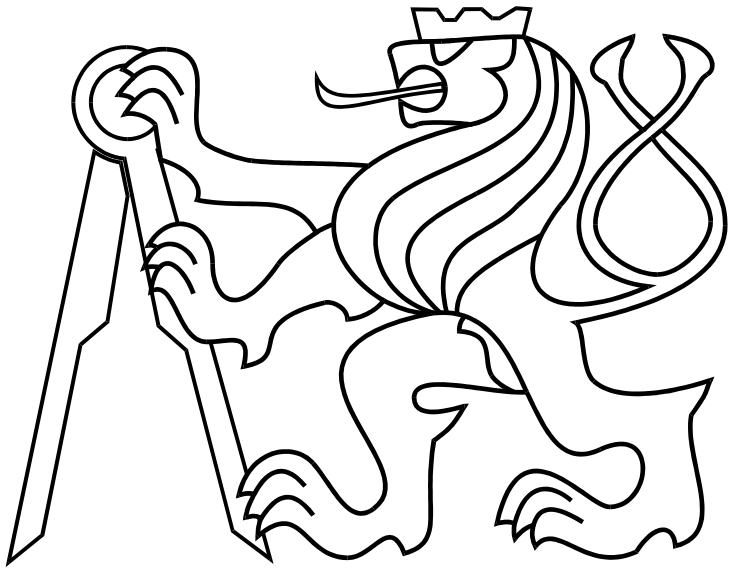
České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra kybernetiky



Diplomová práce

Simulace chování animátů a jeho vizualizace

Karel Kohout

21.5.2004

TODO : Zde se vloží zadání s podpisem vedoucího katedry, případně fotokopie u druhého výtisku

Anotace

Tato diplomová práce se zabývá simulací agentů-animátů a navazuje na předchozí práce výzkumné skupiny MRG vedené doc. Ing. Pavlem Nahodilem, CSc. na katedře kybernetiky FEL ČVUT v Praze. Práce se snaží o návrh prostředí pro potřeby simulací umělého života a to takovým způsobem, aby tato práce mohla být ku prospěchu dalším diplomantům této skupiny. Navržené prostředí využívá moderních poznatků z oblasti umělého života (hybridní architektura), ale i nejnovějších metod vizualizace. A to vizualizace prostředí, ve kterém se agenti pohybují, a také parametrů agentů v čase. Tento přístup podporuje analýzu simulace a lepší identifikaci vzorů chování. Největším přínosem této práce je návrh a implementace architektury prostředí. Neméně významným cílem této práce je formulovat nový standard prostředí, používaný nejen mnou nebo členy naší skupiny MRG, ale i dalšími výzkumnými skupinami. Od počátku tohoto projektu jsem koordinoval činnost zainteresovaných členů při vývoji příslušného softwaru. Při návrhu této architektury byl kladen velký důraz na modulárnost celé aplikace.

Abstract

This diploma work is focused on agents-animates simulation. This work follows up with former research of a MRG working group of Prof. Assoc. Nahodil Pavel, PhD. on the department of cybernetics at FEE CTU in Prague. This work concludes design of new environment for artificial life simulation. This environment should be a basis for other group members in their diploma theses. Designed environment uses modern approaches in artificial life area (hybrid architecture) and modern visualization methods as well. Modern visualization approaches are used in both the visualization of an environment in which agents live and the visualization of agent parameters over time. This serves better analysis and better behavioral pattern recognition. The main benefit of this work is design and implementation of environment architecture due to official assignment. Last but not least aim is to formulate and postulate a quite new standard of environment used not only by me or our MRG group but also in other research groups. Since the beginning of this project I have coordinated research and software development among engaged students as well. While designing this architecture special care was applied to modularity of the whole application.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval pod vedením svého vedoucího diplomové práce samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu (v části Reference).

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne ……. …….

podpis

Poděkování

Na tomto místě bych chtěl vyjádřit svůj dík všem, kteří svými připomínkami, náměty i kritikou přispěli ke kvalitě této práce. Především vedoucímu diplomové práce Doc. Ing. Pavlu Nahodilovi, CSc. za jeho odborné vedení. Také všem kolegům ze skupiny Mobile Robots Group a Ing. Davidu Řehořovi ze skupiny Computer Graphics Group FEL ČVUT v Praze.

Mé poděkování patří také mým rodičům, mé sestře a mé drahé přítelkyni Janě, za jejich trpělivost, nezanedbatelnou morální podporu a korekturu této práce.

Obsah

Úvod

Tato diplomová práce je součástí výzkumu skupiny mobilní robotiky působící na katedře kybernetiky na FEL ČVUT v Praze pod vedením Doc. Ing. Pavla Nahodila, CSc.

Předmětem výzkumu této skupiny je vývoj hardwarových a softwarových prostředků pro autonomní mobilní roboty (více informací viz [28]).

Pro snahy uměle realizovat vybrané přírodní procesy (jako kolektivní chování, růst, samoreprodukci, rozmnožování, evoluci apod.) se vžilo souhrnné označení „umělý život“ *Artificial Life*, zkráceně *ALife*. Ten se zabývá studiem skutečného života aplikováním globálních pravidel, které existují v biologických formách života na bázi uhlíku a znovuvytvořením tohoto vývoje v jiných formách, např. na bázi křemíku. ALife je jakousi alternativní formou života – „život vytvořený člověkem místo přírody“. Výzkum v oblasti ALife je založen na spojení biologického výzkumu a výzkumu v informační technice. Snahou je více využívat také nové poznatky z dalších oborů, jako jsou fyzika, sociologie, psychologie a filozofie. Nosným pilířem výzkumu a praktických aplikací ALife jsou stále simulátory, na nichž se testují vyvinuté algoritmy řízení robotů s prvky ALife [12].

Novější oblastí zájmu je také vizualizace parametrů mobotů a jejich vývoj v čase. V současné době, ve spolupráci se skupinou počítačové grafiky (Computer Graphic Group) FEL ČVUT, se vyvíjí software pro vizualizace parametrů mobotů (agentů). Jedním z cílů, které si tato práce klade, je využít právě tuto vizualizaci při hodnocení výsledků.

Cílem této práce je návrh simulačního prostředí umělého života. Jeho součást je i analýza současných simulačních prostředí a zvážení jejich použití. Na základě této analýzy jsem se rozhodl navrhnout vlastní simulační prostředí odpovídající požadavkům nejen mé diplomové práce, ale i diplomových prací kolegů. Funkčnost navržené architektury prostředí i kompletní implementace ověřím simulací agenta v různých situacích. V průběhu celého řešení se snažím využívat výhod moderní počítačové vizualizace. To znamená, zobrazit prostředí, ve kterém se agenti pohybují, i analýzu získaných dat, pomocí moderních technologií počítačové grafiky.

V Kapitole 1 ze zabývám analýzou problému návrhu software pro mobilní robotiku. Seznamuji se základními problémy a řešeními v oblasti robotiky a umělého života. Poskytuji i přehled o několika vybraných světových simulačních prostředí.

V kapitole 2 jsem detailně popsal navrženou abstraktní architekturu. Každému prvku architektury jsem věnoval jednu podkapitolu obsahující rozbor problému a navrhnutá řešení.

Kapitola 3 jsem zaměřil na konkrétní implementaci architektury. Popisuji postupy a techniky které jsem použil pro vytvoření aplikace. Tuto kapitolu jsem rozčlenil do podobných kapitol jako návrh abstraktní architektury. Každý prvek návrhu abstraktní architektury je zde rozebrán z hlediska implementace.

Kapitolu 4 jsem věnoval uživatelskému a programátorskému manuálu. V uživatelské části jsou popsány ovládací prvky aplikací a podmínky pro jejich spuštění. Programátorský manuál je věnován popisu technologie `JavaDoc`, jíž je veškerý zdrojový kód dokumentován. Samotný programátorský manuál se nachází na přiloženém CD.

Kapitola 5 je věnována experimentům a jejich výsledkům. Jsou zde popsány provedené experimenty ověřující funkci prostředí i některých dílčích algoritmů.

V závěru jsou shrnuty vytyčené a dosažené cíle práce. Také je nastíněn další možný vývoj navrženého prostředí.

Zdrojové kódy ani programátorská dokumentace nejsou pro svou rozsáhlost v textu této diplomové práce uvedeny. Jsou však na přiloženém CD.

1 Analýza problému - současný stav

Tato kapitola popisuje podklady a shrnuje znalosti, které jsou nutné pro úspěšné řešení celého problému. Mobilní robotika a umělý život, je věda interdisciplinární a tedy vyžaduje od řešitele široké spektrum znalostí. Od technických, jakými jsou znalosti programovacích jazyků, neuronových sítí, genetických algoritmů, multiagentních systémů, komunikačních protokolů, vizualizačních technik atd., až po znalosti z jiných vědních oborů jako je psychologie, etologie, biologie a další. Tato kapitola si klade za cíl ke každému z těchto dílčích problémů poskytnout krátké vysvětlení a případné zájemce o tu kterou oblast odkázat na příslušné zdroje, ze kterých jsem čerpal. V části 1.1 se věnuji problémům mobilní robotiky a řízení robotů. Část 1.2 jsem věnoval základním pojmům z etologie. V části 1.3 poskytuji krátký přehled o multiagentních systémech. V kapitola 1.4 popisuji jednotlivé oblasti mobilní robotiky. Kapitolu 1.5 jsem věnoval krátkému popisu několika vybraných simulačních prostředí.

1.1 Řízení inteligentních robotů

Tato kapitola si klade za cíl seznámení s základní problematikou mobilních robotů a současným stavem robotiky. Jako zdroje pro tuto kapitolu posloužili [5], [6], [10], [17].

1.1.1 Historie a světová pracoviště

Dnes je požadována vitalita a pohyblivost robotů. Jejich pohyblivost značně rozšiřuje možnosti robota a jím vykonávaných úkonů. S nasazením robotů se dnes nesetkáváme už jen v průmyslu a vesmírném výzkumu, ale i v dalších odvětvích lidské činnosti jako je například zdravotnictví. Současný výzkum se snaží robotovi vdechnout „život“ na základě poznatků z přírody. Roboti nejen vyhlížející, ale i chovající se jako zvířata, nazýváme animáty.

Moderní postupy používané dnes v mobilní robotice jsou záležitostí uplynulého desetiletí. Výzkum v této oblasti již není jen doménou vědeckých pracovišť universit, předmětem vojenského výzkumu nebo součástí výzkumu NASA, ale proniká i do soukromého sektoru. A to jak v podobě populárních robotických hraček (za všechny jmenujme psa AIBO firmy SONY), tak v podobě soukromých firem zabývajících se výrobou robotů. Zájemce o bližší informace nechť využije služeb sítě Internet.

Pracovišť a organizacích zabývajících se právě výzkumem v oblasti robotiky je dnes poměrně mnoho. Zde uvádím jen ty nejznámější.

V první řadě je to laboratoř Umělé inteligence při *Massachusetts Institute of Technology* (MIT) v Kalifornii USA. Současný výzkum zahrnuje mimo jiné tři roboty. Prvním je robotický mazlíček *Yuppy*. Dalším je robotické kolečkové křeslo pro invalidy

Wheelesly. A posledním je *Pheebles*. Robot vybavený kamerou pro detekci a vyhýbání se překážkám a manipulátorem, který by mohl být použit pro průzkum Marsu. K nejznámějším v minulosti vyzkoušeným robotům na tomto pracovišti patří roboti jako *Frankie* (robot kombinující nízké pořizovací náklady a široké spektrum působnosti), *TJ* (robot rozumějící jednoduché psané angličtině, je schopen přesunu do slovně popsané lokace), *Toto* (robot pohybující se v neznámém prostředí s tvorbou vnitřní mapy), *Allen* nebo *Herbert* (roboti jejichž úkolem bylo mechanickou rukou sbírat plechovky a shromažďovat je na jednom místě).

Z evropských pracovišť zmiňme *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle* (IRIDIA) na *Université Libre de Bruxelles*.

Ve výzkumu aplikace znalostí etologie genetickými algoritmy je na předním místě *School of Cognitive and Computing Sciences* (COGS) na *University of Sussex*.

Ze současných výzkumů prováděných NASA jmenujme *Field Integrated Design & Operations rover* (FIDO) vyvíjený *Jet Propulsion Laboratory* (JPL), *California Institute of Technology*, který je zamýšlen pro budoucí průzkum planety Mars. Nebo projekt *Robot Work Crew* (RWC), který se zabývá výzkumem spolupráce a koordinace robotů. Kdy roboti mají za úkol uchopit dlouhý předmět, každý na jednom konci, a společně jej přemístit na určené místo.

A na závěr nechť je zmíněna soutěž vypsaná *The Defense Advanced Research Projects Agency* (DARPA) pro autonomní pozemní vozidla. Motivace pro tvůrce byla velmi vysoká, jelikož vypsaná cena za první místo je $1,000,000. Tato cena měla být udělena vozidlu, které se dostane z Los Angeles do Las Pegas jako první a před koncem pevně daného časového limitu. Vozidla musí být autonomní, bez posádky a pozemní. Nesmějí být dálkově řízena, ani nesmí být během závodu prováděn přenos dat mezi vozidlem a jeho týmem. Veškeré výpočetní vybavení vozidla musí být přímo součástí vozidla. Závod se uskutečnil částečně po silnicích a částečně mimo ně. Nejsou kladeny žádné požadavky na velikost ani tvar vozidla. Zúčastnit této soutěže se bohužel mohou pouze týmy a obyvatelé Spojených států. Nicméně i tato vypsaná soutěž svědčí o popularizaci mobilní robotiky. V roce 2004 se žádnému ze soutěžících nepodařilo úspěšně závod dokončit. Byla vypsána odměna $2,000,000 pro první vozidlo, které úspěšně trať dokončí.

1.1.2 Problematika mobilních robotů

Naším záměrem je vyvinout mobilního robota, schopného pohybu a vykonávání jemu zadané činnosti v různých prostředích, bez potřeby zásahu do jeho řídícího programu. Možnost libovolného pohybu robota a různorodost prostředí, ve kterých se může vyskytnout značně zvyšují objem dat, jež musí robot zpracovávat. Je třeba také počítat s dynamikou prostředí, čímž se mění možnosti interakcí robota s prostředím (např. vyčerpání zdroje energie). Požadavky na mobilního robota lze shrnout do tří bodů:

- odezva v reálném čase,

- adekvátnost rozhodnutí by měla být úměrná změně prostředí,
- robustnost systému.

Odezva sytému by měla být tak častá, aby se robot dokázal vyrovnat s dynamikou prostředí (tzv. výpočetní racionalita). Do dynamiky prostředí lze zahrnout nejen změnu relací s okolními objekty, ale také zprávy od ostatních robotů. Robot by měl být schopen pohybovat se v prostředí autonomně bez vrážení do okolních objektů. A měl by být schopen je aktivně nebo pasivně využívat.

Pro zajištění adekvátního rozhodnutí je potřeba hlídat časovou dynamiku prostředí. Je dobré rozlišovat pasivní dynamiku prostředí od dynamiky aktivní (z pohledu mobota). Aktivní dynamika prostředí zahrnuje historii akcí prováděných mobotem a k čemu tyto aktivity vedly. Ze zkušenosti aktivní dynamiky prostředí je potom možné vyvodit nejvhodnější reakci nebo připravit plán pro adekvátní rozhodnutí. Na učení má vliv pouze aktivní dynamika prostředí, z které je možné vyvozovat souvislosti mezi akcemi.

Popis prostředí v němž se robot může pohybovat je velmi složitý a je obtížné tento popis získat. Jak z hlediska paměťové, tak z hlediska časové náročnosti prohledávání tohoto prostředí a orientace v něm. Proto se snažíme spíše o maximální robustnost systému, než o přesný popis prostředí.

Pro zajištění autonomního chování robota je nutné plánování a tvorba lokálních cílů, které aproximativně vedou ke cíli globálnímu. K tvorbě plánu lze přistupovat dvěma odlišnými způsoby.

První z nich je klasický přístup (shora dolů), v zahraničí literatuře označován jako *knowledge based* nebo *top-bottom*. Tento přístup spočívá v tvorbě globálního plánu. Tedy dochází k prohledávání stavového prostoru. Jako příklad mějme robota, který se má v místnosti dostat z jednoho bodu do druhého. Robot na základě informací ze senzorů navrhne optimální cestu a vydá se po ní. Stavový prostor, který musí při tom prohledat, bývá většinou rozsáhlý, a tedy jeho prohledávání by mohlo zabrat neúměrně dlouhou dobu. Pokud by se robotovi do cesty v průběhu přesunu postavila překážka, je robot nucen přeplánovat již navrženou cestu, aby se vyhnul kolizi. Plánování sebou nese riziko kombinatorické exploze možných řešení. Robot, který příliš dlouhou dobu plánoval, by nemusel být schopen přizpůsobit se dynamice prostředí. Je tedy nutné určit jistou hranici a přijmout i suboptimální řešení. Tedy najít vhodný okamžik, když je nutné přestat plánovat a začít jednat. Výsledkem plánování je posloupnost akcí, které se provedou jako celek.

Druhý způsob návrhu je reakční přístup (zdola nahoru), v zahraničí literatuře označován jako *behavioral based* nebo *bottop-up*. Tento přístup vychází maximální dekompozice, tedy složením elementárních akcí se snaží docílit kýženého cílového chování. Tímto způsobem se k problému staví například na MIT s jejich autonomními agenty. Do této skupiny spadají také řídící systémy vytvářené na půdě naší mobotické skupiny, které byly navrženy s využitím poznatků etologie. Výběr příští akce robota je prováděn na základě okamžité informace o okolí a vnitřním stavu robota. Tím je zaručena rychlost odezvy v reálném čase i jistá robustnost vzhledem k prostředí, v němž se robot pohybuje. Popis situace (aktuální informace ze senzorů robota plus informace o vnitřním

stavu robota) a k ní adekvátní akce je uložena v reakční bázi robota. Tu lze interpretovat jako genetickou výbavu robota. Z reakční báze je v každém okamžiku vybrána akce odpovídající současné situaci bez uvažování historie. Jako příklad opět vezměme robota, který se má přemístit z jednoho bodu do druhého. Reakční robot zvolí pohyb směrem k cíli. V případě, že narazí na překážku, změní svoji dráhu. Ale i tento přístup má své nevýhody. Jedním z nich je návrh reakční databáze tak, aby nebyla příliš paměťově náročná, ale pokrývala dostatek situací a akcí pro autonomní činnost robota. Důležité je zajištění asociativního přístupu k reakční bázi. Tedy aby se k dané situaci automaticky vybavila reakce. Výsledná rychlost odezvy robota na vzniklou situaci pak bude záviset na návrhu reakční báze a schopnosti najít v ní požadovanou akci. Je také nutné brát v úvahu formát dat přicházejících ze senzorů a dobu potřebnou pro jejich zpracování případně úpravu do takového tvaru, aby bylo možno rychle prohledat reakční bázi. Další nevýhodou je absence strategické predikce. Pokud by robot měl hrát například šachy, byla by reakční báze značně rozsáhlá.

Nabízí se tedy skloubení obou přístupů v hybridní architekturu, využívající obou možných přístupů zdola nahoru i shora dolů. Vniká tím ovšem další problém. Kde zvolit hranici mezi oběma přístupy tak, abychom maximálně využili výhod obou přístupů [7].

1.1.3 Současný stav simulátorů chování

V současné době, kdy výpočetní síla i běžných počítačů je dostačující i pro komplexní simulace mnoha robotů, je patrná tendence výzkumu umělého života nejen na robotech, ale hlavně na simulátorech pro tento účel navržených. Z hlediska výzkumu chování, na který se práce naší skupiny zaměřuje, je tento přístup výhodný z několika důvodů. Jedním důvodem jsou jistě pořizovací náklady na výstavbu robota. Podstatnějším důvodem, pro přesun výzkumu do oblasti simulací, je ideální prostředí bez šumů a poruch. V počítačové simulaci se nemusíme zabývat šumem v senzorických datech, rozpoznáváním, a dalšími s roboty spojenými úlohami. Můžeme se tedy plně soustředit na výzkum chování.

V dnešní době existuje celá řada simulátorů a prostředí různých autorů volně stažitelných ze sítě Internet. Pouze zlomek je uveden v [P02]. Většina z nich demonstruje pouze jednu oblast ze světa umělého života. Některé se soustředí na celulární automaty, jiné na biomorfy, mravenčí kolonie, boidy a další oblasti.

Aplikací snažících se o simulaci robotů na bázi etologických principů je málo, a komplexnějších a rozšiřitelnějších prostředí ještě méně. Za všechny jmenujme StarLogo (viz 1.5.1, [35]), Swarm (viz 1.5.2, [36]), Cormas (viz 1.5.3, [37]) a Xraptor (viz 1.5.4, [38]).

Tohoto nedostatku si všimla naše výzkumná skupina a tato práce se zabývá návrhem komplexního, modulárního, rozšiřitelného a co nejvíce se přírodě blížícího prostředí, ve kterém by šli provádět simulace chování navržených architektur agentů.

1.2 Základní pojmy etologie

Návrh řídícího programu mobilního robota je inspirován poznatky biologických oborů, především etologie. Ta je označována jako srovnávací věda o chování živých tvorů. Pro účely robotiky jsou podstatné části, jež se zabývají chováním, adaptivní modifikací chování a učením. Jako zdroj zde posloužily Lorenzovy Základy etologie [1], a také [7], [8], [10].

Živé organismy jsou schopny udržovat, popř. zvyšovat svoji vlastní uspořádanost a tím se zcela zásadně liší od neživé přírody. Ta podléhá fyzikálním zákonům, jako je zákon růstu entropie. Růst organismu tedy není pouze jev kvantitativní, kdy narůstá hmota, ale i kvalitativní, neboť roste složitost vnitřní struktury organismu. Všechny živé organismy však mají geneticky definovaný životní cyklus, který definuje průběh a délku jednotlivých fází života organismu jako je růst, dospělost, stárnutí a smrt. Zásadní rozdíl mezi živými a neživými systémy je tedy v tom, že základní stavební prvky organismu - buňky jsou schopny adaptace a určité autonomní činnosti (např. přerušený nerv doroste opět do původního místa jinou cestou) a jsou nositelé genetické informace, jež specifikuje činnost a vlastnosti dané buňky i celého organismu. Zde leží obrovská propast mezi složitostí a vlastnostmi živých organismů a realizačními možnostmi současné techniky. Proto v technické praxi není snaha živé systémy kopírovat, ale využívat principy, na kterých jsou založeny a ty pak vhodným způsobem technicky realizovat.

Kapitola se podává krátký přehled o několika základních pojmech jako je chování (1.2.1), učení (1.2.2), vrozený spouštěcí mechanismus (1.2.3) a apetenční chování (1.2.4).

1.2.1 Chování

Příčinnou určitého typu chování jsou vnitřní i vnější podněty. Vnitřní podněty souvisí se stavem systému (hlad, sexuální pud nebo pud sebezáchovy). Vnější podněty přicházejí z okolního prostředí (jedinec zaznamenal jídlo, jedince opačného pohlaví nebo přítomnost dravce). Vnitřní podněty většinou spouští určité chování (hledání potravy, jedince opačného pohlaví nebo útěk před dravcem) a vnější podněty ovlivňují způsob průběhu a realizace vybraného chování (potrava v dosahu - dojde ke konzumaci; je známa poloha potravy - dojde k přesunu na místo a konzumaci; není známa poloha potravy - začne hledání potravy atd.). V přírodě probíhá výběr chování na základě reakcí na příchozí podněty (ať už vnější, nebo vnitřní).

