača a pre umožnenie opätovného prezerania vyevolvovaných jedincov.

5.1.1 Reprezentácia jedinca

Jedinec bude reprezentovaný pomocou m-chromozómu. Ten je tvorený nešpecifikovaným množstvom génov. Gén bude obsahovať identifikátor správania, ktoré reprezentuje. Parametre správania budú tiež zapísané v géne, a to vo forme koreňových stromov. Koreňový strom reprezentuje funkciu. Všetky jeho nelistové vrcholy budú ohodnotené funkciami a listy operandami. Funkcie pre ohodnotenie vrcholov sa budú voliť z nasledujúcej množiny {+, -, *, /, sin, cos}. Táto množina sa počas testovania možno rozšíri o ďalšie funkcie. Operandy budú tvoriť konštanty, alebo premenné z hráčovho modelu sveta. Príklad koreňového stromu je na nasledujúcom obrázku .



Obr. 26 Koreňový strom s Readovým lineárnym kódom každého podstromu (čísla mimo stromu)

Koreňový strom bude reprezentovaný Readovým lineárnym kódom a vektorom ohodnotení vrcholov. Readov lineárny kód tvorí počet potomkov stromu a kódy potomkov. Napríklad strom na obrázku (č. 26) má kód 21200200. Prvé číslo je počet potomkov koreňa (2), nasledujú kódy potomkov (1200 a 200). Počet koreňových stromov závisí na množstve parametrov konkrétneho behavioru. Jednotlivé typy génov sa teda budú líšiť. Vo všeobecnosti bude gén obsahovať čas,

identifikátor správania a parametre správania (vo forme koreňových stromov).

5.1.2 Fitness

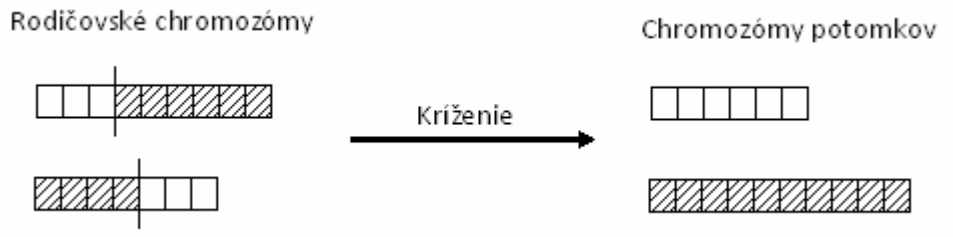
Funkcia fitness bude závislá od priblíženia sa ku očakávanej polohe hráča (pri učení pohybu), spotrebovanej energii a času, za ktorý sa tam dostal. Pravdepodobne to bude váhovaný súčet týchto parametrov.

5.1.3 Spôsob výberu rodičov

Rodičovské páry sa budú vyberať pomocou rulety podľa hodnoty fitness. Tento spôsob výberu simuluje náhodné udalosti v živote jedincov. Počet vybraných jedincov sa určí experimentovaním.

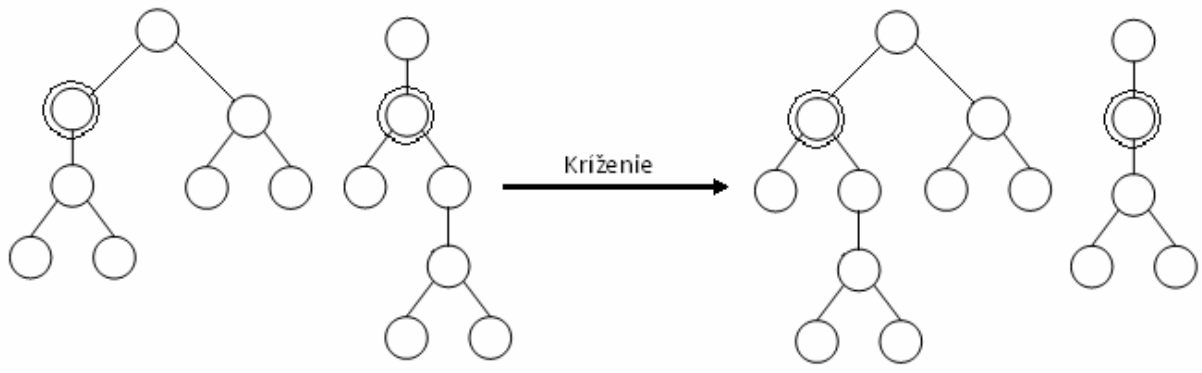
5.1.4 Operácie kríženia a mutácie

Dvaja jedinci sa budú krížiť jednoduchým jednobodovým krížením m-chromozómov. To znamená, že sa zvolia dva náhodné body kríženia (jeden v každom jedincovi) a gény za týmito bodmi sa vymenia. Mutácia jedinca bude pridanie alebo odoberanie génov.



Obr. 27 Kríženie dvoch m-chromozómov

Pri krížení jedincov sa vyberú aj náhodné gény z jedného rodiča. K ním sa následne nájdu gény rovnakého typu z druhého rodiča. Ak druhý rodič obsahuje takéto gény prebehne kríženie aj na nich. Gény sa budú krížiť tak že sa skrížia ich stromy. Krížiť sa budú stromy s rovnakým indexom. Teda tie, ktoré reprezentujú rovnaký parameter správania. Mutácia génu bude mutácia jeho stromov a zmena času tohto génu.
Kríženie koreňových stromov spočíva vo výmene ich podstromov. Vyberú sa teda dva podstromy (z každého stromu jeden) a tie sa navzájom vymenia (obr. 28). Mutácia stromu bude výmena náhodného podstromu za nový náhodne vygenerovaný strom.



Obr. 28 Kríženie dvoch koreňových stromov

5.1.5 Výber jedincov do novej generácie

Jedince do novej generácie sa budú vyberať ruletou podľa hodnoty fitness, alebo sa vyberú najlepšie jedince. Ruletový výber simuluje náhodné udalosti. No na druhú stranu výber najlepších jedincov zabráni strate dobrých génov zlým výberom.

5.1.6 Vykonanie jedinca

Vykonanie jedinca znamená postupné vykonanie pohybov, ktoré sú v jeho génoch. Každý gén jedinca reprezentuje určité správanie (určitý pohyb) v čase. Takže v určitom čase sa musia vykonať pohyby určené pre tento čas. Ak jedinec obsahuje viacero génov s rovnakým časom a správaním, vykoná sa len prvý z nich.
Jedince sa budú vykonávať sériovo a medzi vykonaním dvoch jedincov sa hráč odpojí a pripojí k serveru. Paralelizmus vykonávania hráčov spočiatku nebude možný, aby sa zabránilo interakcii hráčov. No v neskorších fázach by interakcia mohla pomôcť. Hráči by totiž mali reagovať na okolie a hlavne by to urýchlilo evolúciu.
Po vykonaní jedinca sa vypočíta jeho fitness.

6. Distribuovaná evolúcia

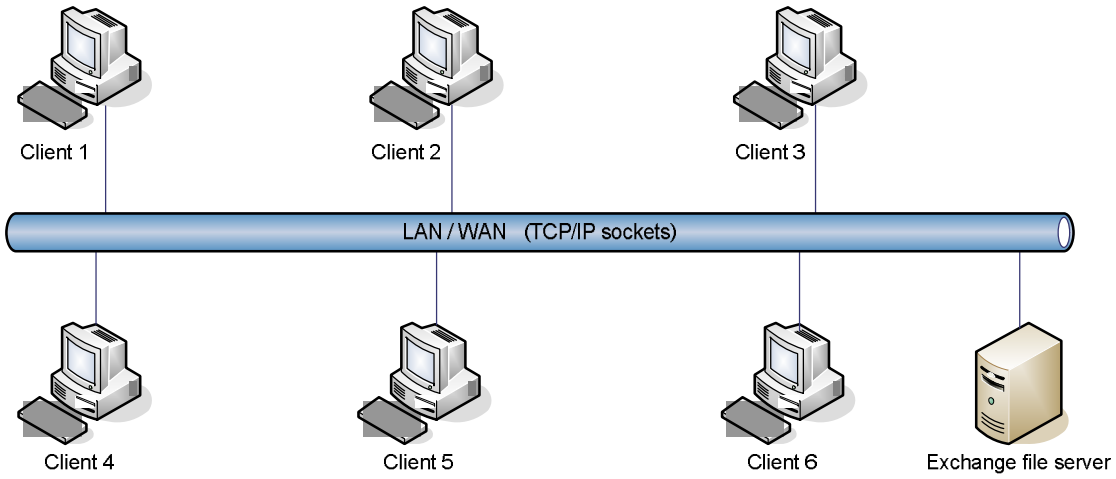
V tejto kapitole bude popísaný evolučný server ktorý sme vytvorili, a spôsob distribúcie evolúcie hráča.

6.1 Evolučný server

Pod týmto pracovným názvom sa rozumie klient - server aplikácia zabezpečujúca prenos dát / súborov cez počítačovú sieť s uchovávaním verzií prenášaných súborov. Aplikácia je prispôsobená požiadavkám evolujúceho agenta:

- Odoslanie a príjem migrujúcich jedincov medzi agentmi.
- Odoslanie a príjem zálohy celej vlastnej generácie.

Server časť sieťovej aplikácie je proces bežiaci na jednom z počítačov pripojených do spoločnej počítačovej siete, lokálnej alebo do internetu s prislúchajúcimi nastaveniami, resp. zabezpečenie prepojenia akýmkoľvek spôsobom, podmienkou je IP adresácia, TCP/IP protokol v komunikácií.



Obr. 29 Fyzická topológia

Komunikácia po sieti je implementovaná pomocou *sockets*. Rozhodnutie využitia *sockets* vzniklo z požiadavky na multiplatformový kód aplikácie, rozšírenosti tejto API v rôznych platformách. *Sockets* zabezpečujú:

- Identifikáciu komunikujúcich koncových bodov pomocou IP adresy
- Spustenie servisu pod ľubovoľným portom, pokiaľ nie je obsadený
- Naviazanie spojenia
- Čakanie na prichádzajúce spojenie
- Odoslanie a prijatie dát
- Uzatvorenie spojenia
- Odchytenie chýb

Socket je abstraktná reprezentácia komunikácie medzi dvoma koncovými bodmi. *Sockets* pracujú pod zmyslom Unix I/O, kde všetko je súbor a prepojenia existuje na princípe rúr a zásobníkov, t.j. po zavedení soketu je naň určená referencia v *Descriptor Table*. Na rozlíšenie od normálneho súboru existuje *Socket Descriptor Data Structure* so štruktúrou:

- *Fimaly*, typ komunikačného protokolu
  - Použitá je AF_INET, t.j. internetwork: UDT, TCP
- *Service*, spôsob reprezentácie prenášaných dát
  - Použitý je SOCK_STREAM, sekvencia bytov vkladaných do rúry.
- *IP address*
- *Port*

Vytvorenie soketu

```
SOCKET socket(

  int af,

  int type,

  int protocol

);
```

Kde *af* = AF_INET, *type* = SOCK_STREAM a *protocol* = IPPROTO_TCP (hodnota 6), kde *protocol* špecifikuje špecifický protokol. Hodnotu IPPROTO_TCP je možné použiť len v prípade *af* = AF_INET (resp. AF_INET6) a *type* = SOCK_STREAM!

Návratová hodnota je *Socket descriptor*, alebo -1 pri chybe.

Spustenie soketu

```
int bind(

  SOCKET s,

  const struct sockaddr* name,

  int namelen

);
```

V stručnosti ide o priradenie vytvoreného soketu k sitovému bodu. Program komunikuje s API soketu a tento soket treba prepojiť s network API. To sa deje pomocou bind príkazu, kde parameter je *socket descriptor,* dátová štruktúra obsahujúca IP adresu sieťového bodu, port a iné.
Funkcia vracia 0 po úspešnom pripojení, -1 v prípade neúspechu.

Počúvanie prichádzajúceho spojenia

```
int listen(
```

```
  SOCKET s,

  int backlog

);
```

Parametrom je vytvorený spustený soket. V prípade neúspechy vracia -1, ináč nula.

Čakanie na spojenie (server strana)

```
SOCKET accept(

  SOCKET s,

  struct sockaddr* addr,

  int* addrlen

);
```

Proces / vlákno volajúce túto funkciu čaká, sa uspí do dobu príchodu dát do soketu inicializujúcich spojenie. Parametrom je soket čakajúcej strany, dátová štruktúra obsahujúce info o pripojujúcej druhej strany. Funkcia vracia *socket descriptor*, cez ktorý je možné komunikovať s pripojujúcou sa druhou stranou.

Pripojenie sa (klient strana)

```
int connect(

  SOCKET s,

  const struct sockaddr* name,

  int namelen

);
```

Ako parameter je vytvorený soket a ten sa nepripojí na lokálny sieťový bod bind(), ale na bod čakajúci na spojenie. Jeho IP a port sa definuje s dátovej štruktúre ktorá je ako parameter.

Odoslanie dát

```
int send(

  SOCKET s,

  const char* buf,

  int len,

  int flags

);
```

Parametrom je už živý soket na ktorý sa odošlú dáta určitej veľkosti. Návratová hodnota je počet skutočne odoslaných dát, ak -1 tak v spojení nastala chyba.

Príjem dát

```
int recv(

  SOCKET s,

  char* buf,

  int len,

  int flags

);
```

Pokiaľ soket ktorý je parametrom je živý, t.j. je nadviazané spojenie, tak pri volaní tejto funkcie sa proces / vlákno dostane do stavu čakania pokiaľ neprime dáta z určeného soketu. Tieto dáta sa uložia do pamäte (veľkosti tretieho parametra), na ktorú ukazuje druhý parameter. Tým sa určí maximálna počet dát, ktoré sa dajú naraz prijať a spracovať v jednom volaní tejto funkcie. Funkcia recv() vracia počet skutočne prijatých dát (>0), alebo v prípade 0 ide o korektné ukončenie spojenia a ak -1, tak v spojení nastala chyba.

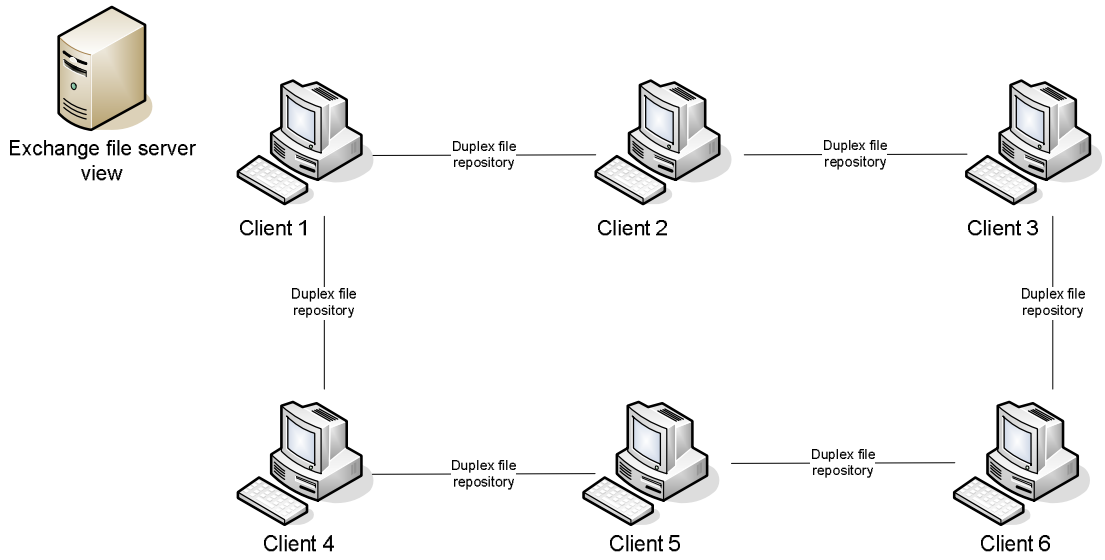
Uzatvorenie soketu

```
int closesocket(

  SOCKET s

);
```

Uzatvorenie daného soketu.

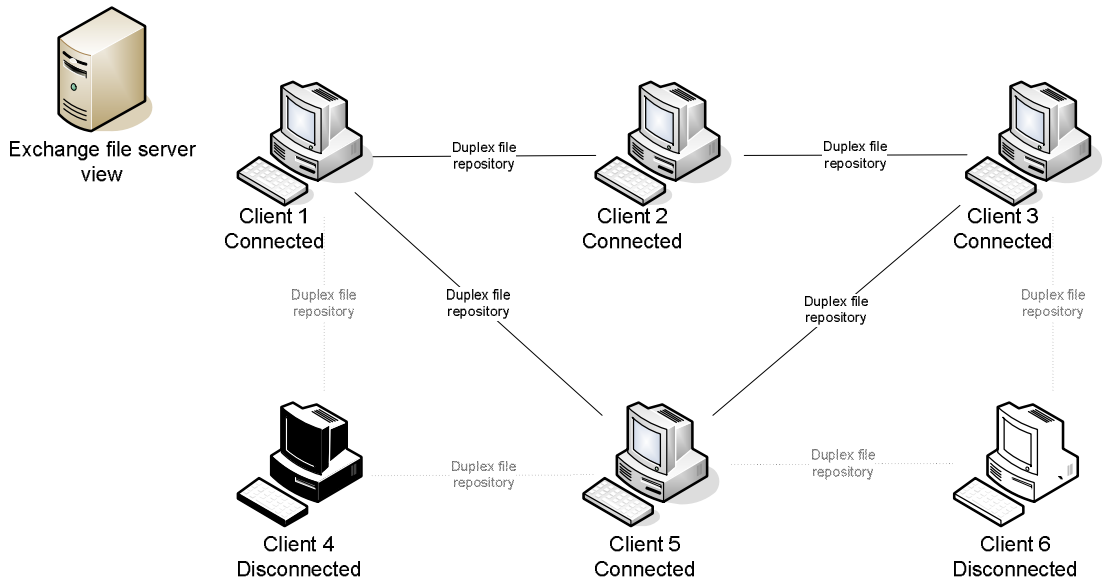
6.1.1 Činnosť serveru

Úlohou serveru je prijímať spojenia viacerých klientov, ktorých počet nie je vopred známi. Musí udržiavať spojenie medzi sebou a nimi. Udržiava si logickú mriežku, podľa ktorej sú klienti navzájom prepojený a odosielajú si navzájom migrujúcich jedincov. Pre jednoduchosť, prvá implementovaná mriežka je kruh podľa obrázku **Chyba! Nenašiel sa žiaden zdroj odkazov.**. Klient 2 prijíma migrujúcich jedincov od klienta 1 a klienta 3. Migrujúci jedinci od klienta 2 sa distribuujú pre klienta 1 a klienta 3.



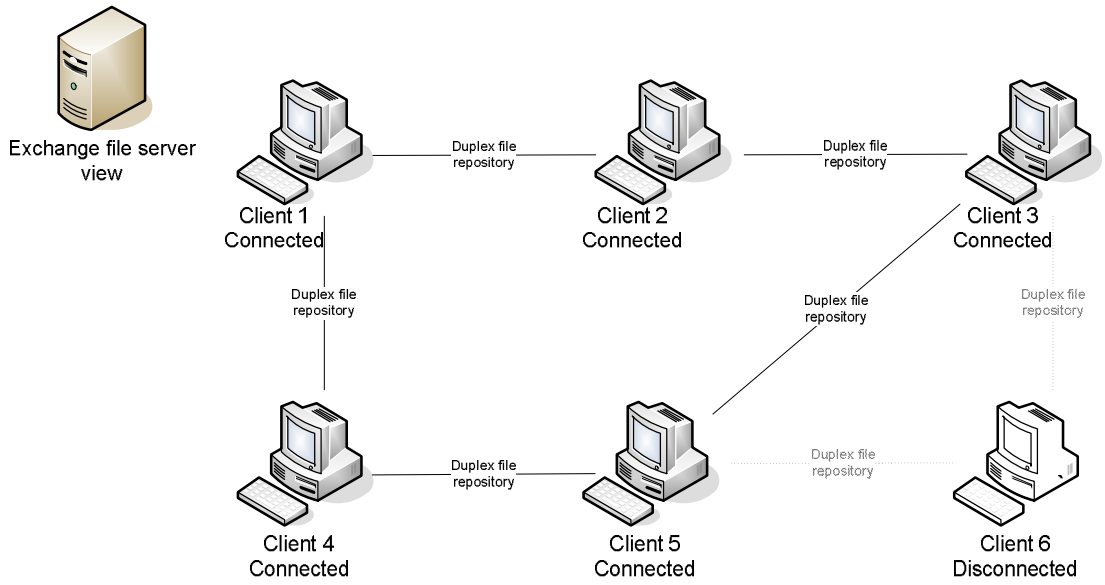
Obr. 30 Logická topológia, prepojenia klientov

Server si uchováva informáciu o všetkých niekedy pripojených klientoch a aj o súčasne pripojených, t.j. pokiaľ sa jeden z klientov odpojí, zostáva naďalej v mriežke, no prepojenia medzi pripojenými klientmi sa upravia tak, aby obišli odpojeného a nečakali od neho nové dáta, ktoré neprídu. Na obrázku **Chyba! Nenašiel sa žiaden zdroj odkazov.** je vidieť zmenu od obrázku **Chyba! Nenašiel sa žiaden zdroj odkazov.**, kde sa klient 4 a klient 6 odpojili a následne sa pripojenia medzi pripojenými zmenili.



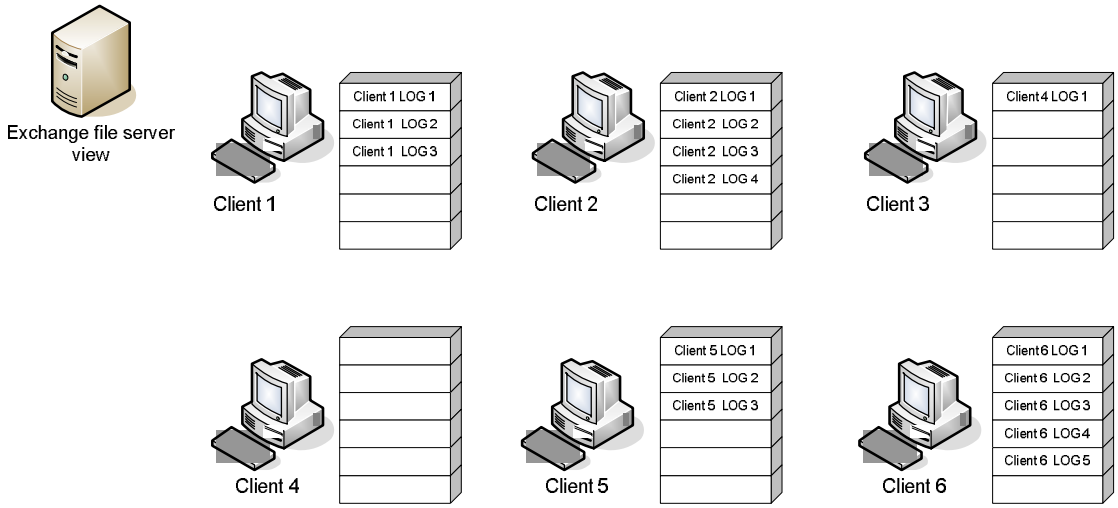
Obr. 31 Logická topológia, prepojenia klientov, dvaja sa odpojili (4 a 6)

Na obrázku **Chyba! Nenašiel sa žiaden zdroj odkazov.** je znázornená situácia, keď sa klient opätovne pripojí a jeho pôvodné miesto v mriežke sa obnoví a prepojeniami medzi susednými klientmi.



Obr. 32 Logická topológia, prepojenia klientov, klient 4 sa opäť pripojil

Ďalej server ponúka možnosť pre klientov uchovávať nimi odosielané zálohy generácií a ich opätovné vrátenie podľa požiadavky. Server si vnútorne uchováva verzie logov (generácii) ako histogram vývoja populácie. Pre možnosť dlhého chodu aplikácie, keby by logy mohli zahltiť voľný priestor na pevnom disku, je definovaný maximálny počet logov (v kóde napevno na hodnotu 100), ktorý keď sa presiahne, počítadlo sa nastaví na štartovaciu hodnotu a logy sa začnú prepisovať.



Obr. 33 Logovanie generácií pre každého klienta separovane

Jedinečná identifikácia klienta

Keďže táto klient – server aplikácia môže bežať aj cez internet a každý klient sa môže nachádzať v inej podsieti, resp. viacerý klienti sa môžu nachádzať pod rovnakou podsieťou, t.j. navonok vystupujú pod rovnakou IP. Identifikácia podľa IP by spôsobila,

že byť v logickej typológii serveru, v jeho mriežke by títo klienti si navzájom prepisovali odoslané dáta. Preto sú identifikovaný podľa ich MAC adresy, jedinečnej pre každú sieťovú kartu na svete.

Server obnova

Po vypnutí serveru sa dáta na pevnom disku uchovajú, ale vnútorná mriežka servera sa stratí a po opätovnom zapnutí servera hoci log súbory pre konkrétnych klientov existujú, ale server o nich nevie. Na obnovu mriežky existujú v adresárovej štruktúre klientov pomocné súbory na obnovu stavu serveru. Zámer je v tom, aby po reštarte servera boli logovacie súbory pre klientov opäť serverom prístupné po sieti. Na migrujúcich jedincov uchovaných na disku servera sa ohľad neberie, tí budú prepísaný. Pomocné súbory:

- InfoClients.inf
  - Umiestnenie: ServerDir/CLIENTS
  - Po pripojení klienta na server sa obsah súboru doplní o jeho IP, MAC a pre rýchlosť a jednoduchosť parsovania súboru aj cestu k dátam daného klienta
- LogInfoFile.inf
  - Umiestnenie: ServerDir/CLIENTS/_KlinetIP_/_KlientMAC_
  - Umiestnenie je cestou získanou z prvého súboru. Súbor je pomocný, obsahuje poslednú verziu log súboru, aby sa nemusel prehľadávať daný adresár.

Server kontroluje či pomocné súbory existujú, poskladá si názov log úboru podľa MAC adresy klienta a poslednej verzie získanej zo súboru LogInfoFile.inf a tento súbor tiež kontroluje na existenciu. Čiže po zásahu do týchto súborov sa vie server vysporiadať tak, aby bol plne funkčný. Obnova vždy prebieha pri spúšťaní serveru. Jeho preskočenie sa dá vykonať pridaním jedného prepínača v konzole pri spúšťaní servera popísaného v používateľskej príručke. Prípadne pred spustením vymazať adresár CLIENTS.

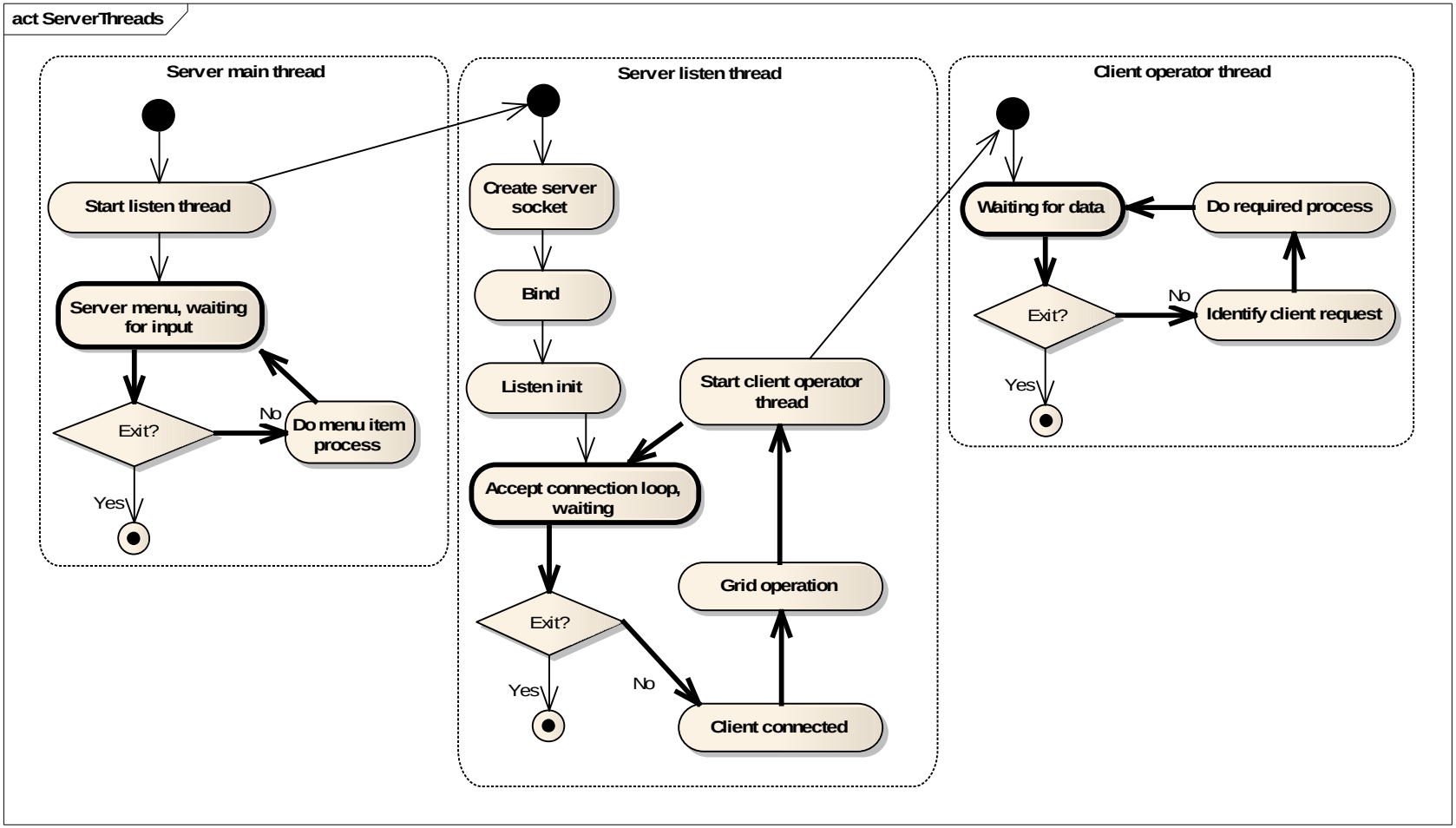
6.1.2 Viacvláknová obsluha klientov

Existujú tri typy vlákien v server časti aplikácie:

1. *Hlavný proces (Server mainl process)*
   Je to proces spustenej binárky servera, spúšťa proces / vlákno riadenia. Jeho cyklom je hlavné menu servera, čaká na vstup používateľa, t.j. voľba z funkcie z ponuky.
2. *Proces riadenia (Server listen thread)*
   Spúšťa sieťovú službu, jeho cyklom je obsluha pripojenia klienta, na ktoré čaká. Po pripojení realizuje operácie nad mriežkou, aby bola zachovaná vyššie popísaná funkčnosť logickej topológie. Spúšťa obslužný proces / vlákno klienta.
3. *Obslužný proces (Client operator thread)*
   Jeho cyklus spočíva v obsluhe prijatej požiadavky od klienta, na ktorú čaká.

Pomocou nižšie opísaného protokolu sa server orientuje v toku prijatých dát, ktoré môžu chodiť v rôznych dávkach, zato striktne v správnom poradí, čo zabezpečí TCP protokol, nad ktorým *Seckets* pracujú.

Obrázok **Chyba! Nenašiel sa žiaden zdroj odkazov.** znázorňuje činnosť jednotlivých procesov, ich cykly, miesta čakania a miesta ukončenia.