1.2.2 Učení a adaptivní modifikace chování

Cílem učení je účelné zlepšení chování. Jedním z druhů učení je učení asociací, které nevyužívá zpětného hlášení úspěchu. Za asociaci se zde považuje spojování dvou po sobě následujících událostí - jakmile došlo k první události, organismus očekává událost druhou. Asociace mezi akcemi v přírodě vzniká pouze v případě jejich vzájemné kauzální souvislosti.

Dalším druhem učení je učení se zpětným hlášením o účinku prováděné akce. Normy přiměřenosti získává v geneticky zakódované informaci od „vrozeného učitele“ (např. popis dědičně koordinovaného pohybu), informace o současném stavu okolí získává pomocí smyslových vjemů. Informace lze v organismu určitou dobu uchovávat, do souvislosti s odměnou se může dostat až později. Pak se vytváří nové „podmíněné spojení“. Během učení na daný podnět dochází ke zvýhodnění určitého chování před ostatními. Při učení se zvyšuje pravděpodobnost, že zvíře použije takový způsob chování, kterým uspokojí své požadavky a pudy.

Naproti tomu stojí instrumentální učení. V tomto případě už je jedinec nucen zvládnout situaci. K tomuto problému se většinou staví tak, že se snaží aplikovat různé způsoby chování. Poté ohodnotí přínosy jednotlivých způsobů chování. Tímto způsobem získává zkušenosti.

Nové způsoby chování vytváří motorické učení. Chování se vytváří na základě krátkých časových odstupů mezi akcemi a dalšími posilujícími podněty. Z takto provázaných akcí se vytváří řetězec jednoho velkého pohybu.

1.2.3 Vrozený spouštěcí mechanismus (AAM)

Vrozený spouštěcí mechanismus dává živému tvoru schopnost rozpoznat relevantní situace v prostředí a uvádět tak do chodu odpovídající dědičně koordinovaný pohyb. Tento spouštěcí mechanismus reaguje na určitou konfiguraci podnětů, která je též nazývaná klíčový podnět. U vrozených spouštěcích mechanismů může docházet i k jistým typům učení, resp. adaptivní modifikaci chování zvyšováním selektivity klíčového podnětu, tj. obohacováním podnětové konfigurace o další znaky (podněty).

1.2.4 Apetenční vyhledávání klidových stavů

Apetence je definován jako stav vzrušení (vybuzení), který trvá tak dlouho, dokud nenastane jistá podnětová situace, označovaná též jako vyhledávaný podnět (*appeted stimulus*). Dojde-li konečně k podnětové situaci, je spuštěno cílové chování (*consumatory action*), apetenční chování přestává a je vystřídáno stavem relativního klidu.

Apetenční chování je tedy účelné chování směřující k vyhledávání takových podnětových situací, které v konečném důsledku vedou k uspokojení potřeb organismu a tím i vymizení buzení k provádění daného typu chování. Proto jsou tyto typy chování souhrnně označovány jako apetenční vyhledávání klidových stavů. Apentence, AAM a instinktivní pohyb patří mezi nejdůležitější faktory určující celkové chování živočichů.

1.3 Výchozí architektury MAS

Multiagentní systém můžeme definovat jako skupinu agentů, kteří jednají nezávisle a autonomně (plnící nějaký designérem zadaný cíl) a současně jsou schopni koordinace

kooperace a vyjednávání mezi sebou nebo s okolím ve kterém jsou situováni. Rozlišujeme zde dvě úrovně návrh agentů (*micro-level*) a návrh společnosti (*macro-level*).

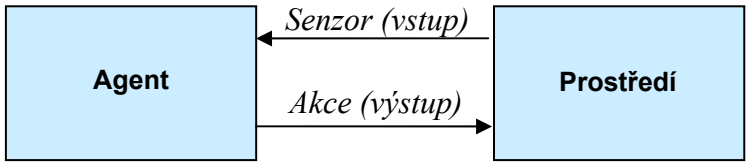
Tato kapitola si klade za cíl seznámení s základní problematikou multiagentních systémů (MAS), vymezení základních pojmů a zavedení formalismu. Jako zdroj pro tuto kapitolu posloužil [3].

1.3.1 Pojem agent a jeho vlastnosti

Agent je zapouzdřený výpočetní systém situovaný v nějakém prostředí schopný flexibilního a autonomního chování za účelem splnění svého cíle. Agent může okolí částečně ovlivňovat. Rozlišujeme několik druhů okolí:

- přístupné vs. nepřístupné,
- dynamické vs. statické,
- deterministické vs. nedeterministické,
- diskrétní vs. spojité.

Okolí odpovídající reálnému světu nazýváme otevřené okolí. To je nepřístupné, dynamické, nedeterministické a spojité. Obrázek č. 1 ukazuje interakci agenta a prostředí.



Obrázek č. 1 – Interakce agent-prostředí

Agent je ***autonomní***. Je proaktivní a cílově orientovaný. Jedná na základě vlastního úsudku tak, aby dosáhl svého cíle, bez nutnosti zásahu, potvrzování nebo inicializace od uživatele. Může se samostatně rozhodovat, může podvádět, může vstoupit/vystoupit do/z komunity.

Agent je ***reaktivní***. Agentovy akce jsou spouštěny událostmi v jeho okolí v reálném čase. Agent je schopen vnímat a jednat.

Agent je ***intencionální***. Má schopnost plnit dlouhodobé cíle a organizovat své chování tak, aby jich dosáhl.

Agent je ***sociální***. Agenti spolupracují v rámci komunity, aby dosáhli společného cíle, jsou si vědomi jeden druhého a uvažují o ostatních agentech. Mohou se sdružovat do koalic a týmů, z kterých mohou mít užitek.

Agent je ***adaptivní***. Agent se dynamicky adaptuje a učí o své prostředí. Je adaptivní na neurčitost a změny v prostředí. Může zlepšovat své sociální postavení.

Agent je ***kooperativní***. Agenti koordinují svojí činnost a vyjednávají za účelem dosažení společného cíle.

Agent je ***mobilní***. Agent se může volně pohybovat v prostředí.

Agent je ***interaktivní***. Agent komunikuje s člověkem, ostatními agenty, prostředím a informačními zdroji.

Agent má ***osobnost***. Agent projevuje vlastnosti jako jsou emoce.

Racionální agent je autonomní, proaktivní, reaktivní a sociální.

1.3.2 Prakticky uvažující agent

Praktickým uvažováním nazýváme rozhodovací proces orientovaný na akci (proces zjišťující co udělat). Lidské praktické uvažování se skládá ze dvou aktivit. Deliberace (*deliberation*) rozhoduje o stavu, který chceme dosáhnout. Koncové uvažování (*means-ends reasoning*) nebo také plánování rozhoduje o tom, jak docílit tohoto stavu. Tento přístup vedl ke vzniku intencí a navržení BDI (*belief-desires-intentions*) modelu. Znalosti (*beliefs*) obsahují stav okolního prostředí (někdy také vnitřní stavy agenta). Cíle (*desires*) určují motivaci agenta, čeho se snaží dosáhnout. Intence (*intentions*) zachycují rozhodnutí jak se chovat, aby bylo dosaženo daného cíle.

Intence jsou mnohem silnější než cíle. Představují agentův lokální závazek ke splnění nějaké akce. V souvislosti s intencemi existují dva problémy. Nekonzistence intencí a znalostí (*intestin-belief inconsistency*), kdy agent věří, že není schopen splnit intenci. Tedy neracionální chování. Nekompletnost intencí a znalostí (*intestin-belief incompleteness*), kdy agent nevěří, že jeho intence jsou splnitelné. To ovšem neznamená, že by intence museli nutně být nesplnitelné, pouze agent nemá žádné informace o splnitelnosti jeho intencí. Tedy toto chování může být racionální.

V souvislosti s BDI modelem zavádíme další pojem a tím je závazek. Zmínili jsme se o něm již při popisu intence. Rozlišujeme tři druhy závazků.

Slepý závazek (*blind commitment*), kdy se agent snaží splnit daný cíl dokud nevěří, že byl splněn.

Jednomyslný závazek (*single minded commitment*), kdy se agent snaží splnit daný cíl dokud nevěří, že byl splněn nebo dokud nevěří že je nesplnitelný.

Otevřený závazek (*open minded commitment*), kdy se agent snaží splnit daný cíl jen pokud stále věří, že je splnitelný.

1.4 Přírodou inspirované roboty a jejich chování

S postupy ve výzkumu distribuovaných robotických systémů je prozkoumáváno stále více aspektů těchto systémů. Parkerová identifikovala 8 základních oblastí – systémy inspirované přírodou, komunikace, architektura, lokalizace, manipulace s objekty, koordinace pohybu, rekonfigurovatelní roboti a učení viz [17]. Počátky mobilní robotiky sahají do 80tých let. Od té doby výzkum v této oblasti velmi pokročil. Tato kapitola se snaží shrnout současný stav mobilní robotiky. Tento vědní obor je stále nový, některé aspekty již byli prozkoumány velmi do hloubky, jiné zůstávají nevyřešeny.

1.4.1 Biologicky inspirované systémy

Velký rozkvět zaznamenala mobilní robotika po uvedení behaviorálního přístupu (*behavioral-based control*) [25], [23]. Tento přístup silně ovlivnil výzkum kooperativní mobilní robotiky. Jelikož je tento přístup motivován přírodou, bylo použito k návrhu chování mobotů poznatků z chování zvířat, převážně hmyzu jako mravenců a včel. Tato architektura našla uplatnění především pro jednoduché lokální rozhodování. Práce v této oblasti ukázala schopnost týmu robotů shlukovat se do skupin, opět se rozdělovat a následovat nějakou trasu. Další oblastí zájmu je simulace systému lovec-kořist (*predator-prey*). Spolupráce v distribuovaném robotnickém systému založená na chování savců, případně lidí se teprve začíná studovat. Příkladem může být robotický fotbal [19]. Přírodou inspirované systémy a jejich aplikovatelost na týmy robotů je již velmi dobře prozkoumána. Méně prozkoumanou oblastí je chování vyšších živočichů, především učení novým chováním.

1.4.2 Komunikace robotů

Komunikace v týmu robotů musela být nutně studována již od počátku multirobotických systémů. Rozlišuje se implicitní a explicitní komunikace. Implicitní komunikace vzniká jako postranní efekt jiných akcí, zatímco explicitní komunikace je konkrétní akt zaměřený na předání informace dalším robotům týmu. Provedenými výzkumy bylo zjištěno, že i malé množství vyměněné informace může vést k velmi dobrým výsledkům [21]. Významná část výzkumu je věnována reprezentaci a jazyku komunikace. Jinou oblastí je zajištění komunikace tolerantní k chybám pomocí distribuované komunikační sítě za účelem spolehlivé komunikace [18].

1.4.3 Architektura, plánování a řízení

Velké úsilí je zaměřeno na vývoj architektur a řízení. Tato oblast se zabývá především výběrem akcí, řešením konfliktů a dalšími. Hlavní otázkou této oblasti je, zda používat specializované architektury pro každý tým robotů, nebo zda lze navrhnou více obecnou architekturu, kterou lze využít pro širokou škálu aplikací. Velmi málo výzkumu bylo věnováno vývoji takové obecné architektury. Příliš obecná architektura by nemusela být použitelná. Tedy je lepší zaměřit se na otázku, zda existující fungující architektury nelze použít i v jiných aplikacích.

1.4.4 Lokalizace, mapování a průzkum

Mnoho úsilí bylo věnováno tomuto problému především pro jednoho autonomního robota. Teprve nedávno začali aplikace na týmy robotů. Téměř veškerý výzkum probíhá pouze ve dvou rozměrech. Většinou je algoritmus navržený pro jednoho agenta použitý pro

celou skupinu, jen velmi málo prací se zaměřuje na využití skupiny robotů za účelem získání přesnějších navigačních dat, která by s jedním robotem nebylo možno získat. Tato oblast je téměř neprozkoumána. Například otázka efektivity týmu robotů v porovnání s jedním robotem.

1.4.5 Manipulace s objekty

Kooperace několika robotů při přenášení, tlačení nebo manipulaci s objekty je obtížnou úlohou. Mnoho úsilí bylo věnováno tomuto aspektu a jen velmi málo z nich bylo demonstrováno na robotech. Avšak tato oblast má velmi mnoho aplikací, což ji řadí mezi nejvíc zkoumané oblasti. Nejvíce demonstrativním příkladem je tlačení objektu dvěma agenty [22]. Téměř většina těchto pokusů se odehrála na rovném povrchu. Velkou výzvou je řešení úlohy manipulace s objekty na nerovném venkovním terénu.

1.4.6 Koordinace pohybu

Tato oblast je velmi populární a zahrnuje především plánování trasy, řízení provozu, držení formací [21], atd. Tato oblast je výborně zvládnuta a prozkoumána. Přesto demonstrace na robotech mají svá omezení. Téměř většina byla navržena pro 2D prostředí. Nejvíce omezujícím problémem je výpočetní náročnost plánovaní trasy.

1.4.7 Rekonfigurovatelní roboti

V této oblasti bylo zatím uděláno jen málo výzkumů [24] a ani nezaznamenala větší zájem v posledních letech. Dřívější práce vedli k robotovi schopnému se rekonfigurovat. Motivací je dosažení funkce na základě tvaru pomocí propojení několika modulů, či robotů za účelem vytvoření tvaru pro danou funkci. Takovýto systém je velmi robustní a dokonce i samoopravitelný. Tato oblast výzkumu je velmi mladá a většina systémů byla realizována jen při jednoduchých úkolech v laboratořích. Tato oblast nabízí mnoho pro další výzkum.

1.4.8 Učení

Velké úsilí bylo věnováno učení multi-agentních systémů [20], avšak menší už multi-robotímu učení. Výzvou jsou především operativní úlohy, ve kterých užitek akce jednoho agenta závisí na akcích ostatních členů týmu. Především proto, že je velmi obtížné určit užitek akce jednoho agenta pro celý tým.

1.4.9 Otevřené otázky v autonomní mobilní robotice

Od počátku mobilní robotiky byl učiněn značný pokrok, především v oblastech biologické inspirace, komunikace a architektur řízení. Pokrok byl také uskutečněn v lokalizaci, koordinaci pohybu a transportu objektů. Výsledky začínají vykazovat i oblasti rekonfigurovatelnosti robotů a učení. Stále ale zůstává mnoho otevřených otázek. Jak lze snadno ovládat tým robotů člověkem? Je pasivní rozpoznání akcí možné?

1.5 Vizualizace chování

K simulování a odzkoušení implementované hybridní architektury, je potřeba programové prostředí, ve kterém se bude tento agent pohybovat. Z tohoto důvody jsem provedl důkladnou analýzu stávajících simulačních prostředí s velkým důrazem na vizualizační schopnosti. Tato kapitola se věnuje několika takovým aplikacím.

1.5.1 StarLogo

StarLogo university MIT je programovatelné modelovací prostředí decentralizovaných systémů – systémů organizovaných bez organizátora, koordinovaných bez koordinátora. Tolik definice na stránkách MIT [35]. StarLogo vychází z programovacího prostředí Logo, které je založené na tzv. želví grafice. StarLogo toto prostředí rozvíjí na simulaci ALife. „Želvy" jsou stvoření která se dají použít k simulaci jevů v živé přírodě jako mravenčí kolonie, hejno ptáků, atd. Aplikace StarLogo je naprogramovaná v Javě, tedy je multiplatformní a je volně ke stažení viz [35]. Tedy je pro její spuštění potřeba *Java Runtime Environment*.

1.5.2 Swarm

Swarm je software pro simulaci komplexních multiagentních systémů, původně vyvíjený na Santa Fe Institute. Cílem autorů je, aby byl Swarm užitečným nástrojem pro vědecké pracovníky v širokém rozsahu oborů. Základem architektury je simulace skupiny soupeřících interagujících agentů. S touto architekturou lze implementovat širokou škálu modelů založených na agentech. K vizualizaci průběhu parametrů simulace dává tento software k dispozici kreslící plátno (*canvas*). Je tedy v plné moci programátora přizpůsobit vizualizaci parametrů jak v závislosti na čase, tak v závislosti na jiných veličinách [36].

1.5.3 Cormas

Multiagentní systémy jsou zde používány k simulaci dynamiky interakcí. Především z ekologického hlediska a nakládání přírodními zdroji. Má několik možností vizualizace, včetně možností vykreslení průběhů parametrů v čase. První možností je

vizualizace prostředí. Druhou vizualizace komunikace. Třetí je výše zmíněná vizualizace průběhů v čase. Avšak tato vizualizace není jako u ostatních při simulaci, ale data jsou ukládána a grafy generovány offline po simulaci. Tato aplikace nenabízí žádné typy grafů, závisí pouze na implementaci, jaké grafy budou vykresleny. Při simulaci se nastaví sondy (*probes*) ty určují, které parametry budou uloženy a použity při vizualizaci [37].

1.5.4 X-Raptor

Xraptor je simulační prostředí multiagentních systémů a jejich prostředí z Johannes Gutenberg-Universität. Je naprogramováno v C++ pro platformu Unix a není k dispozici ve verzi pro Windows. Je však možno volně stáhnout manuál a zdrojový kód. Agenti se mohou pohybovat ve 2D nebo 3D prostředí a mohou zabírat objem v tomto prostředí. Agent má k dispozici informaci o svém okolí a může své okolí ovlivňovat. Agenti jsou řízeni uživatelsky definovaným řídícím jádrem. Typickým řídícím jádrem je takové, které se snaží zajistit co nejdelší přežití agenta [38].

2 Navržená abstraktní architektura WAL

Při analýze současného stavu v oblasti simulací umělého života ve skupině vedené doc. Ing. Pavlem Nahodilem, CSc. na FEL ČVUT v Praze (viz. například v [7], [8], [16], [28]) i ve světě (viz kap. 1.5) s cílem najít platformu, na které by bylo možné provádět námi požadované simulace, jsem došel k závěru, že žádná plně neodpovídá našim požadavkům. Jak již bylo zmíněno výše, většina volně šiřitelných platforem se zabývá jen jednou oblastí umělého života. Ať už simulací celulárních automatů, mravenčích kolonií, vývoje živočichů na základě genetického programování a další. Žádná ze mnou prozkoumaných se nezabývá simulací umělého života na obecnější úrovni. A téměř žádná neposkytuje k této simulaci ještě nástroje pro analýzu simulace. O návrh takového simulačního prostředí jsem se pokusil ve spolupráci s kolegy, kteří jej upotřebí ve svých diplomových pracích. Požadavky bych zformuloval do několika bodů:

- možnost simulovat různé fenomény umělého života od celulárních automatů (*CA*), boidů, biomorfů, mravenčích kolonií atd., až po složité sociálně se chovající agenty,
- možnost exportu vnitřních dat ze simulace tak, aby byla možná vizualizace parametrů,
- snadná modifikovatelnost simulace a její krokování,
- poutavá vizualizace světa agentů, za účelem větší popularizace oboru,
- hodnotná vizualizace průběhů parametrů simulace v čase,
- modulárnost,
- snadná rozšiřitelnost,
- přenositelnost.

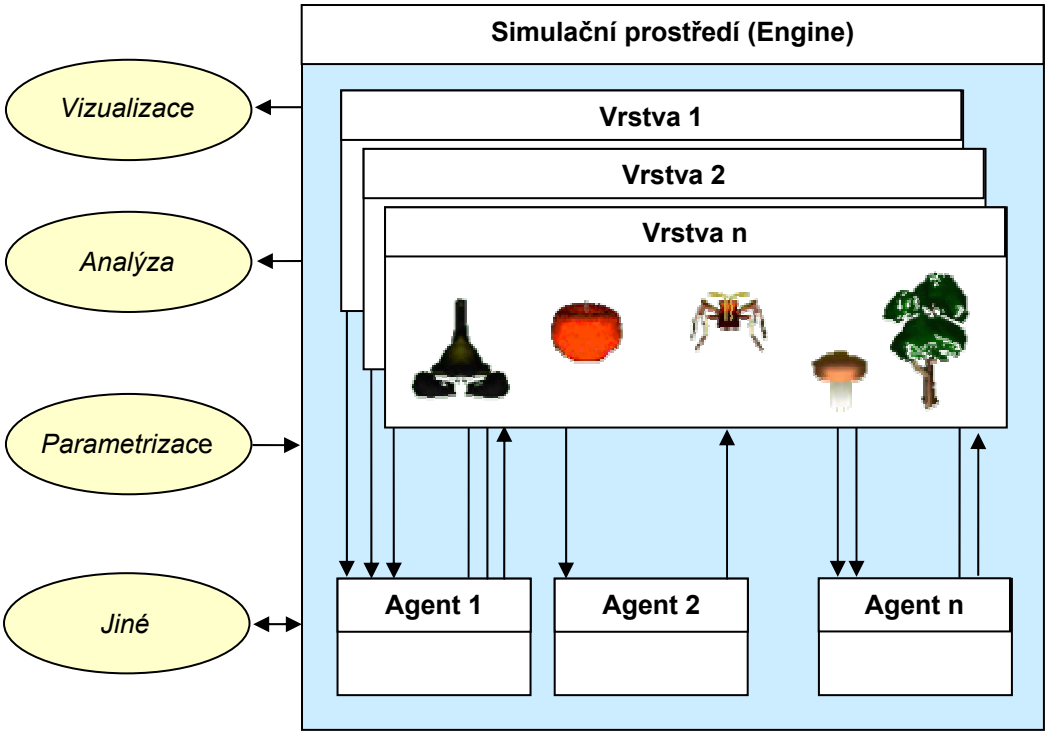
Mým cílem je tedy vytvořit takové prostředí, které nebude sloužit jen k presentaci výsledků mé diplomové práce, ale k presentaci výsledků celé naší výzkumné skupiny. Motivací je vyvinout simulační prostředí natolik modulární, aby vyhovovalo každému, kdo bude mít zájem o výzkum v oblasti umělého života, ale nebude chtít přitom začínat od programování simulačního prostředí. Chtěl bych poskytnout nástroj, ve kterém bych dal prostor svým následovníkům, aby se zabývali více do hloubky problémy sociálního chování, genetickému vývoji agentů, algoritmům učení, srovnáním chováním, atd. Mým cílem je také udělat aplikaci uživatelsky přívětivou, aby k ní měla přístup i široká veřejnost. Tento projekt jsem nazval ***World of Artificial Life***, zkráceně **WAL**.

V průběhu řešení, tak jak jsem se seznamoval s různými mechanismy a postupy, jsem rozdělil svou práci na dvě části. První je ***WAL Abstraktní Architektura*** zkráceně **WALA**[2]. Jedná se o obecný návod nebo předpis, jak by měla vypadat aplikace pro simulaci umělého života, aby její prvky byly co možná nejvíce přenositelné mezi konkrétními implementacemi. Vidím zde jistou paralelu s FIPA Abstract Architecture (viz. [48], [P01]), jejíž studium, za účelem nalézt komunikační protokol, mě na tento nápad přivedl. Samotná architektura byla navržená pod mým vedením ve velmi úzké spolupráci

s mými následovníky. Svůj přínos tedy nevidím jen v tvorbě návrhu, ale i v organizaci a koordinaci součinnosti celé skupiny. Bylo nutné zjistit, jaké požadavky na aplikaci mají jednotliví členové skupiny. Tyto následně unifikovat, specifikovat a v neposlední řade je implementovat. Účelem mé práce je podpora přenositelnosti agentů mezi různými prostředími založenými na této architektuře, bez nutnosti úprav kódu. Cílem **WALA^2** je zajistit, aby každý, kdo se rozhodne ji dodržet a implementovat v ní svého agenta, mohl tohoto agenta simulovat v každém z prostředích naimplementovaných v této architektuře. Předpokládám, že to přinese ovoce při hodnocení výsledků, kdy se agenti budou moci porovnat v referenčním prostředí nebo bude možné vyzkoušet, jak se agent bude chovat v jiném prostředí, implementované někým jiným. Výsledky takovýchto pokusů, mohou být užitečné pro zlepšení jak algoritmů, tak i architektur agentů.

Druhou částí je konkrétní implementace této architektury (více viz. kapitola 3). Tedy naprogramováni jednoho takového prostředí. Část implementace vidím i ve vývoji uživatelsky přívětivých nástrojů, podporujících implementaci abstraktní architektury. To by mělo zajistit přístup širší veřejnosti, za účelem popularizace oboru umělé inteligence, robotiky a umělého života. Pro tyto účely jsem navrhnul jednoduché aplikace sloužící ke konfiguraci simulačního programu bez nutnosti znát formát konfiguračních souborů.

WAL Abstraktní Architektura (tedy **WALA^2**) představuje modulární, blokovou architekturou samotné platformy a schématicky ji popisuje Obrázek č. 2.



Obrázek č. 2 – WAL Abstraktní architektura - celkový pohled

Tato architektura byla navržená tak, aby byla možná parametrizace simulace, jak v iniciační fázi tak ve fázi běhu simulace. Možnost distrubuovanosti částí aplikace, tedy například mít tělo agenta (senzory a aktuátory) na jiném počítačích než mysl, nebo

simulovat na výkonném serveru a vizualizaci již méně výpočetně náročnou provádět na vzdáleném počítači. Dle mého názoru největším přínosem je rozdělení prostředí do vrstev. Tento krok zjednodušuje a zpřehledňuje simulaci a na druhé straně poskytuje možnost popsat i velmi složitá a komplexní prostředí.

2.1 Mechanismus aplikace - engine

Tato kapitola popisuje engine, který je základním prvkem architektury. Engine je jen jakýsi programový a spustitelný obal pro celou aplikaci. Nejprve je funkce a účel enginu slovně rozebrána (kap. 2.1.1). Následně jsou krátce specifikovány požadavky na funkci (kap. 2.1.2). Mým přínosem pro engine byl především návrh serveru a klienta pro vzdálený přenos (více o tomto tématu viz. 2.11, 3.11).

2.1.1 Teoretický rozbor

Hlavní částí aplikace je samotné simulační prostředí, zde nazývané engine nebo také platforma. Výraz simulační prostředí nebude používán, protože by mohlo snadno dojít k záměně s pojmem prostředí, kterým se míní svět agentů, ve kterém žijí a který vnímají prostřednictvím svých senzorů. Engine zastřešuje celou aplikaci a řídí její běh na úrovni programu. To znamená, že poskytuje celé aplikaci synchronizaci v podobě časových impulsů jednotlivých kroků, obsahuje rozhraní pro další podpůrné aplikace (viz dále) a obsahuje, udržuje a zpravuje své části. Těmi jsou vrstvy a agenti (podrobně rozebráno v 2.3.1 a 2.7.1). Kromě těchto dvou objektů se žádné další objekty v simulaci nevyskytují. Tedy i neživé objekty jako stromy, potrava, voda, zdi a další jsou také agenti. Mají některé parametry shodné s „živými“ agenty, ale na rozdíl od nich se nemohou například pohybovat, nebo rozhodovat sami o sobě. Tím je dána možnost vyvíjet se v čase i jiným agentům v prostředí, jako jsou například rostliny.

V jednom kroku simulace deleguje engine všechny vrstvy k vyhodnocení akcí všech agentů a jim odpovídajících změn prostředí. Po provedení všech změn následuje další krok simulace. Engine synchronizuje běh celé simulace.

Delegováním změn v prostředí na vrstvy dochází k distribuci řízení. Vrstvy by mohli být implementovány jako jednotlivá vlákna a běžet paralelně (na počítačích s jedním procesorem pseudoparalelně). Paralelizace však v tomto případě není triviální, jelikož mezi jednotlivými vrstvami vznikají vazby. Problém synchronizace se tak stává obtížným. Sekvenční řešení také není zcela vhodné. Jelikož změna v jedné vrstvě se může projevit změnou v jiné vrstvě a tedy si vynutí další průchod. Tímto způsobem rapidně roste složitost algoritmu a může dojít k zacyklení.

Triviálním řešením je použití jen jedné vrstvy. To však problém neřeší, nýbrž jej obchází. Nicméně pro méně náročné simulace nevyžadující více vrstev, jako jsou celulární automaty, boidi, floidi, biomorfové, atd. plně dostačuje.

Další řešení je sekvenční běh, kdy uspořádáme vrstvy do hierarchie. Tuto možnost lze využít jen v případě jednosměrného zřetězení vrstev. Potom se sekvenčně zpracuje

nejdříve vrstva s nejvyšší důležitostí. Její změny se propagují do dalších vrstev, které už zpětně nadřazenou vrstvu neovlivní. Toto řešení dostačuje ve většině požadovaných aplikací.

Posledním řešením je paralelní běh vrstev. V tomto případě je však nutná synchronizace. Ta se stává složitou v případě zpětných vazeb mezi vrstvami. Jak již bylo uvedeno výše ve většině případů jsou vrstvy provázány jen jednosměrně a proto postačí sekvenční řešení.

2.1.2 Specifikace engine

Pro engine by především mělo platit:

- je spustitelnou částí aplikace,
- zapouzdřuje celou aplikaci,
- poskytuje synchronizaci běhu programu,
- řídí a synchronizuje běh vrstev,
- obsahuje komunikaci s okolím,
- udržuje seznam všech objektů (vrstvy a agenti),
- udržuje datovou strukturu pro export,
- zajišťuje ukládání a obnovení simulace ze souboru.

Tyto vlastnosti by měla splňovat každá implementace.

2.2 Rozhraní engine–okolí (datová reprezentace)

Tato kapitola popisuje datové rozhraní mezi enginem a okolím. Předepisuje jak data simulace poskytovat navenek. V první části kapitoly je rozbor možností, které pro datovou reprezentaci přicházeli v úvahu (kap. 2.2.1). Následně jsou krátce specifikovány požadavky na datovou reprezentaci (kap. 2.2.2).

2.2.1 Teoretický rozbor

Rozhraní mezi simulačním prostředím a jeho okolím jako jsou například vizualizace, analýza nebo parametrizace, je důležitou součástí aplikace. Z hlediska rychlosti zpracování dat a tedy i celé simulace je vhodné vyměňovat si s okolím binární informace. Zvažoval jsem i výměnu informací v již existujícím formátu. Jako nejvýhodnější se mi jevilo XML. To je však velmi redundantní a jeho zpracování při větším objemu dat je pomalé. Proto se stalo jen jednou z možností exportu vnitřních dat.

Engine zpřístupňuje svá data a také data vrstev a agentů prostřednictvím interního sběru dat. Jakákoli externí aplikace má možnost tyto data přijmout a naložit s nimi za jakýmkoliv účelem. Tím myslím, že engine poskytuje navenek všechna data, která se budou autorovi jevit relevantní a naprogramuje jejich export. To znamená i privátní data,

ke kterým by neměli mít jiné komponenty ani enginu přístup. Důležité ale je, aby interface z engine byl dostatečně flexibilní a dobře dokumentován.

Vnitřní reprezentace dat není v abstraktní architektuře předepsána. Co však tato vnitřní reprezentace musí splňovat, je pevný formát, v jakém informace poskytuje navenek. Zde připadají v úvahu dvě varianty. XML nebo vlastní formát binárního souboru.

XML je dobře dokumentován a existují k němu celá řada vývojových nástrojů usnadňujících práci. Výhodou také je, že XML zápis popisuje data. Toto se bohužel také stává nevýhodou. Budeme-li pracovat nad většími soubory (řádově MB) je zpracování takového souboru značně pomalé. Typicky bude těchto souborů více. Doporučuje se, aby každý agent měl svůj konfigurační soubor. Tedy v takovém případě bude práce s daty velmi pomalá. Jistým východiskem je použití parsování XML pomocí SAX, které je rychlejší než klasické DOM (více v [26], [27]).

Binární reprezentace je naproti tomu velmi rychlá. Pokud ale vytvoříme binární soubor standardní serializací v jazyce Java, půjde jen ztěží přečíst z jiného programovacího jazyku. Objekt vytvořený za běhu programu zaniká v momentě skončení programu. Chceme-li tento objekt respektive jeho instanci zachovat, nazývá se tento objekt ***perzistentním objektem***. Proces uložení takového objektu do souboru se nazývá ***serializace*** [2]. Jelikož mi jde o přenositelnost a možnost tyto data posílat i po sítí a tam zpracovat, je tato možnost nevhodná. Proto je nutné za tímto účelem navrhnout vlastní postup zápisu dat simulace do binárního souboru tak, aby je tyto data mohl autor spolupracující aplikace opětovně rekonstruovat.

Řešení této situace vidím v binárním formátu dat s možností konverze do XML. Pro aplikace psané v jazyce Java, typicky další aplikace navazují na tuto, naprogramované ve skupině MRG, použiji binární reprezentaci standardní serializací. Objekty v souboru uložené jsou tudíž známé a práce s nimi je nenáročná. Pro externí aplikace napsané v jiných jazycích je nutné použít vlastní binární zápis (viz 3.2.2) či XML zápis.

2.2.2 Specifikace rozhraní

Navržené rozhraní má následující vlastnosti:

- zapouzdřuje data aplikace,
- umožňuje výpis data do XML,
- umožňuje výpis dat do podoby posílané po síti,
- umožňuje zápis do binárního souboru.

Tyto vlastnosti by měla splňovat každá implementace.

2.3 Vrstvy navrženého prostředí

Tato kapitola popisuje vrstvy prostředí. První část kapitoly poskytuje vysvětlení co se vrstvami myslí a jaké jsou druhy vrstev (kap. 2.3.1). Druhá část specifikuje činnost vrstev (kap. 2.3.2).

2.3.1 Teoretický rozbor

Vrstvy jsou hlavním stavebním kamenem prostředí. Definují svět, ve kterém se agenti pohybují. Slouží k logickému oddělení operací, které by jinak všechny musel řídit engine.

Vrstva je logicky oddělitelná složka prostředí, která je schopná vystupovat samostatně a s ostatními vrstvami tvoří dohromady prostředí jako celek. Všichni agenti, mající na vrstvu nějaký vliv nebo které ovlivňuje vrstva, budou propagovány do této vrstvy. Tedy vrstva má plné informace pro výpočet svých hodnot. V každém kroku na pokyn enginu provede vrstva vyhodnocení akcí agentů. Na jejich základě zmodifikuje sama své vlastnosti a poskytne nová senzorická data agentům.

Vrstva musí poskytovat možnost zaregistrovat agenta a odregistrovat agenta z vrstvy (podotýkám, že i neživé objekty jsou agenti). Dále musí být schopná naplnit senzory zaregistrovaných agentů hodnotami. To znamená, že každá vrstva spolupracuje jen s určitým typem senzorů, které umí naplnit. V každém kroku vrstva přečte výstupy agentů, tedy jejich spuštěné akce. Vyhodnotí zda jsou akce možné, zmodifikuje prostředí a poskytne agentům nová senzorická data.

V průběhu návrhu vyvstala otázka, jak aktualizovat stav vrstvy. Zda lze spočítat hodnotu v příslušném místě vrstvy na základě vlastností vrstvy nebo zda udržovat ve vrstvě kompletní informaci o rozložením hodnot (jakousi mapu). Je zřejmé, že výpočet hodnoty v určitém místě půjde z principu jen u některých vrstev. Vrstvy jsem rozdělil na dva druhy a nazval je bodová vrstva a gradientní vrstva.

Bodová vrstva je taková, kde lze vypočítat hodnotu v požadovaném místě, nebo kde je tato hodnota předem známá z dat obsažených ve vrstvě (tato data na rozdíl od gradientní vrstvy nejsou kompletním popisem hodnot ve všech místech vrstvy).

Gradientní vrstva je taková, kde z její podstaty nelze hodnotu v bodě spočítat a je nutné udržovat informaci o hodnotách ve všech místech vrstvy – mapu. To nutně vyvolává otázku, jak za účelem tvorby mapy diskretizovat prostor a jaké úrovně hodnot volit. Tato volba je však ponechána na konkrétní implementaci.

Vrstvy je možné dělit ještě z jednoho hlediska. A to na vrstvy s pamětí a vrstvy bez paměti. Vrstvy bez paměti jsou takové, které neuvažují svůj předchozí stav. Jejich aktuální stav lze spočítat z aktuálního stavu objektů, proto vrstva bez paměti odpovídá vrstvě bodové. Analogicky vrstva gradientní odpovídá vrstvě s pamětí.

Typ vrstvy vyplývá z její podstaty. Lze snadno nahlédnou, že vrstva udržující polohy agentů bude vrstvou bodovou, a vrstva udržující například teplotu bude vrstva gradientní.

Kolize ve vrstvách jsou dalším, poměrně závažným problémem. Z principu může dojít ke kolizím v každé vrstvě. Vrstva je zodpovědná za kolize v ní a musí je ošetřit. V případě, že nastane kolize ve více vrstvách, jak se bude vzniklá situace řešit? Jak jsem výše uvedl, vrstvy jsou jednosměrně hierarchicky seřazeny. Pokud nastane kolize ve vyšší vrstvě, kolize se ošetří, a není tedy propagována do nižších vrstev. Další velmi důležitá otázka, související s kolizemi je atomičnost akcí. Jelikož tento návrh počítá s vektorovým prostorem není odpověď na atomickou akci zcela triviální. Co je atomická akce, která se spustí a kdy už se nepředává řízení zpět mysli agenta? Zpuštěnou akci agenta vyhodnocuje vrstva. Agent může akci dokončit jen částečně. Vrstva ho tedy informuje, že akci provedl, ale akce dopadla jinak, než agent zamýšlel. Například, agent ve vzdálenosti 0,5 m od zdi se rozhodl spustit akci pohyb dopředu o jeden metr. V tomto případě vrstva jeho akci vyhodnotí a informuje agenta, že urazil pouze pul metru a navíc se mu zvýšila bolest (v důsledku nárazu).

Jelikož fyzické tělo agenta a jeho senzory jsou také součástí prostředí, je nutné je v prostředí interpretovat. Vrstva musí mít informaci o tom, jaký prostor v ní agent zabírá. Také senzory a efektory by měli zabírat prostor. Tím bude umožněna stavba složitějšího těla agenta. K této variantě v žádném případě nechci zavřít cestu a tudíž se s ní v návrhu abstraktní architektury musí počítat.

2.3.2 Specifikace vrstev

Vrstvy, které jsem navrhnul mají následující vlastnosti:

- vrstvy reprezentují prostředí,
- naplňují v každém kroku senzory všech agentů,
- rozhoduje o výsledku akce agentů,
- řeší kolize.

Tyto vlastnosti by měla splňovat každá implementace.

2.4 Vizualizace simulačního prostředí

Tato kapitola se zaměřuje na vizualizaci simulačního prostředí. První část kapitoly vysvětluje jaké možnosti vizualize jsou k dispozici (kap. 2.4.1). Druhá část specifikuje činnost vizualizace (kap. 2.4.2).

2.4.1 Teoretický rozbor

Dosud popsaná simulace je pouze změna stavů objektů uložených v paměti počítače. Úkolem vizualizace je zobrazení těchto objektů ve formě přijatelnější pro člověka. Snahou je také zobrazit simulaci tak, aby zaujala i širší veřejnost. Z principu

návrhu architektury je možné použít jakoukoliv vizualizační techniku. Pevně definovaná forma, v níž engine poskytuje data, je umožňuje zpracovat a na jejich základě zobrazit stav simulace.

Navržený systém vrstev prostředí poskytuje výhodu pro vizualizaci. Filtrováním přijatých dat je možné zobrazit odděleně stav v jednotlivých vrstvách.

Vizualizace dostává data od enginu několika způsoby. První a nejjednodušším je integrace vizualizace přímo do enginu. To ale neodpovídá abstraktní architektuře (viz. Obrázek č. 2). Navzdory tomu je tato možnost velmi často používána pro potřeby odladění simulace a jako zjednodušený náhled.

Další možností je externí vizualizace schopná pracovat s binárními objekty jazyku Java. Potom je možné ukládat data přímo do binárního souboru nebo tyto binární data posílat po síti.

Poslední možností je externí vizualizace, která není schopná pracovat s binárními daty jazyka Java, typicky aplikace napsaná v jiném programovacím jazyce. Zde lze za cenu pomalejšího zpracování použít XML formát dat a nebo binární zápis. Druhý způsob je touto prací preferován.

2.4.2 Specifikace vizualizace

Vizualizace mající zájem o informace z simulace by měla splňovat:

- externí modul,
- přijímá data od enginu (i po síti),
- zobrazuje svět agentů.

Tyto vlastnosti by měla dodržet každá implementace.

2.5 Off-line analýza simulace

Analýzou je zde myšleno zpracování dat simulace a její vyhodnocení. Za tímto účelem byl ve spolupráci s CGG FEL ČVUT vyvinut nástroj Visual Analysis Tools© zkráceně VAT (podrobněji popsán v [13], [14], [15]). Na tomto místě je pouze teoretický rozbor vhodných analytických nástrojů (viz. 2.5.1) a požadavky na ni (viz. 2.5.2). Podrobnější popis nástroje VAT je možné nalézt v 3.5.

2.5.1 Teoretický rozbor

Jak jsem již v kap. 1.5 naznačil, přístup k analýze a vyhodnocování průběhu simulace je různý. Většinou se jedná o online nebo offline vykreslení průběhu několika veličin v čase. Při složitějších simulacích, kdy agent má řádově desítky parametrů nastává problém, jak průběhy těchto parametrů v čase vizualizovat.

Vykreslení všech průběhů do jednoho grafu se nezdá příliš vhodným řešením, neboť při relativně malém počtu parametrů začíná být graf velmi nepřehledný, což znemožňuje kýženou analýzu chování. Vykreslení každého průběhu do samostatného grafu odstraňuje nepřehlednost, ale analýze chování nenapomáhá, jelikož hledání souvislostí parametrů v těchto grafech je obtížné. Kombinací obou protichůdných přístupů by se dalo dosáhnout optima, kdy by byla analýza možná. Ale není nezbytné se vzdávat některých vazeb nebo přehlednosti. Při využití třetího rozměru pro vizualici dochází k zpřehlednění analýzy a počítačová grafika zná řadu metod, jak zobrazovat prostory vyšších dimenzí. Proto je silně podporováno využití 3D metod analýzy. To má velmi mnoho výhod, jelikož není nutné velikost parametrů mapovat na průběh v čase nebo popřípadě barvu čáry, ale i na tvar, výšku a šířku objektu a další. Další výhodou je možnost citlivostní analýzy.

2.5.2 Specifikace analýzy

Analytické nástroje mají tyto vlastnosti:

- offline nebo online vyhodnocení simulace,
- rychlé získání přehledu o komplexní simulaci,
- zpětná analýza zajímavých situací,
- zobrazení vztahů mezi vstupy a akcemi.

Tyto vlastnosti by měla splňovat každá implementace.

2.6 Parametrizace simulovaného světa

Tato kapitola se zaměřuje na parametrizaci simulace. V části 2.6.1 definuje co si pod parametrizací představuji a rozebírá výhody parametrizace. Část 2.6.2 shrnuje požadavky na parametrizaci. Za přínos považuji možnost zpětného chodu simulace.

2.6.1 Teoretický rozbor

Parametrizace poskytuje možnost ovlivnění simulace, ať již se jedná o počáteční nastavení simulace (inicializace), či o změnu simulace za běhu. Změnou simulace za běhu je myšlena možnost změnit některý parametr agenta či vrstvy při běhu simulace. Například zadat agentovy nový cíl nebo snížit teplotu v některém místě tepelné vrstvy. K parametrizaci také patří možnost vytvářet objekty simulace za běhu. Například vytvoření nového agenta (ještě jednou podotýkám, že agentem jsou i neživé objekty, tudíž si zde můžeme pod pojmem vytvořit nového agenta představit třeba přidání jídla na určitou pozici v prostředí).

Nebylo by tedy nutné předpovídat průběh simulace nebo ho designovat dopředu a k tomu vytvořit příslušný iniciační soubor se situací (kdy by byl pevně zadán počet agentů,

jejich polohy, atd.). Bylo by možné spouštět dlouhodobější simulace s tím, že by se scénáře dali upravovat za běhu aplikace, podle toho jak by se situace vyvíjela. To by podle mě mělo velmi pozitivní vliv na výsledky simulací. Navíc, v kombinaci s online vizualizačními metodami analýzy, by bylo možné sledovat zajímavé momenty a na jejich základě případně měnit běžící scénář.

Do parametrizace jsem zařadil i zastavení a krokování simulace. Její uložení na disk a nahrání z disku. Jedním prvkem, který zatím simulacím chybí, je možnost zpětného chodu simulace ve smyslu návratu k už simulovanému. Toto není jednoduchou záležitostí, jelikož se musí uložit kompletní stav simulace. Možnost zpětného trasování není abstraktní architekturou vyžadována, ale byla by jistě přínosem.

Parametrizace simulace za běhu programu je poměrně složitá a náročná oblast. Tento problém by se dal velmi elegantně vyřešit pomocí datové reprezentace udržované prostředím. Vizualizační a analytické moduly, tuto datovou reprezentaci, nebo její část přijímají a dále ji zpracovávají. Parametrizační modul by naproti tomu enginu poslal novou datovou reprezentaci. Agenti stejně tak jako propagují svá data do této datové reprezentace, by byli schopni tyto data také číst a následně modifikovat svoje parametry a vlastnosti dle přání uživatele.

2.6.2 Specifikace parametrizace

Parametrizace zajišťuje především:

- možnost ovlivnění simulace za běhu,
- krokování simulace,
- zpětné krokování simulace.

Tyto vlastnosti by měla splňovat každá implementace.