Obr. 34 Činnosť serveru, procesný pohľad

6.2 Klient evolučného serveru

Požiadavky na klienta boli také, aby pracoval sekvenčne, t.j. nebolo treba riešiť synchronizáciu medzi viacerými vláknami. Musí pracovať tak, že pokiaľ server nebude bežať, alebo nastane chyba pri spojení, nesmie zatuhnúť, ináč by jeden agent v rámci evolúcie neprispieval svojimi výsledkami do hľadaného výsledného jedinca.
Klient obsahuje nasledovné funkcie:

- Pripojenie na server
- Odoslanie migrujúceho jedinca
- Príjem migrujúcich jedincov
- Odoslanie generácie na archiváciu
- Príjem archivovanej generácie
- Odpojenie od servera

Každá z operácii je inicializovaná dátami protokolu, ktorý je nižšie popísaný.

6.3 Info-klient evolučného servera

Je to klient slúžiaci na odoslanie požiadavky a následný príjem informácii zo servera, ktoré obsahujú počet pripojených klientov (alebo odpojených), aktuálna verzia migrujúceho a log súboru, IP a MAC adresy klientov. Info-klient je samostatná aplikácia ktorá vzdialene získava informácie zo servera v sieti. Info-klient bol implementovaný aj priamo do serveru, takže obsluha sieťovej aplikácie ak má fyzický prístup k serveru, tak prostredníctvom jeho menu si môže nechať vypísať rovnaké informácie, aké by dostal vzdialeným pripojením sa info-klientom.

```
Server info
1) IP: 192.168.1.11, MAC: 0013cea03e75, MigrVer: 1, LogVer: 3, Disconnected
```

Obr. 35 Výpis info-klienta

Na obrázku Obr. 35 je príklad výpisu info-klienta, obsahuje:

- Poradové číslo klienta
- IP adresu klienta
- MAX adresu klienta
- Verziu migrujúceho jedinca
- Verziu logu
- Príznak, či je klient pripojený, alebo nie.

Protokol

Pri SOCK_STREAM spojení bolo potrebné zaviesť poriadok v odosielaných dátach, resp. ich riadenú správu. Tok dát medzi klientom a serverom nie je konštantný, najme keď komunikácia prebieha cez internet. Funkcia soketu recv() disponuje parametrom dátového zásobníka určitej veľkosti, do ktorého sa prichádzajúce dáta zapisujú. Ak je spojenie obmedzené, slabá prenosová rýchlosť, vysoká chybovosť, TCP protokol sa v pozadí stále snaží doručiť všetky dáta v správnom poradí. O túto skutočnosť sa implementácia aplikácie opiera spolu s odchytávaním výnimiek (odpojenie druhej strany {0}, fatálna chyba v spojení {-1}). Odoslané dáta sa do pomysleného dátovodu po sieti spájajú / pripájajú na koniec fronty, t.j. ak by server nestíhal prijímať prvý

odoslaný súbor a klient by odoslal ďalší, tak v reťazci prijímaných dát by za posledným bajtom prvého súboru hneď nasledoval prvý druhého súboru a funkcia recv() nevie, kde sa ukončuje prvé odoslanie a kde začína druhé, ona jednoducho prijíma, čo jej na vstup príde.

Zavedený bol teda protokol, dohoda medzi klientom a serverom, režijné dáta v prúde dát medzi týmito dvoma koncovými bodmi tak, aby počty odoslaných a prijatých dát vždy na každej strane komunikácie sedeli a keď nie, vyvodiť výnimku, ošetriť smerom k normálnemu chodu aplikácie. Protokol definuje typy operácií medzi klientom a serverom. Dáta protokolu / réžie obsahujú:

- Typ operácie
- Názov súboru
- Dĺžka súboru
- Kontrolný znak

Typ operácie	Názov súboru	Dĺžka súboru	Kontrolný znak
1 bajt	32 bajtov	8 bajtov	1 bajt

Typ operácie

Definuje, čo sa má v ďalších poliach protokolu očakávať a aké dáta budú za protokolom nasledovať. Typy operácii sú nasledovné:

Typ / číslo operácie	Kto odosiela	Popis
0	Klient, server	Odoslanie súboru / bloku dát, znakov
1	Klient	Požiadavka na príjem migrujúcich jedincov iných agentov
2	Klient	Odoslanie migrujúceho jedinca
3	Klient	Inicializácia klienta, odoslanie jeho MAC adresy
4	Klient	Odoslanie logu / generácie
5	Klient	Požiadavka na príjem logu / generácie
6	Info server	Info klient v rámci serveru, požiadavka o výpis
7	Info klient	Požiadavka o výpis
8	Server	Hlavný proces informuje riadiaci, že ukončuje činnosť, t.j. korektné vypínanie serveru
9	Klient	Slušné informovanie servera, že sa ide odpojiť

Operácia 0, v pole *Názov súboru* sa naplní skutočným názvom súboru z pevného disku, aby na oboch stranách boli totožné názvy. Toto je ale pre evolujúceho klienta zbytočné, pretože on si žiadne dáta lokálne neodkladá (v zmysle komunikácie so serverom), ale táto možnosť je vhodná pri hľadaní chýb, alebo iných aplikáciách, ktoré túto klient – server aplikáciu budú využívať. Dĺžka súboru v bajtoch sa uloží do ďalšieho poľa protokolu.

Operácia 7, odoslanie výpisu už nie je režírované protokolom. Server na základe požiadavky odošle na daný soket výpis a následne uzavrie korektne spojenie, takže na strane servera nenastane zatuhnutie, prijme dát a potom informáciu o uzavretí spojenia.

Operácia 8, server posiela sám sebe dáta cez soket, aby bola aplikácia korektne ukončená, tak sa musí uvoľniť čakanie na prichádzajúce spojenie a to práve tak, že prichádzajúce spojenie sa vytvorí (sám na seba) a cez protokol sa dozvie, že sa má riadiaci proces ukončiť.

Názov súboru je obmedzený na 32 bajtov, t.j. s bodkou a príponou, ak existuje.

Dĺžka súboru je odosielaná znakovo, čo je nie úsporné na odosielané dáta, ale implementácia bola

jednoduchšia a rýchlejšia oproti implementácie binárneho čísla. Teraz najväčšia hodnota je osem miestne číslo 99 999 999, čiže takmer 100 MB (104857600 B) maximálne veľkého prenášaného súboru z pohľadu protokolu. Toto obmedzenie na veľkosť súboru je veľmi vzdialené od veľkosti súborov prenášaných počas chodu evolúcie.

Kontrolný znak, bol zavedený ako poistenie správnosti prijatia protokolu. Jeho hodnota je '@', po prijatí všetkých 42 bajtov protokolu sa kontroluje posledný na prítomnosť znaku '@'. Teoreticky nemožná situácia je, že by avizovaná dĺžka súboru v protokole bola menšia ako skutočný počet odoslaných dát, no napriek tomuto výroku sa testuje správnosť protokolu, lebo ak by táto situácia nastala, prijímateľ by očakával protokol, ale pritom by bol prijímaní zvyšok súboru. Pravdepodobnosť že daný znak bude na svojom mieste aj v tom to prípade je veľmi malá. Oveľa menšia pravdepodobnosť je, že by táto situácia vôbec nastala, jedine cielene treťou narušiteľskou stranou.

Stav

Kód aplikácie je písaný pre GNU Linux a Windows OS. Počas implementácie bola aplikácia testovaná najme pre GNU Linux, pod ktorým táto aplikácia bude na konci našej tímovej práce bežať. Pod Windows OS bola aplikácia využívaná na debugovacie účely. Keďže robocup3D agent je aplikácia pod GNU Linux a jeho súčasťou je klient ExchangeFile aplikácie, je v tomto zmysle vhodné uvažovať len o multiplatformovej server časti aplikácie. Pre tento účel treba aplikáciu doladiť nastaveniami winsocket2, v súčasnej verzií aplikácie je použitie serveru pod Windows OS nepoužiteľné, ale možné s menšou námahou naštudovania a zavedenia nastavení, kód aplikácie je kompilovateľný pod oboma OS.

7. Prototyp

Ako prototyp sme zvolili vytvorenie jednoduchého agenta, ktorý bude vedieť komunikovať zo serverom a ukázať funkčnosť modelov udalostí. Jeho modely nebudú ešte spĺňať užitočnú funkciu.

Ciele:

1. Prvým cieľom je preukázať, že nami zvolená kombinácia častí jednotlivých modulov a ich architektúra je implementovateľná.
2. Ďalším cieľom je overenie správnosti komunikačnej vrstvy prevzatej z agenta Zigorata. Treba overiť, či je vrstva kompletná, teda či obsahuje spracovanie všetkých možných príkazov a správ.
3. Posledným cieľom je otestovanie systému modelov udalostí prevzatého z agenta Hazarda, či je dostatočný pre nové prostredie a jeho požiadavky, alebo potrebuje rozšírenia.

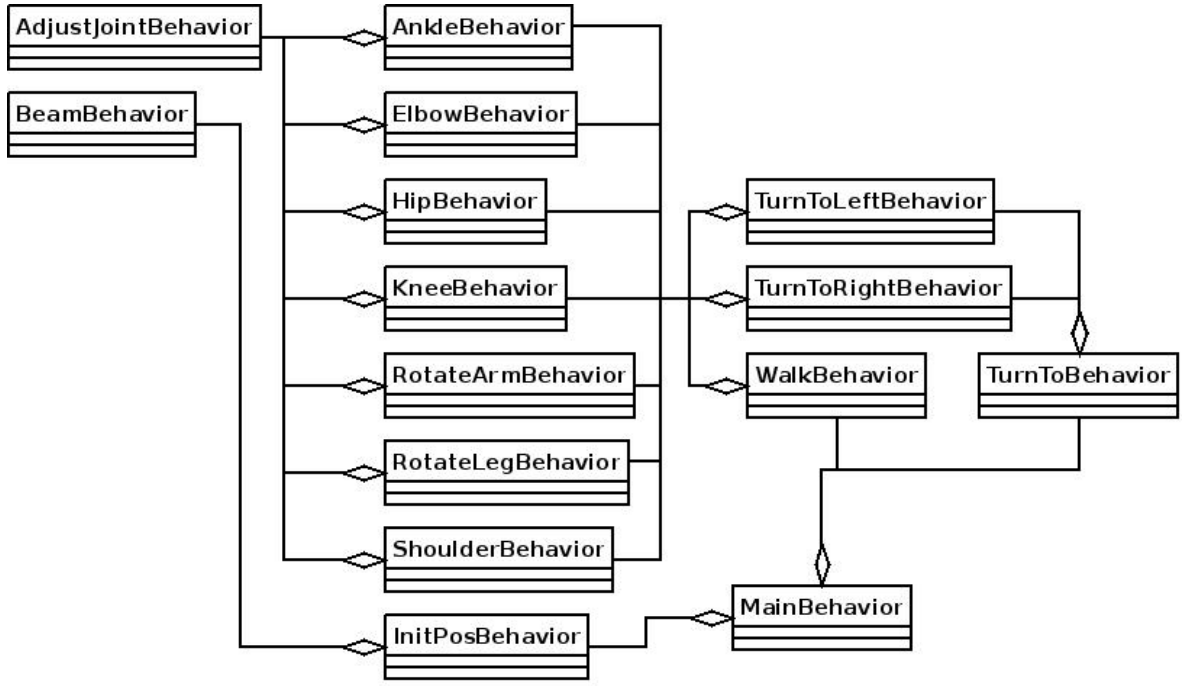
7.1 Inštalácia

K tomuto postupu kompilácie hráča je potrebné mať nainštalované vývojové prostredie Eclipse s podporou pre C++ (v operačnom systéme Ubuntu balíky *eclipse*, *eclipse-cdt*). Najprv je potrebné vytvoriť nový prázdny C++ projekt s názvom *NeuroticAgent*. Eclipse vytvorí k novému projektu aj rovnomenný adresár, ktorý nájdeme vo *workspace* adresári v ktorom sme projekt vytvorili. Do tohto adresára skopírujeme zdrojové kódy hráča. V eclipse, potom stačí zvoliť *Project-> Build Project*. Skompilovaného hráča nájdeme v podadresári projektu, s názvom *Debug*. Pre jeho spustenie je potrebné mať spustený server.

7.2 Štruktúra hráča

Hráč je rozložený do modulov – správaní („behaviors“). Tie majú rôzne úrovne a hierarchicky sa navzájom volajú. Správania sú triedy C++. Okrem BasicBehavior, má každá z nich má funkciu *behave*, ktorá je volaná pri aplikovaní daného správania.

Prototyp obsahuje základné motorické moduly správania a jeden ukážkový modul z vyššej vrstvy, ktorý vykonáva chodenie(iba staticky naprogramované). Štruktúra previazanosti správaní v prototype je nasledovná:



Obr. 36 UML diagram štruktúry správaní

7.3 Činnosť hráča

Pri spustení hráča sa hráč pripojí k serveru a zavolá sa hlavný modul správania – MainBehavior. Ten potom ďalej rozhoduje, čo sa bude diať. V prototype sa tento modul snaží otočiť smerom k súperovej bránke a napredovať k nej pomocou modulov TurnToBehavior a WalkBehavior. Tie zas volajú ďalšie moduly pre základné motorické funkcie.

7.4 Popis správaní

1. ***AdjustJointBehavior*** – Podľa zadaných hodnôt priamo nastaví uhlovú rýchlosť kĺbu.

   vstup: ID kĺbu a dva uhly určujúce jeho natočenie

   výstup: TRUE ak boli zadané hodnoty akceptované, v opačnom prípade FALSE

2. ***AnkleBehavior*** – Nastaví uhol členku.

   vstup: voľba medzi pravou/ľavou nohou, dva uhly určujúce jej natočenie, rýchlosť akou sa zmena má vykonať

   výstup: TRUE ak boli zadané hodnoty akceptované, v opačnom prípade FALSE

3. ***BasicBehavior*** – Základná trieda hráča, z ktorej dedia všetky ostatné triedy, nemá žiadnu špeciálnu funkcionalitu.

4. ***BeamBehavior*** – Vysiela *beam* príkaz serveru. *Beam* príkaz prenesie hráča na požadovanú pozíciu (nie počas hry).

vstup: X a Y súradnice, uhol – smer natočenia hráča

výstup: TRUE ak boli zadané hodnoty akceptované, v opačnom prípade FALSE

5. ***ElbowBehavior*** – Nastaví uhol lakťa.

   vstup: voľba medzi pravou/ľavou rukou, uhol určujúci jej natočenie, rýchlosť akou sa zmena má vykonať

   výstup: TRUE ak boli zadané hodnoty akceptované, v opačnom prípade FALSE

6. ***HipBehavior*** - Nastaví uhol bedrového kĺbu.

   vstup: voľba medzi pravou/ľavou nohou, dva uhly určujúce jej natočenie, rýchlosť akou sa zmena má vykonať

   výstup: TRUE ak boli zadané hodnoty akceptované, v opačnom prípade FALSE

7. ***InitPosBehavior*** – Initalizuje hráča a nastaví všetko potrebné pre jeho štart.

8. ***KneeBehavior*** - Nastaví uhol kolena.

   vstup: voľba medzi pravou/ľavou nohou, uhol určujúci jej natočenie, rýchlosť akou sa zmena má vykonať

   výstup: TRUE ak boli zadané hodnoty akceptované, v opačnom prípade FALSE

9. ***MainBehavior*** – Trieda rozhodujúca na základe zložitejších procedúr o nasledujúcej postupnosti akcií. V prototype táto trieda vykonáva samotné chodenie – postupnosť jednotlivých menších elementárnych akcií súvisiacich s chodením.

10. ***RotateArmBehavior*** - Nastaví uhol otočenia ramena.

    vstup: voľba medzi pravým/ľavým ramenom, uhol určujúci jej natočenie, rýchlosť akou sa zmena má vykonať

    výstup: TRUE ak boli zadané hodnoty akceptované, v opačnom prípade FALSE

11. ***RotateLegBehavior*** - Nastaví uhol otočenia nohy.

    vstup: voľba medzi pravou/ľavou nohou, uhol určujúci jej natočenie, rýchlosť akou sa zmena má vykonať

    výstup: TRUE ak boli zadané hodnoty akceptované, v opačnom prípade FALSE

12. ***ShoulderBehavior*** - Nastaví uhol plecom.

    vstup: voľba medzi pravým/ľavým plecom, dva uhly určujúce jeho natočenie, rýchlosť akou sa zmena má vykonať

    výstup: TRUE ak boli zadané hodnoty akceptované, v opačnom prípade FALSE

13. ***TurnToBehavior*** – Otočí hráča na požadovaný uhol.

    vstup: uhol, o ktorý sa má hráč otočiť

    výstup: TRUE ak boli zadané hodnoty akceptované, v opačnom prípade FALSE

14. ***TurnToLeftBehavior*** – Otočí hráča požadovaným na požadovaný uhol smerom doľava.

    vstup: uhol, o ktorý sa má hráč otočiť smerom doľava

    výstup: TRUE ak boli zadané hodnoty akceptované, v opačnom prípade FALSE

15. ***TurnToRightBehavior*** - Otočí hráča požadovaným na požadovaný uhol smerom doprava.

    vstup: uhol, o ktorý sa má hráč otočiť smerom doprava

    výstup: TRUE ak boli zadané hodnoty akceptované, v opačnom prípade FALSE

16. ***WalkBehavior*** – Hráč kráča rovno v smere, v ktorom je otočený.

7.5 Záver

Implementáciou a otestovaním prototypu sme overili navrhnuté architektonické riešenie, funkčnosť a možnosť využitia existujúcej implementácie pri práci v ďalšom semestri.

7.5.1 Overenie architektonického návrhu agenta

Prototyp využíva modulárnu architektúru agenta *Hazard* a moduly správania, ktoré využívajú funkcionalitu agenta *Zigorat*. Relatívne bezproblémové integrovanie architektúry (*Hazard*) a dostupnej funkcionality (*Zigorat*) indikuje dobrý architektonický návrh agenta.

Najväčšou zmenou oproti modulárnemu systému agenta *Hazard* bolo nahradenie pôvodného spôsobu registrovania modulov správaní v systéme na úrovni implementácie. Pôvodné využitie frameworku *Zeitgeist* sme vyhodnotili ako zbytočné, nakoľko ani v agentovi *Hazard* sa jeho využitie zatiaľ neukázalo ako opodstatnené. Využitie jednoduchšej implementácie viedlo k eliminovaniu zbytočných závislostí a zjednodušenia na úrovni kódu, kompilácie, testovania. Ak v budúcnosti nájdeme dostatočné uplatnenie a vyhodnotíme jeho použitie ako opodstatnené, sme pripravený sa k tomuto frameworku vrátiť.

7.5.2 Overenie implementácie agenta *Zigorat*

- **Overenie komunikačnej vrstvy agenta *Zigorat*:** Komunikačná vrstva zabezpečovala odosielanie správ na server, prijímanie správ a obnovovanie informácií o okolitom svete. Overením na úrovni analýzy kódu, analýzy posielaných/prijímaných správ a samotnej funkcionality robota môžeme konštatovať, že táto vrstva pracuje správne.
- **Overenie funkcionality na vyššej úrovni:** Podobne sme overili moduly správania pre pohyb jednotlivých kĺbov, na vyššej úrovni moduly chodenia a otáčania.

Funkcionalita vytvoreného prototypu nepokryla komunikačný protokol úplne. Z nášho pohľadu je však najdôležitejšia overenie funkcionality, ktorú predpokladáme pri implementácii evolučného algoritmu (založený na pohybe kĺbov).

7.5.3 Overenie nefunkcionálnych požiadaviek na server

Ako už bolo spomenuté (*10.2.2 Komunikácia robota*), evolučný algoritmus bude vyžadovať opakované pripájanie/odpájanie sa na/od server/serveru v krátkych časových intervaloch. Z využitím prototypu sme tiež overili schopnosť servera zvládnuť spomenuté zaťaženie.

Prototypovanie z pohľadu splnenia cieľov považujeme za úspešné. Za dôležité považujeme hlavne overenie funkcionality súvisiacej s implementáciou evolučného algoritmu a overenie architektonického návrhu agenta. Prototypovaním sme minimalizovali riziko výskytu problémov pri implementácii v letnom semestri

8. Externé testovanie

Externé testovanie spočíva v testovaní nášho riešenia projektu inou, nezainteresovanou osobou. Vybrali sme si brata jedného člena nášho tímu, Michala Kolesára. Testovanie sa uskutočnilo 17. 05. 2008 v externom prostredí mimo softvérového štúdia na FIIT STU v Bratislave, kde sme aplikáciu spúšťali. Testerovi bolo stručnej vysvetlené, čo celkovo naša aplikácie robí, a akým spôsobom bude aplikáciu testovať. Náš evolujúci hráč je v progresívnom stave, takže nie je možné testovať jeho požadované výsledné správanie sa, ale testy sme zvolili na distribuovanú evolúciu, ktorú je vhodné použiť aj v budúcnosti, resp. naši nasledovníci po použití distribuovanej evolúcie by mali prísť rýchlejšie k výsledkom a testovaním chceme objaviť prípadné chyby, resp. potvrdiť bezchybnú funkcionalitu. Sledovanie generovaných jedincov je druhé testovanie: vhodnosť nastavení parametrov, či podľa nich postupne fitness stúpa, alebo nie.

Priebeh testovania

Testovanie prebehlo spôsobom overovania distribuovanej evolúcie, spustenie servera, pripájanie a odpájanie klientov. Sledovanie generovaných jedincov na monitore RoboCup 3D servera a hodnotenie priebežných výsledkov evolúcie. Boli spustených traja hráči, t.j. traja klienti na troch počítačoch lokálnej siete a zároveň na jednom z nich bol spustených server na distribúciu migrujúcich jedincov. Bol spustený na strane klienta zápis prijatých migrujúcich jedincov a logovaných generácií do súboru a testujúci sledoval adresáre jednotlivých klientov, či ich obsah korešponduje s výpismi a súbormi na strane servera, t.j. veľkosť a názov súborov. Testujúci využíval aj informácie z Info klienta o počte prenesených migrujúcich jedincov, počte logovaných generácií a informáciu o pripojení klienta. Na záver sme mu dali vyplniť dotazník, ktorý je uvedený v prílohe H.

Záver

Externé testovanie viac pobavilo nášho testujúceho, ako nás potešili výslední jedinci evolúcie. Po zhodnotení dojmov a uzáverov testujúceho pracovala distributívna evolúcia správne, mimo poznámky, že by bolo vhodné použiť GUI v tejto sieťovej aplikácií.

9. Zhodnotenie

Primárnou úlohou do druhého semestra bolo implementovať evolučný algoritmus a overiť ho na evolúcii niektorých pohybových úkonov.

Vstupom do tohto semestra bol prototyp, schopný vykonať základné pohyby (otočenia kĺbov). Predpokladom pre úspešnú evolúciu bolo správne fungovanie týchto základných pohybov. Vzhľadom na ich množstvo, sme sa rozhodli tento proces zautomatizovať, vytvoriť testovací *framework* a overiť funkcionalitu výstupu prvého semestra aj týmto spôsobom. Výstupy testovania nám okrem iného priniesli predstavu o štatistických odchýlkach pri pohyboch, ktoré sme označili ako prijateľné.

Evolučný algoritmus sme implementovali podľa návrhu a integrovali ako nové správanie hráča. Jeho funkcionalitu sme sa rozhodli overiť na evolúcii vstávania z polohy ležmo.

Vzhľadom na beh simulácie v reálnom čase sme sa dodatočne rozhodli implementovať distribuovanú evolúciu, ktorá by mala z teoretického hľadiska priniesť lepšie výsledky ako niekoľko nezávislých evolúcií. Tento krok však priniesol ďalšie nároky na implementáciu, integráciu a testovanie. Dôsledkom bol výrazný posun prác na projekte.

9.1 Testovanie a ladenie

Proces evolúcie je všeobecne náročný najmä na čas a to platilo aj v našom prípade. Už v prvom semestri sme preto na samotné ladenie a testovanie vyhradili dostatok času. S príchodom distribuovanej verzie sa tento čas skrátil a k pôvodným parametrom evolúcie (napr. fitnes funkcia, parametre kríženia a mutácie) pribudli ďalšie, ktoré bolo treba upraviť (napr. migrácia jedincov).

Simulácia sa ukázala ako náročná na výpočtové prostriedky a limitujúca vzhľadom na softvérové (*Linux*). Vykonávanie na osobných počítačoch teda nebolo vo väčšom rozsahu možné a evolúcia bola obmedzená najmä na priestory *softvérového štúdia*. Tento fakt so sebou priniesol hlavne problémy s manažmentom procesu evolúcie. Činnosti ako zber dát a kontrolu procesu sme tak vykonávali v niekoľkodňových intervaloch.

Ďalším z problémov boli vysoké požiadavky na stabilitu aplikácie. Evolúcia mala fungovať prakticky nepretržite. V tomto prípade bola problémom samotná stabilita simulačného prostredia a chyby našej časti aplikácie. Tieto sa však často prejavili až po niekoľkých hodinách behu. V tomto prípade bolo treba chybu odstrániť a simuláciu s výraznou časovou stratou spustiť znovu. Zotavenie sa procesu evolúcie sme čiastočne riešili opakovaným spúšťaním v slučke.

Problémy rôzneho charakteru sprevádzali najmä distribuovanú verziu evolúcie. Preto sme sa nakoniec rozhodli odladiť parametre jednoduchej evolúcie a až potom prejsť na distribuovanú verziu.

9.2 Výsledky

V súčasnosti sme vo fáze ladenia fitnes funkcie evolúcie. Podarilo sa nám určiť a upraviť niektoré kľúčové parametre. Medzi tieto môžeme zaradiť penalizáciu jedincov, ktorý porušujú stabilitu simulácie a parameter špecifikujúci výšku trupu jedinca pri vstávaní. Zvýšenie váh priradených týmto parametrom viedlo k „rozhýbaniu“ jedincov (dovtedy boli relatívne úspešnými aj tí, ktorí sa vôbec nehýbali).

Niektoré výsledky už boli prezentované v rámci prednášok s tematikou ***Robocup*** ako súčasť konferencie *IITSRC*.

9.3 Pokračovanie

Ďalšie možnosti postupu už boli naznačené. Je potrebné pokračovať v ladení evolúcie úpravou parametrov a prechod na distribuovanú verziu.

Pod priebeh evolúcie sa výrazne podpísala aj nestabilita simulačného prostredia, ktorú sme žiaľ neboli schopní ovplyvniť. Často nedeterministické rozpadávanie hráča sa ukázalo ako výrazne limitujúce. Prísľubom do budúcna je fakt, že vývoj simulačného prostredia stále napreduje. Od začiatku projektu boli vydané už dve nové verzie. Prvá priniesla minimálne úpravy, ale neriešila pre nás kritickú stabilitu. Druhá vyšla doslova pred pár dňami s výraznými zmenami. Prechod by si však vyžadoval netriviálne implementačné zásahy a testovanie.

Toto však budú zrejme úlohy pre ďalších potenciálnych pokračovateľov projektu. Bola by škoda ďalej nepodporiť takto dobre rozbehnutý projekt. Motiváciou by mohla byť možná účasť na súťaži zručnosti, alebo využitie evolvovaných pohybov priamo v hre. Na záver je dôležité poznamenať, že podľa našich informácií nemá použitie evolúcie podobným spôsobom v našej kategórii obdobu.

10. Revízia č.1

Táto kapitola vznikla ako reakcia na posudok tímu č.11 odovzdaný dňa 22.11.2007 na dokumentáciu k projektu. Doplnená je kapitola 2.2 o simulačnom prostredí a kapitola 5 obsahujúca špecifikáciu požiadaviek.

10.1 Kapitola 2.2 Simulačné prostredie

Do tejto kapitoly sme sa na základe posudku rozhodli zaradiť podkapitolu o všeobecnom popise servera ako aj monitora ktorý slúži na vizualizáciu zápasu v simulovanom 3d futbale.

10.1.1 Soccer Server

Server zabezpečuje simuláciu robotického futbalu. Simuluje hru hráčov ovládaných nezávislými programami (agentmi). Architektúra systému agentov, server je typu klient-server. Agent v tejto architektúre reprezentuje klienta. Reálne sa hry zúčastňuje niekoľko agentov, kde každý ovláda jedného hráča hrajúceho za jeden z dvoch súperiacich tímov.

Server plní najmä nasledujúce úlohy:

- Uchováva aktuálny stav hry (skóre, čas, ...) a aktuálny stav jednotlivých hráčov.
- Prijíma informácie od agentov a na ich základe upravuje stav príslušných hráčov na ihrisku (agent ovláda efektory hráča).
- Simuluje pohyb hráčov a lopty na základe fyzikálnych zákonov a následne upravuje ich stav.
- Stará sa o dodržiavanie pravidiel a vyhodnocuje štandardné situácie na ihrisku.
- Informuje agentov o aktuálnom stave príslušných hráčov a stave samotnej hry (agent prijíma informácie z perceptorov hráča).
- Informácie o stave hry a stave jednotlivých hráčov posiela monitorom (pozri kapitolu Monitor).

Na základe typu simulácie existuje niekoľko typov serverov robotického futbalu.

- 2D server: simulácie prebieha v 2D prostredí. Hráči sú reprezentovaný kruhmi s určitým polomerom.
- 3D server (Sphere): simulácia prebieha v 3D prostredí a hráči sú reprezentovaný guľami s určitým priemerom a hmotnosťou.
- 3D server (Humanoid): simulácia prebieha v 3D prostredí a hráči sú reprezentovaný humanoidmi. Ich telo pozostáva z niekoľkých častí navzájom prepojených kĺbmi.

V našom prípade sa venujeme simulácii s humanoidnými typmi hráčov a využívame preto príslušný server.

10.1.2 Monitor

Monitor predstavuje okrem agenta ďalší typ klienta komunikujúceho so serverom.

Monitor plní najmä nasledujúce úlohy:

- Prijíma zo servera informácie o stave hry a jednotlivých hráčov.
- Vizuálne reprezentuje simuláciu na základe prijatých informácií.
- Umožňuje zaznamenať simuláciu do tzv. simulačného logu.
- Umožňuje prehrať zaznamenanú simuláciu v simulačnom logu.

Protokol využívaný na komunikáciu so serverom môže byť využitý aj na implementáciu trénera. Protokol monitora totiž umožňuje ovplyvniť stav hry a hráčov na ihrisku a tým iniciovať rôzne herné situácie.

10.2 Kapitola 5. Špecifikácia požiadaviek

Táto kapitola rozširuje pôvodnú kapitolu č.5 na základe posudku tímu č.11.

10.2.1 Schopnosti robota

Robot bude mať len základné pohybové schopnosti orientácie v priestore. Primárne sa bude jednať o schopnosť postaviť sa, prípadne udržanie rovnováhy. Tieto schopnosti však nebudú robotovi implementované priamo. Bude sa ich učiť prostredníctvom rôznych evolučných algoritmov. Zašpecifikujeme len počiatočnú polohu a metriky ktoré budú robotovi hovoriť že dosiahol svoj cieľ. Zvyšok už necháme na evolučnom vývoji.

10.2.2 Komunikácia robota

Robot bude pracovať už na novom type servera vo verzií 0.5.6. Komunikácia bude prebiehať prostredníctvom TCP protokolu. Počas evolúcie sa bude robot opakovane pripájať na server. Pri každom pripojení sa vykoná jedna generácia v evolučnom procese. Výsledné hodnoty sa zapíšu do súboru a robot sa odpojí. Následne sa proces bude opakovať. Robot sa pripojí, zo súboru si načíta predošlú generáciu, prebehne evolúcia a nové hodnoty sa opäť uložia do súboru.

10.2.3 Architektúra robota

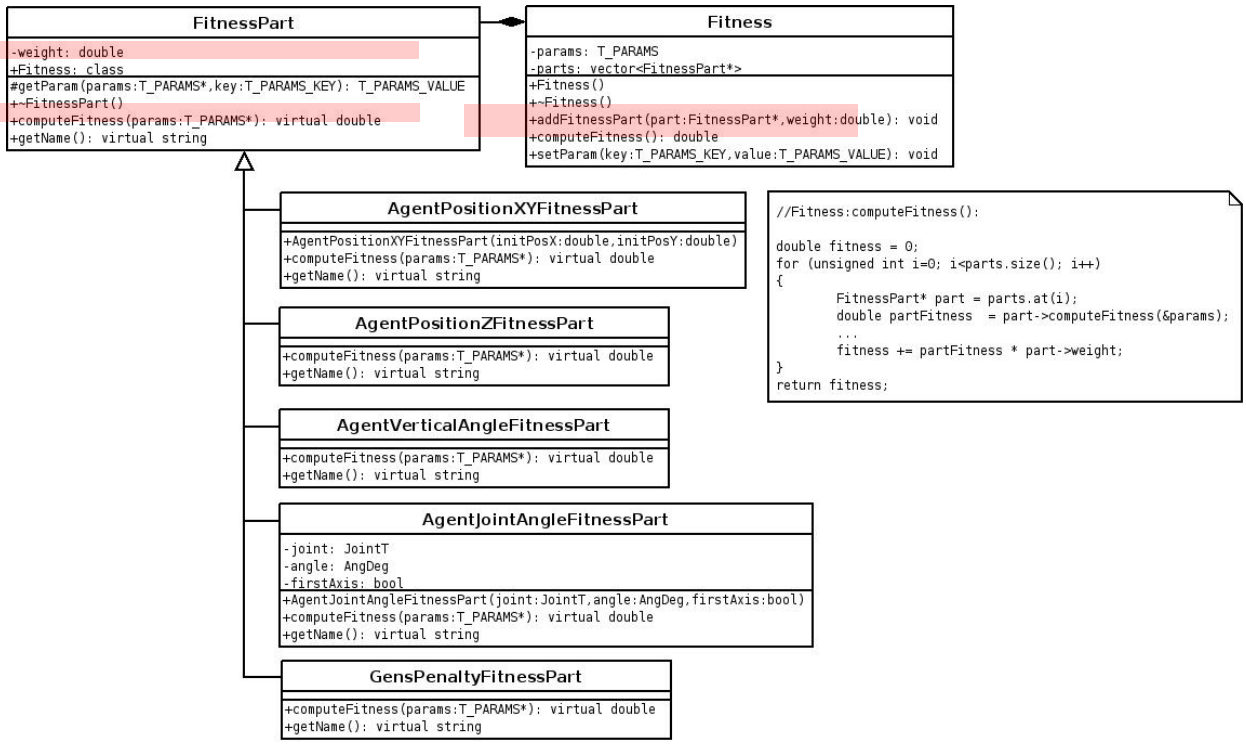
Architektúru robota sme sa rozhodli prebrať z minuloročného projektu tímu 6th sense. Táto architektúra sa vyznačuje dobrou škálovateľnosťou a rozšíriteľnosťou. Použitý typ robota bol však určený pre starší typ servera kde bol výzor hráča značne zjednodušený (guľa). Z tohto dôvodu sme sa rozhodli komunikačnú časť t.j. časť ktorá komunikuje so serverom prebrať od humanoidného robota Zigorata. Tento robot je určený na nový typ servera a jeho komunikačná časť je dobre rozpracovaná. Ďalej by sme chceli do zdrojového kódu vniesť štandardy aby bol prehľadnejší a ľahko čitateľný.

10.2.4 Špecifikácia evolúcie

Evolučné algoritmy simulujú evolúciu, ktorá prebieha v prírode. Tento proces by mohol poskytnúť možnosť, ako nechať hráča aby sám získal základné schopnosti. Plánujeme otestovať možnosť využitia evolučných algoritmov. Budú využité na naučenie základných pohybov ako napríklad postavenie sa zo zeme, chôdza a pod. Ak sa evolučné algoritmy ukážu ako efektívny prístup, môžeme sa pokúsiť aplikovať ich aj na naučenie zložitejších pohybov. Použijeme kombináciu metód evolučného programovania a m-chromozómov. Z evolučného programovania využijeme reprezentáciu funkcie v tvare „koreňového stromu". Ten bude reprezentovaný Readovým lineárnym kódom. M-chromozómy budú predstavovať jedincov premenlivej dĺžky.

10.2.5 Fitness funkcia

Na ohodnotenie evolvovaných jedincov slúži fitnes funkcia. V našom prípade má funkcia (*Fitness*) niekoľko zložiek (*FitnessPart*). Každá zložka sa podieľa na celkovej hodnote normovanou hodnotou v intervale <0;1> prenásobenou váhou fitnes zložky (*FitnessPart.weight*). Spôsob výpočtu celkovej fitnes je zrejmý z obrázka (obr. Obr. 37).



Obr. 37 Diagram tried fitnes funkcie.

Aktuálne je pri evolúcii možné využiť nasledujúce fitnes zložky:

- AgentPositionXYFitnessPart
  - Zohľadňuje vzdialenosť hráča od želanej pozície (v rovine *XY* ihriska).
  - Závislosť je nepriamo úmerná.
- AgentPositionZFitnessPart
  - Zohľadňuje *Z* súradnicu pozície hráča na ihrisku (výška trupu).
  - Závislosť je priamo úmerná (pre *Z<=maxVýškaTrupu*, nepriamo pre *Z> maxVýškaTrupu*, *maxVýškaTrupu* = výška trupu, pri ktorej je hráč vzpriamený).
- AgentVerticalAngleFitnessPart
  - Zohľadňuje uhol odklonenia vertikálnej osi hráča od normály ihriska (hráč stojí vzpriamene: uhol=0, hráč stojí vzpriamene na hlave: uhol=180).

  - Závislosť je nepriamo úmerná.
- GensPenaltyFitnessPart
  - Zohľadňuje počet penalizovaných génov.
  - Závislosť je nepriamo úmerná.
- AgentJointAngleFitnessPart
  - Zohľadňuje odklon natočenia kĺbu od želanej pozície.
  - Závislosť je nepriamo úmerná.

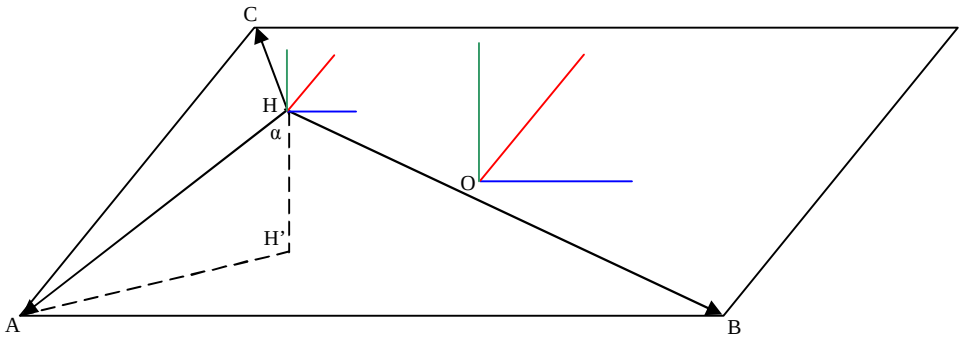
Pozícia hráča na ihrisku

Cieľom je vypočítať pozíciu hráča v súradnicovej sústave ihriska pomocou informácií poskytovaných perceptorom videnia.

Obr. 38 znázorňuje sústavu hráča (H) a sústavu ihriska (O). Algoritmus využíva na výpočet polohy 3 statické body na ihrisku (A,B,C), ktoré neležia na jednej priamke a ležia v rovine ihriska (tyčky bránky, rohové zástavky).

Postup (zjednodušený) výpočtu polohy:

1. A,B,C v sústave H (vstup)
2. **n** = **AB**[2] x **AC** (normála ihriska)
3. α = uhol(**n**, **HA**)
4. |HH'| = cos(α) (súradnica *Z* pozície hráča v sústave O)
5. |AH'| = sin(α) * |AH|
6. |BH'| = sin(α) * |BH|
7. |CH'| = sin(α) * |CH|
8. Vypočítané vzdialenosti priemetu hráča do roviny ihriska od troch bodov sa zachovávajú aj v sústave O.
9. Výpočet súradníc *X*,*Y* pozície hráča v sústave O prienikom kružníc (A v sústave O; |AH'|)[3], (B v sústave O; |BH'|), (C v sústave O; |CH'|).



Obr. 38 Ilustrácia algoritmu na výpočet pozície hráča.

Pôvodný algoritmus (prebraný z hráča *Zigorat*) počítal súradnice *X*,*Y* pozície hráča v sústave O ako jeden z prienikov kružníc (A v sústave O; |AH|), (B v sústave O; |BH|). Okrem relatívne nepresného výpočtu pôvodný algoritmus neposkytoval súradnicu *Z*.

Pozíciu hráča na ihrisku momentálne využívajú len niektoré zložky fitnes funkcie (pozri. *Fitnes*):

- *AgentPositionXYFitnessPart*
- *AgentPositionZFitnessPart*

[2] Vektor B-A.
[3] Kružnica so stredom A a polomerom |AH'|.

Okrem pozície samotného hráča používame rovnaký algoritmus na výpočet vektorov súradnicových osí sústavy H v sústave O. Postup výpočtu vektoru osy *X*:

1. BodNaOsyX = AlgoritmusNaVýpočetPozície(A-(1,0,0), B-(1,0,0), C-(1,0,0) v sústave H)
2. PozíciaHráča = AlgoritmusNaVýpočetPozície(A, B, C v sústave H)
3. **PozíciaHráča BodNaOsyX** (vektor osy *X* sústavy H v sústave O)

Takto je možné určiť presnú orientáciu hráča v sústave ihriska, čo sa využíva napríklad pri zložke fitnes funkcie *AgentVerticalAngleFitnessPart* (pozri *Fitness*).

Pozícia častí hráča

Cieľom je vypočítať pozíciu jednotlivých častí tela hráča (noha, ruka, ...) v sústave ihriska. Algoritmus vychádza z informácií o rozmeroch jednotlivých častí hráča a z natočenia príslušných kĺbov.

Informáciu o natočení kĺbov je možné získať z perceptorov. Rozmery častí tela hráča a ich vzájomné vzťahy (prepojenia kĺbmi) sú popísané v modely hráča (*soccerbot056.rsg*).

Nasledujúci úryvok popisuje vzťah ľavého ramena (*leftshoulder*) k trupu robota (*body*) a ich prepojenie kĺbom (*UniversalJoint*), s dvoma stupňami voľnosti (*setAxis1*, *setAxis2*). Stred kĺbu (osí otáčania) je od stredu ramena posunutý o vektor *setAnchor*. Stred ramena je od stredu trupu posunutý o vektor
(*- $UpperTorsoLength / 2.0 - $ShoulderLength / 2.0 - $TorsoCylinderLength / 2.0, 0, $UpperTorsoHeight / 2.0 - $ShoulderHeight / 2.0*).

```
(def $LeftShoulderPosX (eval $UpperTorsoPosX –
       (eval $UpperTorsoLength / 2.0) -
       (eval $ShoulderLength / 2.0) -
       (eval $TorsoCylinderLength / 2.0)
       )
)
(def $LeftShoulderPosY $UpperTorsoPosY)
(def $LeftShoulderPosZ (eval $UpperTorsoPosZ +
       (eval $UpperTorsoHeight / 2.0) -
       (eval $ShoulderHeight / 2.0)
       )
)

(nd Transform
       (setName leftshoulder)
       (setLocalPos $LeftShoulderPosX $LeftShoulderPosY $LeftShoulderPosZ)
       ...
       (nd UniversalJoint
              (attach ../boxBody ../../body/boxBody)
              (setAnchor 0.0 0.0 0.0)
              (setAxis1 1.0 0.0 0.0) ; move around the x-axis
              (setAxis2 0.0 1.0 0.0) ; move around the y-axis
          ...
       )
)
```

Pomocou modelu je možné vytvoriť transformáciu, ktorá transformuje všetky body sústavy ramena do sústavy trupu. Mapovanie hodnôt modelu na hodnoty transformácie je zrejmé z nasledujúceho úryvku kódu.

```
Matrix leftShoulderTransform = bodyTransform;
transform.Translate(
       -UpperTorsoLength/2.0-ShoulderLength/2.0-TorsoCylinderLength/2.0,
       0,
       UpperTorsoHeight/2.0-ShoulderHeight/2.0);
```

```
//+anchor
transform.Translate(0.0, 0.0, 0.0);
transform.RotateX(uhol1);
transform.RotateY(uhol2);
//-anchor
transform.Translate(0.0, 0.0, 0.0);
```

Rovnakým spôsobom je možné vytvoriť transformácie pre všetky časti tela hráča. Pozície jednotlivých častí tela momentálne nevyužívame. V budúcnosti však môžu nájsť rôzne uplatnenie (napríklad ako vstup fitnes funkcie pre evolúciu pohybu končatín).

11. Literatúra

[1] RoboCup home page
http://www.robocup.org/02.html – oficialna stranka robocuou, (20.10.2007).

[2] Socer Simulation
http://wiki.cc.gatech.edu/robocup/index.php/Soccer_Simulation, (21.10.2007).

[3] Rules for 3D Soccer Simulation Competition during RoboCup 2007 Atlanta,
http://www.uni-koblenz.de/~murray/robocup/rc07/rules/3d_rules.pdf,
1. júl 2007.

[4] Hamid Reza Mohseni Nejad, Roya Helmmat Pour, Mohamad Mayanani, Amir Kharmandar, Hamid Bakhtiyari, Hojjat Nikan, Hamed Tirdad: Simulation of a Humanoid Soccer Robot, Team Description Proposal for RoboCup 2007, Sama 3D,
http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/Sama3D.pdf

[5] Xu Yuan, Tan Yingzi: SEU-3D 2007 Soccer Simulation Team Description,
http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/SEU-3D-TDP07.pdf

[6] Zigorat RoboCup teams,
http://zigorat3d.googlepages.com/home2

[7] Xichao Xia, Haonam Ma: DNU_Explorer 3D Team Description 2007,
http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/DNU_Explorer_3D_TDP_2007.pdf

[8] Ling Gao, Guangbin Cui, Pu Li, Hongxia Chai, Xiaomin Tang.: Fantasia 2007 Team Description,
http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/fantasia2007tdp.pdf

[9] Yoichi Setoguchi, Nobuhiro Ito: The World Model for Autonomous Soccer Agents, NAITO-Strikers,
http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/NAITO-StrikerS2007.pdf

[10] Saeid Akhavan, Mohammad Babaeizadeh, Hamid Reza Hasani, Arefeh Kazemi, Hoda Safaeipour : UI-AI3D Team Description,
http://www.uni-koblenz.de/~murray/robocup/rc07/Binaries/tdp/uiai2007TDP.pdf

[11] S-expresion,
http://en.wikipedia.org/wiki/S-expression, (20.10.2007)

[12] Vladimír Kvasnička: Genetický algoritmus,
http://www2.fiit.stuba.sk/~kvasnicka/NeuralNetworks/5.prednaska/GA_background.pdf

[13] V. Kvasnička, J. Pospíchal, P. Tiňo: Evolučné algorikmi, Slovanská technická univerzita v Bratislave, 2000, ISBN 80-227-1377-5

[14] Obst, O., Rollmann - M., Spark, A Generic Simulator for Physical Multi-agent Simulations, 2005.

[15] Smith, R.: Open Dynamics Engine v0.5 User Guide, 2006, http://ode.org/ode-latest-userguide.html#sec_3_5_0, (14.11.2007).

[16] Writing your own agent, Simspark Wiki,http://simspark.sourceforge.net/wiki/index.php/Getting_Started#Linux (14.11.2007).

[17] The RoboCup Soccer Simulator, rcssserver3D 0.5.6, http://sourceforge.net/project/showfiles.php?group_id=24184 (14.11.2007).

[18] The RoboCup Soccer Simulator, Email Archive: sserver-three-d, http://sourceforge.net/mailarchive/forum.php?forum_name=sserver-three-d (14.11.2007).

[19] Little Green BATS: Installing Robocup 3D simulation server on Ubuntu, http://www.littlegreenbats.nl/?q=node/70 (12.12.2007).

[20] RoboCup at FIIT, http://www2.fiit.stuba.sk/robocup/2007/turnaj_menu.htm

12. Prílohy

12.1 Príloha A - Príklad dát prijatých zo servera

```
(time
      (now 2884.39))
(GS
      (t 0.00)
      (pm BeforeKickOff))
(GYR
      (n torso)
      (rt 0.00 0.01 0.04))
(See
      (F1L
            (pol 35.14 126.87 -5.19))
      (F2L
            (pol 21.61 -169.21 -8.45))
      (F1R
            (pol 40.44 44.00 -4.51))
      (F2R
            (pol 29.45 -7.85 -6.19))
      (G1L
            (pol 26.94 141.85 -7.58))
      (G2L
            (pol 22.58 160.35 -9.05))
      (G1R
            (pol 33.55 29.64 -6.08))
      (G2R
            (pol 30.16 14.50 -6.76))
      (B
            (pol 13.04 71.57 -14.00))
      (P
            (team RoboLog)
            (id 10)
            (pol 10.50 90.01 0.00)))
(UJ
      (n laj1_2)
      (ax1 0.00)
      (ax2 -0.00))
(UJ
      (n raj1_2)
      (ax1 -0.00)
      (ax2 0.00))
(HJ
      (n laj3)
      (ax 0.00))
(HJ
      (n raj3)
      (ax 0.00))
(HJ
      (n laj4)
      (ax 0.00))
(HJ
      (n raj4)
      (ax -0.00))
(HJ
      (n llj1)
      (ax -0.00))
(HJ
      (n rlj1)
      (ax 0.00))
```

```
(UJ
      (n llj2_3)
      (ax1 0.00)
      (ax2 -0.00))
(UJ
      (n rlj2_3)
      (ax1 -0.00)
      (ax2 0.00))
(HJ
      (n llj4)
      (ax 0.00))
(HJ
      (n rlj4)
      (ax -0.00))
(FRP
      (n lf)
      (c 0.10 0.08 -0.05)
      (f -0.42 -0.30 12.22))
(UJ
      (n llj5_6)
      (ax1 0.00)
      (ax2 0.00))
(FRP
      (n rf)
      (c -0.10 0.08 -0.05)
      (f 0.41 -0.28 12.30))
(UJ
      (n rlj5_6)
      (ax1 -0.00)
      (ax2 -0.00))
```

12.2 Príloha B - Inštalácia servera

Postup by mal fungovať pre distribúcie *Dapper Drake (6.06)*, *Edgy Eft (6.10)* a *Feisty Fawn (7.04)*. Postup by mal byť podobný aj pri ostatných distribúciách ako *Fedora*, *Suse*, *Debian*.

- o – príkazy.
- • – komentáre k príkazom.

12.2.1 Povolenie *Universe* a *Multiverse* repozitárov

- o `sudo gedit /etc/apt/sources.list`
- • Podľa inštrukcií v súbore povoliť *Universe* a *Multiverse* repozitáre.
- o `sudo apt-get update`

12.2.2 Inštalácia potrebných balíčkov

- o `sudo apt-get install g++ ruby1.9 ruby1.9-dev libode0-dev libboost-dev libsdl-dev libfreetype6-dev libdevil-dev autoconf automake1.9 libtool freeglut3-dev tetex-extra cvs xlibs-dev libtiff4-dev libslang1-dev libboost-thread-dev libmng-dev libpng12-dev`
- • V prípade problémov s *xlibs-dev* treba použiť *xlibs-static-dev*.

12.2.3 Vytvorenie liniek pre ruby

- o `sudo rm /usr/bin/ruby`
- o `sudo ln -s /usr/bin/ruby1.9 /usr/bin/ruby`
- o `sudo ln -s /usr/lib/libruby1.9.so /usr/lib/libruby.so`

12.2.4 Stiahnutie zdrojových súborov servera z CVS repozitára

- • Zdrojové súbory sa stiahnu do adresára */home/<home_užívateľa>/rcsoccersim/*.
- o `cvs -d:pserver:anonymous@sserver.cvs.sourceforge.net:/cvsroot/sserver login`
- • Prípadný prompt na heslo potvrdiť enterom.
- o `cvs -z3 -d:pserver:anonymous@sserver.cvs.sourceforge.net:/cvsroot/sserver co -P rcsoccersim/rcssserver3D`

12.2.5 Build a inštalácia servera

- o `cd rcsoccersim/rcssserver3D/`
- o `./bootstrap`
- o `./configure`
- • V prípade chyby súvisiacej s *ruby.h* treba odinštalovať *ruby* balíky, *ruby* stiahnuť a nainštalovať manuálne zo zdrojových súborov.
  - o `sudo apt-get remove libruby*`
  - • Stiahnuť zdrojové súbory *ruby* z http://www.ruby-lang.org/en/downloads/.
  - o `sudo tar -xzf ruby-<stiahnutá_verzia>.tar.gz`
  - o `cd ruby-<stiahnutá_verzia>/`
  - o `./configure --enable-shared`
  - o `make`

  - sudo make install
  - cd /usr/lib
  - sudo ln -s /usr/local/lib/libruby.so.1.8
  - sudo ldconfig
  - cd /home/<home_užívateľa>/rcsoccersim/rcssserver3D/
  - ./configure
- make
- sudo make install
- sudo gedit /etc/ld.so.conf
- Do súboru pridať riadok '*/usr/local/lib*' ak sa v ňom ešte nenachádza.
- sudo ldconfig

12.2.6 Inštalácia modelov a textúr

- cd /usr/local/share/rcssserver3d
- sudo wget http://downloads.sourceforge.net/sserver/rcssserver3d-0.5.6-data.tar.gz?use_mirror=mesh
- sudo tar -xzf rcssserver3d-0.5.6-data.tar.gz

12.2.7 Spustenie servera

- cd
- simspark

12.2.8 Spustenie defaultného agenta

- agentspark

12.3 Príloha C - Popis tried evolúcie

Trieda strom

Koreňový strom vyjadrujúci funkciu.

Premenné

- Vektor int (RLK) – Readov lineárny kód stromu
- Vektor structov – Vektor ohodnotení vrcholov stromu
    - Struct: union(id(int), integer, float), typ(int) – Typ označuje čím je vrchol ohodnotený. Podľa typu sa bude brať jedna z troch hodnôt unionu. Tieto hodnoty označujú identifikátor funkcie, identifikátor premennej a konštantu.

Konštruktory

- Konštruktor pre náhodný strom.

Metódy

- Vytvor náhodný strom – Pre potreby mutácie.
- Výmena podstromu, náhodného alebo daného – Mutácia a kríženie
- Výpočet svojej hodnoty
- Nájdi podstrom začínajúci na indexe x v RLK
- Vytvor a vráť kópiu podstromu zo seba ktorá začína na indexe x

Trieda gén

Gén chromozómu (jedinca).

Premenné

- Čas – Čas, kedy s má gén vykonať. Čas bude relatívny na začiatok vykonávania génu
- Id behavioru – Identifikátor správania.
- Vektor stromov – Parametre správania.

Konštruktory

- Konštruktor pre náhodný gén

Metódy

- Vykonanie génu
- Mutácia génu – Mutácia stromov a zmena času.
- Kríženie dvoch génov (výmena stromov alebo kríženie stromov)
- Nastav čas génu

Trieda chromozóm

Jedinec populácie.

Premenné

- Vektor génov – DNA informácia chromozómu
- Fitness – Ohodnotenie jedinca.

Konštruktory

- Konštruktor pre náhodný chromozóm
- Konštruktor s danými génmi – Pre potreby kríženia.
- Konštruktor zo súboru – Pre načítanie jedincov zo súboru.

Metódy

- Ulož do súboru
- Nastav fitness
- Vykonávanie (??? Zatiaľ nevyriešené)
- Vráť zoznam génov pre čas – Pre potreby vykonania jedinca.

Behavior evolúcia

Tento behavior bude zabezpečovať evolúciu.

Premenné

- Zoznam chromozómov – Populácia jedincov
- Stav – Stav behavioru. Označuje či sa vykonávajú jedinci, či sa vytvára nová generácia, ...
- Parametre evolúcie – pravdepodobnosti mutácií, veľkosť populácie, počet rodičovských párov, ...

Konštruktory

- Konštruktor pre vytvorenie behavioru s náhodnou populáciou
- Konštruktor pre vytvorenie behavioru s načítaním populácie zo súboru

Metódy

- Ulož populáciu do súboru
- Vykonaj jedinca (čas) – Vykoná jedinca. Gény jedinca majú relatívny čas, preto bude potrebný prepočet z absolútneho času.
- Ohodnoť jedinca – Vracia fitness jedinca.
- Reprodukcia (výber rodičov, kríženie a mutácia)
- Vytvorenie novej generácie

12.4 Príloha D – CD nosič

Na 1. priloženom CD nosiči sa nachádza dokumentácia, ako aj zdrojové kódy serveru a hráča. CD má nasledujúcu štruktúru:

- \
- \Dokumentácia (obsahuje dokumentáciu k projektu)
- \Sources\player (obsahuje zdrojové kódy hráča neurotics agent)
- \Sources\server (obsahuje zdrojové kódy servera vo verzií 0.5.6)
- \Web (obsahuje webové stránky tímu)

Na 2. prilo6enom CD nosiči sa nachádza samotný produkt ako celok. CD je live distribúcia operačného systému Linux. Podrobnosti o jeho spustení nájdete v používateľskej príručke.

12.5 Príloha E – Posudok prezentácie prototypu tímu č. 11

Úvod

Tento dokument obsahuje posudok prototypu tímu č. 11, a jeho prezentáciu dňa 18.12.2007. Prezentácia sa konala v kabinete Ing. Ivana Kapustíka.v miestnsti D-110.

Prezentácia

Prezentácia bola veľmi pútavo predvedená viacerými členmi tímu. Do prezentácie bol zahrnutý vyčerpávajúci popis použitého komunikačného protokolu na komunikáciu medzi serverom a hráčom. Ďalej sme boli oboznámení s cieľmi projektu tímu č. 11. Tento tím si nekladie za cieľ vytvoriť plnohodnotného hráča schopného hry. Namiesto toho chcú vytvoriť slušný, dobre štruktúrovaný a multiplatformový základ pre ďalších pokračovateľov RoboCup-u 3D na našej fakulte. Príjemným spestrením prezentácie bolo aj predvedenie vstávania robota, ktoré však pracovalo len na základe naskriptovaných pohybov rozdelených do viacerých fáz.

Prototyp

Prototyp nám bol predvedený a mohli sme si tak overiť jeho funkčnosť. Predvádzanie prebiehalo prostredníctvom komunikácie hráča ktorý bežal na systéme Mac OSX a servera ktorý bežal na Linuxe. Pri prezentácií sme videli že komunikácia ako aj jej ukončenie prebiehalo korektne. Veľmi pozitívne hodnotíme aj konštrukciu a využitie analyzátora vstupu, ktorý rozpoznáva údaje vysielané serverom a ukladá ich do vhodnej štruktúry.

Zhodnotenie

Celkovo prototyp aj jeho prezentáciu hodnotíme veľmi pozitívne. Všetky potrebné fakty boli odprezentované, spolu s funkčnosťou celého prototypu. Myslíme si, že tento projekt bude mať určite prínos hlavne v budúcich rokoch pre ďalšie tímy. Prinesie im možnosť vyvíjať hráčov nezávisle od platformy na ktorej beží server.

12.6 Príloha F – Návod na použitie

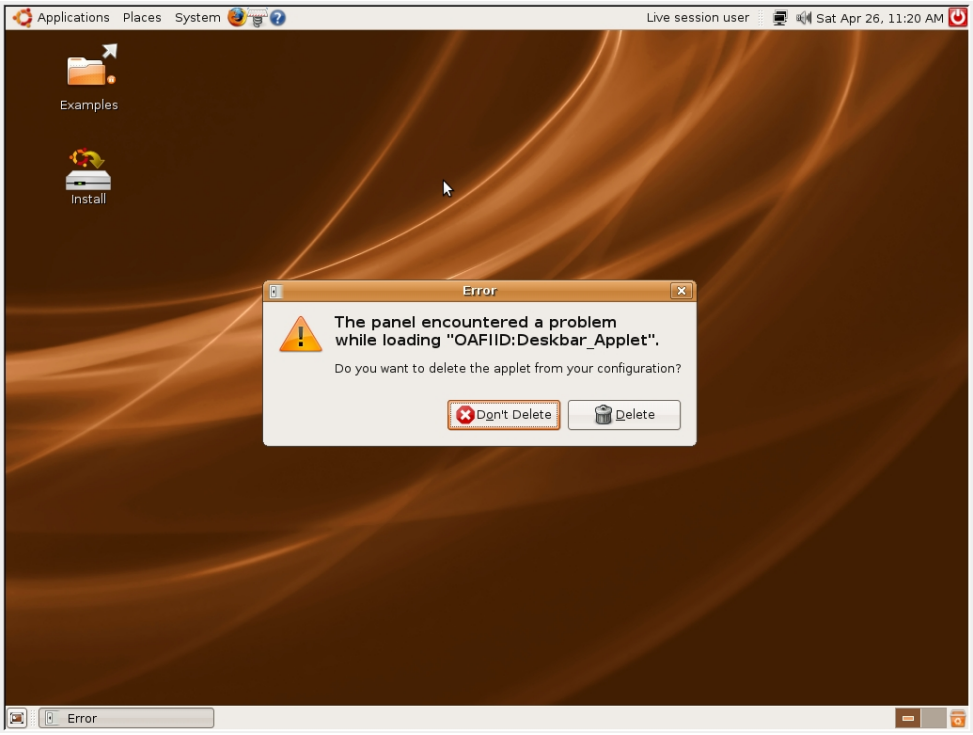
V tejto prílohe sa oboznámime s návodom ako spustiť evolúciu jedného alebo dokonca viacerých hráčov za pomoci dodaného evolučného CD nosiča, na to špeciálne určeného. Na spustenie distribuovanej evolúcie viacerých hráčov budeme potrebovať viac kópií evolučného CD.

12.6.1 Spustenie evolučného servera

K tomuto kroku je potrebné mať počítač so schopnosťou bootovať z disku CD. Hneď po zapnutí počítača vložíme evolučné CD do mechaniky a počkáme na zobrazenie úvodného menu ako na nasledujúcom obrázku.

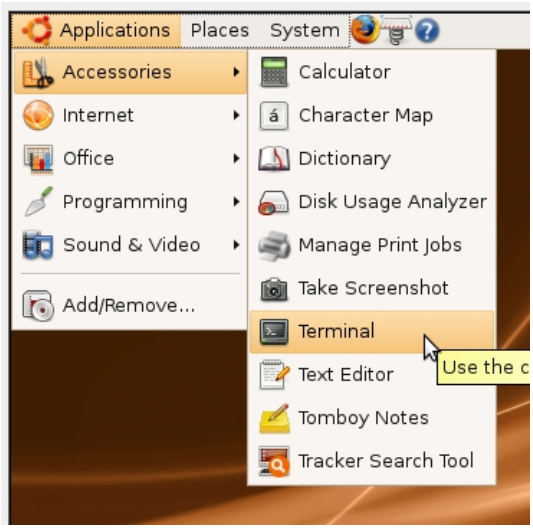
Obr. 39 Úvodná obrazovka evolučného CD

Spustenie CD prebehne automaticky po 30 sekundách alebo voľbou prvej možnosti *„Start or Install Ubuntu“*. Nasleduje samotný bootovací proces až do zobrazenia pracovnej plochy ako je zobrazené na nasledujúcom obrázku.