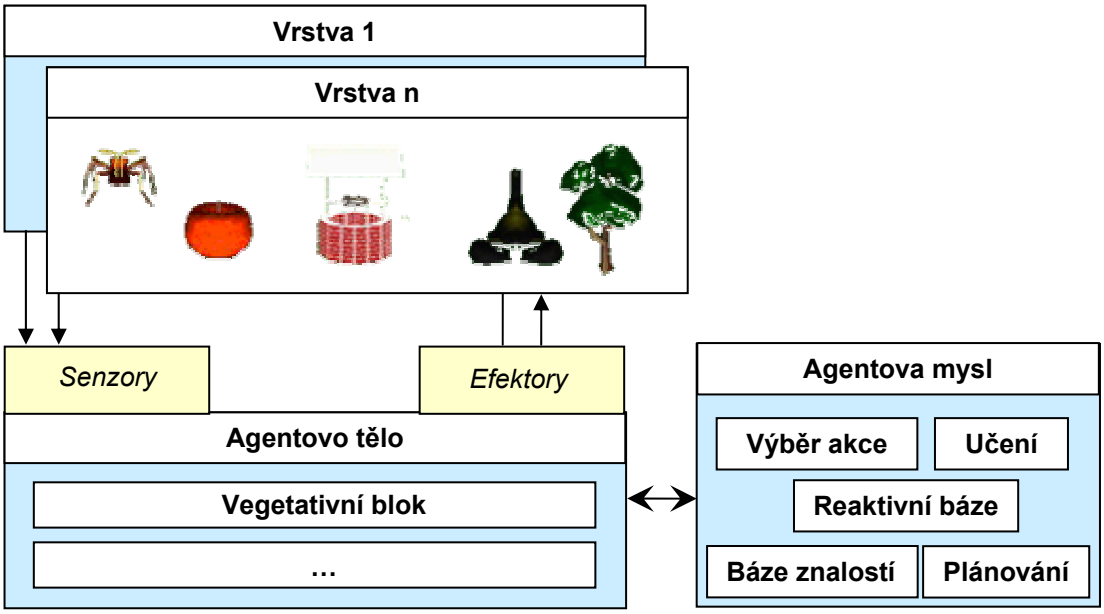
2.7 WAL agent – výchozí specifikace

V současnosti se stále více dostávají do popředí společenství robotů-agentů. Obecně se tento fenomén nazývá MAS – multiagentní systémy a jejich řízení [3]. V přírodě se tyto struktury vyskytují ve všech stupních složitosti počínaje geny, buňkami, mnohobuněčnými strukturami, přes rostliny, zvířata, skupiny zvířat až po jejich společenství, druhy, podtřídy a třídy. Stejně jako v přírodě je i v robotice jasné, že mnohdy jeden superinteligentní a tím finančně značně nákladný robot svými schopnostmi zaostává za „rojem“méně chytrých, jednodušších, a tudíž i podstatně levnějších robůtků. V množství je síla a v jednoduchosti také. Rovněž spektrum možných typů úloh implementovatelných na společenství vzájemně spolupracujících robotů je mnohem širší a pestřejší než v případě pouze jednoho robota. A právě pro řízení společenstev robotů se více než kdy předtím s výhodou uplatňují přístupy umělého života. Při realizaci inteligentního chování se přístup

ALife (biologický model) inspiruje hlavně přírodními fenomény, zatímco klasická umělá inteligence (racionální model) je založena na logice, lingvistice, racionalitě a jen částečně na algoritmizovatelných vzorech převzatých z přírody. Další výrazný rozdíl mezi přístupy UI a ALife je v předmětu zkoumání. Umělá inteligence se tradičně zabývá komplexní lidskou činností, např. hraním šachů, pochopením textu, diagnostikou chorob apod., zatímco ALife se zabývá základními prvky přirozeného chování s důrazem na přežití v prostředí. Většina dosavadních simulací přístupu ALife se opírá o algoritmy, které umožňují robotům, jakožto uměle vytvořeným bytostem, vyvíjet se a přizpůsobovat prostředí v němž žijí [11], [12].

Tato kapitola nejprve poskytuje popis agenta (viz. 2.7.1) a posléze specifikuje požadavky na jeho implementaci (viz. 2.7.2).

2.7.1 Teoretický rozbor



Obrázek č. 3 – Rozhraní vrstvy-agentovo tělo a agentovo tělo-agentova mysl

Architekturou agenta se zde budu zabývat jen zčásti. **WALA**^{2} odděluje tělo agenta, (fyzickou reprezentaci v prostředí) a mysl agenta (rozhodovací mechanismy). Agentovo tělo je součástí prostředí a tedy patří do návrhu prostředí. Mysl agenta naproti tomu komunikuje s tělem jen prostřednictvím dat ze senzorů. Za senzory je považována i interocepce (tedy vnitřní stav agenta). Interocepce je zde reprezentována jednak vlastním stavem senzorů a efektorů (některý z efektorů nebo senzorů může být poškozen a agent respektive jeho mysl, by o tom měla mít informaci, aby vyloučila jeho použití) a také vegetativním blokem (*Vegetative system block*) navrženým Ing. Kadlečkem [7]. Poté co jsou agentovi nastavena data senzorů, agentovo tělo tyto data pošle mysli, která je zpracuje. Jakým způsobem je zpracuje není předmětem této abstraktní architektury. Architektura mysli agenta je popsána například v ([3], [6], [7]). Mysl situaci vyhodnotí a posílá zpět vybranou akci (popřípadě akce možné provést paralelně). Tělo se tyto akce

pokusí vykonat použitím svých efektorů. Vrstva tělu sdělí výsledek akce nastavením nových dat senzorů. Blokové schéma rozhraní v architektuře ukazuje Obrázek č. 3.

Stejně jako u vrstev i zde se naskýtá otázka, jaký způsobem předávat řízení mezi agenty. Reprezentovat každého agenta jako vlákno (*Thread* – viz [2]) se zdá být nejpřirozenějším řešením pro agenty. Problém synchronizace akcí agentů zde není tak palčivý. Navíc v reálném životě jsou akce více agentů synchronizovány jen z jejich vlastní vůle, ne z jejich podstaty. Tato vize je hezká, bohužel v praxi špatně realizovatelná, jelikož musíme vyhodnocovat kolize agentů. A také by se potom mohlo lehce stát, že reaktivní agent, jehož vlákno neprovádí téměř žádný výpočet bude několikanásobně rychlejší než agent hybridní (který navíc i plánuje). Proto zde v abstraktní architektuře podporuji volbu reprezentace agenta jako vlákna, jen upozorňuji na možné problémy. Porovnáním těchto přístupů hybridní architektury vs. optimalizovaných agentů na jednu činnost se bude věnovat jiná diplomová práce. Protože abstraktní architektura si klade za cíl podpory dalších diplomových prací, bylo nutné se zde o tomto problému zmínit.

2.7.2 Specifikace WAL-agenta

Vlastnosti agenta jsou následující:

- klíčový objekt celé simulace,
- **WALA²** se zabývá pouze návrhem těla agenta,
- exportuje data do globální datové struktury,
- zprostředkovává rozhraní mezi myslí a prostředím,
- zaujímají prostor, mají polohu.

Tyto vlastnosti by měla splňovat každá implementace.

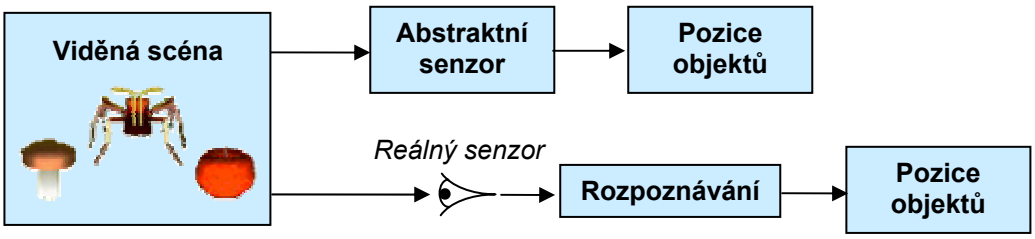
2.8 Rozhraní agentovo tělo-vrstvy, senzory

Tato kapitola popisuje první ze dvou prvků rozhraní mezi agentem a prostředím. V části 2.8.1 se věnuje účelu a funkci senzorů. V části 2.8.2 uvádí jejich úlohu v prostředí.

2.8.1 Teoretický rozbor

Prostřednictvím senzorů agent vnímá okolí kolem sebe a také svůj vlastní stav. Podle toho jaké senzory bude mít agent k dispozici, tak bohatou informaci o prostředí bude mít. Důležitá věc je uvědomit si, že obcházíme rozpoznávání. Tedy nezabýváme se otázkami, které se například v robotice nutně musí řešit, jako je počítačové vidění nebo tvorba vnitřního modelu ze sonarických dat. Reálný robot nebo i živočich má velmi komplexní senzory a tedy informace které poskytují musí projít fází rozpoznávání. My však tuto fázi obcházíme volbou takových mechanismů, které poskytují již rozpoznaná

data (abstraktní senzor). Tuto situaci schématicky naznačuje Obrázek č. 4. Reálný senzor je zde zobrazen jako oko, které je příkladem velmi komplexního senzoru. Rozpoznávání obcházíme z toho důvodu, že se chceme soustředit na jinou oblast. Tedy na výzkum chování agenta, ne na výzkum jeho schopnosti rozpoznat objekty prostředí.



Obrázek č. 4 – Rozdíl mezi reálnými a abstraktními senzory

Vrstva musí znát typy senzorů, aby je uměla naplnit daty. Tedy každý kdo chce přidat nějakou vrstvu, musí nutně k ní přidat i senzory, s kterými vrstva umí pracovat, či implementovat naplnění jíž existujících senzorů touto vrstvou. Popřípadě při přidávání nového senzoru modifikovat vrstvu tak, aby uměla pracovat i tímto novým senzorem. Senzory jakožto součást prostředí znají přesná data. Je v nich uložená přesná číselná hodnota. Chceme-li se abstraktní architekturou více přiblížit realitě, musíme definovat filtry těchto hodnot. Filtry převedou přesnou číselnou hodnotu na fuzzy hodnotu. Například hodnotu vzdálenosti objektu 1m převedou na hodnotu „blízko“, atd.

Dalším problémem je zaměření pozornosti. Všechny agentovi senzory jsou v jednom kroku simulace naplněny. Ne však všechny údaje z nich agenta zajímají. Ba co víc, agent by neměl z principu byt schopny všechny informace zpracovat. Což ho bude nutit k zaměření pozornosti na data, která ho zajímají nejvíce. Ostatní data nebudou v tu chvíli podstatná. Agentova mysl musí rozhodnout jaká data jsou pro agenta nejdůležitější a tato data si od těla vyžádat.

2.8.2 Specifikace senzorů

Vlastnosti a činnost senzorů jsou:

- vstupní bod při komunikaci prostředí s myslí,
- vrstva musí být schopná daný senzor naplnit,
- zaujímají prostor, mají polohu.

Tyto vlastnosti by měla splňovat každá implementace.

2.9 Rozhraní agentovo tělo-vrstvy, efektory

Tato kapitola popisuje druhý prvek rozhraní mezi agentem a prostředím. Kapitola 2.9.1 je věnována účelu a funkci efektorů. Kapitola 2.9.2 upřesňuje jejich úlohu v prostředí.

2.9.1 Teoretický rozbor

Prostřednictvím efektorů agent ovlivňuje své okolí (interaguje s ním). Efektory jsou podobně jako senzory, zobecněné (abstraktní). Tedy neuvažujeme efektory typu noha, nebo dokonce klouby. Neřešíme inverzní ani dopředné kinematické úlohy. Typickým příkladem efektoru v prostředí je pohyb daným směrem o danou vzdálenost. Nicméně abstraktní architektura nebrání pečlivějšímu zaměření na efektory a jejich detailnější rozpracování. Jelikož efektory stejně jako senzory zabírají prostor, lze principiálně sestavit z několik efektorů například nohu a teprve pomocí složení instrukcí pro takovéto efektory zajistit pohyb. Projekty zabývající se vývojem stvoření učících se pohybovat pomocí genetických algoritmů je několik (viz například [39]).

2.9.2 Specifikace efektorů

Vlastnosti a činnost efektorů jsou:

- výstupní bod při interakci prostředí s myslí,
- agentův nástroj ovlivnění prostředí,
- spouštějí agentem vybranou akci.

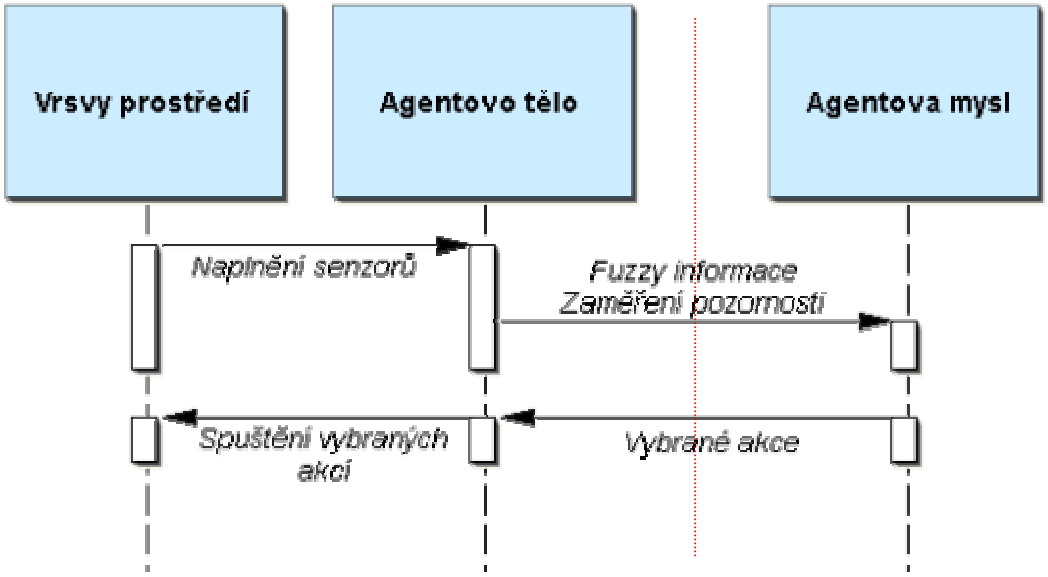
Tyto vlastnosti by měla splňovat každá implementace.

2.10 Rozhraní agentovo tělo-agentova mysl

Tato kapitola se věnuje rozhraní mezi agentovým tělem a myslí. Tyto dvě části jsou v **WALA²** odděleny a tedy je nutné popsat jejich rozhraní. V části 2.10.1 je poskytnut slovní komentář k této problematice. V části 2.10.2 jsou uvedeny požadavky.

2.10.1 Teoretický rozbor

Výše jsem se zmínil o programovém oddělení mysli od těla. Od tohoto si slibuji, možnost přenosu mysli mezi různými těly. Mysl je ale pořád vázána na informace, které dostává od těla. Musí být schopná jim rozumět a umět je zpracovat. Mysl není univerzálně přenositelná. Je přenositelná v rámci implementace používající pevný výčet senzorů. Což bude v praxi téměř vždy, jelikož autor konkrétní implementace bude znát typy senzorů, které použil nebo které byli použity, tedy při návrhu mysli je schopen data z nich zpracovat.



Obrázek č. 5 – UML Sekvenční diagram komunikace mezi agentem a vrstvami

Jednotlivé fáze komunikace popisuje Obrázek č. 5. Komunikace mezi agentem a vrstvami je komunikací vnitřní. Tedy není nutné ji provádět pomocí zasílání dat. Naopak komunikace mezi tělem a myslí (v obrázku odděleno tečkovanou čarou) je považována za zasílání zpráv a je možná řešit přenosem přes různá přenosová media (například Internet).

2.10.2 Specifikace rozhraní

Toto rozhraní zajišťuje následující činnosti a má následující vlastnosti:

- číselná informace filtrována,
- komunikace vrstvy-tělo může probíhat interně,
- komunikace tělo-mysl by měla probíhat externě.

Tyto vlastnosti by měla splňovat každá implementace.

2.11 Komunikace

Tato kapitola pojednává o komunikaci uvnitř i vně simulačního enginu. Nejen tedy o komunikaci mezi agenty, ale i o komunikaci jednotlivých modulů vzájemně. Teoretický rozbor (viz. 2.11.1) se věnuje možnostem komunikace. Specifikace (viz. 2.11.2) shrnuje požadavky na komunikaci.

2.11.1 Teoretický rozbor

Nejprve se zabývejme komunikací na úrovni celého enginu. Možnost mít senzory na jiných místech než aktuátory a mysl je velmi lákavá. Otázkou zůstává, zda v této chvíli

je toto nezbytně nutné, a zda snaha o tuto distribuovanost nebude spíše na škodu (z důvodu pomalého zpracování při přenosu po síti). Tato možnost však nesmí zůstat do budoucna uzavřená a abstraktní architektura ji podporuje svojí modulárností. Rozšířením pomocí předřadných bloků lze tuto funkcionalitu dodat. Po připsání předřadných komunikačních částí by nebyl problém mít zcela distribuovanou architekturu.

Komunikace probíhá také na úrovni agentů. Rozumí se tím šíření zpráv zasílaných jedním agentem druhému nebo skupině jiných agentů za účelem dorozumění. Zde se snažíme přiblížit reálné situaci, kdy o šíření zprávy rozhoduje prostředí, respektive příslušná vrstva prostředí za šíření zodpovědná. Ta vyhodnotí, kteří agenty zprávu „uslyší“, případně informaci zašumí, pozmění, atd. Prostředí tedy ví, jaké senzory a efektory agenti mají k dispozici a na jakou informaci mají právo. Pokud agent chce poslat zprávu jinému a použije k tomu médium (akustické vlnění), které může zachytit i třetí agent, není důvod třetímu agentu zprávu zaslat. Ten ji nemusí nijak zpracovávat, nemusí ji věnovat pozornost, jeho příslušný senzor bude touto zprávou naplněn.

2.11.2 Specifikace komunikace

Komunikace slouží především k:

- navržená bloková architektura umožňuje distribuovatelnost,
- komunikace na úrovni agentů vyhodnocují přenosové vrstvy.

Tyto vlastnosti by měla splňovat každá implementace.

3 Realizovaná implementace WAL

WALA² je abstraktní architektura. Doporučuje komponenty a techniky tvorby prostředí tak, aby bylo modulární a bylo schopno komunikovat s řadou externích aplikací. Důraz je také kladen na přenositelnost agentů. Tedy oddělení reprezentace agenta v prostředí od řízení chování agenta (jeho mysli). Pro import agenta stačí naprogramovat jeho reprezentaci v prostředí. To bude tvořit interface mezi prostředím a myslí, která může být převzata od jakéhokoliv agenta. Je zřejmé, že přenos mysli agenta, při jehož návrhu se počítalo s prostředím **WAL,** bude o mnoho snazší než převod agenta, který byl naprogramován zcela nezávisle.

Tato kapitola popisuje konkrétní implementaci jednotlivých částí ve stejném pořadí jako byli teoreticky popsány výše (viz kap. 2). Je tedy popisem konkrétních řešení a zároveň programátorským manuálem. Je proto ilustrována řadou UML schémat (více o UML viz [51], [P19]) a obsahuje odkazy na implementované třídy.

3.1 Mechanismus aplikace - engine

Hlavní mechanismus celé simulace obstarává grafické uživatelské rozhraní sloužící k řízení simulace a jejímu ovlivnění. Největší výhodou a přínosem je možnost zastavení simulace a zpětného chodu simulace. To se děje pomocí ukládání stavu simulace do souborů v adresáři ***hist*** simulace. Díky nevhodné manipulaci se soubory s náhodným přístupem v jazyce Java, jsem se rozhodl každý krok simulace uložit jako jeden soubor obsahující veškeré hodnoty nutné k zrekonstruování kroku. Tyto soubory jsou pojmenovány *datax.dat*, kde x je identifikátor kroku simulace. Pro čtvrtý krok simulace je tedy vytvořen soubor *data3.dat* (číslováno od 0). Tyto soubory jsou po ukončení aplikace automaticky smazány, aby při příštím spuštění aplikace nekolidovali s nově vznikajícími soubory. K provedení zálohy těchto souborů je nutné simulaci uložit do jiného adresáře, buď ručním zkopírováním souborů historie, nebo pomocí ovládacích prvků v hlavní nabídce engine.

Ukládání kroků má několik výhod. Při zpětném chodu se načítají předchozí stavy celého prostředí. Při opětovném chodu dopředu je zde možnost simulovat znovu kroky, které už byli uloženy, nebo se vrátit pomocí uložených souborů do místa zastavení a pokračovat v simulaci. Tedy je možné vyzkoušet opakovatelnost simulace tím, že simulaci zastavíme, vrátíme jí o několik kroků a tyto kroky simulujeme znovu. Porovnáním obou stavů, do kterých jsme se dostali, lze získat zajímavá data o opakovatelnosti a determiničnosti agentů.

Engine je implementován v balíčku ***engine***. Jeho třídy jsou ***Engine.java*****,** hlavní spustitelná třída enginu, která vytváří okno enginu a startuje TCP/IP server. ***EngineFrame.java*** definuje vzhled hlavního okna a obsluhuje události posílané uživatelem

(kliknutí myši, stisknutí tlačítka, atd.). ***Layer.java*** definuje vrstvy prostředí (více viz 3.3). ***EngineMenu.java*** definuje vzhled menu enginu. ***MyPanel.java*** se stará o vykreslení simulace. ***DataTable.java*** zobrazuje parametry agenta vybraného kliknutím myši. ***HistoryManager.java*** se stará o korektní uložení kroků simulace do souborů a jejich opětovné načtení. ***Adapter.java*** adapter pro zobrazovanou tabulku údajů o agentovi. Součástí balíčku ***engine*** je i další balíček ***engine.action***. Tento balíček obsahuje obsluhu událostí posílaných hlavním menu enginu.

3.1.1 Grafické uživatelské rozhraní

Hlavní obrazovka se skládá ze 3 částí. Je to grafické okno, informační panel a ovládací panel (více viz 4.1.1). Grafické okno je velmi zjednodušené, protože neslouží k presentaci simulace, nýbrž k rychlému se zorientování v simulaci. Celá simulace by mohla probíhat bez tohoto okna, protože vizualizaci prostředí obstarává externí modul. Toto okno jsem zde ponechal ze dvou důvodů. Hlavním důvodem je stále ještě zaintegrování parametrizace do aplikace. Její návrh jako externího modulu překračuje rámec této práce a po dohodě s vedoucím diplomové práce od něj bylo ustoupeno. Dalším důvodem je odladění funkčnosti aplikace, kdy i z jednoduchého grafického znázornění lze lépe poznat správný či špatný běh simulace, než z výpisu proměnných na konzoli.

K presentaci simulovaného světa slouží externí moduly, které dostávají od enginu data přes TCP/IP protokol (více v části 3.4 a 3.11). Dalším prvkem je tabulka vlastností vybraného agenta. Na ploše lze na agenta kliknout myší a nechat zobrazit podrobnější údaje o něm. Posledním prvkem je řídící panel s údaji o celém prostředí, jako je počet vrstev, počet agentů a objektů ve vrstvě a krok simulace. Ovládání aplikace je popsáno v kapitole 4.

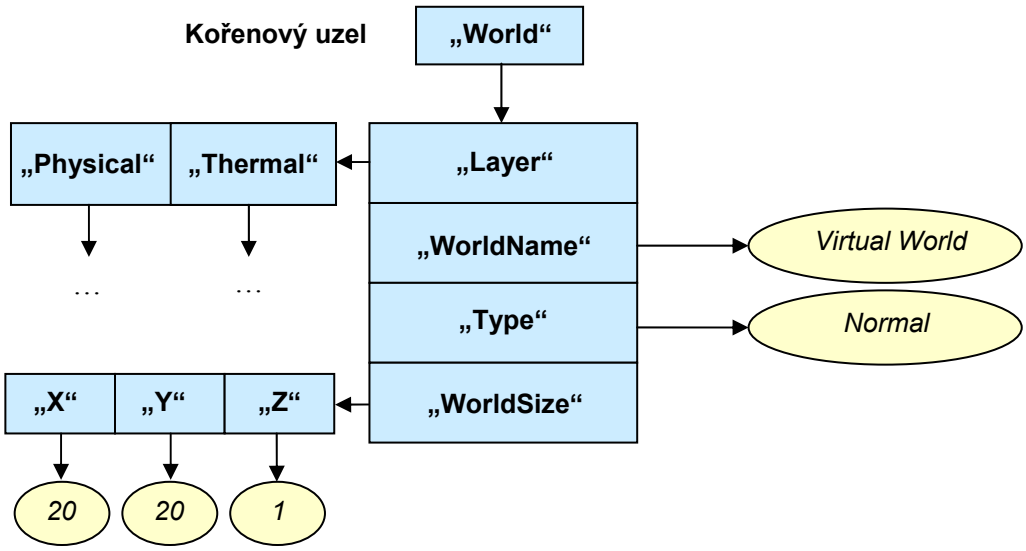
3.2 Rozhraní engine-okolí (datová reprezentace)

Tato kapitola se věnuje popisu rozhraní mezi enginem a okolím. Nejprve je osvětlena vnitřní datová reprezentace (viz. 3.2.1). Posléze je proveden návrh binárního formátu dat (viz. 3.2.2).