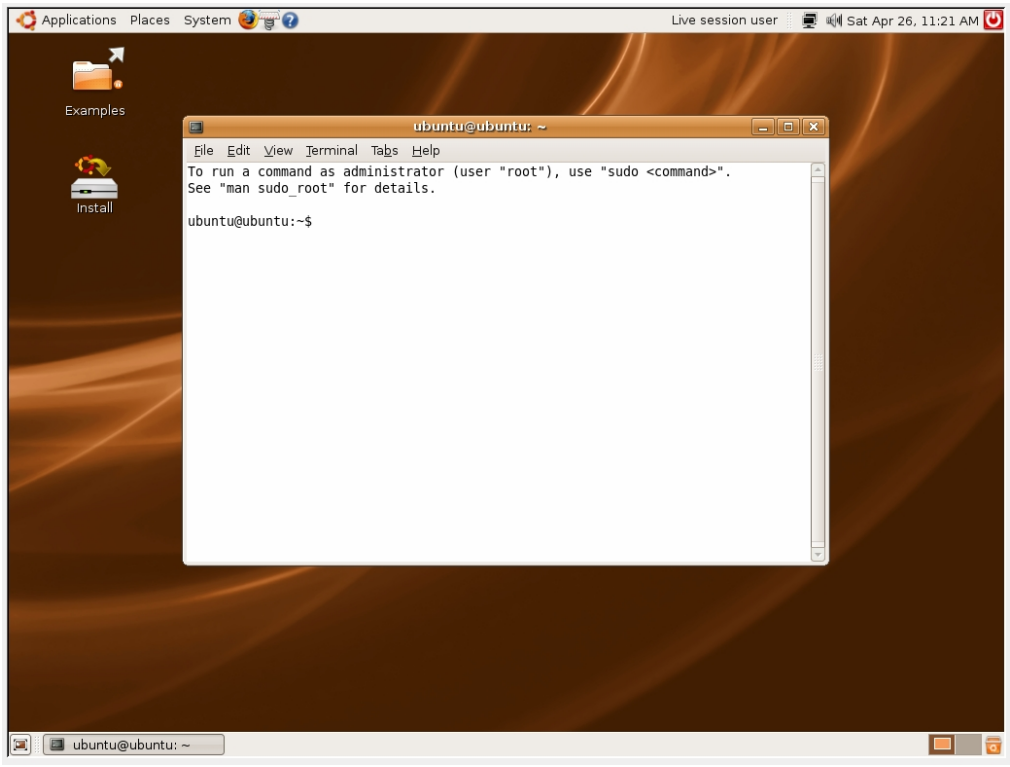
Obr. 40 Pracovná plocha evolučného CD

Chybovú hlášku si nemusíme všímať a môžeme ju odklepnúť kliknutím na tlačidlo *Delete*. Spustenie servera neprebieha automaticky a preto je ho potrebné pustiť z príkazového riadku. Najprv je potrené si pustiť terminál z menu *Applications -> Accessories -> Terminal*.



Obr. 41 Spustenie aplikácie Terminal

Po zvolení sa nám zobrazí okno príkazového riadku ako ukazuje nasledujúci obrázok.



Obr. 42 Okno terminálu

Po otvorení okna terminálu je potrebné spustiť server s príslušnými parametrami ako je IP adresa a port na ktorom bude server počúvať. Najprv si pomocou príkazu *ifconfig* zistíme svoju IP adresu.

```
ubuntu@ubuntu:~$: ifconfig
```

Po potvrdení dostaneme nasledujúci výpis, kde je tučným písmom vyznačená IP adresa nášho počítača:

```
eth0 Link encap:Ethernet  HWaddr 00:19:db:ea:d3:db
     inet addr:92.60.60.187  Bcast:92.60.63.255
Mask:255.255.248.0
     inet6 addr: fe80::21b:77ff:fea5:251c/64 Scope:Link
     UP BROADCAST MULTICAST  MTU:1500  Metric:1
     RX packets:0 errors:0 dropped:0 overruns:0 frame:0
     TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:1000
     RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
     Interrupt:216 Base address:0xa000
```

Teraz môžeme pustiť server príkazom *server.bin IP_ADRESA PORT*. Ako IP adresu použijeme tú čo sme zistili z výpisu príkazu *ifconfig*. Ako číslo portu by sme mali použiť nejaké vysoké v rozmedzí od 20000 do 30000.

```
ubuntu@ubuntu:~$: cd /ExchangeServer/bin
ubuntu@ubuntu:~$:./server.bin 92.60.60.187 20087
```

Pozn.: Pre podrobnejšie informácie o použití evolučného serveru pozri kapitolu číslo 12.6.5

12.6.2 Spustenie hráča

Pre spustenie hráča budeme taktiež potrebovať evolučné CD. Pri zavádzaní a spúšťaní samotného CD postupujeme rovnako ako pri evolučnom serveri. Po spustení terminálu je však potrebné postupovať inak. Najprv je potrebné spustiť na pozadí ***simspark*** simulačný server na ktorom sa bude háč vykonávať.

```
ubuntu@ubuntu:~$: loop_command simspark >& /dev/null &
```

Príkaz loop_command zabezpečí že aj po páde servera sa tento spustí znova. Ďalej musíme v konfiguračnom súbore nastaviť správnu adresu a port.

```
ubuntu@ubuntu:~$: cd /NeuroticsAgent/bin/
ubuntu@ubuntu:~$:mcedit evolution_config.txt
```

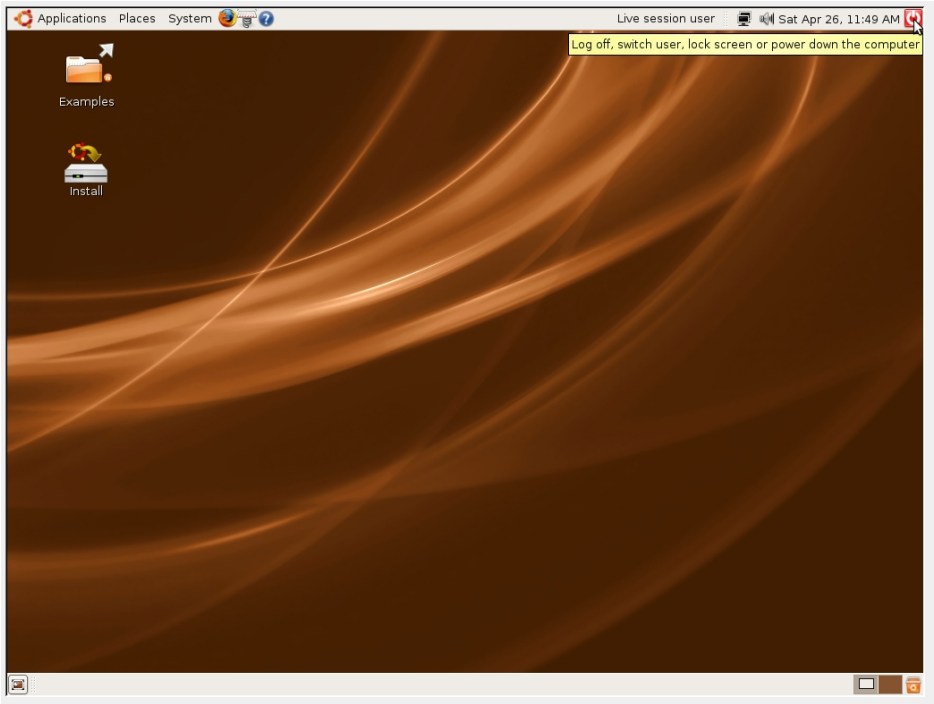
V tomto súbore sa nachádza riadok s IP adresou, a na riadku hneď pod ňou je číslo portu. Tieto hodnoty zmeníme na tie, ktoré sme zadávali pri spúšťaní evolučného servera. Súbor uložíme stlačením F2 a potvrdením. Z editora odídeme stlačením klávesu ESC. Následne môžeme spustiť hráča.

```
ubuntu@ubuntu:~$: loop_command ./NeuroticsAgent
```

Pokiaľ sme zadali správnu IP adresu a port, hráč by sa mal napojiť na evolučný server, vypýtať si uložených jedincov ak existujú, a začať svoju vlastnú evolúciu. Takisto ako simulačný server, ak hráč je spustený s príkazom ***loop_command,*** pre prípad pádu a následného znovu spustenia.

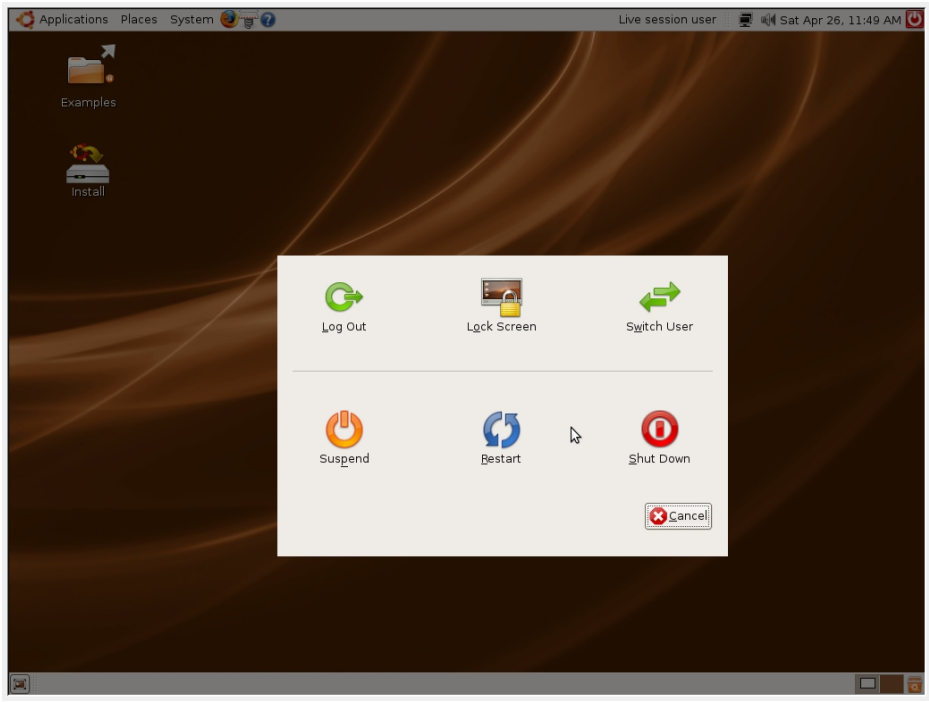
12.6.3 Ukončenie hráča alebo evolučného servera

Pre ukončenie práce evolučného servera alebo hráča postupujeme nasledujúcim spôsobom. Nie je potrebné manuálne nič ukončovať. Stačí kliknúť na ikonu vypnutia počítača v menu v pravom hornom rohu obrazovky, ako vidno na nasledujúcom obrázku.



Obr. 43 Voľba ukončenia práce, a vypnutia počítača

Po zvolení ešte určíme či sa má daný počítač reštartovať (*Restart*), alebo celkom vypnúť (*Shut Down*).



Obr. 44 Vypnutie alebo reštart počítača

Po ukončení všetkých procesov bude vysunuté evolučné CD. Vyberieme CD z mechaniky

a stlačíme klávesu ENTER pre vykonanie reštartu alebo vypnutia počítača.

12.6.4 Zapnutie monitoru

Monitor je súčasť simulačného servera pre RoboCup 3D. Jeho podrobnejší opis bol uvedení v Revízií č. 1 v podkapitole Monitor. Tento monitor je štandardne v našom spracovaní servera vypnutý. Je to z výkonnostných dôvodov, kedy samotné zobrazovanie tohto monitoru zaberá určitý procesorový čas (nakoľko nevyužíva hardvérovú akceleráciu), ktorý sme chceli radšej použiť na urýchlenie výpočtov evolúcie. Tento monitor však hlave pre prezentačné účely je možné znova zapnúť nasledujúcim spôsobom.

Najprv otvoríme príslušný súbor príkazom:

```
sudo gedit /usr/local/share/rcssserver3d/simspark.rb
```

V tomto súbore odkomentujeme riadok č. 9, súbor uložíme a spustíme server príkazom

```
simspark
```

12.6.5 Evolučný server

Zoznam zdrojových súborov:

- main.cpp
- server.cpp
- server.h
- ClientInfo.cpp
- ClientInfo.h
- Protocol.cpp
- Protocol.h
- DataSet.h
- ListenThreadData
- headers.h
- socketHeaders

Pre Windows OS naviac:

- pthreadVC2.lib
- pthreadVC2.dll
- pthread.h
- semaphore.h

Kompilácia

Windows OS

1. Vytvorenie Win32 konzolovú aplikáciu
2. Pripojiť v nastaveniach projektu tieto .lib: pthreadVC2.lib Ws2_32.lib
   Kde Ws2_32.lib je súčasťou MSVS 2005
3. Skompilovať projekt

Windows OS cygwin

>g++ -o server.bin main.cpp server.cpp ClientInfo.cpp Protocol.cpp

GNU Linux

> g++ -o server.bin main.cpp server.cpp ClientInfo.cpp Protocol.cpp -lpthread

Spustenie

Pre príklad nech existuje sieť s maskou podsiete 255.255.255.0 a server bude spúšťaný na počítači s IP 192.168.1.2 s portom na servis 123.

Windows OS

>server.exe 192.168.1.2 123

Windows OS cygwin, GNU Linux:

>./server.bin 192.168.1.2 123

Spustenie cez obnovy mriežky z predchádzajúceho chodu servera

Stačí pripísať ako posledný tretí parameter X

Windows OS

>server.exe 192.168.1.2 123 X

Windows OS cygwin, GNU Linux:

>./server.bin 192.168.1.2 123 X

Menu servera

Prístupný je Info klient priamo v servery a ukončenie aplikácie. V zdrojovom kóde je zakomentovaná možnosť cez menu meniť IP adresu a port, kde pobeží daná služba (pokiaľ je daná IP priradená k jednému zo sieťových kariet počítača).

Vypnutie servera

Odporúčané je vypínať server pomocou menu zadaním hodnoty 8!

12.6.6 Používateľská príručka klienta

Zoznam zdrojových súborov:

- main.cpp
- Client.cpp
- Client.h
- Protocol.cpp
- Protocol.h
- headers.h
- socketHeaders

Obsahuje aj main.cpp s main() funkciou, čiže klient môže byť aj samostatná aplikácia na testovacie účely.

Kompilácia

Windows OS

1. Vytvorenie Win32 konzolovú aplikáciu
2. Pripojiť v nastaveniach projektu Ws2_32.lib
   Kde Ws2_32.lib je súčasťou MSVS 2005
3. Skompilovať projekt

Windows OS cygwin, GNU Linux

>g++ -o client.bin main.cpp Client.cpp Protocol.cpp

Spustenie

Pre príklad nech existuje spustený server na počítači s IP adresou 192.168.1.2 a s portom servisu 123.

Windows OS

>client.exe 192.168.1.2 123

Windows OS cygwin, GNU Linux:

>./client.bin 192.168.1.2 123

Spustenie Info klienta

Stačí pripísať ako posledný tretí parameter X

Windows OS

>client.exe 192.168.1.2 123 X

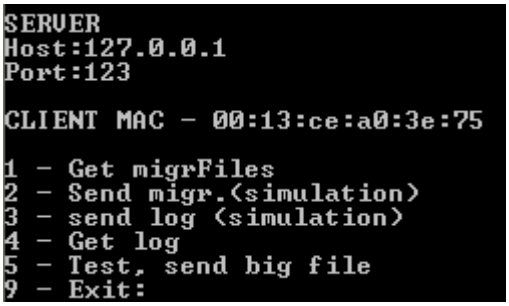
Windows OS cygwin, GNU Linux:

>./client.bin 192.168.1.2 123 X

Menu klienta

Pre testovacie účely je možné odolať migrujúcich jedincov, prijať migrujúcich,

odoslanie logu, prijatie logu, odoslanie veľkého súboru na otestovanie spojenia.



Vypnutie klienta

Vypnutie je cez 9.

Integrácia klienta do agenta

Podstatnejšia je integrácia klienta do iných aplikácii, konkrétne do agenta evolúcie.

```
#include "Client.h"

CClient client;

//Pripojenie sa na server
char*       cHost;
char*       cPort;
unsigned short int nPort;

cHost = "192.168.1.11";
cPort = "22087";
nPort = atoi(cPort);

client.Connect(cHost, nPort);

//Odoslanie dat, retazec znakov
char* cpFileStream;
int   nLength;

cpFileStream      = "Data jedinca";
nLength           = (int)strlen(cpFileStream);

client.SendStream(cpFileStream, nLength);

//Prijatie suborov
char**      cFiles      = (char**)malloc(sizeof(char*) * 2);
int*        nFileSizes  = (int*)malloc(sizeof(int) * 2);
int*        nFilesCount = (int*)malloc(sizeof(int));

client.ReceiveFiles(cFiles, nFileSizes, nFilesCount);
//Vsetky parametre su vystupne
//nFilesCount     - naplni sa poctom prijatych soborov
//nFileSizes      - pole velkosti suborov
//cFile           - pole retazcov, t.j. obsahov prijatych suborov

//indexy v poliach cFiles a nFileSizes navzajom koresponduju
```

```
//Odoslanie logu
char* cFileStream;
int   nLength;

cpFileStream      = "Data jedinca";
nLength           = (int)strlen(cpFileStream);

client.SendLog(cFileStream, nLength);

//Prijatie logu
char* cLog = NULL;
int LogSize;

client.ReceiveLog(&cLog, &LogSize);

//Odpojenie sa
client.Disconnect();
```

12.6.7 Testovanie správaní

Zapnutie testovania:
Testy sa v projekte NeuroticAgent používajú priamym prekladom zdrojového kódu agenta. V súbore *main.cpp* je treba odkomentovať riadok s makrom *#define TESTING 1*. Toto spôsobí, že sa agent preloží a výsledný binárny súbor bude v testovacom móde. Program bude v tomto prípade namiesto evolučného správania spúšťať testovacie správanie, ktoré spustí svoje zaregistrované testy s parametrami, aké boli poskytnuté.

Vytvorenie nových testov:
Triedy Testov pre jednotlivé kĺby by mali dediť z triedy TestJoint. Je potrebné aby mali obsiahnuté správanie ktoré je potrebné na testovanie kĺbu cez makro *USE_MODULE*. Druhou nutnosťou je preťaženie chránenej metódy *executeBehavior()*. V tejto metóde sa musí zavolať dané správanie s argumentami *a1*, prípadne *a2* a *speed*, ktoré určujú uhly pre kĺb a rýchlosť pohybu.

Pre vytvorenie zložitejších (nie kĺbových) testov momentálne neexistuje základná trieda. Je potrebné vychádzať priamo z triedy Test a implementovať všetky abstraktné metódy.

Registrácia a nastavenie testov:
V súbore *main.cpp* sa ďalej v metóde *main()* nachádza oblasť oddelená makrom testing (*#ifdef TESTING*). V tejto oblasti je vytvorené semienko náhodnosti pre generátor náhodných čísiel. Ďalej je získaný vektor testov, do ktorého sú postupne pridávané inštancie jednotlivých testov. Pridanie nového testu spočíva vo vytvorení inštancie testu (na hromade), a následné vloženie do vektoru *tests* pomocou metódy vektora *push_back()*. Testy sú navrhnuté tak, aby mali argumenty pre spúšťanie nastavené cez konštruktory.

Výstup testov:
Ak pri behu testov nenastane kritická chyba (nie zlý výsledok, ale chyba programu), je výsledkom každého jedného testu výpis. Tento výpis obsahuje priemer nameraných hodnôt, variáciu a normálnu odchýlku. Obsahuje aj textovú správu ako pomocnú informáciu.

12.6.8 Evolúcia

Nastavenie evolúcie
Všetky potrebné nastavenia sú v súbore „evolution_config.txt“. Význam hodnôt zodpovedá podľa poradia nasledovným parametrom evolúcie:

- Veľkosť populácie
- Počet rodičovských párov
- Percento elitných jedincov v populácii. Elitný jedinci budú automaticky zaradený do novej generácie.
- Čas vykonávania jedinca
- Čas od začiatku vykonávania jedinca, kedy sa začne počítať fitness. Musí byť menší alebo rovný ako čas vykonávania jedinca
- Maximálna hĺbka stromu – maximálna hĺbka náhodne vygenerovaného stromu.
- Maximálny počet potomkov uzla stromu.
- Maximálna absolútna hodnota konštanty v strome.
- Pravdepodobnosť mutácie stromu.
- Pravdepodobnosť mutácie génu.
- Pravdepodobnosť kríženia génov.
- Maximálny počet génov v náhodne vygenerovanom géne
- Minimálny počet génov v náhodne vygenerovanom géne
- Pravdepodobnosť mutácie chromozómu (jedinca).
- Maximálny počet génov pridaných počas mutácie.
- IP adresa paralelizačného servera.
- Port paralelizačného servera.
- Počet migujúcich chromozómov (jedincov).
- Frekvencia migrovania jedincov v generáciách.
- Počet logovaných chromozómov (jedincov) na paralelizačnom servery.
- Frekvencia logovania jedincov na paralelizačnom servry v generáciách.
- Vypínač lokálneho logovania jedincov (1/0 – zapnuté/vypnuté). Ak je lokálne logovanie zapnuté bude sa každú generáciu prepisovať súbor „evolution_dump.bin“ aktuálnou populáciou.
- Pauza pri znovupripojení. Ak je príliš krátka hráč môže mať hodnotu aktuálnej polohy v čase začiatku vykonávania chromozómu nesprávnu, aj keď sa začne vykonávať na správnom mieste (napr. v prípade ak bol predchádzajúci chromozóm príliš „divoký“ a hráč sa rozpadol).

Súbor v súčasnosti nepodporuje komentáre. V budúcnosti sa jeho formát pravdepodobne zmení a hodnoty v ňom budú dostatočne okomentované. Ak hráč po spustení súbor nenájde bude používať defaultne nastavenie. Toto nastavenie je možné nájsť v zdrojových kódoch hráča v triede „EvolutionConfig“ (súbor /evolution/evolution_config.h(cpp)). Bližšie informácie o obsahu a význame tejto triedy nájdete v technickej dokumentácii.

Spustenie evolúcie
Evolúcia sa spúšťa tak, že sa v každom cykle behu hráča zavolá metóda „MainBehavior::evolution()“. Momentálne sa táto metóda volá v „MainBehavior::makeDecision()“ (v súbore /others/AgentDecision.cpp).

Vytvorenie inštancie evolúcie
Evolúcia sa vykonáva ako správanie hráča. Konkrétne ide o „EvolutionBehavior“. V

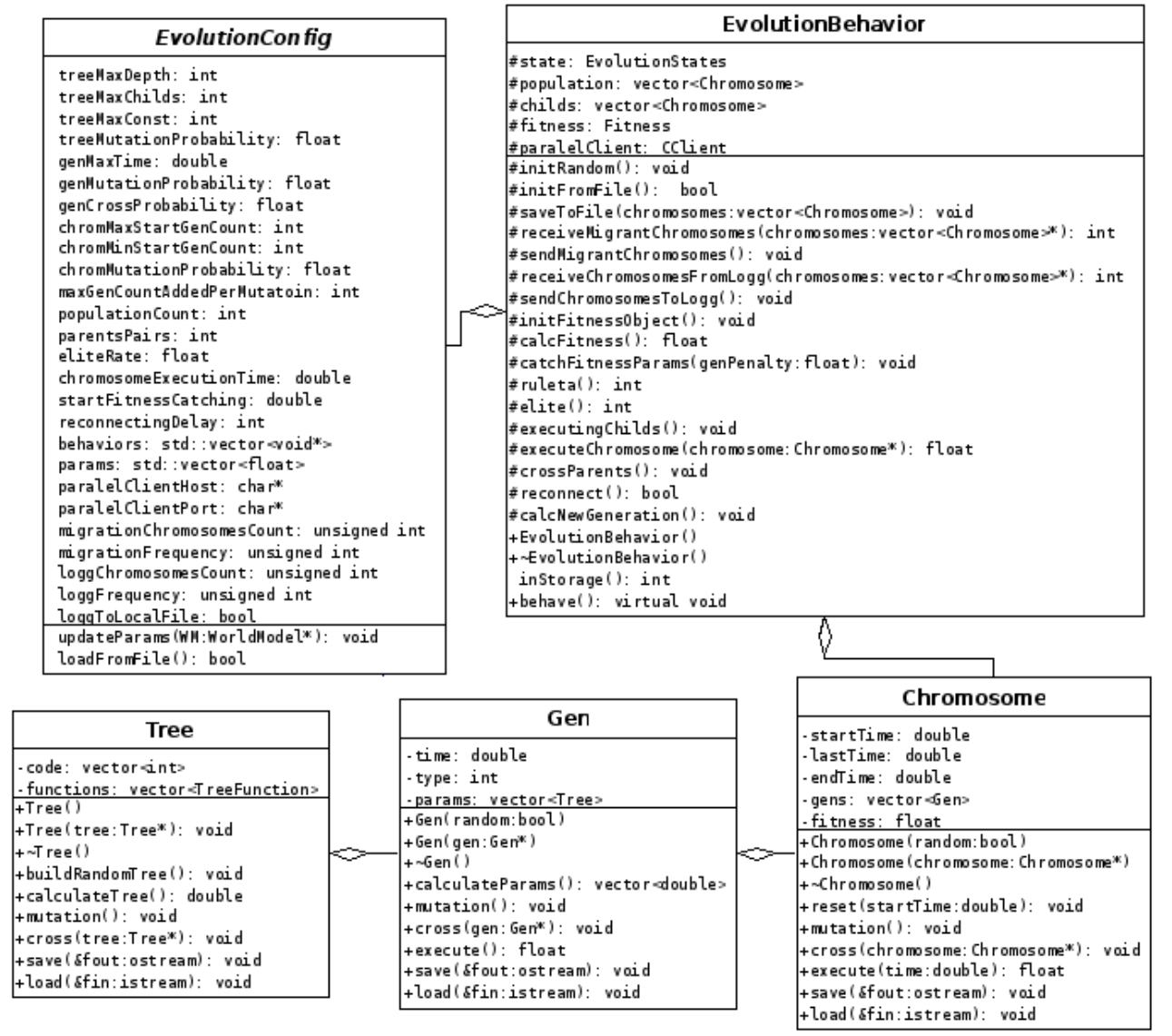
konštruktore tejto triedy sa vytvorí fitness objekt a hráč pokúsi pripojiť na paralelizačný server. Ak sa mu to podarí, vyžiada si od neho svojich posledných logovaných jedincov a migrujúcich jedincov.

Inicializácia
Evolúcia sa inicializuje najprv zo súboru „evolution_dump.bin“ a ak sa to nepodarí (súbor neexistuje) inicializuje sa náhodne. Pri inicializácii zo súboru sa načítajú jedinci zo súboru a následne sa im vymaže fitness. Jedinci sa budú znova vykonávať (pre prípad zmeny fitness funkcie).

Priebeh evolúcie
Vykonanie potomkov a ich pridanie do pôvodnej generácie.
Pred vykonaním každého jedinca sa hráč znovupripojí na simulačný server.
Výber jedincov do novej generácie.
Do novej generácie sa pridajú elitný jedinci a doplní sa pomocou ruletového výberu.
Kríženie jedincov.
Jedince na kríženie sa vyberajú ruletovým výberom.

Evolúcia sa skončí len ak sa vypne hráč (prípadne padne server :).

Fitness
Fitness sa počíta pomocou objektu triedy „Fitness“. Ten sa inicializuje volaním metódy „`EvolutionBehavior::initFitnessObject`“ v konštruktore triedy „`EvolutionBehavior`“. Počas vykonávania jedinca sa od času určeného v konfiguračnom súbore, začne v každom cykle počítať fitness (metóda „`EvolutionBehavior::catchFitnessParams`“). Výsledná hodnota fitness sa potom vypočíta pomocou váhovaného súčtu tak, aby jedince, ktoré splnia požadovanú úlohu skôr, mali fitness lepšiu (metóda „`EvolutionBehavior::calcFitness`“).



Obr. 45 Diagram tried evolúcie

12.7 Príloha G – Technická dokumentácia

V tejto prílohe uvádzame technickú dokumentáciu k nášmu produktu. Jedná sa konkrétne o hráča, evolučný server a LiveCD na ktorom sú distribuované. Budú tu podrobne opísané ich funkcie, vstupy a výstupy.

12.7.1 LiveCD

Už v predchádzajúcich kapitolách sme niekoľkokrát uviedli, že simulačný server ***simspark*** je možné spúšťať len pod operačným systémom Linux. V laboratóriu, ktoré nám bolo k dispozícií, však nebol ani jeden počítač, ktorý by mal Linux nainštalovaný.
A tak sme sa rozhodli, že použijeme LiveCD distribúciu. Táto voľba nám zabezpečila, že sme mohli využiť temer každý počítač, schopný bootovať z CD. Ako LiveCD distribúciu sme využili Ubuntu vo verzií 7.10. Na toto LiveCD sme potom skompilovali ***simspark***, simulačný server, v príslušnej verzií. Následne sme na CD umiestnili nášho hráča, spolu s evolučným serverom.
Pre naše testovacie účely sme zaviedli štartovací skript, ktorý vždy pri štarte CD nahral a spustil z SVN repozitára najnovšiu verziu hráča a evolučného servera. Táto funkcionalita vo výslednom produkte nebude prítomná, nakoľko SVN server bude zrušený po skončení aktuálneho semestra. Ako spustiť hráča, alebo evolučný server manuálne, je však možno nájsť v prílohe ***F – návod na použitie***.

12.7.2 Evolúcia

V tejto časti budú opísané technické parametre prostriedkov evolúcie.

Trieda EvolutionConfig
Obsahuje konfiguračné parametre objektov, ktoré sa používajú pri evolúcii.

Zoznam funkcií:

- *static int treeMaxDepth* – Maximálna hĺbka náhodne vytvoreného stromu.
- *static int treeMaxChilds* – Maximálny počet potomkov uzlu v strome.
- *static int treeMaxConst* – Maximum absolútnej hodnoty konštanty v strome.
- *static float treeMutationProbability* – Pravdepodobnosť mutácie stromu.

- static double genMaxTime – Maximálny čas spustenia génu.
- static float genMutationProbability – Pravdepodobnosť mutácie génu.
- static float genCrossProbability – Pravdepodobnosť kríženia génu.

- *static int chromMaxStartGenCount* – Maximálny počet génov v náhodne vytvorenom chromozóme.
- *static int chromMinStartGenCount* - Minimálny počet génov v náhodne vytvorenom chromozóme.
- *static float chromMutationProbability* – Pravdepodobnosť mutácie chromozómu.
- *static int maxGenCountAddedPerMutatoin* – Maximálny počet génov pridaných pri mutácii.

- *static int populationCount* – Veľkosť populácie.
- *static int parentsPairs* – Počet rodičovských párov.
- *static float eliteRate* – Percento novej populácie vybranej elitizmom.
- *static double chromosomeExecutionTime* – Čas vykonávania chromozómov.
- *static double startFitnessCatching* – Čas, po ktorom sa začnú zbierať údaje pre výpočet fitness.
- *static int reconnectingDelay* – Pauze po rekonektovaní.

- *static std::vector<void*> behaviors* – Vektor správaní, ktoré sa používajú v génoch chromozómov.
- *static std::vector<float> params* – Vektor premenných z vnútorného sveta hráča, ktoré sa používajú v stromoch (trieda tree opísaná nižšie). Tieto parametre je potrebné pravidelne obnovovať (metóda EvolutionConfig::updateParams).

- *static char* paralelClientHost* – IP adresa paralelizačného servera.
- *static char* paralelClientPort* – Port paralelizačného servera.

- *static unsigned int migrationChromosomesCount* – Počet migrujúcich jedincov.
- *static unsigned int migrationFrequency* – Frekvencia migrovania (počet generácii).
- *static unsigned int loggChromosomesCount* – Počet jedincov zaslaných na logovanie na server.
- *static unsigned int loggFrequency* – Frekvencia logovania (počet generácii)

- *static bool loggToLocalFile* – Ak je true jedinci sa loguju (okrem paralelizačného severa) aj na lokálny disk.

- *static void updateParams(WorldModel* WM)* – Metóda vyprázdni a znovu naplní vektor „EvolutionConfig::params“.
- *static bool loadFromFile()* - Načíta parametre evolúcie zo súboru „evolution_config.txt“. Okrem vektorov „behaviors“ a „params“.

evolution_types.h

Obsahuje definície typov, použitých v evolúcii.

enum UnarFunctions – Unárne funkcie pre triedu „tree“.
inv – Unárny mínus.
sinus - Sínus
cosinus - Kosínus
unarFunctioncCount – Počet unárnych funkcií.

enum BinarFunctions – N-nárne funkcie pre triedu „tree“.
pluss = unarFunctioncCount - „+“
minuss - „-“
krat - „*“
deleno - „/“
functionsCount – Počet funkcií.

enum TreeTypes – Typy vrcholov v strome.
funkcia – Každý nelistový vrchol je tohto typu. Predstavuje funkciu, ktorá sa má vykonať nad jeho potomkami.
operand – List obsahujúci jednu z premenných z vektora „EvolutionConfig::params“.
konstanta – List obsahujúci konštantu.
typesCount – Počet typov.

struct TreeFunction
float hodnota;
int typ;

struct EvolutionPopulationDump
unsigned int size;
char *data;

enum EvolutionStates
initEvolutionState,
reconnectiongState,
executingChromosomsState,
calcNewGenerationState,
showingBestState,
testState

Trieda Tree

Strom reprezentuje funkciu, pomocou ktorej sa vypočítava hodnota niektorého parametra. Strom sa evolvuje podľa princípov genetického programovania.

- *vector<**int**> code* – Readov lineárny kód (RLK) stromu. Vytvára sa rekurzívne ako zreťazenie počtu potomkov vrchola a RLK podstromov z ľava do prava.
- *vector<TreeFunction> functions* – Vektor funkcií jednotlivých vrcholov, zoradených v rovnakom poradí ako sú v „code“.
- *Tree()* - Konstruktor náhodného stromu. Hraničné parametre stromu sú v triede EvolutionConfig
- *Tree(Tree* tree)* – Konštruktor pre kopírovanie stromu daného parametrom tree.
- ***int** findSubTree(**unsigned int** start)* – Funkcia vyhľadá koniec podstromu začínajúceho na indexe start v code. Vráti index prvého vrchola, ktorý už nepatrí do daného podstromu.
- ***double** calculateTree()* - Metóda vypočíta aktuálnu hodnotu stromu. (Podľa aktuálnej hodnoty premenných vo vektore EvolutionConfig::params).
- ***void** mutation()* - Metóda pre mutáciu stromu. Mutácia spočíva vo výmene podstromu náhodným podstromom.
- ***void** cross(Tree* tree)* – Kríženie stromu so stromom daným parametrom. Stromy si vymenia náhodné podstromy a mutujú sa podľa pravdepodobnosti danej v triede EvolutionConfig.

Trieda Gen

Gén reprezentuje spustenie jedného zo základných správaní, ktoré sú vo vektore EvolutionConfig::behaviors.

- ***double** time* – Čas spustenia génu.
- ***int** type* – Typ génu (index vektora EvolutionConfig::behaviors).
- *vector<Tree> params* – Vektor parametrov správania, ktoré gén reprezentuje. Parametre sú vo forme Koreňových stromov.
- *Gen(**bool** random)* – Konstruktor génu. Ak je parameter rovný true, vytvorí sa náhodný gén. Inak bude gén prázdny (pre účely kopýrovania génu).
- *Gen(Gen* gen)* – Konštruktor pre kopírovanie génu.
- ***void** mutation()* - Mutácia génu. Mutuje sa čas génu a stromy vo vektore params.
- ***void** cross(Gen* gen)* – Kríženie génu s génom zadaným parametrom gen. Krížiť sa môžu len gény rovnakého typu. Krížia sa tak, že sa vymenia ich časy a krýžia sa ich parametre (stromy v params).
- ***float** execute()* - Gén sa vykoná a vráti hodnotu penalizácie. Penalizuje sa použitie neprimeraných parametrov (vysoká rýchlosť alebo uhol).

Trieda Chromosome

Reprezentuje jedinca evolúcie (celkové správanie).

- ***double** startTime* – Čas začiatku vykonávania jedinca.
- ***double** endTime* – Doba vykonávania jedinca.
- *vector<Gen> gens* – Vektor génov jedinca.

- *Chromosome(**bool** random)* – Konštruktor chromozómu. Ak je hodnota parametra rovná true vytvorí sa náhodný jedinec. Opačný prípad sa využíva len pri kopírovaní chromozómov (aby sa zbytočne ne vytváral náhodný jedinec keď ho ajtak prepíšeme).
- *Chromosome(Chromosome* chromosome)* – Konštruktor pre kopírovanie chromozómov.
- ***void** reset(**double** startTime)* – Metóda pripraví jedinca na vykonanie. Musí sa spustiť pred vykonávaním. Parameter startTime predstavuje aktuálny čas simulácie.
- ***float** execute(**double** time)* – Vykonávanie jedinca. Parameter time predstavuje aktuálny čas simulácie.
- ***void** mutation()* - Odstránenie niektorých génov, mutovanie zostávajúcich a pridanie nových, náhodne vytvorených génov.
- ***void** cross(Chromosome* chromosome)* – Kríženie jedincov. Jednobodové kríženie, pričom indexy kríženia sú navzájom nezávislé. Jedince sa následne mutujú.

Trieda EvolutionBehavior

Správanie Reprezentujúce evolúciu.

- *EvolutionStates state* – Stav v ktorom sa evolúcia nachádza.
- *vector<Chromosome> population* – Populácia evolúcie. Všetky jedince v tomto vektore už musia byť vykonané (musia mať fitness).
- *vector<Chromosome> childs* – Populácia potomkov. Jedince, ktoré ešte neboli ohodnotené funkciou fitness (neboli vykonané).
- *CClient paralelClient* – Klient paralelizačného servera.
- *EvolutionBehavior()* - Konštruktor evolúcie.
- ***void** initRandom()* - Inicializácia evolúcie vygenerovaním náhodných jedincov.
- ***bool** initFromFile()* - Inicializácia evolúcie z lokálneho súboru „evolution_dump.bin“.
- ***void** saveToFile(vector<Chromosome> chromosomes)* – Uloženie populácie do lokálneho súboru „evolution_dump.bin“.
- ***int** receiveMigrantChromosomes(vector<Chromosome>* chromosomes)* – Prijatie migrujúcich jedincov z paralelizačného servera.
- ***void** sendMigrantChromosomes()* - Poslanie migrujúcich jedincov na paralelizační server.
- ***int** receiveChromosomesFromLogg(vector<Chromosome>* chromosomes)* – Prijatie posledných jedincov logovaných na paralelizačný server.
- ***void** sendChromosomesToLogg()* - Poslanie jedincov na logovanie na paralelizačný server.
- *void initFitnessObject()* - Vytvorenie a inicializácia fitness objektu.
- *float calcFitness()* - Výpočet fitness.
- *void executingChilds()* - Vykonávanie potomkov.
- *float executeChromosome(Chromosome* chromosome)* – Vykonávanie jedincov.
- *void crossParents()* - Výber rodičov a ich kríženie.
- *void calcNewGeneration()* - Výber jedincov do novej generácie.

12.7.3 Distribuovaná evolúcia

V tejto prílohe uvádzame technické doplnky ku kapitole o distribuovanej evolúcií.

Adresárová štruktúra

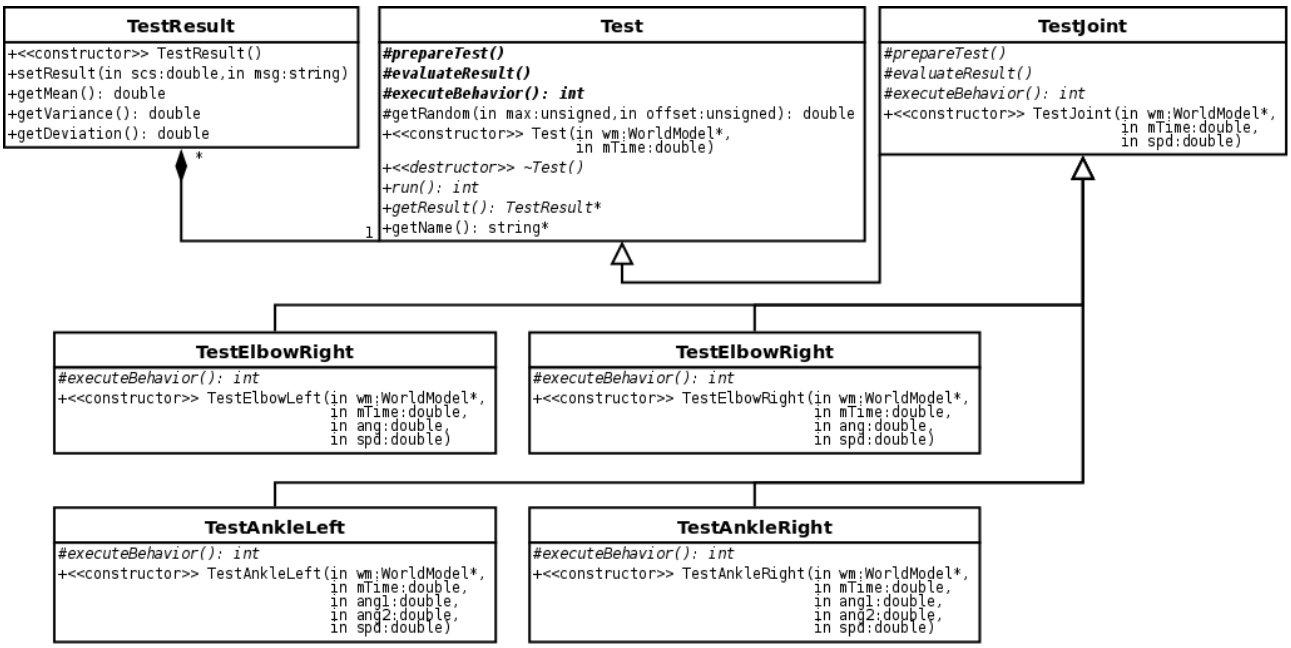
V mieste spustenia binárky servera sa vytvorí adresár CLIENTS. V čase pripojenia klienta sa v tomto adresári vytvorí ďalší s názvom IP, pod ktorou klient vystupuje (v LAN priamo jeho, alebo IP podsiete v ktorej sa nachádza). V adresári s názvom IP klienta je vytvorí adresár s názvom MAC klienta a v ňom sa uchovávajú jeho súbory. Takže v prvej línii sú IP adresy, ktoré ak sú IP podsiete, budú tieto adresáre obsahovať viaceré adresáre s MAC adresou daných klientov odtiaľ pripojených.

Názvy súborov

Pre iné účely si server uchováva aj verzie migrujúcich jedincov do súborov. Názov súboru pre migrujúce jedince je MAC_klienta.verzia, názov súboru pre log generácie je MAC_klienta_LOG.verzia. Na obrázku klient v MAC adresou 00.13.CE.A0.3E.75 odoslal dvoch jedincov a dve generácie.

12.7.4 Testovanie správaní hráčov

Základom všetkých testov je trieda Test a jej výsledok, trieda TestResult. Všetky testy sú konkrétnou implementáciu triedy Test. Trieda TestJoin predstavuje spoločný základ pre testy jednotlivých kĺbov.



Obr. 46 ***Architektúra tried testovaní správaní***

Trieda TestResult:

Trieda TestResult slúži na uchovanie výsledku testovania. Stará sa o medzi výpočty pre určovanie priemeru, variácie a štandardnej odchýlky.

Metódy:
konštruktor *TestResult()* - vytvorí prázdny výsledok
void setResult(double scs, string &msg) – nastaví výsledok. Scs predstavuje mieru úspechu testu, msg je textová pomocná správa. TestResult sa postará o výpočet priemeru a ostatných štatistických údajov.
double getMean() - získa priemernú hodnotu pre výsledok testu.
double getVariance() - získa variáciu pre výsledok testu.
double getDeviation() - získa štandardnú odchýlku pre výsledok testu.

Trieda Test:

Trieda test slúži ako základ pre všetky testy. Je to abstraktná trieda, ktorej metódy je nutné implementovať v podtriedach.

Metódy (protected):)
virtual void prepareTest() - metóda sa spúšťa automaticky z metódy run() keď je test prvý krát spustený. Táto metóda pripraví test na spustenie, napríklad generovanie nových náhodných čísiel prebieha tu.
virtual void evaluateResult() - metóda vyhodnotí výsledok testu, spúšťa sa z metódy *run()*. V tejto metóde sa určí hodnota úspešnosti testu a uloží do výsledku TestResult.

virtual int executeBehavior() - metóda vykoná dané správanie. Každý test obsahuje jedno, alebo viac správaní, ktoré chce testovať. Táto metóda sa spúšťa z metódy *run()* a zabezpečuje vykonanie testovaných správaní.

Metódy (public):
konštruktor *Test(WorldModel* wm, double mTime)* – vytvorí novú inštanciu triedy. Prvý argument je WorldModel, ktorý je potrebný pre získavanie informácií o svete a hráčovi. Druhý argument je maximálny čas, ktorý má test bežať v sekundách.
virtual int run() - metóda run je používaná testovacím správaním. Je to metóda, ktorá sa volá každé kolo testovania. Metóda je virtuálna, avšak obsahuje základnú implementáciu.
virtual TestResult getResult()* - metóda vráti ukazateľ na výsledok testu. Metóda je virtuálna, avšak obsahuje základnú implementáciu.
string getName()* - vráti meno testu. Každý konkrétny test by mal určovať svoje meno do premennej *name*. Táto metóda vracia dané meno.

Trieda TestJoint:

Trieda TestJoint obsahuje základné spoločné prvky pre implementáciu testov jednotlivých kĺbov.

Metódy:
konštruktor *TestJoin(WorldModel *wm, double mTime, double spd)* – podobne ako u triedy Test, avšak naviac je argument rýchlosti, ktorý určuje konštantnú rýchlosť pohybu pri vykonávaní správania na kĺbe.

12.7.5 Hráč

V tejto kapitole bude opísaný model hráča, ako aj popis funkcií a tried ktoré využíva.

Funkcie v main.cpp:

```
int main(int argc, char* argv[])
```
Hlavná funkcia. Vytvorí a prelinkuje všetky rôzne triedy. Ako vstup má argumenty z príkazového riadku

```
bool StartAgent(TRoboCupConnection **con, WorldModel **wm, ActHandler
**act, Formations **form, MAIN_BEHAVIOR_TYPE **mainBehavior)
```

Funkcia na spustenie hráča. Na vstupe má prepojenie na robocup(con), model sveta(wm), obsluhu hráča(act), formáciu(form) a správanie(mainBehavior). Funkcia vráti true ak sa hráča podarí inicializovať, false ak nie.

```
void StopAgent(TRoboCupConnection **con, WorldModel **wm, ActHandler **act,
Formations ** form, MAIN_BEHAVIOR_TYPE **mainBehavior)
```

Funkcia na ukončenie hráča. Na vstupe má prepojenie na robocup(con), model sveta(wm), obsluhu hráča(act), formáciu(form) a správanie(mainBehavior). Funkcia cez operátor *delete* zmaže všetky triedy a tým hráča ukončí.

```
bool ConfigureArguments(int argc, char* argv[])
```

Funkcia ktorá nastaví patričné premenné podľa vstupu z príkazového riadku.

Funkcie v BasicBehavior:
Trieda pre základné správanie.

```
static void init(ActHandler * _ACT,WorldModel * _WM,TRoboCupConnection *
_connection);
```

Funkcia pre inicializáciu správania. Na vstupe má prepojenie na obsluhu hráča(`_ACT`), model sveta(`_WM`) a robocup(`_connection`).

Funkcie v MainBehavior:
Trieda rozširujúca základné správanie.

```
void behave();
```

Hlavná slučka hráča. Táto metóda je volá aktualizačné metódy modelu sveta, potom sa indikujú nové informácie. Za týmito akciami nasleduje hlavné „rozmýšľanie“ agenta, ktoré pošle príkazy serveru.

Funkcie v AdjustJointBehavior:
Trieda pre priame nastavenie uhlovej rýchlosti kĺbu.

```
virtual bool behave( JointT id, double dSpeed1, double dSpeed2 =
UnknownDoubleValue );
```

Funkcia kĺbu id nastaví uhlovú rýchlosť dSpeed1, prípadne dSpeed2 ak má kĺb dve osy.

Funkcie v AnkleBehavior:

Trieda pre nastavenie uhlov kotníku.

```
virtual bool behave( SideT side, AngDeg valueFront, AngDeg valueOpen,
double dSpeed );
```

Funkcia nastaví rýchlosť otáčania ľavému/pravému (závisí od parametru side) členku. valueFront určuje uhol vzhľadom na nohu a valueOpen natočenie. dSpeed určí rýchlosť motorov.

Funkcie v BeamBehavior:
Trieda pre priame nastavenie polohy hráča na ihrisku.

```
virtual void behave(double dx, double dy, double dang);
```

Funkcia nastaví polohu hráča na súradnice dx a dy, s natočením dang.

Funkcie v ElbowBehavior:
Trieda pre nastavenie uhlu lakťu.

```
virtual bool behave( SideT side, AngDeg value, double dSpeed );
```

Funkcia nastaví rýchlosť otáčania ľavému/pravému (závisí od parametru side) lakťu. valueFront určuje uhol otvorenia lakťu. dSpeed určí rýchlosť motora.

Funkcie v HipBehavior:
Trieda pre nastavenie uhlov bedrového kĺbu.

```
virtual bool behave( SideT side, AngDeg valueFront, AngDeg valueOpen,
double dSpeed );
```

Funkcia nastaví rýchlosť otáčania ľavému/pravému (závisí od parametru side) bedrovému kĺbu. valueFront určuje uhol vzhľadom na telo a valueOpen natočenie. dSpeed určí rýchlosť motorov.

Funkcie v ShoulderBehavior:
Trieda pre nastavenie uhlov ramena.

```
virtual bool behave( SideT side, AngDeg valueFront, AngDeg valueOpen,
double dSpeed );
```

Funkcia nastaví rýchlosť otáčania ľavému/pravému (závisí od parametru side) ramenu. valueFront určuje uhol vzhľadom na telo a valueOpen natočenie. dSpeed určí rýchlosť motorov.

Funkcie v InitPosBehavior:
Trieda pre inicializáciu hráča.

```
virtual void behave();
```

Funkcia nastaví všetko potrebné pre spustenie hráča.

Funkcie v KneeBehavior:
Trieda pre nastavenie uhlu kolena.

```
virtual bool behave( SideT side, AngDeg value, double dSpeed );
```

Funkcia nastaví rýchlosť otáčania ľavému/pravému (závisí od parametru side) kolenu. valueFront určuje uhol otvorenia kolena. dSpeed určí rýchlosť motora.

Funkcie v RotateArmBehavior:

Trieda pre nastavenie uhlu ramena. Ekvivalentné s ShoulderBehavior, ale nastavuje iba prvý uhol.

```
virtual bool behave( SideT side, AngDeg value, double dSpeed );
```

Funkcia nastaví rýchlosť otáčania ľavému/pravému (závisí od parametru side) ramenu. valueFront určuje uhol otvorenia ramena. dSpeed určí rýchlosť motora.

Funkcie v RotateLegBehavior:
Trieda pre nastavenie uhlu nohy. Ekvivalentné s HipBehavior, ale nastavuje iba prvý uhol.

```
virtual bool behave( SideT side, AngDeg value, double dSpeed );
```

Funkcia nastaví rýchlosť otáčania ľavej/pravej (závisí od parametru side) nohy. valueFront určuje uhol otvorenia nohy vzhľadom na telo. dSpeed určí rýchlosť motora.

Funkcie v StaticBehavior:
Trieda pre vykonanie statického správanie.

```
virtual void behave();
```

Funkcia volá ostatné správania podľa napevno nastavenej sekvencie.

Funkcie v TurnToBehavior:
Trieda pre otočenie hráča.

```
virtual bool behave( AngDeg ang, bool &start );
```

Funkcia začne sekvenciu správaní vedúcich k otočeniu hráča o požadovaný uhol ang, vstupný parameter start indikuje či ide o prvé volanie tohto správania (ak áno funkcia ho nastaví na hodnotu false).

Funkcie v WalkBehavior:
Trieda pre spustenie kráčania hráča.

```
virtual bool behave( bool & start );
```

Funkcia začne sekvenciu správaní vedúcich ku kráčaniu hráča v smere, v ktorom je otočený, vstupný parameter start indikuje či ide o prvé volanie tohto správania (ak áno funkcia ho nastaví na hodnotu false).

Funkcie v behavior_storage.h:
V tomto súbore sú uložené makrá a dátové štruktúry, pre inteligentné(aby z každej triedy bola alokovaná vždy iba jedna inštancia) referencie na triedy správania.

```
#define DECLARE_CLASS(name) int name::inStorage = -1;
```

Makro, ktoré nastaví statickú premennú inStorage triedy s názvom name na hodnotu -1, čo indikuje, že táto trieda správania ešte nemá vytvorenú žiadnu inštanciu, čo je štandardný stav pri štarte.

```
#define USE_MODULE(name, path)
```

Makro, ktoré pridá do triedy, v ktorej deklarácii ho voláme, inteligentnú referenciu na triedu správanie s názvom name.

12.8 Dotazník pre externého testera

Dotazník

Meno testera: *Michal Kolesár*

Pozn.: Odpovedajte na otázku zakrúžkovaním jedného z čísel 1 až 5. Čísla 1 až 5 predstavujú hodnotenie podobné známkam v škole. Známka jeden teda predstavuje najväčšiu spokojnosť, či súhlas a naopak, známka päť predstavuje najväčšiu nespokojnosť či nesúhlas.

Otázky

Otázka: Ako sa Vám páčilo testovanie? **J**

Odpoveď: 1 2 3 4 5

Otázka: Ako ste boli spokojný s testovacím prostredím?

Odpoveď: 1 2 3 4 5

Otázka: Ako hodnotíte kvalitu evolučného serveru?

Odpoveď: 1 2 3 4 5

Otázka: Ako hodnotíte kvalitu samotného hráča?

Odpoveď: 1 2 3 4 5

Otázka: Ako hodnotíte kvalitu samotnej evolúcie?

Odpoveď: 1 2 3 4 5

Otázka: Ako hodnotíte predvedené výsledky evolúcie?

Odpoveď: 1 2 3 4 5

Poznámky:

.....

.....

.....

V Bratislave dňa: Podpis Testera

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

RoboCup – tretí rozmer
(Riadenie projektu)

Tím č. 17: Neurotics
Členovia tímu: Aleš Katona, Gabriel Braniša, Peter Čimo,
Tomáš Labuda, Juraj Šimon, Ján Kolesár
Študijný program: IS/SI
Ročník prvý, inžinierske štúdium
Email: team17.robocup@gmail.com
Web: http://www2.dcs.elf.stuba.sk/TeamProject/2007/team17is-si/

Vedúci projektu: Ing. Marián Lekavý
Ak. rok: 2007/2008

Obsah

Zoznam obrázkov

Zoznam tabuliek

1. Úvod

1.1 Účel dokumentu

V tomto dokumente sa nachádzajú informácie ohľadne riadenia a komunikácie v našom tíme.

1.2 Štruktúra dokumentu

Dokument je rozdelený na niekoľko častí. V prvej časti sa nachádza štruktúra tohto dokumentu. V druhej časti sa nachádza naša ponuka k predmetu Tvorba softvérového systému v tíme. Tretia kapitola obsahuje náš pôvodný hrubý plán na zimný semester ako aj skutočný zjemnený plán. Súčasťou je aj Ganttova schéma. V tretej kapitole uvádzame rozdelenie úloh v tíme, tak z dlhodobého, ako aj z krátkodobého hľadiska. Štvrtá kapitola obsahuje všetky naše zápisnice od 12.10.2007 do 9.11.2007. V piatej kapitole sú zadefinované štandardy tímu ktoré bude dodržiavať počas celého trvania projektu. V šiestej časti spomíname použitú literatúru. V prílohe A sa potom spomína ponuka nášho tímu. V prílohe B sa nachádzajú zápisnice zo všetkých stretnutí tímov. V prílohe C uvádzame štandardy tímu a v prílohe D sa nachádza zjemnený plán za obdobie celého zimného semestra.

2. Ponuka

V prílohe A sa nachádza ponuka nášho tímu pôvodne vytvorená pre projekt ALUMNI.

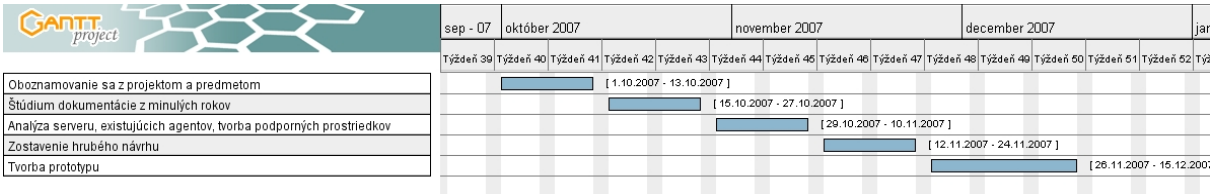
3. Plán projektu

Táto časť dokumentu obsahuje hrubý predbežný plán ktorý sme si stanovili na začiatku projektu. Pri tvorbe tohto dokumentu a po preštudovaní našich zápisníc sme zostavili skutočný plán podľa ktorého sme naozaj postupovali. Z tohto plánu sme zostavili Ganttovu schému.

3.1 Hrubý predbežný plán projektu

Obdobie	Popis	Stav
1.-2. týždeň	Oboznamovanie sa s projektom a predmetom	Ukončené
3.-4. týždeň	Štúdium dokumentácie z minulých rokov	Ukončené
5.-6. týždeň	Analýza serveru, existujúcich agentov, tvorba podporných prostriedkov	Ukončené
7.-8. týždeň	Zostavenie hrubého návrhu	Ukončené
9.-12. týždeň	Tvorba prototypu	Ukončené

Tab. 1 Hrubý predbežný plán na zimný semester



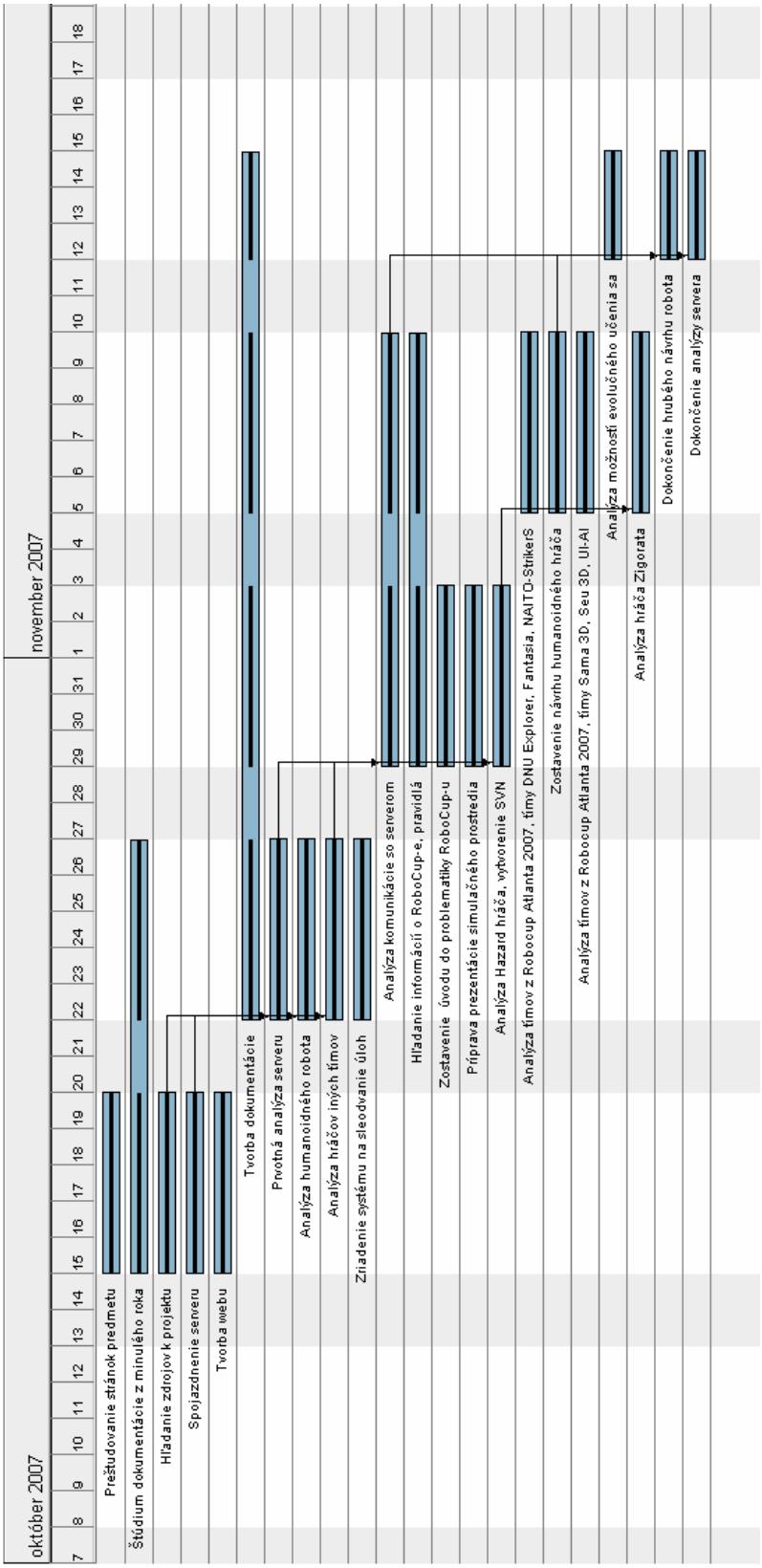
Obr. 1 Ganttova schéma hrubého plánu

3.2 Zjemnený plán za obdobie od 12.10.2007 do 15.11.2007

Úloha	**Začiatok**	**Koniec**	**Zodpovedný**
Preštudovanie stránok predmetu	*15.10.2007*	*20.10.2007*	Všetci
Štúdium dokumentácie z minulého roka	*15.10.2007*	*27.10.2007*	Všetci
Hľadanie zdrojov k projektu	*15.10.2007*	*20.10.2007*	Gabriel Braniša, Tomáš Labuda
Spojazdnenie serveru	*15.10.2007*	*20.10.2007*	Aleš Katona, Ján Kolesár
Tvorba webu	*15.10.2007*	*20.10.2007*	Peter Čimo, Juraj Šimon
Tvorba dokumentácie	*22.10.2007*	*15.11.2007*	Juraj Šimon
Prvotná analýza serveru	*22.10.2007*	*27.10.2007*	Tomáš Labuda
Analýza humanoidného robota	*22.10.2007*	*27.10.2007*	Ján Kolesár
Analýza hráčov iných tímov	*22.10.2007*	*27.10.2007*	Gabriel Braniša
Zriadenie systému na sleodvanie úloh	*22.10.2007*	*27.10.2007*	Peter Čimo
Analýza komunikácie so serverom	*29.10.2007*	*10.11.2007*	Tomáš Labuda
Hľadanie informácií o RoboCup-e, pravidlá	*29.10.2007*	*10.11.2007*	Ján Kolesár
Zostavenie úvodu do problematiky RoboCup-u	*29.10.2007*	*3.11.2007*	Gabriel Braniša
Príprava prezentácie simulačného prostredia	*29.10.2007*	*3.11.2007*	Aleš Katona
Analýza Hazard hráča, vytvorenie SVN	*29.10.2007*	*3.11.2007*	Peter Čimo
Analýza tímov z Robocup Atlanta 2007, tímy DNU Explorer, Fantasia, NAITO-StrikerS	*5.11.2007*	*10.11.2007*	Peter Čimo
Zostavenie návrhu humanoidného hráča	*5.11.2007*	*10.11.2007*	Gabriel Braniša, Aleš Katona
Analýza tímov z Robocup Atlanta 2007, tímy Sama 3D, Seu 3D, UI-AI	*5.11.2007*	*10.11.2007*	Juraj Šimon
Analýza možností evolučného učenia sa	*12.11.2007*	*15.11.2007*	Ján Kolesár
Analýza hráča Zigorata	*5.11.2007*	*10.11.2007*	Gabriel Braniša, Aleš Katona
Dokončenie hrubého návrhu robota	*12.11.2007*	*15.11.2007*	Aleš Katona
Dokončenie analýzy servera	*12.11.2007*	*15.11.2007*	Tomáš Labuda

Tab. 2 Zjemnený plán za obdobie 15.10.2007 až 15.11.2007

3.3 Ganttova schéma



Obr. 2 Ganttova schéma za obdobie 15.10.2007 až 17.12.2007

3.4 Zjemnený plán za obdobie od 19.2.2008 do 18.5.2008

Úloha	**Začiatok**	**Koniec**	**Zodpovedný**
Implementácia evolúcie	*19.2.08*	*26.2.08*	Ján Kolesár, Aleš Katona
Korekcie dokumentácie	*19.2.08*	*26.2.08*	Gabriel Braniša, Tomáš Labuda
Hľadanie statických správaní v iných tímoch	*19.2.08*	*26.2.08*	Juraj Šimon, Peter Čimo
Vytvorenie jadra testovacieho systému pre správania	*27.2.08*	*4.3.08*	Aleš Katona, Peter Čimo
Pokračovanie v implementácií evolučného algoritmu	*27.2.08*	*4.3.08*	Ján Kolesár
Nahrávanie videa z OpenGL v linuxe, vytvorenie pralalizačného systému	*27.2.08*	*4.3.08*	Gabriel Braniša
Vytvorenie matematického modelu pre fitness	*27.2.08*	*4.3.08*	Tomáš Labuda
Analýza a prevzatie statických správaní	*27.2.08*	*4.3.08*	Juraj Šimon, Peter Čimo
Vytvorenie jadra testovacieho systému pre správania	*5.3.08*	*11.3.08*	Aleš Katona
Dokončovanie implementácie evolučného algoritmu a prispôsobenie pre paralelizáciu	*5.3.08*	*11.3.08*	Ján Kolesár
Vytvorenie paralelizačného systému	*5.3.08*	*11.3.08*	Gabriel Braniša
Vytvorenie matematického modelu pre fitness	*5.3.08*	*11.3.08*	Tomáš Labuda
Prevzatie statických správaní a zmapovanie možností spustenia klienta.	*5.3.08*	*11.3.08*	Juraj Šimon, Peter Čimo
Ladenie evolúcie	*12.3.08*	*26.3.08*	Ján Kolesár
Dokončenie fitness funkcie	*12.3.08*	*26.3.08*	Tomáš Labuda
Pokračovanie v implementácií evolučného servera	*12.3.08*	*26.3.08*	Gabriel Braniša
Statické správania	*12.3.08*	*26.3.08*	Juraj Šimon, Peter Čimo
Implementácia testovania pre jednotlivé správania	*12.3.08*	*26.3.08*	Aleš Katona
Pokračovať vo vývoji testovacieho frameworku pre správania	*27.3.08*	*2.4.08*	Aleš Katona, Peter Čimo
Pokračovať v testovaní evolúcie.	*27.3.08*	*2.4.08*	Ján Kolesár
Dokončovanie fitness, integrácia s evolúciou.	*27.3.08*	*2.4.08*	Tomáš Labuda
Práca na vývoji servera pre	*27.3.08*	*2.4.08*	Gabriel Braniša

distrib. evolúciu			
How STL vector works? (Akým spôsobom uchováva vložené prvky)	*27.3.08*	*2.4.08*	Peter Čimo
Vytvoriť spúštacie skripty pre server (zabezpečiť znovuspustenie pri páde servera).	*27.3.08*	*2.4.08*	Juraj Šimon
Možnosti odstránenia závislosti servera na latexu.	*27.3.08*	*2.4.08*	Juraj Šimon
Aké sú možnosti použitia sieťového disku s linuxom pre server?	*27.3.08*	*2.4.08*	Juraj Šimon
Ladenie evolúcie	*3.4.08*	*9.5.08*	Aleš Katona, Ján Kolesár
Dohliadanie na evolúciu	*3.4.08*	*9.5.08*	Juraj Šimon, Peter Čimo
Vypracovanie technickej dokumentácie	*3.4.08*	*9.5.08*	Všetci
Ladenie distribuovanej evolúcie	*3.4.08*	*9.5.08*	Gabriel Braniša
Externé testovanie	*11.5.08*	*12.5.08*	

Tab. 3 Zjemnený plán od 19.2.08 do 18.5.08

4. Rozdelenie úloh v tíme

V našom projekte sme analyzovali nasledujúce krátkodobé a dlhodobé úlohy.