3.2.1 Vnitřní datová struktura verze 1.0

Za účelem výměny dat s externími moduly byla navržena jednoduchá stromová datová struktura. Ta je implementována pomocí asociativních polí (*Hashtable*, *Hashmap* v jazyce Java [2]). Vytvořil jsem dva základní typy uzlů. První je uzel obsahující další uzly (tento uzel jsem označil typ 1). Druhý je uzel obsahující hodnotu, tento uzel je označen typem podle proměnné 2 – celočíselná hodnota (*int*), 3 – řetězec znaků (*String*), 4 – číslo v plovoucí řádové čárce s dvojnásobnou přesností (*double*), 5 – číslo v plovoucí řádové čárce s jednoduchou přesností (*float*).

Příklad uložení informací v této datové struktuře viz Obrázek č. 6. Uzly obsahující další uzly jsou značeny bíle vyplněným obdélníkem, uzly obsahující hodnotu modře vyplněnou elipsou. Z hlediska kompatibility s XML, kde musí existovat jeden kořenový uzel, i zde musí být jen jeden kořenový uzel.



Obrázek č. 6 – Ukázka části datové struktury

Do datové struktury se prvky přidávají pomocí hodnoty a adresy. Hodnota je konkrétní informace, kterou chceme uložit. Adresa jsou jména uzlů od kořenu k listu, oddělená lomítkem. Například adresa jména světa by byla World/WorldName. V adrese jsou rozlišována malá a velká písmena, adresy world/ a World/ jsou tedy dvě rozdílné adresy. To plyne z podstaty jazyka Java, který rozlišuje velká a malá písmena (*case-senzitive*).

Tato datová reprezentace byla označena za verzi 1.0. Je velmi jednoduchá, což přináší celou řadu nevýhod. Implementace pomocí asociativních polí neumožňuje uložit pod jeden klíč více hodnot. Pro malé a jednoduché přenosy dat jako mezi enginem a VRML vizualizací postačí. VRML modul z důvodů popsaných níže (3.4.4) vyžaduje značnou jednoduchost. Pro další použití enginu se počítá s dalšími verzemi datového stromu, který umožní mít pod jedním klíčem seznam prvků a tím se stane podstatně více universálnějším.

Implementace datové struktury se nachází v balíčku ***data***. Samotná datová struktura je implementována ve dvou třídách. ***SimpleDataNode.java*** implementuje datové struktury a ***Address.java*** je třída pro práci s adresou.

3.2.2 Binární zápis pro komunikaci s okolím

Datové struktury vytvořené v jazyce Java a posílané přes internet, lze bohužel přijímat jen v aplikacích napsaných také v tomto jazyce a navíc musí přijímací aplikace znát třídy objektů, které přijímá. Pro mnou navržený a naprogramovaný vizualizační modul toto řešení postačilo. Avšak cílem této práce je navrhnout takové řešení, které je

přenositelné, je zde požadavek, aby přijímající aplikace nemusela znát třídy přijímaných objektů a navíc aby mohla být napsána v libovolném programovacím jazyce. Tento požadavek vedl k návrhu a implementaci binární podoby datové reprezentace.

Komunikace enginu s okolím probíhá jako klient-server aplikace. Externí modul mající zájem o data ze simulace se přihlásí k enginu pomocí nějakého komunikačního média. Já jsem implementoval TCP/IP protokol. Ale je možné využít i jiných technologií, jako například pojmenovanou rouru (*pipe*).

Po přihlášení k médiu, tedy po úspěšném připojení na TCP/IP socket, který jsem zvolil v souladu s rokem vzniku jako 2004, proběhne „přihlašovací sekvence“, kdy se externí modul a engine domluví na verzi komunikace. Jako každý klient se i externí moduly budou v jistém smyslu hlásit nebo registrovat u engine (serveru). To má několik důvodů:

- engine by měl vědět, že do konkrétního komunikačního media má posílat data,
- engine by měl vědět co je na druhém konci a co od toho může očekávat (minimálně by měl mít informaci o typu modulu, který se snaží o komunikaci),
- engine by měl vědět, jakou verzi komunikace modul umí.

Klient tedy pošle serveru zprávu *OP_CONNECT*

```
OP_CONNECT {
        char moduleName[8];
        unsigned long version;
}
```

kde *moduleName* je krátká identifikace modulu například „vizu“ a *version* je verze komunikace, jakou modul zvládá. Poté co server zprávou *ACCEPT* potvrdí, že modul zná a je schopen s ním v dané verzi komunikovat, pošle modul žádost o identifikátory stromu v engine *OP_REQUEST_IDENT_MAP*. Server pošle zpět překladovou mapu *OP_IDENT_MAP* obsahující dvojice řetězec a identifikátor. Identifikátory byli zavedeny z důvodu úspory místa. Jakmile se jednou pošlou celé řetězce s názvy uzlů a externí modul i engine mají převodní tabulku, není nutné posílat jako adresu uzlu textovou zprávu, ale pouze identifikátor.

```
OP_IDENT_MAP {
        data = {oneRecord}+;
        oneRecord = string, id;
}
```

kde *{}* značí blok a + značí jeden a více výskytů bloku. String bude null-terminated string libovolné délky a id je číslo typu long.

Většina externích modulů nebude vyžadovat všechna data v každém okamžiku od engine. Ten by jim data měl „filtrovat“. Specifikaci filtru (části podstromu dat engine) posilá externí modul. Tento filtr bude sekvence cest ke kořenům podstromu.

```
OP_TREEFILTER {
        data = { path }+ oddelovac;
        path = { nodeID }+;
}
```

kde *nodeID* je kód uzlu dle domluvené převodové tabulk. Identifikátor 0 bude rezervován na oddělovač.

Po takovéto domluvě, kdy je jasné co modul chce, může engine začít posílat data. Ve většině případů není posílán celý strom, ale jen jeho části. Každá taková část bude uvozena cestou k jejímu kořeni ve stromu. Pak následuji datové úseky (*chunky*). Zpráva posílaná enginem externímu modulu po navázání komunikace:

```
Msg {
        long opType;
        long dataSize;
        char data[];
}
```

kde *opType* je identifikace zprávy a *dataSize* je délka dat.

```
OP_ENGINE_DATA {
        data = timeStamp {path, oddelovac, subtreeData}+;
        path = {nodeID}+;
        oddelovac = '\0';
        subtreeData = chunk;
        chunk = chunkID, chunkSize, {chunkData};
        chunkData = {chunk}* | konkrétní data (byte, short, int, long, float, double, string);
}
```

kde *timeStamp* je časová známka simulace typu long, *chunkID* je identifikátor datového úseku typu long a *chunkSize* je velikost datového úseku v bytech.

Data uvnitř datového úseku umí externí modul zpracovat (parsovat). Pokud tyto data nezná, tak je bude ignorovat, z toho důvode je nutné uvádět velikost datového úseku.

Na závěr této kapitoly je uvedena tabulka datových typů a jejich velikostí Tabulka č. 1. Tato tabulka vychází z definice typů proměnných jazyku Java [2]. Pouze řetězec zakončený nulou není jazykem Java podporován a tedy musel být vytvořen pomocí sekvence bytů s ASCII kódy znaků, zakončený nulou.

Typ	Binární kód	Délka	Délka	Min. hodnota	Max. hodnota
-	-	[bit]	[byte]	-	-
byte	0001	8	1	-128	127
short	0010	16	2	-32768	32767
int	0011	32	4	-2147483648	2147483647
long	0100	64	8	-9223372036854770000	9223372036854770000
float	0101	32	4	±1,4E-45	±3.4E38
double	0110	64	8	±4.9E-324	±1.7E308
String	0111	n*8+8	n+1	0	255

Tabulka č. 1 – Datové typy

Vzhledem k rozsáhlosti a časové náročnosti celé práce zůstala binární datová reprezentace pouze ve fázi návrhu a ranných testů implementace. Nicméně odevzdáním diplomové práce nekončí tento projekt a budu se dále podílet na jeho zlepšování, což zahrnuje i plné otestování této binární datové struktury.

Implementace binární datové struktury se nachází v balíčku ***data***. Samotná datová struktura je implementována ve třídě ***DataNode.java***, která je odvozena od třídy ***SimpleDataNode.java***. Tuto třídu rozšiřuje o konverzi do binární podoby. Tato třída navíc využívá služeb třídy ***Binary.java*** z balíčku ***utils***. Ta slouží k převodu primitivních datových typů na pole bytů. V balíčku ***data.binary*** jsou připraveny dvě třídy ***BinaryTest.java*** a ***ReadTest.java*** sloužící k ověření správné funkci binárního zápisu.

3.3 Vrstvy navrženého prostředí

Vrstvy jsou reprezentací prostředí. Obsahují seznam zaregistrovaných agentů. Těm poskytují data na jejich senzory vyhodnocením stavu ve vrstvě a zpracovávají agentem spuštěnou akci. To znamená, že vyhodnotí, zda agentova akce je možná a zmodifikují prostředí podle výsledku akce tak, aby mohla v dalším kroku poskytnout nová data na senzorech agentů. Vrstva je implementována v balíčku ***engine*** třídou ***Layer.java***.

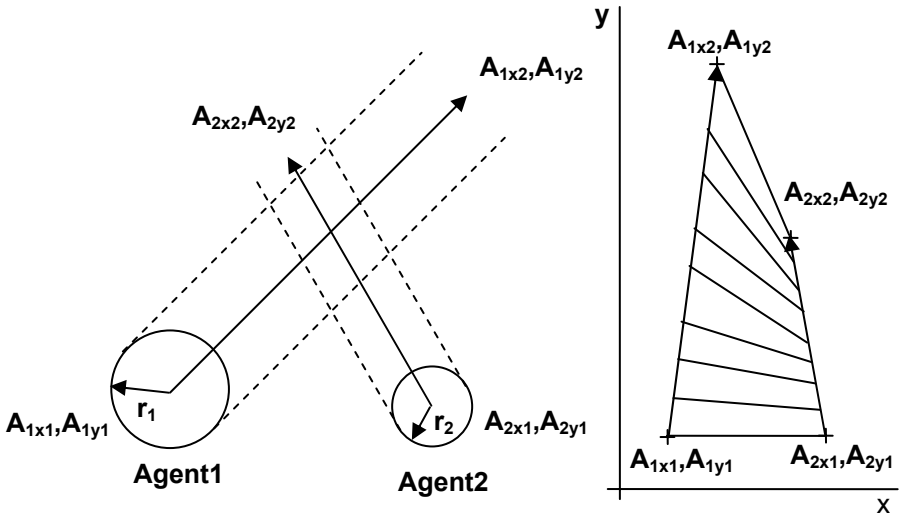
Tato kapitola je věnována detekcí kolizí vrstvou (viz 3.3.1), výsledku kolize (viz 3.3.2), a také druhům implementovaných vrstev (viz 3.3.3).

3.3.1 Detekce kolizí

Kolize je jeden z prvků, který v simulačních aplikacích bývá opomíjen. Protože naším cílem je vývoj softwarových prostředků pro mobilní roboty, je řešení kolizí klíčové. V navrženém algoritmu se snažím nejen informovat agenty o srážce s jiným agentem, ale i brát v potaz velikost a sílu agenta a podle toho určit výsledek kolize.

Je zde velmi důležité rozlišit mezi detekcí kolize z pohledu agenta a detekcí kolize z hlediska prostředí. Zde navržený algoritmus se zabývá detekcí kolize z hlediska prostředí. Jelikož prostředí je simulované, je nutné kolizi detekovat a poskytnout agentům informaci o tom, že ke kolizi došlo. V reálných aplikacích není nutné tento problém řešit, v nich je nutné řešit kolizi z pohledu agenta.

Základní a velmi jednoduchý algoritmus k detekci kolizí navrhl kolega Jára Altmann [44]. Tento algoritmus počítá s agenty kruhového tvaru s různými poloměry a různými rychlostmi. Vychází z myšlenky že pohyb obou agentů v rámci jednoho simulačního kroku se děje v čase t = <0,1> po úsečce. Za tento čas urazí agenti určitou dráhu danou rychlostí, jakou se dokáží pohybovat. Celou situaci demonstruje Obrázek č. 7. V levé části je zobrazena situace kolize agentů. Velikost vektorů vyjadřuje jejich rychlost. V pravé části jsou v rovině x-y zobrazeny odpovídající si úseky z hlediska probíhajícího času.



Obrázek č. 7 – Schématické znázornění kolize (vlevo), mapování pohybu (vpravo)

Označme počáteční pozici prvního agenta jako $\mathbf{A_{1x1},A_{1y1}}$ a počáteční pozici druhého agenta jako $\mathbf{A_{2x1},A_{2y1}}$. Analogicky označme koncové pozice jako $\mathbf{A_{1x2},A_{1y2}}$ a $\mathbf{A_{2x1},A_{2y1}}$. Označme poloměry agentů $\mathbf{r_1}$ a r_2. Spočtěme směrové vektory obou agentů

$$\mathbf{A_{1sx}} = \mathbf{A_{1x2}} - \mathbf{A_{1x1}}, \quad \mathbf{A_{1sy}} = \mathbf{A_{1y2}} - \mathbf{A_{1y1}},$$
$$\mathbf{A_{2sx}} = \mathbf{A_{2x2}} - \mathbf{A_{2x1}}, \quad \mathbf{A_{2sy}} = \mathbf{A_{2y2}} - \mathbf{A_{2y1}}.$$

Z nich vypočtěme rozdíly směrových vektorů $\mathbf{d_{sx}},\mathbf{d_{sy}}$ a rozdíly počátků $\mathbf{d_x}$ $\mathbf{d_y}$.

$$\mathbf{d_{sx}} = \mathbf{A_{2sx}} - \mathbf{A_{1sx}}, \quad \mathbf{d_{sy}} = \mathbf{A_{2sy}} - \mathbf{A_{1sy}}$$
$$\mathbf{d_x} = \mathbf{A_{2x1}} - \mathbf{A_{1x1}}, \quad \mathbf{d_y} = \mathbf{A_{2y1}} - \mathbf{A_{1y1}}.$$

Čas dotyku obou vektorů je dán řešením kvadratické rovnice

$$\left(\mathbf{d_{sx}^2} + \mathbf{d_{sy}^2}\right)\cdot \mathbf{t}^2 + \left(\mathbf{d_{sx}} \cdot \mathbf{d_x} + \mathbf{d_{sy}} \cdot \mathbf{d_y}\right)\cdot \mathbf{t} + \mathbf{d_x^2} + \mathbf{d_y^2} = 0.$$

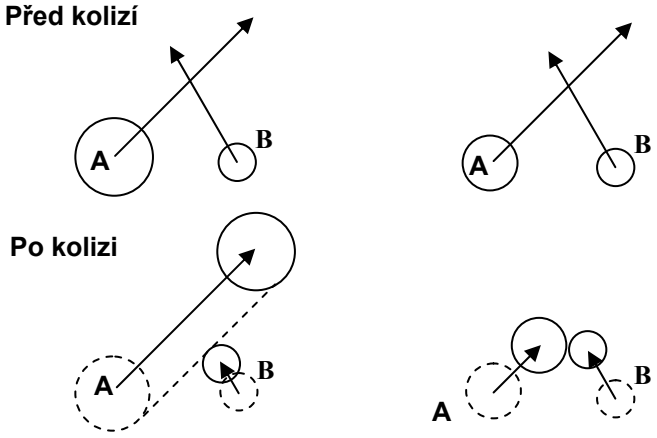
Tato rovnice počítá dobu střetu středů kruhů, tedy dobu po jaké dojde k průsečíku směrových vektorů. Je tedy nutné ji korigovat o poloměry obou agentů. Výsledná rovnice pro výpočet doby kolize

$$\left(\mathbf{d}_{\mathbf{sx}}^2 + \mathbf{d}_{\mathbf{sy}}^2\right)\cdot \mathbf{t}^2 + \left(\mathbf{d}_{\mathbf{sx}}\cdot \mathbf{d}_{\mathbf{x}} + \mathbf{d}_{\mathbf{sy}}\cdot \mathbf{d}_{\mathbf{y}}\right)\cdot \mathbf{t} + \mathbf{d}_{\mathbf{x}}^2 + \mathbf{d}_{\mathbf{y}}^2 - \left(\mathbf{r}_1 + \mathbf{r}_2\right)^2 = 0\,.$$

Tento algoritmus správně vyhodnotí i případ, kdy k průsečíku vektorů dojde, ale první agent by již urazil takovou dráhu, že ke kolizi nedojde. Výchozí situace zpracovaná tímto algoritmem musí být bez kolize. Pokud by vlivem špatného rozmístění agentů při inicializaci došlo ke kolizi, nebudou se moci agenti během celé simulace pohybovat.

3.3.2 Dopad kolize na agenty

Známe-li čas kolize, lze snadno dopočítat místo kolize. Problémem, který je nyní potřeba vyřešit, je rozmístění agentů po kolizi. Zde jsem stál před otázkou zda zavést agentům hmotnosti a hybnosti a na fyzikálním základě dle jejich kinetických energií určit srážku. To by dalo simulaci věrnější podání. Znamenalo by to přidat agentovy další vlastnosti a implementovat některé fyzikální zákony. Jelikož touto prací prostředí **WAL** teprve vzniká, rozhodl jsem se tuto problematiku obejít pomocí několika předdefinovaných výsledků. Ty jsem navrhl podle dvou atributů agentů. Jedním z nich je agentova velikost (předpokládám přímou úměru s hmotností) a druhým je agentova síla. Pokud jeden z agentů má některý parametr významně vyšší než druhý agent (více než dvojnásobný), pak je tomuto agentovi dovoleno dokončit jeho plánovanou trajektorii a druhý agent je umístěn tak, aby umožnil druhému projít. Pokud jsou oba agenti srovnatelní, ani jeden z nich nedokončí svůj pohyb a zůstanou v místě kolize (viz. Obrázek č. 8).



Obrázek č. 8 – Výsledek kolizí při různých parametrech agentů

Algoritmus výpočtu časů kolize je implementován v balíčku ***environment.collisions***. Jeho součástí jsou tři třídy ***Collision.java*** obsahující algoritmus výpočtu času dotyku agentů. ***CollisionDemo.java*** je spustitelná aplikace demonstrující funkci detekce kolizí. ***CollisionFrame.java*** definující grafické okno pro demonstraci.

3.3.3 Typy implementovaných vrstev

Ve své práci jsem implementoval tři vrstvy. Je to vrstva fyzická, tepelná a atmosférická. Tyto vrstvy postačily pro základní pokusy a ověření funkčnosti návrhu.

Fyzická vrstva slouží k uchování pozice agentů a také prostoru zabraného agenty v tomto prostředí. Toto je také jediná vrstva, ve které zatím byl implementován systém kolizí.

Tepelná vrstva byla implementována jako jediná gradientní vrstva. Je však řešena velmi jednoduše. Od detailnější implementace tepelné vrstvy byla z časových důvodů a po dohodě s vedoucím diplomové práce upuštěno.

Atmosférická vrstva slouží k uchování informace o denním čase. V budoucnu může uchovávat i informace o tlaku a počasí. A budou tedy možné simulace, kdy se agenty budou schovávat před deštěm.

3.4 Vizualizace simulačního prostředí

Tato kapitola je věnována otázce vizualizace světa agentů. Nejprve jsou rozebrány přínosy vizualizace ve třech rozměrech (3.4.1). Dále je uvedeno krátké seznámení s technologií VRML (3.4.2). Poté je popsána knihovna vytvořených modelů (3.4.3). Jako poslední je uvedena kapitola shrnující obtíže vizualizace právě technologií VRML a specifikuje podmínky za kterých je tato vizualizace možná (3.4.4).

3.4.1 Přínosy vizualizace světa ve 3D oproti 2D

Pro výzkum algoritmů chování, učení a další z oblasti umělého života není 3D svět nezbytně nutný. Jakou přidanou hodnotu může mít převod do třetího rozměru? **WALA^2** plně třetí rozměr podporuje, i když já jsem ho v mé práci neimplementoval. Nic však nestojí v cestě i 2D prostor vizualizovat ve 3D a tím udělat první krok. Touto cestou jsem se rozhodl vydat. Vedlo mě k tomu několik důvodů.

Cílem **WALA^2** je vytvořit simulační aplikaci tak, aby bylo možné, „napojit “ na simulaci jakoukoliv vizualizaci, ať už 2D nebo 3D. Dalším podstatným atributem práce je také oddělení umělo-inteligentního enginu od grafického. Na základě použitého interface by pak bylo možné vizualizovat svět jakýmkoliv způsobem, bez ohledu na engine a bez nutnosti zásahů do něj. Navíc by změna vizualizace, analýzy a sledování průběhu simulace byla velmi jednoduchá. Toto jsem chtěl ověřit a napojit na simulaci ve 2D prostředí 3D vizualizaci.

Dalším plusem vizualizace ve 3D je možnost sledovat simulaci očima samotného agenta. Na základě toho vyhodnotit jeho chování. Zde by byla nutná spolupráce s lidmi z přírodovědeckých oborů. Jelikož my jako lidé často hodnotíme chování agentů z našeho lidského hlediska, s uvažováním všech našich lidských možností. Často se nemůžeme vcítit do kůže agenta a tak uznat jeho chování z jeho hlediska za dobré.

Další výhodou 3D vizualizace by byla možnost tvorby avatara. Avatar je termín používaný v oboru počítačové grafiky pro reprezentaci uživatele ve virtuálním světe. Tento by pak mohl interagovat s agenty přímo v jejich prostředí nebo je tam jen pasivně pozorovat a analyzovat chování. Jak by mohl aktivně zasahovat? Jednou z možností je jen pokročilé 3D rozhraní parametrizace. Druhou by byla i interakce s agenty, která by umožňovala vystupovat jako učitel (učit je například za pomoci imitace).

To jsou jen plány do budoucna, které se mi jeví jako nesmírně zajímavé. Tedy s nimi v návrhu abstraktní architektury počítám.

3.4.2 Technologie VRML

VRML (Virtual Reality Modeling Language) je otevřený, rozšiřitelný průmyslový standard jazyka pro popis 3D scény nebo světa na internetu [30]. S VRML pluginem do některého z internetových prohlížečů (Internet Explorer, Netscape, Mozila atd.) lze snadno prohlížet distribuované interaktivní 3D světy, ve kterých nechybí text, obrázky, animace, zvuky, hudba a dokonce ani video. VRML 2.0 (viz [31], [32], [P09]) podporuje komplexní 3D animace a simulace a je schopno spolupracovat s JavaScriptem, nebo Javou. VRML je navrženo k použití na internetu, ale lze použít i na intranetu nebo na lokálním počítači. VRML se také snaží být univerzálním výměnným formátem pro 3D grafiku a multimedia.