4.1 Krátkodobé úlohy

Činnosť	**Zodpovedný**
Analýza hráčov iných tímov	Peter Čimo, Juraj Šimon, Gabriel Braniša, Aleš Katona
Tvorba špecifikácie a návrhu	Aleš Katona
Analýza serveru a komunikácie	Tomáš Labuda
Inštalácia podporných webových prostriedkov	Peter Čimo
Inštalácia serveru, hráčov	Aleš Katona, Gabriel Braniša

Tab. 4 Krátkodobé úlohy v tíme

4.2 Dlhodobé úlohy

Úlohy	**Zodpovedný**
tvorba dokumentácie a manažér vývoja, vedúci tímu	Juraj Šimon
Manažér kvality, udržiavanie webu	Peter Čimo
Analytik, Tvorba plánovo	Gabriel Braniša
Technické a podporné činnosti	Aleš Katona
Manažér komunikácie	Ján Kolesár
Systémový analytik	Tomáš Labuda

Tab. 5 Dlhodobé úlohy v tíme

4.3 Podrobnejší opis úloh

Vedúci tímu

- Komunikácia v tíme
- Prideľovanie úloh
- Kontrola plnenia úloh

Manažér kvality

- Dodržiavanie štandardov

Analytik, Systémový analytik

- Analýza údajov z externých zdrojov

Technické a podporné činnosti

- Systémové programovanie
- Podpora vývoja

Manažér komunikácie

- Komunikácia s ostatnými členmi tímu
- Synchronizácia medzi členmi tímu

Udržiavanie webu

- Manažment webového sídla
- Manažment SVN repozitára

Tvorba dokumentácie

- Vytváranie a spájanie častí dokumentácie do jedného celku

Manažér vývoja

- Konzultácia úloh s vedúcim projektu
- Návrh nových úloh
- Koordinácia úloh

5. Zhodnotenie projektu

V tejto kapitole by sme chceli zhodnotiť ako sme sa držali prvotného plánu projektu a prečo a ako sme sa od neho odkláňali.

V prvom semestri sme oproti pôvodnému hrubému plánu o niečo rýchlejšie zvládli fázu oboznámenia sa s projektom a študovania dokumentácií iných tímov z minulých rokov. Je to preto, lebo takýchto tímov nebolo veľa, a takisto informácií na internete o samotnom RoboCup-e 3D nieje veľké množstvo, a tak ich štúdium nezabralo toľko času. Samotná príprava prostriedkov pre vývoj, ako je inštalácia serveru a podporných aplikácií (SVN), nám trvala približne dva týždne, teda toľko koľko bolo predpokladané v pôvodnom pláne. Pri tvorbe hrubého návrhu sme sa však pozdržali asi o dva týždne. V tomto smere prebehlo množstvo diskusií na stretnutiach tímu a konzultovali sme mnohé možnosti návrhu.
Samotná tvorba prototypu sa tiahla už od doby analýzy RoboCup serveru. Už vtedy sme si vyhliadli vhodného hráča, na ktorom budeme stavať. V poslednej fáze zimného semestra sme hráča patrične upravili, a skombinovali s hráčom jedného z tímov z minulých rokov. Celkovo sme teda dodržali celkový plán prác, avšak niektoré fázy trvali inú dobu ako boli naplánované.

V druhom, semestri sme si plán prác naplánovali o niečo podrobnejšie ako v zimnom semestri. Po začatí semestra sme sa hneď pustili do dokončovacích prác na prototype. Jednalo sa len o menšie zmeny potrebné na to, aby sme mohli začať implementovať evolúciu samotnú. Popri tejto činnosti sme taktiež zapracovávali zmeny do dokumentácie podľa posudku druhého týmu. Tieto dve fázy nám trvali asi dva týždne, teda toľko, koľko sme si naplánovali. Samotná implementácia tried evolúcie a evolučného behavioru nám však trvala o niečo dlhšie, ako dva týždne v pôvodnom pláne. Bolo to zapríčinené pomerne veľkou zmenou v návrhu evolúcie. Evolúciu sme prerábali na distribuovanú, bežiacu zároveň na viacerých počítačoch. Toto spôsobilo zdržanie asi tri až štyri týždne oproti pôvodnému plánu. Samotnú evolúciu sa nám teda podarilo spustiť so značným oneskorením. Po spustení sme potom ešte týždeň, či dva, odlaďovali zistené chyby. Zhodnotili sme výsledky, a rozhodli sa evolúciu spustiť na viacerých počítačoch samostatne, namiesto použitia distribuovanej evolúcie. Naše zistenie sú opísané v dokumentácií k produktu. Celkovo teda sme v letnom semestri trocha zaostali za pôvodným plánom z dôvodu značnej zmeny v návrhu evolúcie. Zistený rozdiel sme sa však snažili čo najrýchlejšie dohnať.

6. Zápisy zo stretnutí

Príloha B obsahuje všetky zápisy z našich stretnutí od 12.10.2007 do 6.12.2007.

7. Štandardy tímu

Príloha C obsahuje štandardné šablóny dokumentov aké používame pri tvorbe zápisnice a šablónu preberacieho protokolu.

7.1 Kódové štandardy

Pri písaní zdrojového kódu, je pre jeho dobrú čitateľnosť, dobré dodržiavať nasledujúce zásady.

7.1.1 Odrážkovanie

Funkcie:

- Typ návratovej hodnoty funkcie, názov funkcie a parametre funkcie v jednom riadku.
- Otvárajúca a zatvárajúca zátvorka na novom riadku bez odsadenia.

```
int mojaFunkcie(int parameter)
{
...
}
```

Cykly, podmienky, … :

- Ak je telo na viac riadkov, otváracia a zatváracie zátvorky na nový riadok, samotné telo odsadiť(tabulátor).

```
if (parameter > 5)
{
     writeln(...);
     ...
}
```

- Ak je telo jednoriadkove pokračovať v aktuálnom riadku bez nového riadku.

```
if (parameter = 9) writeln(...);
```

- V cykoloch používať názvy premenných *i, j, k, l,* …

```
for (int i;i < 4;i++) for (int j;j < 4;j++) writeln(...);
```

- Za príkazom *if, while,* … nechať medzeru.
- V podmienke pri zátvorkách nenechávať medzeru.

```
if (parameter = 9) ...
```

- Pri aritmetických a logických operátoroch nechávať medzeru z oboch strán.

```
tmp += 5 + 6 * 53;
```

- Kľúčové slovo *else* písať buď na nový riadok, pri jednoriadkovom *if* bez zátvoriek, alebo za ukončovaciu zátvorku, otváraciu zátvorku potom písať za *else* na tom istom riadku.

```
if (parameter > 5)
{
     writeln(...);
} else {
     writeln(...);
}
```

alebo:

```
if (parameter = 9) writeln(...);
else writeln(...);
```

Triedy, structy:

- Pri dedeni, za nazvom triedy, ktora dedi bodkociarka bez medery a za nou mezdera.
- Otvárajúca a zatvárajúca zátvorka na novom riadku bez odsadenia.
- Kľúčové slova ako *private, protected, public* bez odsadenia.
- Premenné a funkcie odsadiť(tabulator).
- Najprv deklarovať premenné až potom funkcie.

```
class MojaTrieda: public MaterskaTrieda
{
private:
     int xxx;

public:
     int a;
     int b;

     MojaTrieda();
     void vypisNieco(char *text);
};
```

7.1.2 Pomenovanie

Premenné a funkcie:

- Začíname malým písmenom, viacslovné premenné píšeme spolu, dalšie slová píšeme s veľkým prvým písmenom.

  Príklad: mojaFunkcia() , mojaPremenna

Triedy a structy:

- Začíname velkým písmenom, viacslovné premenné píšeme spolu, dalšie slová píšeme s veľkým prvým písmenom.
  Príklad: MojaTrieda
- Behavior triedy majú v názve ako posledné slovo „*Behavior*“
  Príklad: BasicBehavior, RunToBallBehavior

Makrá:

- Celý názov píšeme veľkými písmenami, slová v názve oddeľujeme podčiarkovníkmi

  Príklad: MOJE_MAKRO

7.1.3 Súbory a adresáre

Pomenovanie súborov:

- Každá trieda je umiestnená v samostatnom .h (deklarácia) a .cpp (telá funkcií) súbore, názov súboru je odvodený z názvu triedy – celý názov malými písmenami, slová oddelené podčiarkovníkom.

  Príklad:
  trieda sa volá RunToBallBehavior
  tak bude v súboroch run_to_ball_behavior.h a run_to_ball_behavior.cpp

Adresáre:

- Behavior moduly – adresár *behaviors/* .
- Ostatné moduly – adresár *others/* .

7.1.4 Ukážky

```
int mojaFunkcie(int parameter)
{
     int tmp;
     int viacSlovnaPremenna;

     if (parameter > 5)
     {
          writeln(...);
     } else {
          writeln(...);
     }

     while (parameter < 4)
     {
          tmp += 5 + 6 * 53;
     }

     if (parameter = 9) writeln(...);
     else writeln(...);

     for (int i;i < 4;i++) writeln(...);

     return 7;
}

struct MojStruct
{
     int a;
```

```
     int b;
};

class MojaTrieda: public MaterskaTrieda
{
private:
     int xxx;

public:
     int a;
     int b;

     MojaTrieda();
     void vypisNieco(char *text);
};

#define MOJE_MAKRO 1
```

8. Použitá literatúra

[1] Bieliková M.: Tvorba systému v tíme I, II, http://www2.fiit.stuba.sk/~bielik/courses/tp-slov/tp-main.html#dokumentacia

9. Prílohy

9.1 Príloha A – Ponuka tímu

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

ALUMNI

Informačný systém pre komunikáciu s absolventmi

Ponuka

Bc. Peter Čimo
Bc. Aleš Katona
Bc. Ján Kolesár
Bc. Tomáš Labuda
Bc. Milan Herda
Bc. Juraj Šimon

Odbor: Softvérové inžinierstvo
Dátum: 30.9.2007

Zadanie

Naša FIIT má záujem vhodnou formou prezentovať verejnosti svojich absolventov. Zároveň by fakulta rada udržiavala s absolventmi neformálny odborný kontakt pomocou webu, a tiež by rada poskytla svojim absolventom rámec na neformálnu odbornú a spoločenskú komunikáciu medzi nimi navzájom. Zámerom projektu je vytvoriť informačný systém, ktorý bude plniť uvedené úlohy.

Ciele systému:

- *Prezentovať základné informácie o absolventoch verejnosti.* Znamená to zabezpečiť vytvorenie a udržiavanie databázy absolventov a vhodne prezentovať základné informácie o jednotlivcovi verejnosti na webe bez obmedzenia prístupu. Predpokladáme stručnú informáciu o absolventovi, kedy študoval, absolvoval, jeho špecializácia, o téme jeho bakalárskej, prípadne diplomovej práce, prípadne abstrakt práce. Tu by bolo vítané rozšíriť prezentáciu o grafické vyjadrenie zamestnanosti a odbornosti absolventov z rôznych hľadísk, pokiaľ dokážeme od nich získať k tomu potrebné údaje.
- *Sprostredkovať fakulte získavanie aktuálnych informácií o absolventoch v praxi.* Ide o aktualizáciu kontaktu, zamestnania, profesijného vývoja, odborného zamerania, oblasti, v ktorej je aktívny, sfér odborného záujmu a pod., tie, ktoré poskytne sám absolvent. Táto oblasť je veľmi citlivá, vyžaduje záujem o kontakt z oboch strán a je podmienená prísnou ochranou údajov s vhodne zorganizovaným autorizovaným prístupom. Je to dôležitá, pre fakultu užitočná úloha, treba ju uvažovať.
- *Umožniť absolventom vzájomnú komunikáciu.* Má to byť jednoduchá a bezpečná komunikácia v informatickej komunite chránená starostlivo navrhnutými prístupovými právami pre skupiny autorizovaných účastníkov. Má slúžiť na neformálnu výmenu informácií v komunite kolegov, rovesníkov, odborníkov z praxe, ktoré komunite poskytne sám účastník. Okrem sprostredkovania kontaktu môže byť úlohou tejto časti systému informovať záujemcov o odborných aktivitách komunity, poskytnúť pre ne priestor – fórum, prípadne ďalšie vhodné činnosti.

Dôležité požiadavky:

- zachovanie bezpečného prístupu k informáciám
- jednoduché rozhranie
- systém bez ďalších hardvérových nárokov
- jednoduchá a bezpečná komunikácia medzi všetkými používateľmi systému
- modularita a rozšíriteľnosť
- škálovateľnosť
- rozhranie pre získavanie štatistických dát pre fakultu
- import základných údajov z fakultných IS
- export do súboru, možnosť umiestniť vybrané údaje ako prezentáciu o absolventoch na pamäťové médiá
- nasadenie do skúšobnej prevádzky v marci 2008

Projekt riešili vlaňajšie tímy č.15 a 18, každému z nich sa podarilo vytvoriť vhodný základ systému. Viac informácií o ich postupoch a dosiahnutých výsledkoch riešenia nájdete na webových stránkach:

- http://www2.dcs.elf.stuba.sk/TeamProject/2006/team18/
- http://www2.dcs.elf.stuba.sk/TeamProject/2006/team15/public_html/

V tomto akademickom roku treba vytvorené systémy analyzovať, zhodnotiť stav, urobiť výber, rozšíriť vybraný systém o novú požadovanú funkčnosť a dotvoriť ho tak, aby ho tohoroční absolventi už mohli používať.

Predstavenie tímu

Bc. Aleš Katona:

Počas bakalárskeho štúdia nadobudol poznatky z jazykov C, C++, Java, C#, Object Pascal a PHP. Témou bakalárskeho projektu bola multiplatformová sieťová knižnica – Lnet. Od roku 2005 je živnostníkom v oblasti IT. Je členom open source tímu Free Pascal a prácu v menšom tíme pozná veľmi dobre. Na strednej škole absolvoval kurz Cisco CCNA, 4 semestre. Pri práci na komerčných projektoch získal praktické poznatky o databázach PostgreSQL, MySQL a jazyku PHP, web serveroch Apache a Lighttpd. Z osobnej iniciatívy získal praktické vedomosti o správe systému Linux.

Bc. Juraj Šimon

Počas bakalárskeho štúdia nadobudol základné poznatky z jazykov C,C++ a Java. Témou jeho bakalárskej práce bol Adaptívny hyper mediálny systém. Od konca roku 2005 je živnostníkom v oblasti IT. Z prác na projektoch získal praktické skúsenosti o programovaní v PHP, Visual Basicu a základy jazyka SQL.

BC. Ján Kolesár

Počas bakalárskeho štúdia získal skúsenosti s programovacími jazykmi C/C++, Java, Pascal, SQL a HTML. Pracoval v prostrediach MS Visual Studio, Eclipse, NetBeans, FireBird Maestro, Delphi a čiastočne v prostredí Lazarus. Skúsenosti s tvorbou a prácou s databázovými systémami ziskal na predmete Databázové systémy a momentálne ich rozvíja v komerčnom projekte s technológiami FireBird a Delphi.

Bc. Peter Čimo

Popri štúdiu sa už niekoľko rokov podieľa tvorbe robustných internetových portálov a internetových obchodov, kde sa snaží uplatňovať vedomosti z bakalárskeho štúdia (postupy charakteristické pre softvérové inžinierstvo). Okrem najrôznejších webových technológií ovláda aj programovacie jazyky C/C++ a Pascal a databázové prostredia ako MySQL, PostgreeSQL či Oracle. Má skúsenosti s vývojom pod operačnými systémami Linux aj Windows.

Bc. Tomáš Labuda

Je absolventom bakalárskeho štúdia na FIIT STU v odbore Informatika. Téma jeho záverečného projektu bola „Modelovanie deployment prostredia pomocou UML2.0“. Vyše dvoch rokov pracuje v softvérovej firme, kde sa špecializuje na vývoj J2EE aplikácií. Má široké znalosti z vývoja webových aplikácií (hlavne na platforme Java, menej PHP a ASP.NET) a využívania webovo orientovaných technológií (Web Services, XML, XSLT, ...). Má rozsiahle znalosti jazyka UML, ktoré nadobudol hlavne pri práci na bakalárskom projekte.

Bc. Milan Herda

Motivácia k tomuto projektu

Počas štúdia na vysokej škole človek nadobudne množstvo skúseností, získa nových priateľov, kolegov alebo spolupracovníkov z prostredia fakulty kde študuje, ako aj z prostredia celej univerzity. Po skončení štúdií a nástupu do zamestnania však mnohé z týchto kontaktov či už vedomky alebo nie zanikajú, strácajú predošlú pevnosť. Je to prirodzený proces, človek vystavený novému prostrediu naväzuje kontakty nové, súvisiace s jeho novým zameraním a cieľmi, a staré kontakty postupne upadajú do zabudnutia.
Je to však škoda, pretože z udržiavaných vzťahov medzi jednotlivými absolventmi navzájom ako aj medzi fakultou a jej absolventmi by mali nesporný prospech obe strany. Na jednej strane fakulta, ktorá by prostredníctvom udržiavaných vzťahov s absolventmi mala akúsi spätnú väzbu a vedela by lepšie zhodnotiť svoje úsilie. Vedela by kde sa jej študentom podarilo zamestnať, ako aj to či pracujú v odpore ktorý vyštudovali. Zároveň by tak mala akýsi archív všetkých absolventov. V neposlednom rade by sa však vedela pochváliť ich úspechmi a prilákať tak nových študentov do daného oboru. Na strane druhej máme absolventov ktorí by z udržiavaných kontaktov mohli získavať nové informácie, pomoc či radu, či dokonca aj naväzovať nové kontakty. Ale aby obe strany mohli ťažiť z takýchto spoločných vzťahov musí tu byť niečo, čo by poskytovalo dané služby. Musel by tu byť systém ktorý by zabezpečoval určitú komunikáciu medzi fakultou a jej absolventmi. V minulých rokoch bolo prostredníctvom tímových projektov na túto tematiku vypracovaných viacero projektov. Bol skonštruovaný základ takéhoto systému, ktorý však ešte nespĺňa všetky požiadavky nasadenia do praktickej prevádzky.
Je teraz na nás preštudovať tieto minulé projekty a to čo už bolo vykonané. Chceme pokračovať v tomto úsilí, a podporiť a vylepšiť tento projekt a pomôcť a popohnať ho na jeho ceste do praktického nasadenia.

Ponuka riešenia

V rámci náväznosti na minuloročné projekty by sme chceli prevziať pozitívne črty týchto projektov a využiť poznatky a skúsenosti členov nášho tímu z praxe. Jednou z hlavných čŕt by mala byť jednoduchosť v zmysle jednoduchého a intuitívneho ovládania, aby sa užívateľ nemusel s aplikáciou učiť pracovať, ale aby s ňou rovno pracoval. Projekt by bol vyhotovený ako moderná „web 2.0“ aplikácia poháňaná výkonným databázovým riešením v pozadí.

Predpokladané zdroje projektu

Vývojové prostriedky:

Pre dokumentačné účely budú použité prostriedky na písanie dokumentov a grafov (MS Word, Open Office) a tvorbu diagramov (Dia).
Vývoj samotného riešenia bude prebiehať v heterogénnom prostredí. Na správu zdieľania sa použije verziovací systém SubVersion. Každý člen tímu si vyberie editor zdrojových kódov podľa vlastných preferencií.

Softvérové prostriedky:
Systém bude vyžadovať web server, avšak nie je nutné vybrať konkrétny web server. Výhodou riešenia ja možnosť systém nainštalovať na rôznych serveroch bez väčších problémov. Jedinou požiadavkou na server je podpora PHP 5 ako modul, alebo cez CGI prípadne FastCGI.
Samotný systém bude implementovaný v jazyku PHP 5.
Pre správu dát použijeme databázový systém PostgreSQL, ktorý bol použitý predošlým tímom, a považujeme ho za vhodný.

Hardvérové prostriedky:
Procesor x86 - 32, alebo 64 bitový, 1 GHz
Pamäť - aspoň 512 MB
Úložný priestor - aspoň 2 GB

Ďalšie štyri témy ktoré nás zaujali

- Tvorba testov s využitím LaTeXu
- Distribuovaný systém na riešenie symetrickej hry
- Analyzátor sociálnych sietí
- Tvorba rozvrhov

Rozvrh členov tímu

		7:00 7:50	8:00 8:50	9:00 9:50	10:00 10:50	11:00 11:50	12:00 12:50	13:00 13:50	14:00 14:50	15:00 15:50	16:00 16:50	17:00 17:50	18:00 18:50	19:00 19:50	20:00 20:50
Po	Čimo	APS			@NP				ML2		@TSST1		@VSS		
	Kolesár	APS						@NP			@TSST1		@VSS		
	Labuda	APS									@TSST1		@VSS		
	Šimon	APS			@NP				ML2		@TSST1		@VSS		
	Herda														
	Katona										@TIST1		@VIS		
Ut	Čimo									MSI		@MSI[1]		@MSI[1]	
	Kolesár									MSI		@MSI[1]		@MSI[1]	
	Labuda									MSI		@MSI[1]		@MSI[1]	
	Šimon									MSI		@MSI[1]		@MSI[1]	
	Herda														
	Katona											BMIS		@BMIS	
St	Čimo								@ML2						
	Kolesár	NS										@NS[2]			
	Labuda	NS				@NS[2]									
	Šimon								@ML2						
	Herda														
	Katona					PDT					FS				
Št	Čimo				NP				ASS						
	Kolesár				NP				ASS						
	Labuda				NP				ASS						
	Šimon				NP				ASS						
	Herda														
	Katona				VI			@VI			AIS				
Pi	Čimo														
	Kolesár														
	Labuda				@NP										
	Šimon														
	Herda														
	Katona								@FS						
Vysvetlivky															
		@cvičenie		prednáška											
		SI		IS											
1 (@MSI)		všetci skupina B – nedeterministické striedanie začiatkov o 17:00 a 19:00													
2 (@NS)		cvičenie = konzultácie => možnosť zúčastniť sa v ľubovolnom termíne (9:00-17:00)													

Kontakt

Nasledujúca tabuľka obsahuje mailové kontakty na všetkých členov tímu.

Meno	**Osobný kontakt**	**Školský kontakt**
Aleš Katona	almindor@gmail.com	xkatonaa@is.stuba.sk
Juraj Šimon	sajmonphoenix@gmail.com	xsimonj@is.stuba.sk
Peter Čimo	gunxter@gmail.com	xcimo@is.stuba.sk
Tomáš Labuda	tomasko.labuda@gmail.com	xlabudat@is.stuba.sk
Ján Kolesár	jkolesar@gmail.com	xkolesarj@is.stuba.sk
Milan Herda		xherdam@is.stuba.sk

9.2 Príloha B – Zápisnice zo stretnutí tímu

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: *1.*
Dátum stretnutia: *12.10.2007*
Čas: *neurčitý*
Miesto: *žiadne, pokyny zaslané prostredníctvom emailu*
Prítomní: -

Stretnutie viedol: -
Zápisnicu vypracoval: *Juraj Šimon*

Kontrola a hodnotenie úloh

Jednalo sa o naše prvé stretnutie a tak sme doteraz nemali zadelené žiadne úlohy k vypracovaniu.

Záznam zo stretnutia

Od Ing. Mariána Lekavého sme dostali email s inštrukciami o rozdelení úloh na nasledujúci týždeň.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Preštudovanie stránok predmetu	všetci
Preštudovať dokumentáciu z minulého roka	všetci
Vyhľadanie odkazov na tematiku robotického futbalu	Tomáš Labuda, Gabrial Braniša
Rozbehať vlaňajšie zdrojové kódy	Ján Kolesár, Aleš Katona
Vytvoriť webové sídlo tímu	Juraj Šimon, Peter Čimo

Zhrnutie

Dostali sme zadelenie úloh na najbližší týždeň, a odkaz na dokumentáciu minuloročného tímu. Stretnutie prebehlo úspešne.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: *2.*
Dátum stretnutia: *19.10.2007*
Čas: *07:00*
Miesto: D202
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Juraj Šimon*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Preštudovanie stránok predmetu	všetci
Preštudovať dokumentáciu z minulého roka	všetci
Vyhľadanie odkazov na tematiku robotického futbalu	Tomáš Labuda, Gabrial Braniša
Rozbehať vlaňajšie zdrojové kódy	Ján Kolesár, Aleš Katona
Vytvoriť webové sídlo tímu	Juraj Šimon, Peter Čimo

Zhrnutie vypracovania úloh

Všetci členovia týmu si dôkladne preštudovali stránky predmetu. Ďalej sme pozerali aj projekty z minulého aj predminulého roka. Tomáš Labuda a Gabriel Braniša vyhľadali na internete niekoľko odkazov na iné tými robotického futbalu a stratégie vo futbale. Tieto odkazy boli umiestnené na našu stránku. Stránku vytvorili Juraj Šimon a Peter Čimo. Ďalej sme vyrobili náš plagát ktorý je taktiež umiestnený na stránke. Ján Kolesár a Aleš Katona sa pokúšali rozbehať server a no zatiaľ niektoré knižnice stále nejdú.

Záznam zo stretnutia

Spoločne sme sa stretli a prebrali náš doterajší postup. Ing. Lekavý nás podrobnejšie oboznámil s úlohou a so skúsenosťami s robotickým futbalom z minulých rokov. Zhodli sme sa na použití novšej verzie serveru kde už figurujú humanoidný roboti. Všetky predošlé projekty boli vytvorené v staršej verzií. Našim cieľom je teda zistiť aké sú rozdiely medzi starou a novou verziou. Úloha rozbehania servera stále ostáva. Ďalšou úlohou bude naštudovať si základnú koncepciu hráča aký sa používa pri novom serveri. Z administratívnych úloh je to potom vytvorenie systému pre zapisovanie úloh a ich trvanie. Ďalej bol zvolený vedúci tímu Juraj Šimon ktorý bude vykonávať aj funkciu správcu dokumentácie. Za dodržiavanie štandardov v zdrojovom kóde bude zodpovedný Peter Čimo. Za dodržiavanie termínov bude zodpovedný Gabriel Braniša. Bol upravený čas stretnutí na 8:00 vo štvrtok.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Príprava a začatie písania dokumentácie, požiadavky	Juraj Šimon
Preštudovať dokumentáciu z minulého roka	všetci
Pohľadať čo najviac info o novom serveri, dokumentácia, fóra, tipy triky, proste všetko	Tomáš Labuda
Vyhľadanie čo najviac informácií o fungovaní humanoidného robota v robotickom futbale	Ján Kolesár
Naplniť systém úloh termínmi, vyhľadať čo najviac humanoidných hráčov na nete info o nich	Gabriel Braniša
Rozbehať vlaňajšie zdrojové kódy	Aleš Katona
Vytvoriť systém na písanie úloh členov tímu	Peter Čimo

Zhrnutie

Bližšie sme sa oboznámili s tematikou robotického futbalu, spresnili termíny stretnutí, administratívne záležitosti a funkcie členov v tíme. Dostali sme rozdelene úloh na ďalší týždeň.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 3.
Dátum stretnutia: *26.10.2007*
Čas: *08:00*
Miesto: D202
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Juraj Šimon*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Príprava a začatie písania dokumentácie, požiadavky	Juraj Šimon
Preštudovať dokumentáciu z minulého roka	všetci
Pohľadať čo najviac info o novom serveri, dokumentácia, fóra, tipy triky, proste všetko	Tomáš Labuda
Vyhľadanie čo najviac informácií o fungovaní humanoidného robota v robotickom futbale	Ján Kolesár
Naplniť systém úloh termínmi, vyhľadať čo najviac humanoidných hráčov na internete info o nich	Gabriel Braniša
Rozbehať vlaňajšie zdrojové kódy	Aleš Katona
Vytvoriť systém na písanie úloh členov tímu	Peter Čimo

Zhrnutie vypracovania úloh

Podarilo sa nám nájsť postup rozchodenia nového serveru. Server sme úspešne spustili. Ďalej sme našli niekoľko typov humanoidov a preštudovali ich schopnosti. Vytvorili sme systém na zápis úloh v podobe Gmail účtu. Naplnili sme toto konto termínmi odovzdaní, prednášok a stretnutí.

Záznam zo stretnutia

Ako obvykle na začiatku stretnutia sme prebrali náš doterajší postup. Ďalej sme sa zhodli na postupe po dobu prvého odovzdávania. Rozdelili sme si úlohy tak, aby sme naplnili tento prvý cieľ. Pokúsime sa rozanalyzovať niekoľko humanoidných hráčov, ich schopností a štruktúry. Ďalej budeme študovať systém komunikácie so serverom aká je štruktúra správ v komunikácií a podobne. Spolu s úvodom do problematiky robotického futbalu tieto informácie budú tvoriť analýzu problematiky. Ďalej preskúmame minuloročného hráča Hazarda a uvážime či bude z neho možné niektoré časti použiť. Napadla nás aj myšlienka evolučného učenia sa takže sa budeme venovať aj skúmaniu tejto problematiky. Nakoniec s pomocou týchto všetkých informácií zostavíme návrh riešenia resp. nového hráča.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Zápisnica, zostavenie dokumentácie	Juraj Šimon
Komunikácia servera, štruktúra komunikácie, hľadanie nových hráčov	Tomáš Labuda
Prihlásenie sa do robocup fór, zistenie typov hier a ich pravidiel, zistenie možností učenia sa pomocou evolučných algoritmov.	Ján Kolesár
Zostavenie všeobecného úvodu do problematiky robotického futbalu, analýza humanoidného hráča	Gabriel Braniša
Analýza robotického týmu hráčov, nastavenie serveru a hráčov na prezentáciu jednoduchej hry	Aleš Katona
Analýza Hazard hráča a jeho použiteľnosti a štruktúry kódu. Vytvorenie SVN repozitára pre uloženie zdrojových kódov.	Peter Čimo

Zhrnutie

Prebrali sme náš doterajší postup a stanovili si cestu k termínu prvého odovzdania. Načrtli sme možnú stratégiu tvorby nového hráča v podobe učenia sa pomocou evolučných algoritmov. Pre zostavenia analýzy preskúmame niekoľko humanoidných hráčov ako aj hráča z minulého roku. Na základe týchto informácií zostavíme potom návrh nášho riešenia humanoidného hráča.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 4.
Dátum stretnutia: *05.11.2007*
Čas: *18:00*
Miesto: DE150
Prítomní: *všetci*

Stretnutie viedol: Juraj Šimon
Zápisnicu vypracoval: *Juraj Šimon*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Zápisnica, zostavovanie dokumentácie	Juraj Šimon
Komunikácia servera, štruktúra komunikácie, hľadanie nových hráčov	Tomáš Labuda
Prihlásenie sa do robocup fór, zistenie typov hier a ich pravidiel, zistenie možností učenia sa pomocou evolučných algoritmov.	Ján Kolesár
Zostavenie všeobecného úvodu do problematiky robotického futbalu, analýza humanoidného hráča	Gabriel Braniša
Analýza robotického týmu hráčov, nastavenie serveru a hráčov na prezentáciu jednoduchej hry	Aleš Katona
Analýza Hazard hráča a jeho použiteľnosti a štruktúry kódu. Vytvorenie SVN repozitára pre uloženie zdrojových kódov.	Peter Čimo

Zhrnutie vypracovania úloh

Začali sme pracovať na zostavení špecifikácie a dokumentácie. Tomáš Labuda spracováva komunikáciu so serverom, úloha je hotová na 50%. Zostáva ešte preložiť niekoľko odstavcov. Ján Kolesár zohnal pravidlá robocupu3d z internetu z turnaja z roku 2007 v Atlante. Gabriel Braniša zostavil všeobecný úvod do robotického futbalu. Peter Čimo analyzoval kód hráča Hazarda z minulého roku a vytvoril SVN ktoré sme zatiaľ nenaplnili. Aleš Katona analyzoval hráčov The little green BATS. Rozbehnúť na serveri sa mu ich žiaľ zatiaľ nepodarilo. Juraj Šimon zostavil zápisnicu a začal postupne zostavovať dokumentáciu z dokumentov posielaných ostatnými členmi týmu.

Záznam zo stretnutia

Na začiatku sme si zhodnotili úlohy ktoré sme mali zadané a stav práce na nich. Každý vyjadril stupeň svojho postupu na úlohe. Bez dlhších rečí sme sa poradili o ďalšom postupe. Tomáš Labuda dokončí dokumentovanie serveru. Ján Kolesár bude pokračovať v štúdiu evolučných algoritmov aby tak mohol zostaviť časť návrhu nášho hráča. Peter Čimo a Juraj Šimon sa podrobnejšie pozrú na hráčov z robocup3d turnaja v Atlante z roku 2007. Aleš Katona spolu s Gabrielom Branišom sa pokúsia zostaviť prvotný hrubý návrh robota.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Analýza tímov z robocup3d turnaja v Atlante: Sama3D, SEU-3D, UI-AI.	Juraj Šimon
Komunikácia servera, štruktúra komunikácie, hľadanie nových hráčov – dokončenie.	Tomáš Labuda
Možnosti evolučného učenia sa, určenie metrík pre meranie úspešnosti rôznych činností robota	Ján Kolesár
Zostavenie prvotného hrubého návrhu robota	Gabriel Braniša, Aleš Katona
Príprava prezentácie serveru a hráčov	Aleš Katona
Analýza tímov z robocup3d turnaja v Atlante: DNU Explorer, Fantasia, Naito Strikers	Peter Čimo

Zhrnutie

Prehodnotili sme náš doterajší postup. Zostavili sme časť dokumentácie obsahujúcej pravidlá robocupu a úvod do robocup prostredia. Pokračujeme v plnení zadaných úloh. Ešte zanalyzujeme niekoľko tímov z robocup3d turnaja v Atlante 2007. Určíme si metriky ktorými budeme merať úspešnosť agenta pri evolučnom učení sa. Pripravíme si prezentáciu reálnej práce servera a hráčov.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 5.
Dátum stretnutia: *09.11.2007*
Čas: *08:00*
Miesto: D202
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Juraj Šimon*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Analýza tímov z robocup3d turnaja v Atlante: Sama3D, SEU-3D, UI-AI.	Juraj Šimon
Komunikácia servera, štruktúra komunikácie, hľadanie nových hráčov – dokončenie.	Tomáš Labuda
Možnosti evolučného učenia sa, určenie metrík pre meranie úspešnosti rôznych činností robota	Ján Kolesár
Zostavenie prvotného hrubého návrhu robota, naplnenie SVN	Gabriel Braniša, Aleš Katona
Príprava prezentácie serveru a hráčov	Aleš Katona
Analýza tímov z robocup3d turnaja v Atlante: DNU Explorer, Fantasia, Naito Strikers	Peter Čimo

Zhrnutie vypracovania úloh

Analýzu sme už skoro dokončili, zostáva dopracovať troch agentov z Atlanty a agenta Zigorata. Komunikáciu servera sme ešte nestihli plne dokončiť. Úspešne sme odprezentovali prostredie robocupu3d a vizualizačné prostredie. Taktiež analýza tímov Sama3D, SEU-3D a UI-AI je hotová a zakomponovaná do dokumentácie.

Záznam zo stretnutia

Na začiatku stretnutia sme oboznámili nášho vedúceho o postupe prác na našich úlohách. Ďalej sme diskutovali o návrhu. Zhodli sme sa že použijeme agenta Hazarda z minulého roku ako model správania. Avšak vzhľadom na to že sa odvtedy server zmenil ako model nižších činností ako je komunikácia so serverom alebo základné pohybové schopnosti z neho už zobrať nemôžeme. Podrobnejšie sme si rozobrali štruktúru agenta Zigorata a rozhodli sme sa že tento model základných činností preberieme od neho. Dohodli sme sa na termíne odovzdania dokumentácie nášmu vedúcemu. Pozreli sme si niekoľko inštruktážnych videí o evolučnom učení sa chodiť, humanoidného robota.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Zostavenie dokumentácie k odovzdaniu, výmena dokumentácie s druhým tímom	Juraj Šimon
Komunikácia servera, štruktúra komunikácie, hľadanie nových hráčov – dokončenie.	Tomáš Labuda
Možnosti evolučného učenia sa, určenie metrík pre meranie úspešnosti rôznych činností robota	Ján Kolesár
Zostavenie prvotného hrubého návrhu robota	Aleš Katona
Analýza hráča Zigorata	Gabriel Braniša
Analýza tímov z robocup3d turnaja v Atlante: DNU Explorer, Fantasia, Naito Strikers	Peter Čimo

Zhrnutie

Prehodnotili sme náš doterajší postup. Prebrali sme náš hrubý návrh a dohodli sa na podrobnostiach. Rozdelili sme si úlohy tak aby spoločne viedli k úspešnému odovzdaniu dokumentácie.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 6.
Dátum stretnutia: 15*.11.2007*
Čas: *08:00*
Miesto: D202
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Juraj Šimon*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Zostavenie dokumentácie k odovzdaniu, výmena dokumentácie s druhým tímom	Juraj Šimon
Komunikácia servera, štruktúra komunikácie, hľadanie nových hráčov – dokončenie.	Tomáš Labuda
Možnosti evolučného učenia sa, určenie metrík pre meranie úspešnosti rôznych činností robota	Ján Kolesár
Zostavenie prvotného hrubého návrhu robota	Aleš Katona
Analýza hráča Zigorata	Gabriel Braniša
Analýza tímov z robocup3d turnaja v Atlante: DNU Explorer, Fantasia, Naito Strikers	Peter Čimo

Zhrnutie vypracovania úloh

Peter Čimo a Gabriel Braniša dokončili analýzu im pridelených hráčov. Aleš Katona v spolupráci s Gabrielom Branišom a Jánom Kolesárom zostavili hrubý návrh hráča. Hráč sa bude učiť svoje schopnosti za pomoci evolučných algoritmov. Možnosti evolučných algoritmov podrobne preskúmal Ján Kolesár pričom určil aj možnosti ohodnocovacích funkcií. Tomáš Labuda dokončil opis servera vo verzii 0.5.6. Juraj Šimon zostavil dokumentáciu k projektu. Dokumentácia bola odovzdaná ako vedúcemu nášho tímu Ing. Mariánovi Lekavému tak aj druhému tímu č.11.

Záznam zo stretnutia

Na začiatku stretnutia sme odovzdali vypracovanú dokumentáciu vedúcemu tímu Ing Mariánovi Lekavému. Obe strany podpísali preberací protokol. Ďalej sme preberali možnosti reprezentácie génov pri evolučných algoritmoch aplikovaných na RoboCup. Následne sme si stanovili náš ďalší postup ktorým je tvorba prototypu. Rozdelili sme si úlohy medzi jednotlivých členov tímu.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Overenie funkčnosti budúcej simulácie na serveri	Tomáš Labuda
Tvorba evolučných algoritmov pre robota	Ján Kolesár
Naplnenie SVN (server a hráč/hráči)	Gabriel Braniša
Tvorba posudku tímu č.11	Juraj Šimon, Aleš Katona
Určenie kódových štandardov	Peter Čimo
Nájdenie univerzálneho komunikačného VoIP nástroja	Aleš Katona

Zhrnutie

Odovzdanie dokumentácie prebehlo v poriadku. Dohodli sme sa na ďalšom postupe. Postupne budeme tvoriť prototyp. Začneme teoretickou tvorbou evolučných algoritmov a praktickým odskúšaním niektorých vlastností servera. Ďalej vytvoríme základný model hráča schopného komunikovať so serverom.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 7.
Dátum stretnutia: 22*.11.2007*
Čas: *08:00*
Miesto: Softvérové štúdio
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Juraj Šimon*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Overenie funkčnosti budúcej simulácie na serveri	Tomáš Labuda
Tvorba evolučných algoritmov pre robota	Ján Kolesár
Naplnenie SVN (server a hráč/hráči)	Gabriel Braniša
Tvorba posudku tímu č.11	Juraj Šimon, Aleš Katona
Určenie kódových štandardov	Peter Čimo
Nájdenie univerzálneho komunikačného VoIP nástroja	Aleš Katona

Zhrnutie vypracovania úloh

Tomáš Labuda overil že je možné na server pripájať a odpájať jedného alebo viacerých hráčov ľubovoľný počet krát. Poukázal aj na problém pri pripojení a vložení hráča na hraciu plochu kedy nie je možné vybrať mu špecifickú polohu ako napr. určité natočenie kĺbov. Gabreil Braniša do SVN vložil zdrojové kódy hráča a serveru. Juraj Šimon vypracoval posudok k projektovej dokumentácií tímu č. 11 a odovzdal ho vedúcemu projektu Ing. Mariánovi Lekavému ako aj tímu č. 11. Peter Čimo vypracoval návrh kódových štandardov. Aleš Katona odporučil ako komunikačný nástroj VoIP nástroj TeamSpeak. Ján Kolesár pokračuje v návrhu evolučných algoritmov.

Záznam zo stretnutia

Na stretnutí sme preberali hlavne návrh kódových štandardov. Ujasnili sme si niektoré nedorozumenia. Peter Čimo vypracuje o dohodnutých štandardoch dokument. Ďalej sme preberali implementáciu modulov do prototypu. Návod pre tvorbu modulov dostal za úlohu Aleš Katona. On následne rozdelí implementáciu všetkých modulov medzi jednotlivých členov tímu. Na konci stretnutia sme odovzdali posudok tímu č. 11 nášmu vedúcemu projektu..

Rozdelenie úloh

Znenie úlohy	**Poverený**
Implementácia modulov	Všetci
Návod na tvorbu modulu	Aleš Katona
Vypracovanie reakcie na posudok	Juraj Šimon

Zhrnutie

Vypracovali sme posudok dokumentácie tímu č. 11. Dohodli sme sa na kódových štandardoch a rozdelili si úlohy implementácie modulov do prototypu.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 8.
Dátum stretnutia: 29.*11.2007*
Čas: *08:00*
Miesto: Softvérové štúdio
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Juraj Šimon*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Implementácia modulov	Všetci
Návod na tvorbu modulu	Aleš Katona
Vypracovanie reakcie na posudok	Juraj Šimon

Zhrnutie vypracovania úloh

Aleš Katona a Peter Čimo spoločne začali implementovať moduly do prototypu. Juraj Šimon vypracoval reakciu na posudok tímu č. 11. Implementácia základných modulov z dôvodu nedostatku času ešte nebola ukončená. Ján Kolesár pokračoval v tvorbe evolučných algoritmov. Gabriel Braniša a Tomáš Labuda sa zaoberali podpornými činnosťami v tíme.

Záznam zo stretnutia

Z dôvodu nedostatku času bolo naše stretnutie rýchle. V stručnosti sme si zhrnuli náš doterajší postup. Prebrali sme implementáciu modulov a jej budúci postup. Dohodli sme sa na vyhodení knižnice *zeitgeist* z implementácie prototypu.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Doplnenie popisu servera do dokumentácie	Tomáš Labuda
Tvorba evolučných algoritmov pre robota	Ján Kolesár
Implementácia prototypu	Gabriel Braniša, Aleš Katona, Peter Čimo
Korekcia chýb v dokumentácií	Juraj Šimon

Zhrnutie

Zostavili sme reakciu na posudok tímu č.11. Implementácia modulov pomaly napreduje. Dokumentáciu je potrebné očistiť od chýb.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 9.
Dátum stretnutia: 6.*12.2007*
Čas: *08:00*
Miesto: Softvérové štúdio
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Juraj Šimon*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Doplnenie popisu servera do dokumentácie	Tomáš Labuda
Tvorba evolučných algoritmov pre robota	Ján Kolesár
Implementácia prototypu	Gabriel Braniša, Aleš Katona, Peter Čimo
Korekcia chýb v dokumentácií	Juraj Šimon

Zhrnutie vypracovania úloh

Tomáš Labuda vypracoval doplnenie popisu servera na základe posudku tímu č.11 ktoré bolo zaradené do dokumentácie. Juraj Šimon pridal do dokumentácie kapitolu revízie, v ktorej reagujeme na posudok tímu č. 11 a dopĺňame zmeny. Ján Kolesár pokračuje v tvorbe implementácií evolučných algoritmov. Zvyšok tímu postupne implementoval väčšiu časť prototypu.

Záznam zo stretnutia

Na začiatku stretnutia sme zhodnotili naplnenosť úloh z minulého týždňa. Preberali sme odovzdávanie projektovej dokumentácie. Dohodli sme približný dátum prezentácie nášho prototypu druhému tímu č.11. Ďalej sme rozoberali čo všetko treba doplniť do dokumentácie a ako sa tieto úlohy rozdelia medzi jednotlivých členov tímu.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Návod na inštaláciu servera	Tomáš Labuda
Opis podrobného návrhu robota, doplnenie špecifikácie evolúcie	Ján Kolesár
Príprava prezentácie pre druhý tím	Gabriel Braniša
Implementácia prototypu, zostavenie dokumentácie prototypu	Peter Čimo, Aleš Katona
Doplnenie riadenia projektu	Juraj Šimon

Zhrnutie

Implementácia prototypu je takmer hotová. Implementácia evolučných algoritmov napreduje zatiaľ pomalším tempom. Sústredíme sa na dokončenie prototypu a dopracovaní dokumentácie.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 10.
Dátum stretnutia: 13*.12.2007*
Čas: *08:00*
Miesto: Softvérové štúdio
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Juraj Šimon*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Návod na inštaláciu servera	Tomáš Labuda
Opis podrobného návrhu robota, doplnenie špecifikácie evolúcie	Ján Kolesár
Príprava prezentácie pre druhý tím	Gabriel Braniša
Implementácia prototypu, zostavenie dokumentácie prototypu	Peter Čimo, Aleš Katona
Doplnenie riadenia projektu	Juraj Šimon

Zhrnutie vypracovania úloh

Tomáš Labuda vypracoval návod na inštaláciu servera. Na podrobnom návrhu a popise prototypu ďalej pracujú Peter Čimo a Aleš Katona. Gabriel Braniša pripravuje prezentáciu pre druhý tým. Juraj Šimon pripravuje dokumentáciu na odovzdanie. Ján Kolesár doplnil špecifikáciu evolúcie.

Záznam zo stretnutia

Na stretnutie sme doniesli predbežnú verziu dokumentácie. Vedúci projektu nám dal cenné rady ako ju ešte vhodne doplniť.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Doplnenie informácií k evolučným algoritmom	Ján Kolesár
Doplnenie informácií k prototypu	Peter Čimo, Aleš Katona
Príprava prezentácie pre druhý tím	Gabriel Braniša
Zhodnotenie prototypu	Tomáš Labuda
Doplnenie riadenia projektu	Juraj Šimon

Zhrnutie

Na stretnutí sme prebrali aké nedostatky ešte má naša dokumentácia a prebrali možnosti ich nápravy.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 11.
Dátum stretnutia: 19*.2.2008*
Čas: 07*:00*
Miesto: Softvérové štúdio
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Peter Čimo*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Návod na inštaláciu servera	Tomáš Labuda
Opis podrobného návrhu robota, doplnenie špecifikácie evolúcie	Ján Kolesár
Príprava prezentácie pre druhý tím	Gabriel Braniša
Implementácia prototypu, zostavenie dokumentácie prototypu	Peter Čimo, Aleš Katona
Doplnenie riadenia projektu	Juraj Šimon

Zhrnutie vypracovania úloh

Peter Čimo a Aleš Katona dokončili implementaciu prototypu a zostavili aj jeho dokumentáciu. Tomáš Labuda vypracoval návod na inštaláciu. Juraj Šimon doplnil riadenie projektu do projektovej dokumentácie. Gabriel Braniša pripravil prezentáciu pre druhý tím.

Záznam zo stretnutia

Na stretnutí sme si porozdeľovali úlohy na nasledujúce týždne. Rozobrali sme niektoré dôležité otázky a snažili sa na ne nájsť odpovede. Rozprávali sme sa o harmonograme na letný semester, termínoch a úlohách. Marián Lekavý poukázal na niektoré chyby v dokumentácií a upozonil, že silná konvergenia v evolucií nieje žiadúca.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Implementácia evolúcie	Ján Kolesár, Aleš Katona
Korekcie dokumentácie	Gabriel Braniša, Tomáš Labuda
Hľadanie statických správaní v iných tímoch	Juraj Šimon, Peter Čimo

Zhrnutie

Na stretnutí sme si povedali čo nás čaká v letnom semestri a určili úlohy.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 12.
Dátum stretnutia: 27*.2.2008*
Čas: 11*:00*
Miesto: Softvérové štúdio
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Aleš Katona*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

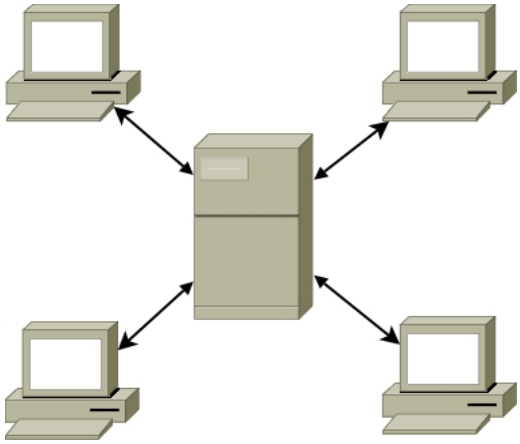
Znenie úlohy	**Poverený**
Implementácia evolúcie	Ján Kolesár, Aleš Katona
Korekcie dokumentácie	Gabriel Braniša, Tomáš Labuda
Hľadanie statických správaní v iných tímoch	Juraj Šimon, Peter Čimo

Zhrnutie vypracovania úloh

Aleš Katona a Ján Kolesár pokračujú v implementovaní evolučného algoritmu. Gabriel Braniša a Tomáš Labuda prezreli dokumentáciu a ponachádzali v nej chyby. Tieto sa posúdia a ďalej posunú na korekciu v dodatkových kapitolách. Juraj Šimon ani Peter Čimo zatiaľ z dôvodu väčšieho vyťaženia v iných predmetoch nehľadali žiadne statické správania.

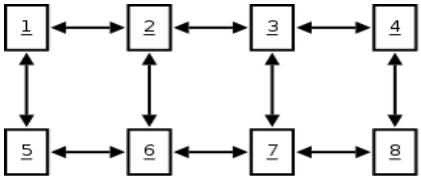
Záznam zo stretnutia

Na stretnutí sme si porozdeľovali úlohy na nasledujúce týždne. Rozobrali sme niektoré dôležité otázky a snažili sa na ne nájsť odpovede. Určili sme si metódu evolúcie ako sieťovú architektúru typu klient-server kde klienti budú vykonávať evolúcia a po určitom časovom úseku si vymenia nekoľko chromozómov.



Obr. č. 1 - Sieťová topológia

Urýchli sa tým evolučný proces nakoľko sa bude prehľadávať priestor riešení do niekoľkých smerov zároveň. To ktorý klient si s ktorým budú vymieňať chromozómy bude určovať konfiguračný súbor servera. Príklad topológie je uvedený na obrázku.



Obr. č. 2 – Evolučná topológia

Ďalej sme sa zhodli na vytvorení testovacieho správania ktoré bude testovať všetky ostatné správania robota.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Vytvorenie jadra testovacieho systému pre správania	Aleš Katona, Peter Čimo
Pokračovanie v implementácií evolučného algoritmu	Ján Kolesár
Nahrávanie videa z OpenGL v linuxe, vytvorenie pralalizačného systému	Gabriel Braniša
Vytvorenie matematického modelu pre fitness	Tomáš Labuda
Analýza a prevzatie statických správaní	Juraj Šimon, Peter Čimo

Zhrnutie

Na stretnutí sme si určili dlhodobejšie úlohy. Ďalej sme si upresnili metodiku testovania ako aj podrobnosti k evolúcií.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 13.
Dátum stretnutia: 5.*3.2008*
Čas: 11*:00*
Miesto: Softvérové štúdio
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Ján Kolesár*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Vytvorenie jadra testovacieho systému pre správania	Aleš Katona, Peter Čimo
Pokračovanie v implementácií evolučného algoritmu	Ján Kolesár
Nahrávanie videa z OpenGL v linuxe, vytvorenie paralelizačného systému	Gabriel Braniša
Vytvorenie matematického modelu pre fitness	Tomáš Labuda
Analýza a prevzatie statických správaní	Juraj Šimon, Peter Čimo

Zhrnutie vypracovania úloh

Ján Kolesár dokončil jadro evolučného algoritmu. Chýba už len výpočet fittness a premenné vnútorného sveta hráča. Gabriel Braniša našiel program na nahrávanie videa a vytvoril základný návrh paralelizačného systému. Tomáš Labuda odhalil nefunkčnosť výpočtu aktuálnej polohy hráča a nahradil ho novým. Ďalej našiel nastavenie servera, ktore umožní získavanie polohy priamo zo servera. Juraj Šimon a Peter Čimo preskúmali niektoré iné tímy a našli dva, ktoré majú zverejnené aj zdrojové kódy so statickými správaniami.

Záznam zo stretnutia

Na stretnutí sme si porozdeľovali úlohy na nasledujúce týždne. Rozobrali sme niektoré dôležité otázky a snažili sa na ne nájsť odpovede. Navrhli sme nasledovné úpravy a riešenia.

Statické sparávania – Budú zapúzdrené v správaniach hráča ako chromozómy. Vďaka tomu sa budú dať využiť ako štartovací bod evolúcie.

Server
- Server bude logovať jedincov populácie. To znamená že klient mu pošle celú svoju populáciu a server ju uloží na disk.
- Server bude posielať jedincov klientom až keď si ich vyžiadajú.
- Logická topológia sa bude vytvárať dynamicky (napr. mriežka).
- Metóda na vyžiadanie najlepšieho jedinca.
- Bude poskytovať len základnú vizualizáciu.
- Bude uchovávať a posielať aj konfiguračný súbor evolúcie.
- Zistiť či by sa dal využiť MIBOOK. Je to sieťový disk s vlastným linuxom. Dal by sa na ňom rozbehať server?

Klient
- Pri výpadku siete musí byť schopný pokračovať v evolúcii izolovane (použitie tiomeuotov).
- Bude obsahovať aj metódu na vykonanie ohodnotených jedincov, pre prípad kontroly stavu bežiacej evolúcie.
- Pri štarte evolúcie si klient nielen vygeneruje náhodnú populáciu, ale vyžiada si jedincov aj od servera. Ak sa teda takto spustí klient v čase, keď už evolúcia beží na iných počítačoch, nezačne úplne od nuly. Získa niekoľko jedincov z už bežiacej evolúcie.
- Klient musí vypisovať svoj aktuálny stav (napr. do príkazového riadku).
- Zmapovať možnosti spustenia klienta. Nepr. live-CD alebo nejaká jednoduchá a rýchla inštalácia.

Fittness
- Zabezpečiť transparentnosť 2 dostupných metód na zisťovanie polohy hráča.
- Metóda na zistenie orientácie v priestore.
- Penalizácia za rozbitie hráča počas vykonávania jedinca.

Ostatné
- Napísať na mailing list otázku či existuje nastavenie servera, ktoré odstráni problémy s rozsypaním hráča, keď používa príliš veľkú silu pri pohybe.

- Vymyslieť silové pole pre potreby ochrany počítača pred vypnutím počas bežiaceho experimentu.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Vytvorenie jadra testovacieho systému pre správania	Aleš Katona
Dokončovanie implementácie evolučného algoritmu a prispôsobenie pre paralelizáciu	Ján Kolesár
Vytvorenie paralelizačného systému	Gabriel Braniša
Vytvorenie matematického modelu pre fitness	Tomáš Labuda
Prevzatie statických správaní a zmapovanie možností spustenia klienta.	Juraj Šimon, Peter Čimo

Zhrnutieie

Na stretnutí sme si určili dlhodobejšie úlohy. Ďalej sme si upresnili návrh servera a niektoré podrobnosti paralelizácie.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 14.
Dátum stretnutia: 12.*3.2008*
Čas: 11*:00*
Miesto: Softvérové štúdio
Prítomní: *všetci*

Stretnutie viedol: Bc. Juraj Šimon
Zápisnicu vypracoval: *Juraj Šimon*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Vytvorenie jadra testovacieho systému pre správania	Aleš Katona
Dokončovanie implementácie evolučného algoritmu a prispôsobenie pre paralelizáciu	Ján Kolesár
Vytvorenie paralelizačného systému	Gabriel Braniša
Vytvorenie matematického modelu pre fitness	Tomáš Labuda
Prevzatie statických správaní a zmapovanie možností spustenia klienta.	Juraj Šimon, Peter Čimo

Zhrnutie vypracovania úloh

Ján Kolesár vykonal niekoľko úprav a opravil niekoľko chýb v evolučnom algoritme. Ten už prebieha korektne, vykonávajú sa kríženia aj mutácie pri tvorbe novej generácie. Sily boli obmedzené aby nedošlo k rozpadu hráča (z dôvodu chýb v serveri). Peter Čimo implementoval serializáciu objektov . Aleš Katona zostavil základ testovacieho frameworku pre testovanie jednotlivých základných správaní. Gabriel Braniša pokračoval v zostavovaní sieťového serveru pre potreby evolúcie. Testoval použitie threadov pre obsluhu klientov s multiplatformovou knižnicou. Juraj Šimon zostavil DVD (lebo na CD sa nevošlo) so skompilovaným serverom a jedným ukážkovým hráčom. DVD je bootovateľné (ubuntu linux) a po nabootovaní je spustené prostredie Gnome kde je následne možné spustiť server ako aj hráča. Tomáš Labuda čiastočne zostavil fitness funkciu pre ohodnotenie polohy hráča na ihrisku v súradnicovom systéme ihriska. Fitnes je váhovaná. Každá váha zodpovedá určitému parametru hráča (poloha, natočenie, poloha kĺbov atď).

Záznam zo stretnutia

Na stretnutí sme si porozdeľovali úlohy na nasledujúce týždne. Rozobrali sme niektoré dôležité otázky a snažili sa na ne nájsť odpovede. Navrhli sme nasledovné úpravy a riešenia.

Server - možnosť kontrolovať postup na diaľku. T.j. monitoring činnosti evolučného servera prostredníctvom internetu. Bude tak jednoduché overovať postup evolúcie u klientov (či nespadli, príp číslo generácie a pod.).

Ostatné - Napísať na mailing list otázku či existuje nastavenie servera, ktoré nespustí grafický monitor pri štarte.

- opraviť chybu mutácií v evolúcií

Rozdelenie úloh

Znenie úlohy	**Poverený**
Ján Kolesár	Ladenie evolúcie
Tomáš Labuda	Dokončenie fitness funkcie
Gabriel Braniša	Pokračovanie v implementácií evolučného servera

Juraj Šimon, Peter Čimo	Statické správania
Aleš Katona	Implementácia testovania pre jednotlivé správania

Zhrnutieie

Na stretnutí sme si určili ako bude prebiehať pokračovanie riešenia úloh.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 15
Dátum stretnutia: 27*.3.2008*
Čas: 11*:00*
Miesto: Softvérové štúdio
Prítomní: *všetci \ {Peter Čimo}*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: Tomáš Labuda

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Ján Kolesár	Ladenie evolúcie
Tomáš Labuda	Dokončenie fitness funkcie
Gabriel Braniša	Pokračovanie v implementácií evolučného servera
Juraj Šimon, Peter Čimo	Statické správania
Aleš Katona	Implementácia testovania pre jednotlivé správania

Zhrnutie vypracovania úloh

Ján Kolesár

- Pokračoval v testovaní evolúcie

Aleš Katona

- Pokračoval vo vývoji testovacieho frameworku správaní (hotový iface)

Gabriel Braniša

- Pokročil;) (práca na serveri (pre distrib. evolúciu))
- Odskúšaná kompilácia linux / win (thready, sockety)

Juraj Šimon

- Vytvoril live DVD (hráč + server)
- Vypol interný monitor servera

Tomáš Labuda

- Práca na fitness (v konečnom štádiu)

Záznam zo stretnutia

Marián Lekavý

- Upozornil na nedostupnosť posledných zápisov zo stretnutí na stránke

Ján Kolesár

- Klient občas padá
  - Zrejme problém so štruktúrou STL vector
- Čo robiť pri páde servera (robocup server)?
  - Opakovať pokus o pripojenie
  - Treba vytvoriť spúštacie skripty pre server – zabezpečiť znovuspustenie pri páde servera
- Treba zistiť obmedzenia na kĺby robota (pre potreby penalizácie)
- Posielať pri distrib. evolúcii celú populáciu?