Proč jsem se rozhodl právě pro VRML? Je to velmi jednoduchý standard pro popis 3D světů. Tedy není složité se jej naučit používat. Má možnost být řízen přes Javascript, nebo přes speciální rozhraní nazývané EAI (*External Authoring Interface*) programem napsaným v jazyce Java. Tedy pro případ aplikace vyvíjené v jazyce Java se mi jevil způsob vizualizace pomocí VRML jednoduchý a přirozený. Největší výhody vidím v přístupu k simulaci z internetového prohlížeče. Tedy jak už jsem nastínil výše, na výkonném serveru by běžela samotná simulace a klientovi připojenému pomocí kombinace Java apletu a VRML prohlížeče by se zobrazoval stav simulace. ***Aplet*** je aplikace napsaná v jazyce Java používaná na www stránkách, tedy nespouštějí se jako aplikace ale otevřením www stránky prohlížečem, který umí s aplety pracovat [4]. Uživatel by mohl tímto světem procházet a pozorovat chování agentů. Další obecnější výhody 3D vizualizace byli již zmíněny výše.

3.4.3 Vytvořená knihovna VRML modelů

Vytvořil jsem malou knihovnu modelů pro účely vizualizace **WAL**. Tato knihovna se nachází v adresáři *vizu*. Pro agenty jsou připraveny tři modely živočichů a to mravenec (***ant***), pavouk (***spider***) a škorpión (***scorpion***). Výběr grafické reprezentace agenta lze

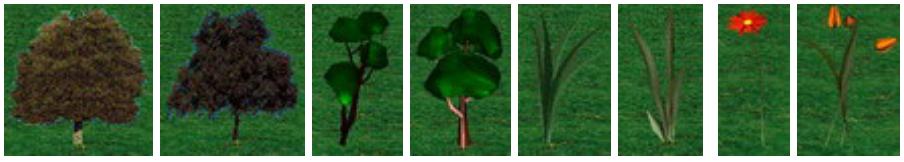
provést interaktivně v konfiguračním programu, či v konfiguračním souboru (více viz. 3.6), tyto modely jsou v adresáři ***vizu/animals***. Obrázek č. 9 ukazuje zmíněné modely.

Obrázek č. 9 – Grafické reprezentace agentů-animátů

Pro ostatní objekty jsem vytvořil několik jednoduchých modelů symbolizujících jídlo a zdroje vody. Ty se nacházejí v adresářích ***vizu/food*** pro jídlo a ***vizu/misc*** pro zdroje vody. Tyto modely zobrazuje Obrázek č. 10.

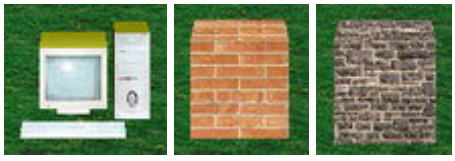
Obrázek č. 10 – Grafické reprezentace zdrojů jídla a vody

Dále jsem vytvořil modely několika přírodních prvků jako jsou květiny a stromy, sloužící jako zpestření prostředí, ty se nacházejí v adresářích ***vizu/trees*** pro stromy a ***vizu/plants*** pro rostliny. Náhled na ně poskytuje Obrázek č. 11.



Obrázek č. 11 – Grafická reprezentace stromů a rostlin

Dále jsem přidal ještě několik objektů, jako počítač symbolizující zábavu pro agenty. Při interakci s tímto objektem se jim redukuje nuda. Modely se nalézají v adresáři ***vizu/misc***. Dalšími objekty jsou zdi, sloužící k omezení pohybu agentů (viz Obrázek č. 12).



Obrázek č. 12 – Grafická reprezentace dalších objektů

3.4.4 Aplikace VRML – omezení použití

Ačkoliv může VRML svět propojený s enginem vypadat působivě, je to za cenu velkých ústupků. I když je VRML ISO/IEC normou, která si vydobyla tu výsadu že je volně šiřitelná, jsou mezi implementacemi rozdíly a podpora tohoto standardu není vždy

úplná. VRML svět samotný nebo komunikující jen s apletem v internetovém prohlížeči je mocným nástrojem pro různá použití. V momentě, kdy jsem ho chtěl použít jako externí modul a dat mu posílat z enginu jsem narazil na mnoho problémů. Jako internetový prohlížeč jsem použil Internet Explorer a v něm plugin Cortona. Pro tento nejpopulárnější VRML prohlížeč, jsem se rozhodl z toho důvodu, že na rozdíl od jiných vyzkoušených ContactBS [33], Blauxxun [34], neměl problémy se zobrazením navržených modelů. V ostatních výše zmiňovaných jsem narazil na mírné odlišnosti v chování té samé scény. Tedy jsem se rozhodl pro Cortonu.

Další problém nastal při pokusu EAI apletu o komunikaci s serverem enginu. Díky bezpečnostním omezením vztahujícím se na aplety, není možné navázat spojení se vzdáleným serverem bez podepsání apletu certifikátem. Aplet bez certifikátu nesmí:

- nahrávat knihovny či definovat nativní metody,
- číst a zapisovat do souborů na straně klienta,
- navazovat síťová spojení s výjimkou svého domovského serveru,
- ukončit práci virtuálního stroje,
- spouštět další programy na straně klienta,
- modifikovat systémové proměnné a některé z nich nemůže ani číst (tzv. citlivé proměnné),
- manipulovat se soubory na lokálním disku.

Samotné podepsání a vytvoření práv apletu pro mě nebylo problémem. Ale s podepsanými aplety umí pracovat pouze novější verze interpreteru JVM (*Java Virtual Machine*). Bohužel tyto nové verze JVM nepodporují EAI rozhraní Cortony. Toto rozhraní podporuje naopak Microsoft VM. Tedy jsem se ocitl v neřešitelné situaci. Abych mohl zprovoznit EAI rozhraní potřeboval jsem Microsoft VM, ale pro komunikaci s okolím zase JavaVM (od firmy Sun).

Tento problém se mi nakonec podařilo obejít, jednou výjimkou v omezeních apletů. Ten může otevřít bez omezení komunikační kanál na server na kterém se nachází, to v žádném případě nemůže ohrozit uživatele a tudíž je to povoleno. Tedy k tomu abych mohl úspěšně komunikovat mezi enginem a VRML modulem vizualizace je nutné spustit internetovou stránku s vizualizací z lokálního serveru. Za tímto účelem jsem nainstaloval volně šiřitelný webový server Apache. Bohužel Microsoft VM je velmi zastaralá, jelikož tato firma ji již několik let nevyvíjí. Tudíž jsem měl funkční verzi vizualizačního modulu, ale některé třídy, které je dnes běžné používat při programování v jazyku Java bohužel tato VM nezná. Z tohoto důvodu bylo nutné pro komunikaci volit velmi jednoduché prostředky a konstrukce jazyka.

Z výše uvedeného vyplývají specifické a mnohdy nepříjemné podmínky pro provoz takového vizualizačního modulu. Proto nemohu tento postup doporučit do budoucna. Pokud by byl dále zájem udržet tuto vizualizaci při životě, existují ještě další cesty zprovoznění VRML. Jedna z nich je použít VRML přímo v okně aplikace pomocí 3D knihovny jazyka Java nazvané Java3D a přídavných knihoven pro práci s VRML Xj3D.

Malá ukázka této spolupráce je v konfiguračním nástroji pro agenta při výběru grafické reprezentace agenta. Samotná inicializace grafického enginu Java3D trvá několik sekund. Knihovna Xj3D je ve vývoji a tudíž není bez chyb. Aplikace se s ní stává nestabilní a velmi pomalou (viz. Tabulka č. 3).

Další možností jak pokračovat ve VRML vizualizaci je vyčkat na dokončení nového formátu X3D, které je přímým následníkem VRML.

Na závěr této kapitoly bych doporučil pro externí modul jazyka VRML nepoužívat. Vizualizace v internetovém prohlížeči se mi jevila jako velmi přirozená a doufal jsem v snadné prohlížení dění ve světe agentů-animátů. Bohužel moje představa se nenaplnila a jak jsem popsal není možné z internetového prohlížeče se připojit k enginu. Vše běží pouze na lokálním počítači.

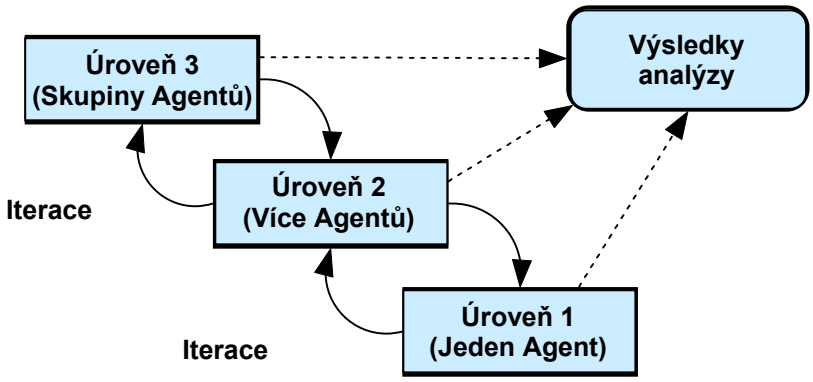
Zdrojové kódy vizualizačního modulu se nachází v balíčku ***vizu***. Jeho součástí jsou třídy ***WalVizualization.java, InfoPanel.java***, ***Client.java*** a ***SimpleAgent.java***. Kde první jmenovaný je apletem, lze ho spustit na webové stránce a zajišťuje zasílání a získávání událostí z VRML světa. Druhý popisuje rozložení grafických a ovládacích prvků. Třetí jmenovaný soubor obstarává příjem zpráv od enginu a následnou modifikaci scény dle přijatých údajů. Poslední jmenovaný soubor slouží ke zpětné rekonstrukci přijatých dat.

3.5 Off-line analýza simulace

Simulace umělého života se stávají čím dál více složitější a tudíž je i těžší pro jejich autory poznat, že simulace probíhá dle jejich představ, nebo tak jak ji navrhli. Proto je nutné spolu s rostoucí komplexitou prostředí používat odpovídající nástroje umožňující zobrazit velké množství parametrů v čase zároveň. Tato potřeba se projevila zhruba před rokem a dala vzniknou nástroji nazvanému VAT (*Visual Analysis Tool*).

3.5.1 Vizuální nástroj pro analýzu – VAT

Tento nástroj byl vytvořen Ing. Davidem Řehořem na žádost naší skupiny. Mým přínosem byla aktivní účast na návrhu vizualizačních nástrojů a konzultace. Nástroj byl autorem nazván Visual Analysis Tools© a publikován na několika světových konferencích (detaily možno zjistit v [13], [14], [15]. Nástroj VAT je dělen dle úrovně podle požadovaného stupně detailu. Blokové schéma jednotlivých úrovní VAT ukazuje Obrázek č. 13. Se svolením autora těchto nástrojů jsem je všechny umístil jako přílohu na CD. Do adresáře ***/Other/VAT*** (viz [P10], [P11], [P12], [P13], [P14]).

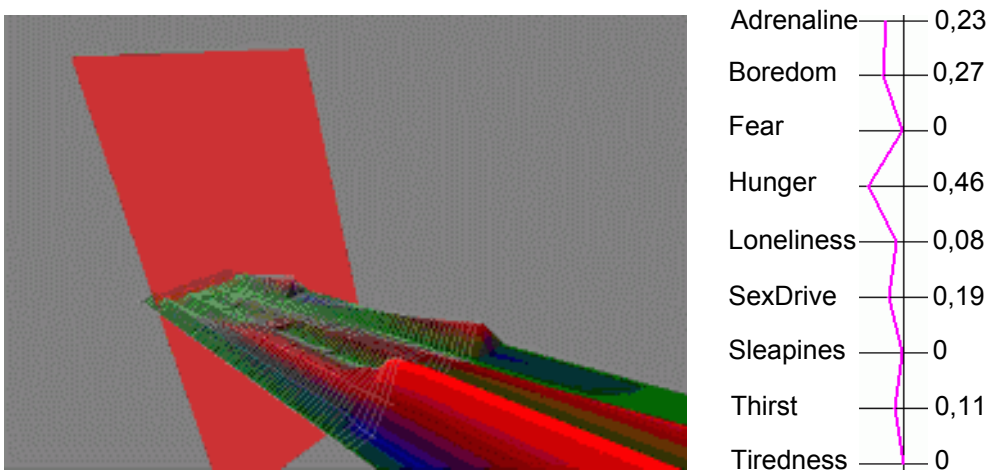


Obrázek č. 13 – Úrovně VAT

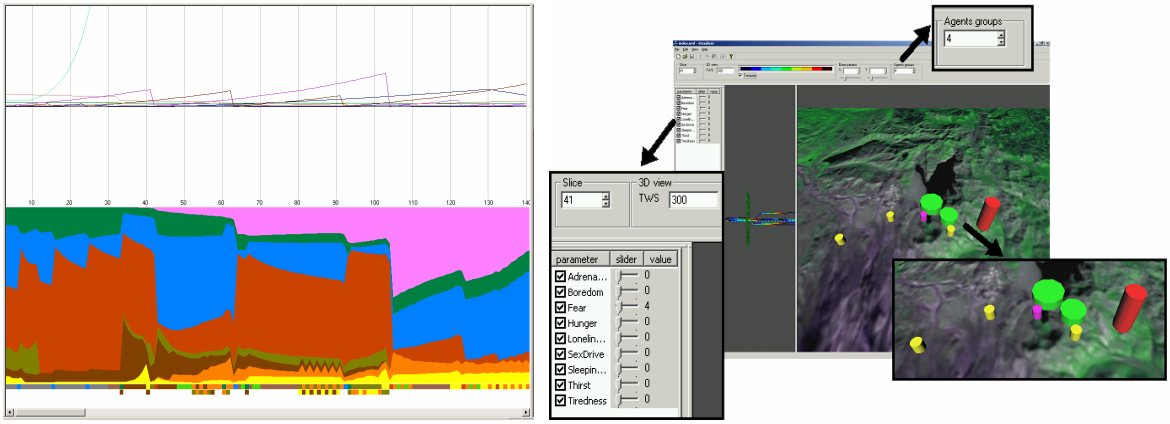
Výhody tohoto přístupu jsou:

- rychlý náhled na komplexní a složitou situaci simulace,
- několik úrovní detailu zpřehledňuje analýzu,
- off-line analýza umožňuje zpětnou analýzu kritických událostí,
- zobrazení vztahů mezi vnějšími stimuly a vnitřními stavy.

Úroveň 1 (VAT1) umožňuje zobrazení parametrů agenta v čase, agentův rozhodovací mechanismu (ASM) a vzájemnou korespondenci mezi vnitřním stavem agenta a výběrem akce. Umožňuje taktéž zobrazit inflexní body. Slouží tedy k analýze vztahů mezi vnitřním stavem agenta a spuštěnými akcemi. Sleduje agentovo chování a jeho rozhodnutí v daných situacích. Ve VAT úrovně jedna je využita technika paralelních souřadnic k zobrazení vnitřního stavu agenta (viz Obrázek č. 14). Tato vizualizační technika klade paralelně osy jednotlivých parametrů. Na ně vynáší jejich velikost. Přidáme-li čas, jako třetí rozměr dostaneme zakřivenou plochu. K zobrazení preferencí akcí agenta je použita barevná mapa (viz. Obrázek č. 15). Jednotlivým agentovým akcím jsou přiřazeny barvy. Velikost preference k akci je vyjádřena zastoupením příslušné barvy ve sloupci. Tento sloupec složený z barev reprezentuje jeden časový okamžik. Poskládáním těchto sloupců vedle seb, vzniká průběh preferovaných a spuštěných akcí v čase. Úrovně 2 a 3 (VAT2 a VAT3) slouží k zobrazení parametrů celé skupiny agentů. K dispozici dává nástroje jako možnost shlukování agentů podle jejich vlastností nebo citlivostní analýzu. Agent je v jednom pohledu reprezentován křivkou v prostoru, reprezentující jeho pozici v čase. V druhém pohledu je reprezentován jako válec. Při citlivostní analýze lze mapovat parametry na poloměr nebo výšku válce a také jako barvu křivky v prostoru. Barva válce odpovídá skupině agenta, do které byl zařazen algoritmem shlukování. S využitím těchto nástrojů lze zkoumat zajímavé skupiny agentů na různých úrovních granularity. VAT2 a VAT3 jsou vyššími analytickými nástroji pro náhled na komunitu agentů.



Obrázek č. 14 – 3D a 2D pohled na paralelní souřadnice



Obrázek č. 15 – Barevná mapa (vlevo), ukázka VAT úrovně 3 (vpravo)

Tyto vizualizační nástroje včetně několika ukázkových souborů s daty jsou přiloženy na CD (viz [P10], [P11], [P12], [P13], [P14]).

3.6 Parametrizace simulovaného světa

Tato kapitola popisuje jak jsem řešil parametrizaci prostředí i samotných agentů. Pokud mají být výsledky průkazné, je nutno zajistit opakovatelnost simulace. Prostředí je nutno nejprve zinicializovat a nadefinovat velikost, je nutné rozmístit agenty, nadefinovat jejich velikost a další. Všechny tyto úkony jsou popsány v této kapitole. Část 3.6.1 se zabývá jazykem konfiguračního souboru. Část 3.6.2 na příkladu demonstruje jak konfigurační soubor vypadá.

3.6.1 Konfigurace enginu v jazyce XML

Jako jazyk konfiguračního souboru jsem zvolil XML [26]. Aby se zajistila kontrola konfiguračního souboru, je nutné zavést při načítání konfigurace kontrolu struktury XML

dokumentu pomocí DTD (*Document type definition*), nebo nověji pomocí XSD (*XML schema definition*). Obě technologie umožňují definovat povolené značky (*tags*) v konfiguračním souboru, přičemž novější standard XML schémat (XSD) jde ještě dále a dovoluje definovat i kvantitativní omezení. Tímto tedy necháváme volně přístupnou konfiguraci prostředí, agentů a všech objektů simulace, avšak použitím DTD nebo XSD zajistíme, že aplikace bude tomuto konfiguračnímu souboru rozumět a bude umět ho správně interpretovat. Dokument, který projde kontrolou vůči příslušnému DTD nebo XSD se nazývá validní (*valid*). Dokument, který dodržuje zásady pro psaní XML dokumentu předepsané konsorciem W3C se nazývá dobře formovaným (*well-formed*) dokumentem.

K práci s XML lze použít dvě odlišné parsovací technologie. Událostně orientované API a stromově orientované API.

Událostně orientované posílá zprávy z parsování (jako například start a konec elementu atd.) přímo aplikaci. Tedy nestaví žádnou vnitřní reprezentaci dokumentu. Aplikace zachází s těmito událostmi stejně jako události z GUI, tedy není nutné ukládat dokument do paměti. Navíc je možné pomocí událostně orientovaného API vytvořit strom a poté s tímto stromem pracovat. Aplikace implementuje ovladače, které pracují s těmi událostmi. SAX je jedním z takových API.

Stromově orientované API mapuje XML dokument na vnitřní stromovou reprezentaci, potom je v aplikaci možná navigace v tomto stromě. Dokument Object Module (DOM) spravované konsorciem W3C doporučuje stromové API pro XML a HTML dokumenty. Stromově orientované API jsou použitelné v širokém spektru aplikací, ale velmi zatěžují systém, zvláště pro velké dokumenty (> 1MB). V některých aplikacích je potřeba stavět vlastní reprezentaci dokumentu. V tomto případě je nešikovné tvořit nejprve stromovou strukturu (implicitně pomocí DOM) jen proto, bychom ji mapovali do struktury jiné.

Pro potřeby úvodní inicializace simulace jsme zvolil DOM přístup pomocí parseru jDOM [P21], kdy inicializační soubory jsou řádově 10kB a tedy tvorba jejich stromové reprezentace pomocí DOM nezatěžuje příliš systém. Navíc se lze v této stromové reprezentaci mnohem lépe pohybovat a vyhledávat než při použití SAX.

3.6.2 Konfigurace světa při spuštění enginu

Konfigurační soubor světa je očekáván v souboru ***DefaultWorld.xml***, pokud není uvedeno jinak v parametrech spouštěné aplikace. Konfigurační soubor je díky zápisu v XML snadno pochopitelný. Zjednodušený konfigurační soubor přibližuje Obrázek č. 16.

```
<?xml version="1.0" encoding="UTF-8"?>
<World xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
DefaultWorld.xsd">
    <WorldName>Virtual World</WorldName>
    <Type>Normal</Type>
    <WorldSize>
        <X>20</X>
        <Y>20</Y>
        <Z>1</Z>
    </WorldSize>
    <Layer>
        <LayerName>Physical</LayerName>
        <Type>Point</Type>
        <Description>Layer containing agents positions in space</Description>
        <Agent>
            <Name ID="Agent1">Smith</Name>
            <Translation>
                <X>3</X>
                <Y>6</Y>
                <Z>0</Z>
            </Translation>
        </Agent>
        <Agent>
            <Name ID="Foodl1">Apple</Name>
            <Translation>
                <X>2</X>
                <Y>9</Y>
                <Z>0</Z>
            </Translation>
        </Agent>
        <Agent>
            <Name ID="Tree1">Oak</Name>
            <Translation>
                <X>13</X>
                <Y>4</Y>
                <Z>0</Z>
            </Translation>
        </Agent>
    </Layer>
    <Layer>
        <LayerName>Termal</LayerName>
        <Type>Gradient</Type>
        <Description>Layer containing teperature in space</Description>
        <Agent>
            <Name ID="Agent1">Agatha</Name>
        </Agent>
    </Layer>
</World>
```

Obrázek č. 16 – Ukázka konfiguračního souboru světa

I když je jazyk XML velmi jednoduchý, a při vhodné a správné volbě značek je i samovysvětlující a intuitivní, nemusí každému jeho použití vyhovovat. Proto jsem vytvořil dva konfigurační nástroje pracující s konfiguračními soubory a interagující s uživatelem prostřednictvím grafického rozhraní. Tato možnost je příjemnější a pohodlnější. Navíc není nutné dokument kontrolovat, zda je dobře formován, nebo zda odpovídá předepsanému

DTD či XSD. Konfigurační soubor je automaticky generován a tudíž se minimalizuje výskyt chyb. Následují ukázky konfiguračních aplikací. První je konfigurační aplikace světa. Ta obsahuje definice vrstev prostředí a v nich nadefinované agenty. Druhou je konfigurační aplikace agenta. Ta umožňuje vytvoření zcela nového agenta pomocí definice jeho bloků jako je vegetativní blok, blok senzorů a blok akcí. Detailnější popis těchto aplikací a práce s nimi viz. 4.

Načítání konfigurace je implementováno v ***environment***. Jeho součástí je pouze třída ***Environment.java***. Tato třída se stará o inicializaci prostředí. Nejprve načte konfigurační soubor světa. Posléze načítá všechny konfigurační soubory agentů. Při nepřítomnosti takového souboru není možné inicializovat prostředí a aplikace končí.

3.7 WAL agent - implementace

Agent je hlavním aktérem celé simulace. Za účelem simulování chování agentů je prostředí navrhováno. Hlavní myšlenkou a novým přínosem je rozdělení agenta na dvě části. Jedna jeho část interaguje s prostředím a objekty v prostředí. Je součástí prostředí a je nazývána agentovo tělo. Druhá část agenta řídí, stará se o rozhodování, plánování a uvažování. Tuto část nazývám agentova mysl. Obě tyto části dohromady tvoří agenta takového jak je znám [3], [9], [16]. Toto rozdělení agenta jsem zavedl z několika důvodů. Jedním z nich je možnost mít mysl lokalizovánu na jiném počítači než engine. Dalším důvodem je přiblížení se více reálné situaci. Agentova mysl by neměla mít přístup k datům těla přímo, ale tělo by tyto hodnoty mělo filtrovat a teprve potom je poskytnout mysli. Uvedu příklad. Pokud agent má na svých senzorech objekt, je v záznamu senzoru uvedena poloha tohoto objektu v x,y souřadnicích. Nechceme však, aby agentova mysl znala tyto hodnoty přesně. Proto můžeme polohu x,y filtrovat takovým způsobem, že se mysl jen dozví, zda je objekt blízko nebo daleko, popřípadě hrubý odhad vzdálenosti.

3.7.1 Prostředky genetického programování pro křížení agentů

Modul genetických algoritmů je součástí enginu. Je výsledkem semestrální práce kolegy Jana Svobody [45]. Umožňuje simulaci reálného předávání genetických informací. Implementuje pouze dvě základní genetické operace a to křížení a mutaci. Výběr vhodných rodičů nového agenta je pak ponechán na „přirozeném výběru“, tzn. že šanci rozmnožit se mají jen ti nejsilnější agenti (tj. takoví agenti, kteří přežili dostatečně dlouho, aby mohli dospět, neboli prokázali schopnost přežít v daném prostředí).

Nevýhoda tohoto způsobu výběru se ukáže v okamžiku, kdy se snažíme naučit agenty určitému chování, které nemá přímo vliv na schopnost agenta přežít. Způsob jak tuto nevýhodu eliminovat, je možno řešit v implementaci přitažlivosti jednotlivých agentů. Čím je agent schopnější chovat se podle určeného způsobu chování, tím je přitažlivější pro jiné agenty (opačného pohlaví) a má větší šanci na rozmnožení.

Jednotlivé geny jsou implementovány jako objekt integer. Tento způsob implementace genu má výhodu snadného převodu na jiné datové typy (double, boolean,

omezený integer a omezený double). A to bez výrazného zvýšení složitosti výpočtu genetických operací. Nevýhoda je pak velikost genu (i logická boolean hodnota se mapuje jako celé číslo a tedy zabírá 4 byty místo 1 bitu. O číselných typech více viz (Tabulka č. 1).

Základní rekombinační operace poskytované modulem jsou ***křížení*** a ***mutace***. Tento operátor kombinuje genomy *n* rodičů (n pohlavní rozmnožování) a vytváří z nich nový genom.

Křížení lze rozdělit dle hloubky křížení na ***binární***, kdy nejmenší část se kterou operujeme je bit a ***hodnotové***, kdy nejmenší část se kterou operujeme je celé číslo typu integer.

Křížení lze též rozdělit podle typu na ***n-bodové*** a ***dle masky***. První zmiňované kříží genom v *n* bodech, přičemž genomy rodičů se uplatní v pořadí, v jakém byly zadány. Křížení dle masky vytvoří genom o délce rovné délce genomu prvního rodiče tak, že každý nejmenší kus genomu (bit nebo integer) náhodně převezme od jednoho rodiče.

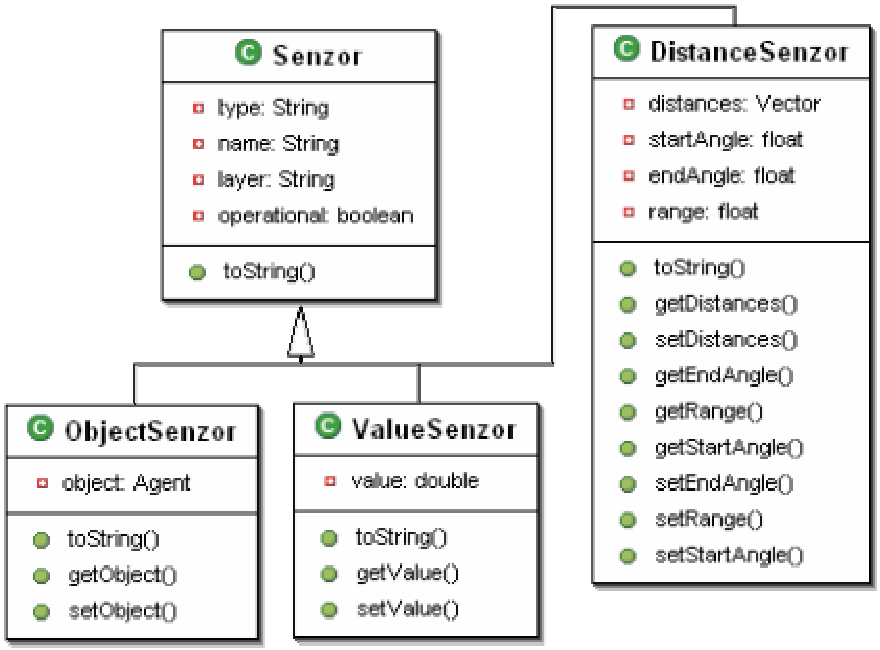
Při provádění operace ***mutace*** je na výběr z těchto možností ***bez mutace, binární mutace*** a ***celočíselná mutace***.

Převodní funkce pro převod genotypu na fenotyp a naopak závisí na architektuře agenta a také na tom, jaké informace se budou do genomu agenta ukládat. Dle toho se budou se tyto operace v různých implementacích lišit.

Výše uvedené funkce jsou součástí balíčku ***genetic***. ***Conversion.java*** obsahuje metody pro konverzi datových typů. Jak již bylo uvedeno výše, gen je reprezentován celočíselnou hodnotou, proto jsou nutné převody mezi touto hodnotou a informací kterou chceme zakódovat do genu. ***GeneticOperation.java*** obsahuje metody pro genetické operace popsané výše. ***Base.java*** je spustitelná aplikace s ukázkou chování genetických algoritmů. ***Frame.java*** definuje podobu ukázkové aplikace.

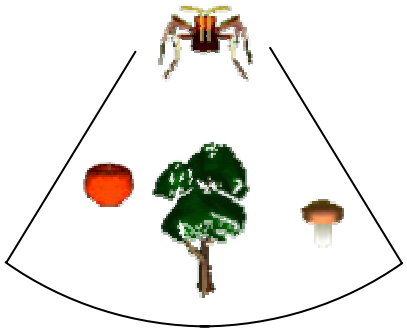
3.8 Rozhraní agentovo tělo-vrstvy, senzory

Pro vnímání okolního světa jsem navrhl pro agenta několik senzorů. Tyto senzory všechny vychází ze základní třídy ***Senzor*** (*interface*). Z této abstraktní třídy byli vytvořeny tři druhy senzorů. tato abstraktní třída pouze obsahuje informace o typu a názvu (identifikátoru) senzoru. UML diagram tříd vytvořených senzorů je ukazuje Obrázek č. 17.



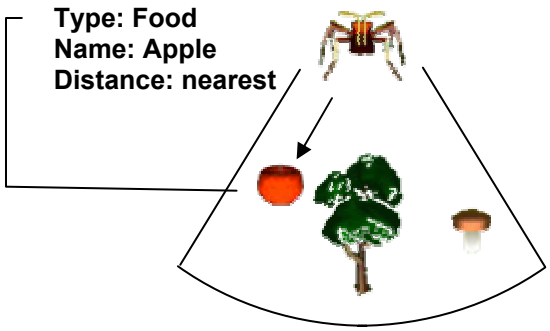
Obrázek č. 17 – UML Diagram navržených senzorů

Prvním je senzor vzdáleností objektů. Jak jsem již uvedl v rozboru, není žádoucí zabývat se otázkou rozpoznávání, tedy agentovy jsou data o přítomnosti objektů v dosahu senzoru přístupná. Tento senzor po naplnění obsahuje seznam všech objektů v jeho dosahu. Dosah tohoto senzoru není omezen pouze poloměrem, ale také úhlem (viz Obrázek č. 18). Tedy lze agentovy poskytnout jen informace o objektech v určité výseči. Počáteční i koncový úhel jsou počítány od směru agenta v radiánech. Kladný směr je tak jak je zvykem proti směru hodinových ručiček. Mohla by vzniknout námitka, že senzor by měl mít na počátku iniciován svůj dosah a tento by neměl jít v průběhu simulace měnit. Tato změna je však připuštěna pro potřeby simulace poškození senzoru. V takové situaci by byla potřeba změnit nastavení dosahu senzoru a proto je umožněno. Všechny vyjmenované senzory lze nalézt v balíčku ***environment.senzors***. Jeho součástí jsou třídy výše zmíněných senzorů ***DistanceSenzor.java***, ***ValueSenzor.java***, ***ObjectSenzor.java***, ***Senzor.java*** a ***ObjectDistancePair.java***. Poslední jmenovaný soubor slouží pouze k zapouzdření informací získaných pomocí senzoru vzdálenosti. Obsahuje jak referenci na objekt který je vnímán, tak jeho polohu a vzdálenost k němu.



Obrázek č. 18 – Senzorický dosah agenta

Dalším senzorem je senzor objektů. Tento slouží ke zkoumání parametrů objektů, které je agent schopen rozlišit. Omezeným počtem senzorů objektů je řešena otázka zaměření pozornosti agenta. Údaje o objektu majícím agentovu pozornost jsou uchovány právě v tomto senzoru (viz Obrázek č. 19).

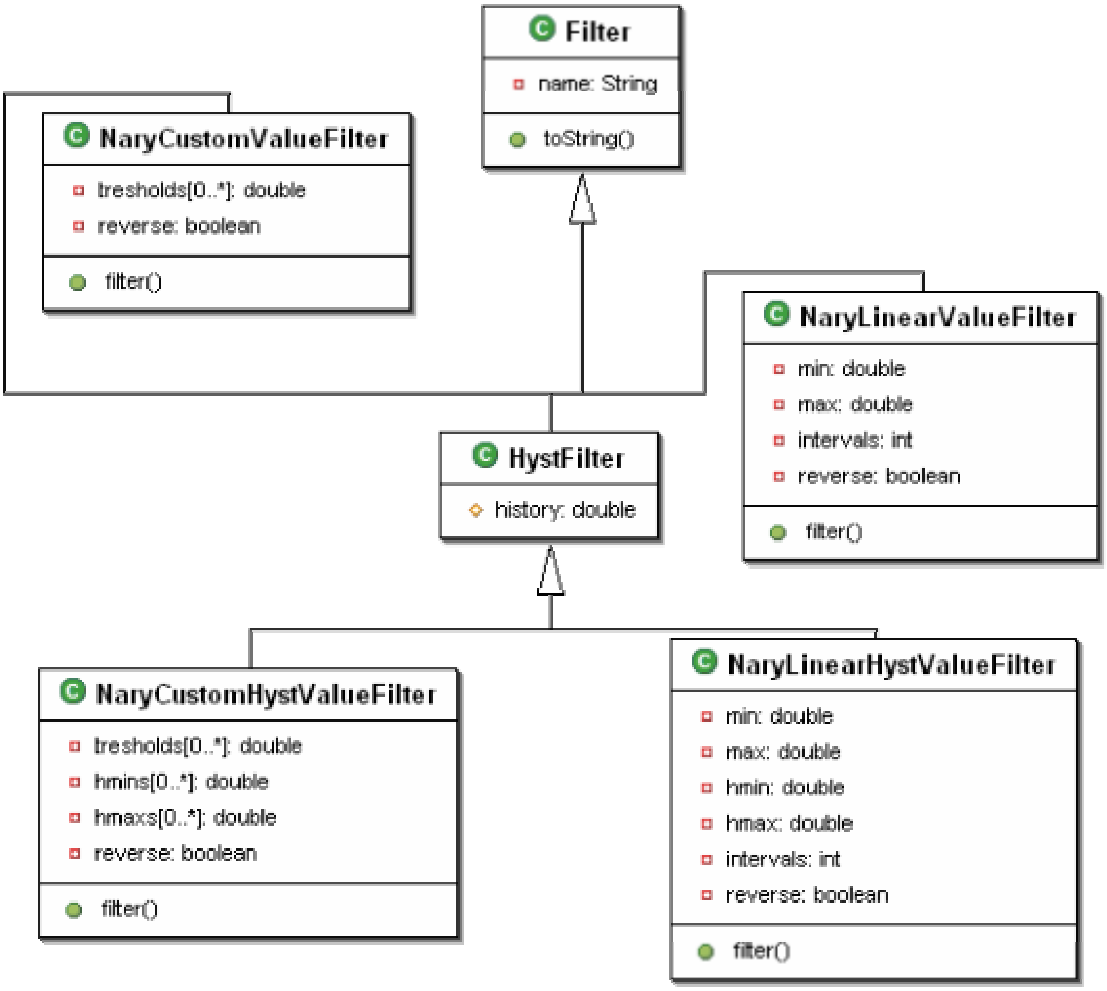


Obrázek č. 19 – Ukázka zaměření pozornosti a senzoru objektů

Posledním implementovaným typem senzoru je senzor hodnoty. Tento senzor slouží k uchování nějaké hodnoty a má dosah pouze velikosti agenta. Je tedy používán k určení hodnoty v místě, kde se agent nachází, jako je například teplota. Nebo k uchování hodnoty, která je pro celé prostředí stejná, jako je například denní doba.

3.8.1 Filtry senzorů

Mysl agenta by neměla mít přístup přímo k číselným hodnotám senzorů. To je také jedním z důvodů separace mysli agenta od jeho těla. Potom lze za senzory zařadit bloky s různými filtry, umožňující zašumět, pozměnit nebo učinit neurčitou informaci ze senzoru. Tímto postupem je docíleno větší přiblížení se realitě ve smyslu vnímání okolního světa. Vidí-li agent strom, neměla by jeho mysl mít k dispozici informaci, že strom je na pozici 10.0,6.0,0.0 a že je od něj vzdálen 3.654 metru. Tyto informace má k dispozici tělo, to je součástí prostředí. Ale mysl by měla dostat informaci například, objekt je napravo a je středně vzdálen. Tuto možnost jsem tedy implementoval pomocí návrhu několika filtrů. Navržené filtry jsou obsaženy v balíčku ***environment.filters***. ***Filter.java*** je společným předkem všech filtrů, definuje jejich základní vlastnosti. ***HystFilter.java*** je jeho rozšíření o definici základních vlastností pro filtry s hysterezí. UML diagram zobrazující dědičné vazby mezi všemi filtry ukazuje Obrázek č. 20.

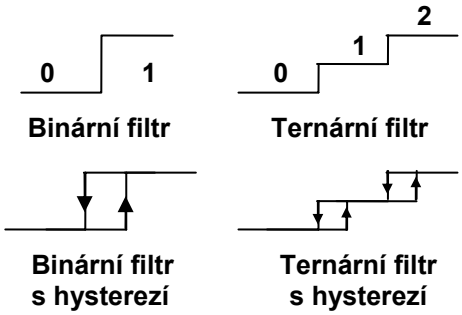


Obrázek č. 20 – UML Diagram vytvořených filtrů

Prvním filtrem je lineární n-ární filtr ***NaryLinearValueFilter.java***. Je mu zadaný úsek hodnot, které budou rozděleny na stejně velkých intervalů. Úsek hodnot se zadává pomocí minimální a maximální hodnoty. Hodnotu která je filtru předána filtr zařadí a klasifikuje pomocí celého čísla. Je možné zvolit pořadí číslování úseků. Zda nejmenší úsek bude číslován od 0 či od n.

Dalším filtrem je lineární n-ární filtr s hysterezí ***NaryLinearHystValueFilter.java***. Tento filtr je velmi podobný předchozímu. Navíc lze zadat horní a dolní mez hystereze. Filtr má tedy paměť a pamatuje si směr, kterým šli hodnoty. Podle historie vyhodnotí, zda aplikovat horní nebo dolní mez hystereze.

Po návrhu těchto filtrů jsem se zamýšlel nad implementací podobných filtrů pro kvadratické či exponenciální rozdělení. Rozhodl jsem se, že takové implementace by byli příliš svazující a tudíž jsem navrhl další filtr zcela obecný. Tomuto filtru je zadáno pole hodnot, které jsou hranicemi úseků ***NaryCustomValueFilter.java***. Tento filtr je k dispozici opět ve verzi s hysterezí ***NaryCustomHystValueFilter.java***, kdy se zadává jak pole úseků tak pole hysterezí. Příklady filtrů ukazuje Obrázek č. 21.



Obrázek č. 21 – Ukázky možných filtrů

3.9 Rozhraní agentovo tělo-vrstvy, efektory

Efektory jsou agentovým nástrojem ovlivnění prostředí. Při simulování umělého života jdou proti sobě dva požadavky. Co možná největší zjednodušení a co možná nejvěrnější kopírování skutečného chování. V případě efektorů jsem zvolil cestu jednoduchosti. Při vybrání akce myslí agenta a předání informace o vybrané akci tělu, tělu tuto akci provede jako jeden celek. Tedy každé akci odpovídá efektor. Toto je významné zjednodušení oproti skutečnosti, kdy zamýšlená akce je rozkládána na nižších vrstvách na pohyby jednotlivých částí živočicha tak, aby byla výsledkem zamýšlená akce. Tato dekompozice v simulacích není potřeba. Jsem si vědom, že pro aplikaci algoritmu chování na robota je nutné tuto úroveň řešit. Zjednodušení, které jsem zde učinil, předpokládá, že robot má již svůj základní software a tedy dokáže přijmout příkazy typů akcí, jako například posuň se dopředu nebo otoč se.

Implementované akce, které agent může povést jsou tyto rozmnožit se (***mate***), přátelit se (***hobnob***), pít (***drink***), jíst (***eat***), otočit se (***turn***), pohyb dopředu (***forward***), pohyb dozadu (***backward***), nedělat nic (***nothing***), spát (***sleep***), odpočívat (***relax***), zaútočit (***attack***), použít (***use***), sebrat předmět (***get***) a předmět opět položit (***drop***) [16]. Všechny tyto akce jsou součástí třídy ***ActuationLayer.java*** z balíčku ***agent***.

3.10 Rozhraní agentovo tělo-agentova mysl

Toto rozhraní je velmi důležité, jelikož je propojením agentova těla a agentova mozku. V této diplomové práci je rozhraní řešeno voláním metod. Rozdělení agenta a mysli vyžaduje funkční komunikační protokol pro rychlý přenos dat. Aby tento protokol byl dostatečně rychlý, je nutné přenášet binární data. Jak již jsem uvedl, binární formát dat nesouvisí přímo s tématem této diplomové práce, a tudíž jsem po dohodě s vedoucím diplomové práce od řešení tohoto problému ustoupil. Oddělení mysli od těla je dle mého názoru přínosná myšlenka, která poslouží k lepší modulárnosti a přenositelnosti celého simulačního prostředí.

3.11 Komunikace

Komunikace může být chápana dvojím způsobem. Jednak je zde problém komunikace agentů mezi sebou a jejich interakcí. Dále je tu problém komunikace s externími moduly. Tyto dva pojmy jsou diametrálně odlišné. Komunikace mezi agenty je navržena řešit přes komunikační vrstvy, konkrétní komunikační vrstva závisí na způsobu dorozumívání agentů. Komunikační vrstvou může být vrstva akustická, vrstva pachová, či vrstva elektromagnetická.

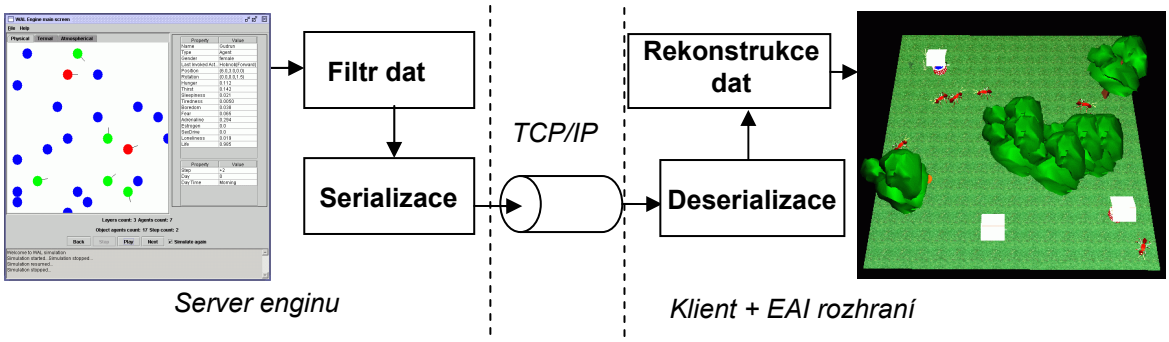
V této kapitole bych se rád věnoval druhému typu komunikace a to komunikaci s moduly a komunikačními protokoly používanými a implementovanými v enginu. Engine při svém startu spouští proces (démon) který je TCP/IP serverem. Ten čeká na spojení na portu 2004. Číslo portu jsem zvolil v souladu s rokem vzniku tohoto serveru. V okamžiku žádosti klienta o spojení na tomto portu je serverem vytvořen nový proces pro komunikaci s klientem. Po ukončení komunikace ze strany klienta je proces, který s ním komunikoval, zrušen. Server je démonem z toho důvodu, aby celý engine při skončení své funkce nečekal na dokončení běhu serveru, který neskončí protože běží v nekonečné smyčce naslouchání na portu.

3.11.1 Komunikace s VRML vizualizací

Komunikace probíhající s VRML je z výše popsaných důvodů řešena co možná nejjednodušším způsobem. Data jsou serializována standardně pro jazyk Java a posílána klientovy. Klient na své straně data přijme a deserializuje. Jelikož zná třídy, které byli poslány, dokáže data snadno přečíst.

Pro vizualizaci nejsou typicky potřeba všechna data, záleží na druhu vizualizace a co všechno se snaží zobrazit. Při zobrazení světa agentů, tak jak jsem je implementoval, jsou důležité informace o polohách a směru agentů a velikosti světa. Proto není nutné ostatní informace o agentech posílat a zbytečně tak zatěžovat komunikaci. Průběh komunikace schématicky naznačuje (Obrázek č. 22).

Třídy zajišťující komunikaci jsou umístěny v balíčku ***comm*** a ***comm.bin***. Oba balíčky obsahují stejné třídy. ***Client.java*** je klientská strana komunikace sloužící k přijímání údajů a jejich zpracování. ***Server.java*** je server, který čeká na připojení klienta, v okamžiku spěšného připojení vytvoří klientovy vlákno ***MultiServerThread.java***, které zajistí obsluhu klienta (zasílání dat). Tyto třídy v balíčku ***comm*** zasílají data serializovaná standardně jazykem Java, kdežto třídy z balíčku ***comm.bin*** zasílají data ve formě posloupnosti bytů a jsou tedy vhodné i pro posílání dat modulům, které jsou napsané v jiných programovacích jazycích.