    - Nie, usporiadať jedincov podľa fitness, …
- Zvýšiť fitness ak robot vykoná želanú akciu skôr ako za pridelený čas

Aleš Katona

- Problém polohy pri znovuspustení hráča
    - Použi beam

Gabriel Braniša

- Ako meniť komunikačnú topológiu (mriežku) pri výpadku uzla?
    - Uchovať pôvodnú topológiu po timeout
    - (timeout) Pridanie nových dočasných prepojení, po pripojení uzla obnoviť pôvodnú topológiu

Juraj Šimon

- Binárky sa budú sťahovať zo svn?
    - Hej
- Závislosť servera na latexu, dá sa odstrániť?
- Vypol interný monitor servera
    - Tomášovi v takomto stave klient občas padá

- Vieme v priebehu 2 týždňov spustiť gridovú evolúciu?
    - Zrejme áno
- Aké sú možnosti použitia sieťového disku s linuxom pre server?

Rozdelenie úloh

Znenie úlohy	**Poverený**
Pokračovať vo vývoji testovacieho frameworku pre správania	Aleš Katona, Peter Čimo
Pokračovať v testovaní evolúcie.	Ján Kolesár
Dokončovanie fitness, integrácia s evolúciou.	Tomáš Labuda
Práca na vývoji servera pre distrib. evolúciu	Gabriel Braniša
How STL vector works? (Akým spôsobom uchováva vložené prvky)	Peter Čimo
Vytvoriť spúšťacie skripty pre server (zabezpečiť znovuspustenie pri páde servera).	Juraj Šimon
Možnosti odstránenia závislosti servera na latexu.	Juraj Šimon
Aké sú možnosti použitia sieťového disku s linuxom pre server?	Juraj Šimon

Zhrnutie

Nie je čo zhŕňať.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 17.
Dátum stretnutia: 9*.4.2008*
Čas: 11*:00*
Miesto: Softvérové štúdio
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Ján Kolesár*

Kontrola a hodnotenie úloh
Z minulého stretnutia boli úlohy rozdelené nasledovne:

Zhrnutie vypracovania úloh
Aleš Katona dokončil template na testovanie správaní. Tomáš Labuda implementoval zisťovanie polohy jednotlivých častí tela hráča. Gabriel Braniša otestoval migráciu jedincov a doplnil TIME-OUT na pripájanie ku paralelizačnému serveru.

Záznam zo stretnutia
10. Na stretnutí sme si porozdeľovali úlohy na nasledujúce týždne. Rozobrali sme niektoré dôležité otázky a snažili sa na ne nájsť odpovede. Gabriel Braniša a Peter Čimo debugovali príjem logovaných hráčov. Juraj Šimom otestoval LIVE-CD. Ján Kolesár debugoval zapojenie fitness do evolúcie.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Dorobit testy základných správaní jednotlivých kĺbov	Aleš Katona
Debug paralelizačného servera a zakomponovania do evolúcie	Gabriel Braniša, Ján Kolesár
Dokončiť zisťovanie polohy jednotlivých častí robota	Tomáš Labuda
Analýza a prevzatie statických správaní	Peter Čimo
Dokončiť LIVE-DC a výstražne ceduľky na počítače.	Juraj Šimon

Zhrnutie
Na stretnutí sme si určili úlohy na dokončenie dlhodobejších cieľov.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 18
Dátum stretnutia: 16*.4.2008*
Čas: 12*:00*
Miesto: Softvérové štúdio
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: Tomáš Labuda

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Dorobit testy základných správaní jednotlivých kĺbov	Aleš Katona
Debug paralelizačného servera a zakomponovania do evolúcie	Gabriel Braniša, Ján Kolesár
Dokončiť zisťovanie polohy jednotlivých častí robota	Tomáš Labuda
Analýza a prevzatie statických správaní	Peter Čimo
Dokončiť LIVE-DC a výstražne ceduľky na počítače.	Juraj Šimon
Okopcit z minuleho	

Zhrnutie vypracovania úloh

Ján Kolesár

- Integrácia evolučného algoritmu a servera na podporu distribuovanej evolúcie.
- Odstraňovanie chyby z minulého stretnutia (viď Gabriel Braniša).

Aleš Katona

- Pokračoval vo vývoji testovacieho frameworku (výpočet štatistických údajov (štandardná odchýlka, ...)).

Gabriel Braniša

- Odstraňovanie chyby servera (príčina – problém čítania zo *socketu* (nekompletné údaje)).

Peter Čimo

- Vytvorenie konfig. súboru pre evolúciu.

Tomáš Labuda

- Vytváranie maticových transformácií pre výpočet polohy častí tela robota (treba refactoring kódu).
- Overenie transformácií renderovaním vypočítaných polôh (OGL).

Záznam zo stretnutia

Ján Kolesár

- Plánuje zaviesť do evolúcie obmedzenie na kĺby.
- Úvahy nad fitness funkciou pri evolúcií pohybu z miesta A do miesta B (obavy zo skákania = problém pri zmene počas pohybu).
    - Zatiaľ neriešiť, nech skáče.
- Problémy s BEAM – umiestnenie hráča na náhodné pozície.

Aleš Katona

- Pri pripojení a odpojení pôsobia na hráča neštandartné sily.

- Dorobiť so testovacieho frameworku otáčanie podľa všetkých osí kĺbov.

Rozdelenie úloh

Znenie úlohy	**Poverený**
Dorobiť so testovacieho frameworku otáčanie podľa všetkých osí kĺbov.	Aleš Katona
Zaviesť do evolúcie obmedzenie na kĺby. Problémy s BEAM.	Ján Kolesár
Refactoring kódu pre transformácie.	Tomáš Labuda
Rozbehanie distribuovanej evolúcie.	všetci

Zhrnutie

Nie je čo zhŕňať.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 19.
Dátum stretnutia: 07.*05.2008*
Čas: 11*:00*
Miesto: Softvérové štúdio
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Gabriel Braniša*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Písanie technickej dokumentácie a používateľskej príručky pre: fitness, testovanie, evolúcia, LiveCD, client-server	Celý tím, každý svoju časť práce na projekte
Odstránenie memory leaks zo sieťovej aplikácie	Gabriel Braniša

Zhrnutie vypracovania úloh

Každý počas týždňa si našiel čas na spísanie technickej dokumentácie a používateľskej príručky. Tieto časti Juraj Šimon skompletizoval do jedného celku a elektronicky odovzdal. Gabriel Braniša odstránil memory leaks z aplikácie a commitol na SVN.

Záznam zo stretnutia

Na stretnutí sme hodnotili výsledky evolúcie, resp. došli do momentu, keď vygenerované súbory bolo treba spätne použiť na zobrazenie výsledného jedinca. Ján Kolesár na tom počas stretnutia pracoval. Po zobrazení sme videli nepostačujúci výsledok a Jano upravil výpočet fitness s tým, že tentoraz budú výsledky lepšie. Na ExchangeFile server sa klienti v hráčoch nemohli napojiť, hoci samostatný klient bez hráča sa napojil bez problémov. Tento iste malý problém sme neriešili do detailu, ale definovali sa nové úlohy súvisiace s blížiacim sa koncom semestra..

Rozdelenie úloh

Úlohy boli definovane v duchu udržania živého projektu aj po skončení našej práce v tímovom projekte, aby naši nasledovníci bez väčších problémov mohli pokračovať v nasej práci ďalej.

Znenie úlohy	**Poverený**
Vytvorenie návodu na vytvorenie LiveCD	Juraj Šimon
Uložiť projekt na verejné SVN, Aleš má za úlohu poslať žiadosť na SourceForce	Aleš Katona
Doviesť cely projekt do stavu prezentácie, t.j. termín pre túto úlohu je dátum RoboCup 2008 na FIIT	Ján Kolesár

Zhrnutie

Na stretnutí sme si určili úlohy súvisiace s ukončením prác. Ako odporúčanie pre nasledovníkov, vhodné je testovanie celej aplikácie cez LiveCD, kde sa môžu ukázať situácie, ktoré sa hneď nepredpokladali v systéme bez LiveCD.

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: 20.
Dátum stretnutia: 14*.05.2008*
Čas: 11*:00*
Miesto: Softvérové štúdio
Prítomní: *všetci*

Stretnutie viedol: Ing. Marián Lekavý
Zápisnicu vypracoval: *Juraj Šimon*

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
Vytvorenie návodu na vytvorenie LiveCD	Juraj Šimon
Uložiť projekt na verejné SVN, Aleš má za úlohu poslať žiadosť na SourceForce	Aleš Katona
Doviesť cely projekt do stavu prezentácie, t.j. termín pre túto úlohu je dátum RoboCup 2008 na FIIT	Ján Kolesár

Zhrnutie vypracovania úloh

Žiadosť na SourceForge bola schválená postupne tam budeme presúvať časti projektu (svn, iso obraz cd, dokumentáciu, ...). Dorábali sme niektoré úpravy na hráčovi a hlavne v dokumentácií.

Záznam zo stretnutia

Na stretnutí sme preberali niektoré pripomienky k dokumentácií. Dočasne sme upustili od distribuovanej evolúcie, a rozhodli sa nechať bežať niekoľko samostatných hráčov s mierne pozmenenými parametrami. Po zhodnotení výsledkov potom určíme jedného najlepšieho, a ten sa potom použije v distribuovanej evolúcií.

Rozdelenie úloh

Úlohy boli rozdelené s ohľadom na koniec semestra. Každý člen tímu bude pracovať na dokončení niektorej z častí.

Znenie úlohy	**Poverený**
Prichystanie prezentácie pre druhý tím	Ján Kolesár
Presun našej SVN na sourceforge SVN	Aleš Katona
Dokumentácia	Juraj Šimon
Zhodnotenie projektu	Tomáš Labuda
Externé testovanie	Gabriel Braniša
Aktualizácia stránok, dokončenie zápisníc	Peter Čimo

Zhrnutie

Nie je čo zhŕňať.

10.1 Príloha C – Štandardy tímu

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

Preberací protokol

Typ projektu: Tímový projekt I

Mená študentov: Juraj Šimon, Peter Čimo, Gabriel Braniša, Tomáš Labuda, Ján Kolesár, Aleš Katona

Študijný program: SI/IS

Názov práce: Robocup tretí rozmer

Predmet prebratia:

1. .....

Odovzdávajúci subjekt: Tím Neurotics

Preberajúci subjekt:.....

....
Podpis zástupcu odovzdávajúcej strany Podpis zástupcu preberajúcej strany

V Bratislave dňa .....

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

Zápisnica zo stretnutia tímu

Číslo stretnutia: .
Dátum stretnutia:
Čas:
Miesto:
Prítomní:

Stretnutie viedol:
Zápisnicu vypracoval:

Kontrola a hodnotenie úloh

Z minulého stretnutia boli úlohy rozdelené nasledovne:

Znenie úlohy	**Poverený**
...	...

Zhrnutie vypracovania úloh

...

Záznam zo stretnutia

...

Rozdelenie úloh

Znenie úlohy	**Poverený**
...	...

Zhrnutie

...

10.2 Príloha D

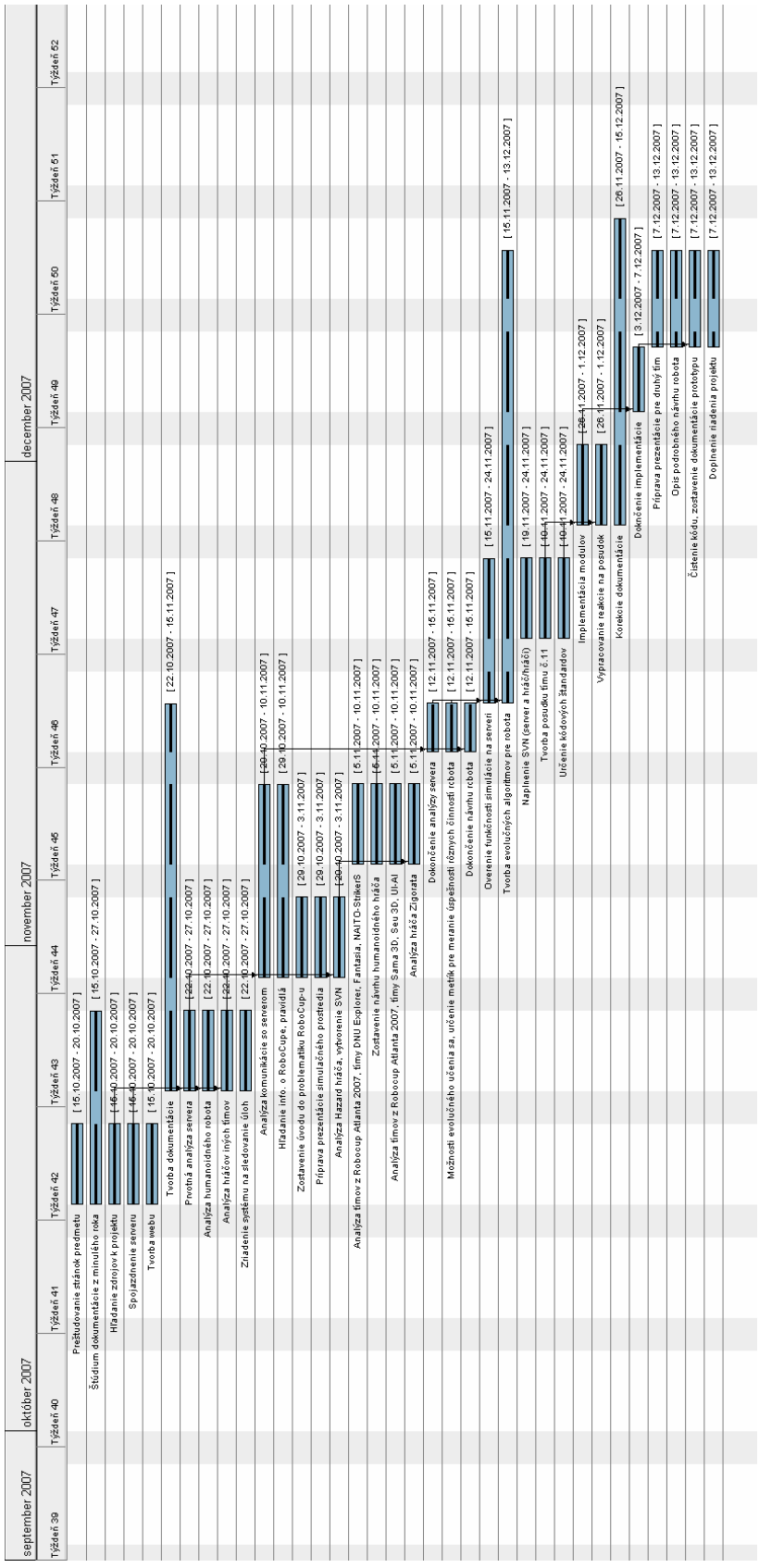
10.2.1 Úplný podrobný plán za zimný semester

Úloha	Začiatok	Koniec	Zodpovedný
Preštudovanie stránok predmetu	*15.10.2007*	*20.10.2007*	Všetci
Štúdium dokumentácie z minulého roka	*15.10.2007*	*27.10.2007*	Všetci
Hľadanie zdrojov k projektu	*15.10.2007*	*20.10.2007*	Gabriel Braniša, Tomáš Labuda
Spojazdnenie serveru	*15.10.2007*	*20.10.2007*	Aleš Katona, Ján Kolesár
Tvorba webu	*15.10.2007*	*20.10.2007*	Peter Čimo, Juraj Šimon
Tvorba dokumentácie	*22.10.2007*	*15.11.2007*	Juraj Šimon
Prvotná analýza serveru	*22.10.2007*	*27.10.2007*	Tomáš Labuda
Analýza humanoidného robota	*22.10.2007*	*27.10.2007*	Ján Kolesár
Analýza hráčov iných tímov	*22.10.2007*	*27.10.2007*	Gabriel Braniša
Zriadenie systému na sleodvanie úloh	*22.10.2007*	*27.10.2007*	Peter Čimo
Analýza komunikácie so serverom	*29.10.2007*	*10.11.2007*	Tomáš Labuda
Hľadanie informácií o RoboCup-e, pravidlá	*29.10.2007*	*10.11.2007*	Ján Kolesár
Zostavenie úvodu do problematiky RoboCup-u	*29.10.2007*	*3.11.2007*	Gabriel Braniša
Príprava prezentácie simulačného prostredia	*29.10.2007*	*3.11.2007*	Aleš Katona
Analýza Hazard hráča, vytvorenie SVN	*29.10.2007*	*3.11.2007*	Peter Čimo
Analýza tímov z Robocup Atlanta 2007, tímy DNU Explorer, Fantasia, NAITO-StrikerS	*5.11.2007*	*10.11.2007*	Peter Čimo
Zostavenie návrhu humanoidného hráča	*5.11.2007*	*10.11.2007*	Gabriel Braniša, Aleš Katona
Analýza tímov z Robocup Atlanta 2007, tímy Sama 3D, Seu 3D, UI-AI	*5.11.2007*	*10.11.2007*	Juraj Šimon
Analýza možností evolučného učenia sa	*12.11.2007*	*15.11.2007*	Ján Kolesár
Analýza hráča Zigorata	*5.11.2007*	*10.11.2007*	Gabriel Braniša, Aleš Katona
Dokončenie hrubého návrhu robota	*12.11.2007*	*15.11.2007*	Aleš Katona
Dokončenie analýzy servera	*12.11.2007*	*15.11.2007*	Tomáš Labuda
Overenie funkčnosti budúcej simulácie na serveri	*19.11.2007*	*23.11.2007*	Tomáš Labuda
Tvorba evolučných algoritmov pre robota	*19.11.2007*	*7.12.2007*	Ján Kolesár
Naplnenie SVN (server a	*19.11.2007*	*23.11.2007*	Gabriel Braniša

hráč/hráči)			
Tvorba posudku tímu č.11	*19.11.2007*	*23.11.2007*	Juraj Šimon, Aleš Katona
Určenie kódových štandardov	*19.11.2007*	*23.11.2007*	Peter Čimo
Nájdenie univerzálneho komunikačného VoIP nástroja	*19.11.2007*	*23.11.2007*	Aleš Katona
Implementácia modulov	*24.11.2007*	*28.11.2007*	všetci
Vypracovanie reakcie na posudok	*24.11.2007*	*28.11.2007*	Juraj Šimon
Doplnenie popisu servera do dokumentácie	*30.11.2007*	*5.12.2007*	Tomáš Labuda
Implementácia prototypu	*30.11.2007*	*5.12.2007*	Gabriel Braniša, Aleš Katona, Peter Čimo
Korekcia chýb v dokumentácií	*30.11.2007*	*5.12.2007*	Juraj Šimon
Návod na inštaláciu servera	*6.12.2007*	*13.12.2007*	Tomáš Labuda
Opis podrobného návrhu robota, doplnenie špecifikácie evolúcie	*6.12.2007*	*13.12.2007*	Ján Kolesár
Príprava prezentácie pre druhý tím	*6.12.2007*	*13.12.2007*	Gabriel Braniša
Opis podrobného návrhu robota	*6.12.2007*	*13.12.2007*	Aleš Katona
Implementácia prototypu, zostavenie dokumentácie prototypu	*6.12.2007*	*13.12.2007*	Peter Čimo
Doplnenie riadenia projektu	*6.12.2007*	*13.12.2007*	Juraj Šimon

Tab. 6 Zjemnený plán od 15.10.2007 do 13.12.2007

10.2.2 Ganttova schéma



Obr. 3 Ganttova schéma za obdobie 15.11.2007 až 13.12.2007

10.3 Príloha E - Plán projektu na letný semester

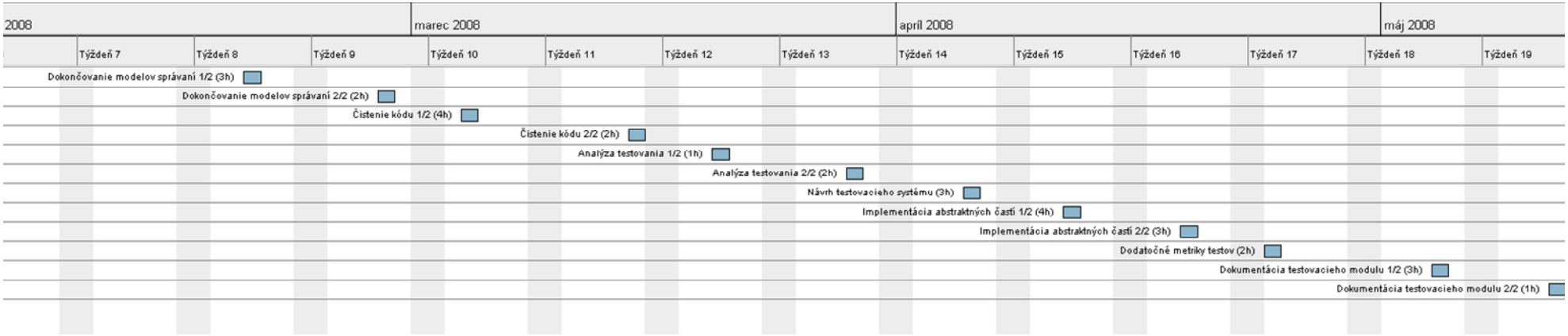
V tejto prílohe uvádzame podrobný plán projektu na letný semester.

10.3.1 Podrobný plán na letný semester

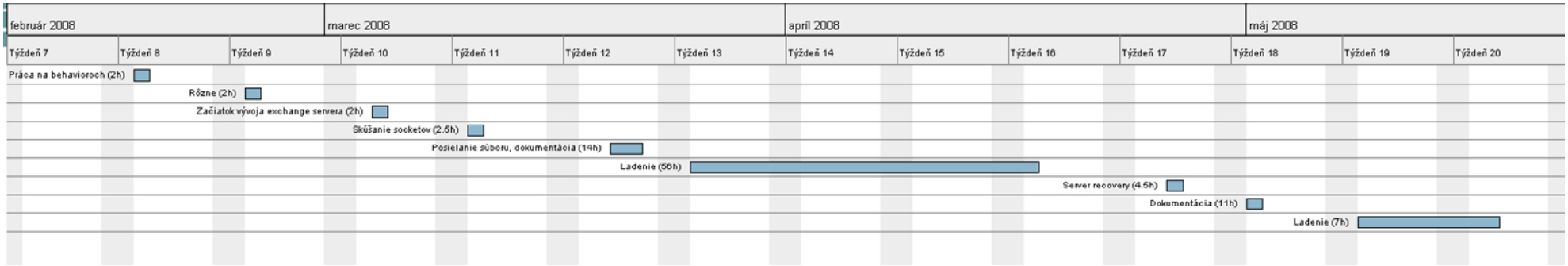
Obdobie	Popis	Podrobnosti
1. týždeň	Zhodnotenie prototypu a práce na zimnom semestri	• Zhodnotenie práce zo zimného semestra • Spresnenie plánu a činností v tíme • Vytýčenie postupu a cieľov projektu v letnom semestri
2. týždeň	Zapracovanie zmien a pripomienok do prototypu a dokumentácie	• Zapracovanie pripomienok z posudku prototypu a dokumentácie
3. – 4. týždeň	Implementácia tried evolúcie	• Implementovanie objektov potrebných k evolúcií (gén, chromozóm, strom, populácia)
5. týždeň	Implementácia evolučného behavioru	• Implementácia správania evolúcie do robota
6. týždeň	Testovanie a dolaďovanie objektov evolúcie a evolučného behavioru	• Zapracovanie objektov evolúcie do správania evolúcie • Otestovanie interakcie správania evolúcie s objektmi evolúcie • Dolaďovanie chýb
7. – 8. týždeň	Spustenie evolúcie	• Spustenie evolúcie s náhodnými hodnotami • Sledovanie evolučného procesu
9. týždeň	Zhodnotenie výsledkov evolúcie	• Vyhodnotenie výsledkov evolučného procesu
10. – 11. týždeň	Úprava evolučných algoritmov na základe výsledkov zhodnotenia a spustenie novej evolúcie	• Zapracovanie výsledkov predošlej evolúcie • Zmena konštánt, ohraničení príp. funkcií • Spustenie nového evolučného procesu s upravenými hodnotami
12. týždeň	Vyhodnotenie výsledkov	• Zhodnotenie výsledkov evolučného procesu ako aj výsledkov celej práce

Tab. 7 Podrobný plán na letný semester

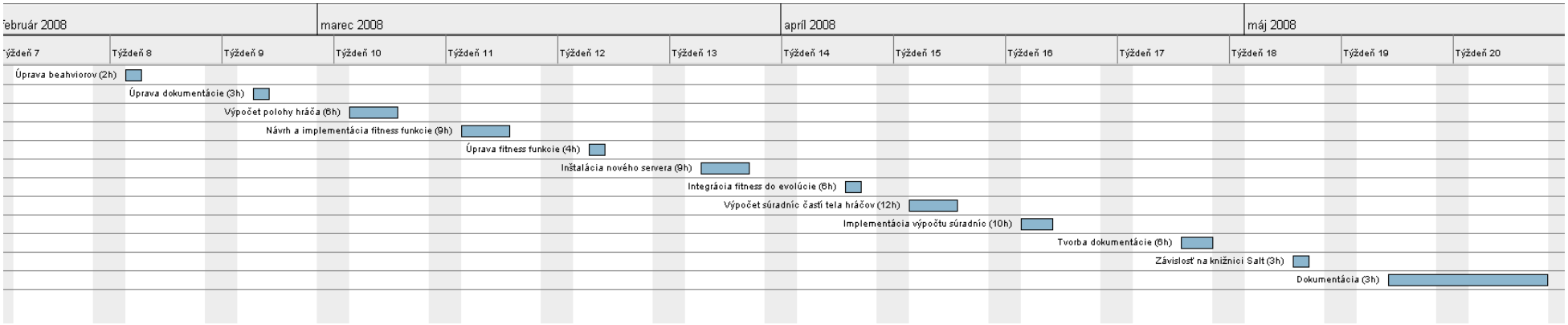
10.4 Príloha F – Podrobný plán práce za letný semester



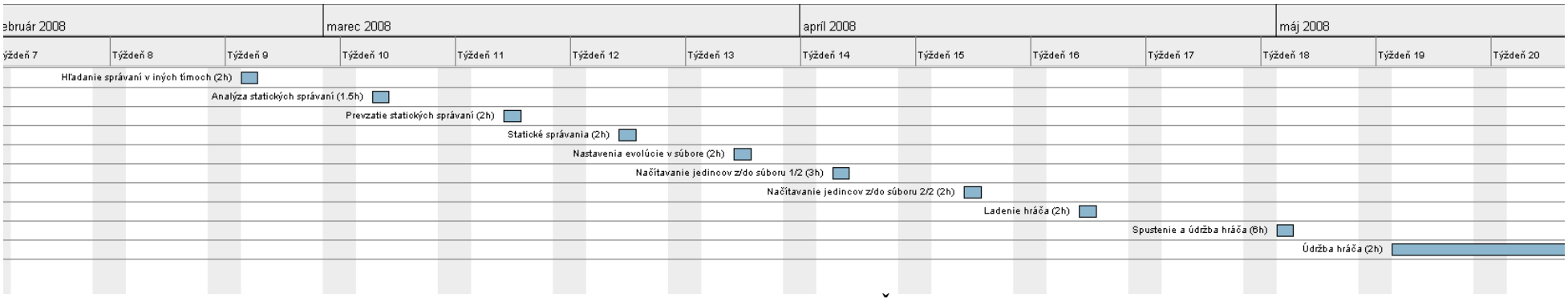
Obr. 4 Plán práce - Aleš Katona



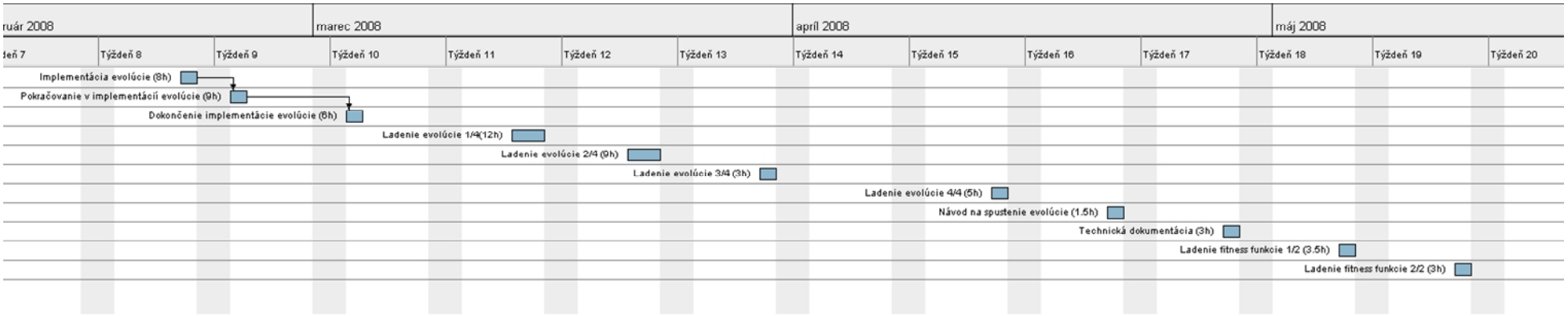
Obr. 5 Plán práce - Gabriel Braniša



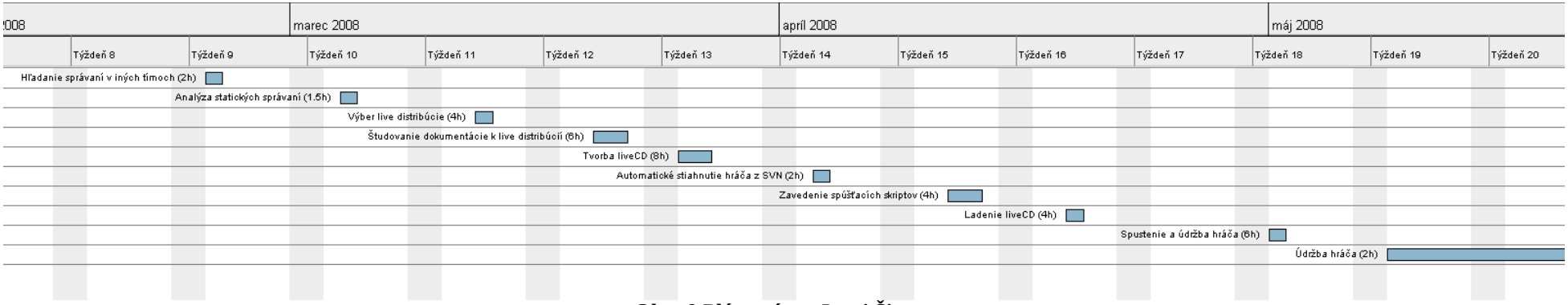
Obr. 6 Plán práce - Tomáš Labuda



Obr. 7 Plán práce - Peter Čimo



Obr. 8 Plán práce - Jána Kolesár



Obr. 9 Plán práce - Juraj Šimon