Obrázek č. 22 – Schéma komunikace s VRML modulem

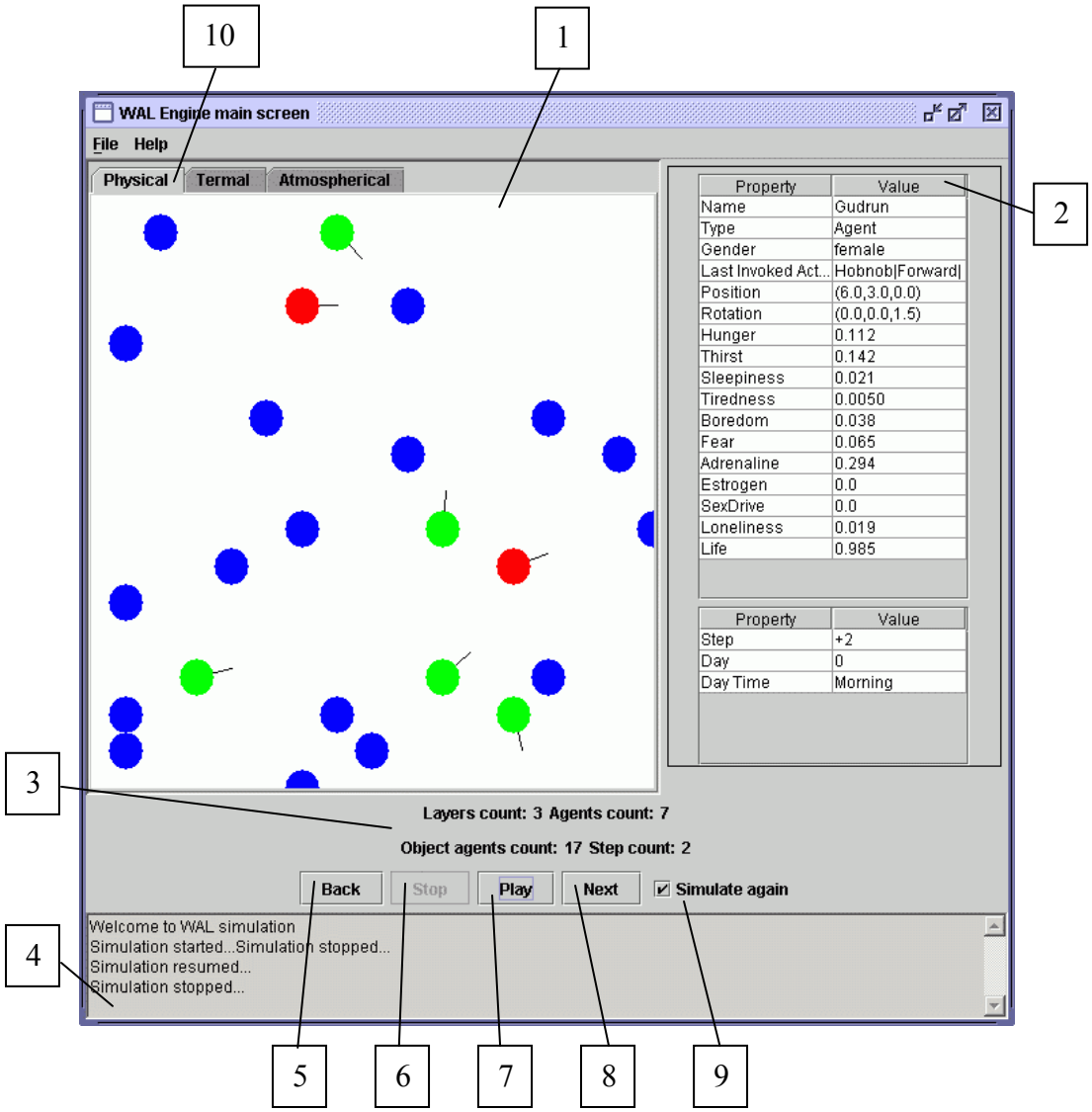
4 Uživatelský a programátorský manuál

4.1 Uživatelský manuál

Tato část se věnuje popisu navržených aplikací z hlediska uživatele. Seznamuje s funkcí a účelem ovládacích prvků, popisuje způsob navigace ve virtuálním světě a také se věnuje konfiguračním aplikacím agenta a prostředí.

4.1.1 Popis hlavního okna enginu

Engine se spouští souborem ***start.bat***. Jeho grafické rozhraní viz Obrázek č. 23.



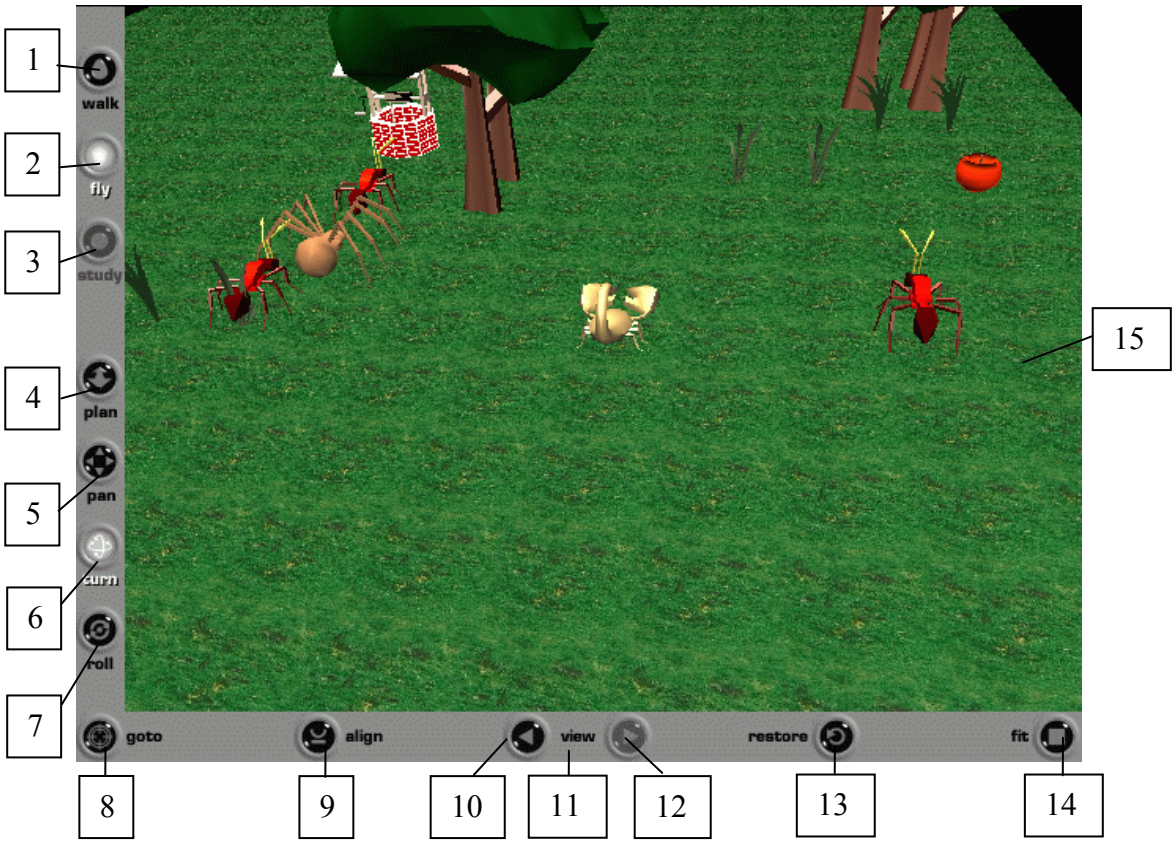
Obrázek č. 23 – Detailní pohled na okno enginu

1. ***Grafické okno*** - slouží k náhledu na průběh simulace převážně při ladění programu. Pro zobrazení situace slouží vizualizační modul (viz dále).

2. ***Panel parametrů*** - zobrazuje parametry vybraného agenta. Po kliknutí v grafickém okně (1) se zobrazí informace o agentu, který je nejblíže kliknuté pozici.
3. ***Informační panel*** - ukazuje počet vrstev, počet agentů a počet agentů-objektů v simulaci. Dále zobrazuje aktuální krok simulace.
4. ***Stavový panel*** - informuje o průběhu simulace a uživatelských akcích.
5. ***Tlačítko Back*** - zpětný chod simulace. Toto tlačítko je aktivní pouze po stisku tlačítka Stop (5). Pomocí tohoto tlačítka se lze vrátit o libovolný počet kroků.
6. ***Tlačítko Stop*** - zastavení běhu simulace.
7. ***Tlačítko Play*** - pokračování simulace po jejím zastavení tlačítkem *Stop* (6).
8. ***Tlačítko Next*** - posune simulaci o jeden krok vpřed.
9. ***Zaškrtávací okénko Simulate again*** – umožňuje při dalším kroku simulovat znovu (okénko je zaškrtlé) nebo načíst soubor z historie (okénko není zaškrtlé).
10. ***Výběr vrstvy*** - vrstva vybraná na záložce je zobrazena v grafickém okně.

4.1.2 Popis Cortona VRML pluginu vizualizačního modulu

Virtuální svět je spouštěn současně s apletem souborem ***start.htm***. Vzhled prohlížeče VRML ukazuje Obrázek č. 24.



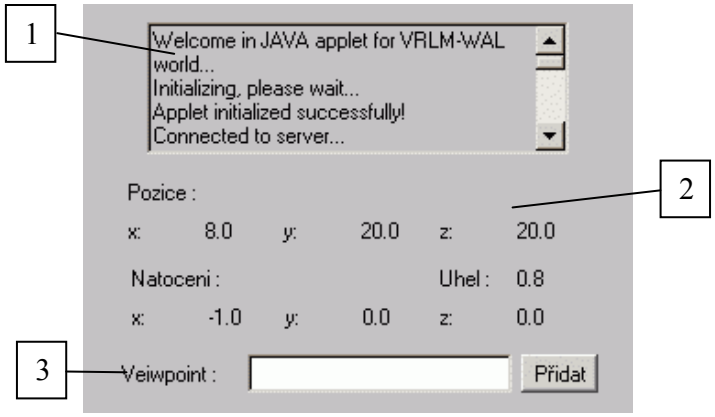
Obrázek č. 24 – Detailní pohled na VRML okno vizualizačního modulu

1. ***Tlačítko WALK*** - v tomto módu je avatar přitahován gravitací k objektům pod ním.
2. ***Tlačítko FLY*** - v tomto módu není avatar přitahován gravitací.

3. ***Tlačítko STUDY*** - v tomto módu avatar není přitahován gravitací a navíc je vypnuta detekce kolizí s objekty. Tento mód není ve vizualizačním modulu povolen.
4. ***Tlačítko PLAN*** - přepíná způsob pohybu na pohyb dopředu a dozadu ve virtuálním světě.
5. ***Tlačítko PAN*** - podobně jako tlačítko *PLAN*, s tím rozdílem, že pohyb funguje pouze dopředu, dozadu, doleva a doprava, tedy jen kolmými směry (v módu *WALK*). Nebo nahoru, dolů, doprava a doleva (v módu *FLY*).
6. ***Tlačítko TURN*** - přepíná způsob pohybu na rotaci pohledu, rotace se děje kolem osy kolmé na pohyb myši se stisknutým levým tlačítkem.
7. ***Tlačítko ROLL*** - přepíná způsob pohybu na rotaci kolem osy kolmé na obrazovku.
8. ***Tlačítko GOTO*** - po stisknutí tohoto tlačítka se změní kurzor myši, po kliknutí na libovolném místě virtuálního světa se na něj avatar přesune.
9. ***Tlačítko ALIGN*** - slouží při ztrátě orientace, srovná polohu avatara, tak aby pohled byl rovnoběžný s rovinou x-y.
10. ***Tlačítko VIEW (šipka vlevo)*** - přepíná mezi přednastavenými navigačními body v prostředí, přepne na předchozí bod.
11. ***Tlačítko VIEW*** - Po stisku tlačítka se rozbalí nabídka s nadefinovanými navigačními body.
12. ***Tlačítko VIEW (šipka vlevo)*** - přepíná mezi přednastavenými navigačními body v prostředí, přepne na následující bod.
13. ***Tlačítko RESTORE*** - slouží při ztrátě orientace, vrátí pozici do aktuálně nastaveného navigačního bodu.
14. ***Tlačítko FIT*** - Posune pozici tak, aby byla vidět celá scéna. Zde může být použito pro oddálení tak, aby byl vidět i měsíc a slunce, umístěné ve značné vzdálenosti.
15. ***Okno zobrazující 3D svět*** - kliknutí pravým tlačítkem myši vyvolá popup nabídku.

Pohyb v 3D světě lze realizovat pomocí směrových šipek klávesnice, či pomocí pohybu myši při stisknutém levém tlačítku. Pro zrychlení pohybu nebo rotace je možné podržet spolu s příslušnou klávesou klávesu SHIFT, CTRL nebo SHIFT+CTRL.

4.1.3 Okno EAI apletu vizualizačního modulu

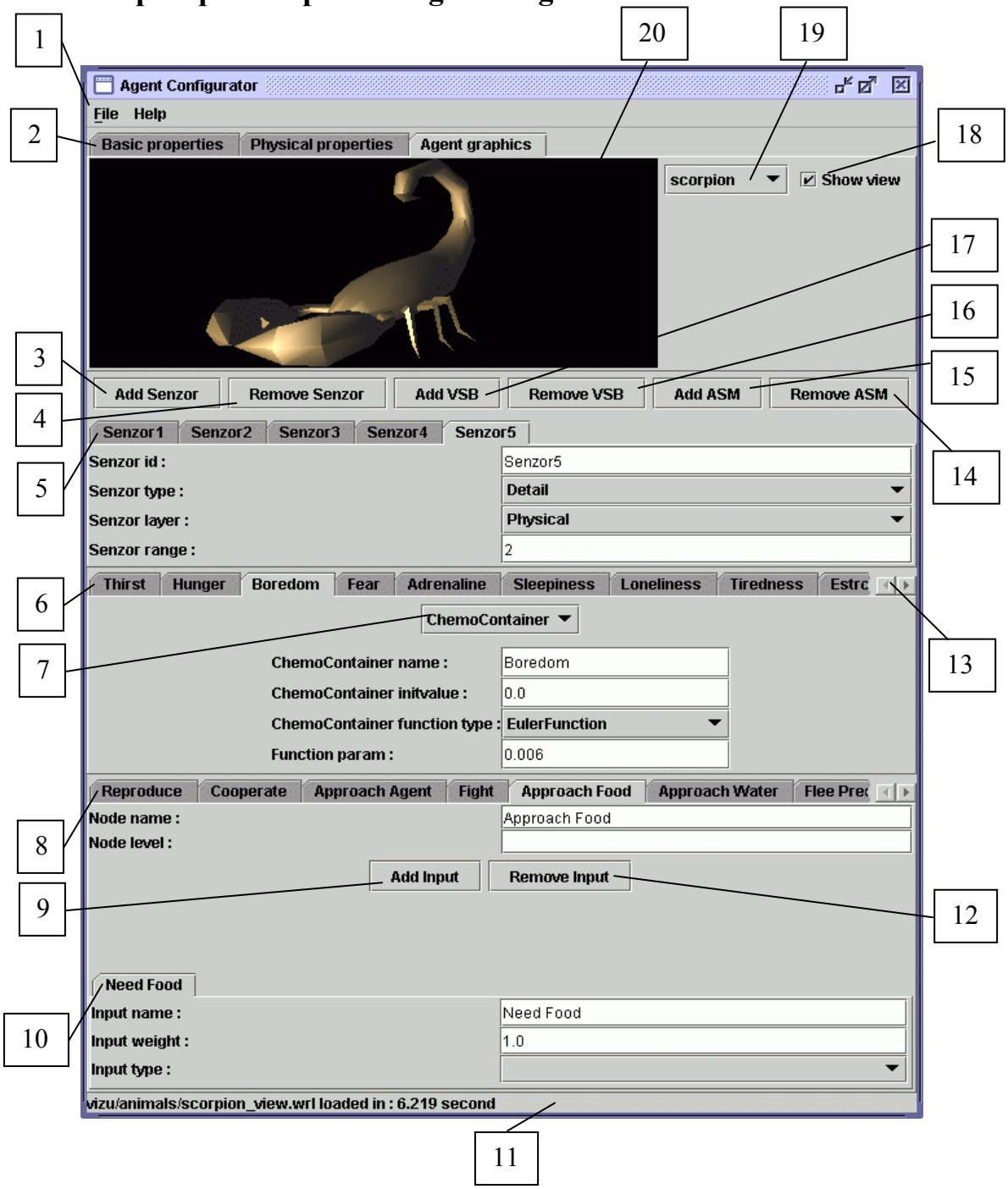


Obrázek č. 25 – EAI aplet

1. ***Informační okno*** - slouží k informování uživatele, zobrazují se zde informace o přijatých zprávách od enginu, o přidaných viewpointech atd.
2. ***Informační část*** - zobrazuje pozici uživatele ve virtuálním světě.
3. ***Přidání vlastního navigačního bodu*** - přidá navigační bod. Pokud uživatel nalezne pro něj výhodný bod na pozorování, může si ho pojmenovat a po přidání se k němu může vracet.

Aplet je spuštěn při načtení HTML stránky ***start.html***. Jeho vzhled viz Obrázek č. 25.

4.1.4 Popis aplikace pro konfiguraci agentů



Obrázek č. 26 – Konfigurační utilita agenta

Aplikace konfigurace agenta se spouští souborem ***startAC3D.bat***, nebo ***startACNo3D.bat***, podle toho zda je nainstalována knihovna Java3D a Xj3D (první jmenovaný) nebo není (druhý). Grafické rozhraní ukazuje Obrázek č. 26.

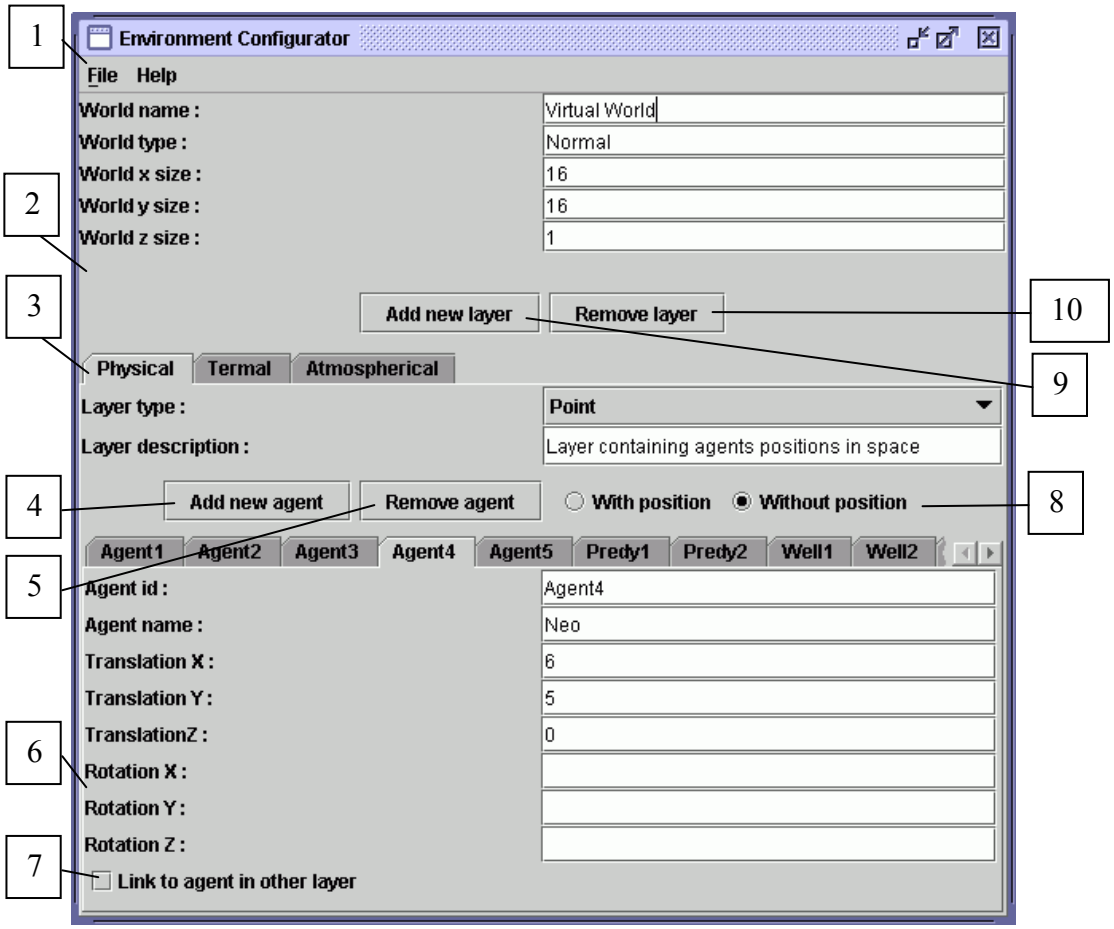
1. ***Hlavní nabídka aplikace*** - slouží k načtení (*Open*) a uložení (*Save*, *Save as*) konfiguračního souboru. Také slouží k opuštění aplikace (Exit) a zobrazení nápovědy (*Help*) a informací o programu (*About*).
2. ***Panel základních vlastností agenta*** - skládá se ze tří záložek:
   a. ***Základní vlastnosti*** (*Basic properties*) - zde se vyplňují parametry jako agentovo jméno (*Agent name*), typ (*Agent type*), subtyp (*Agent subtype*), pohlaví (*Agent gender*), barva agenta (*Agent color*), preferovaná barva agenta (*Preferred Color*), typ prostoru který agent zabírá (*Space taken type*) a jeho parametry (*Radius*).
   b. ***Fyzické parametry*** (*Physical properties*) – zde se vyplňují parametry jako je síla (*Agent strength*), obratnost (*Agent dexterity*), vitalita (*Agent vitality*), rychlost (*Agent speed*) a délka života (*Agent lifelength*).
   c. ***Grafické reprezentace*** (*Agent graphics*) – záložka pro výběr grafické reprezentace agenta (viz také 18,19,20).
3. ***Tlačítko Add senzor*** - přidá záložku nového senzoru do panelu senzorů (viz. 5).
4. ***Tlačítko Remove senzor*** - odebere vybranou záložku senzoru z panelu senzorů (viz. 5). Proti nechtěnému smazání senzoru je přidán potvrzovací dialog s jménem senzoru.
5. ***Panel se záložkami senzorů*** - obsahuje záložky senzorů, které obsahují detailní informace o senzoru. Vyplňuje se identifikátor senzoru (*Senzor id*), typ senzoru (*Senzor type*), vrstva která je odpovědná za plnění senzoru (*Senzor layer*) a dosah senzoru (*Senzor range*). Senzory se přidávají a odebírají pomocí tlačítek *Add senzor* (3) a *Remove senzor* (4).
6. ***Panel se záložkami chemikálií vegetativního bloku*** (*Vegetative system block*) - přidání a odebrání nové chemikálie se děje pomocí tlačítek *Add VSB* (viz. 17) a *Remove VSB* (viz 16). Na záložce prvku je nejprve nutné vybrat jeho typ (viz 7), podle toho se liší vyplňované údaje:
   a. ***ChemoContainer*** - zde se vyplňují parametry kontejneru chemikálie. Jsou to jméno kontejneru (*ChemoContainer name*) shodné s jménem záložky doplněné automaticky, počáteční hodnota (*ChemoContainer initvalue*), typ časové funkce (*ChemoContainer funtion type*) a její parametr (*Function param*).
   b. ***ChemoEmitter*** - zde se vyplňují parametry emitoru chemikálie. Jsou to jméno emitoru (*ChemoEmitter name*) shodné se jménem záložky doplněné automaticky, práh (*ChemoEmitter threshold*), zisk (*ChemoEmitter gain*), název výstupu (*Output name*) a váha výstupu (*Output weight*).
   c. ***Chemical*** - zde se vyplňují parametry chemikálie. Jsou to jméno chemikálie (*Chemical name*) shodné se jménem záložky doplněné automaticky a její počáteční hodnota (*Chemical value*).

7. ***Výběr typu prvku vegetativního bloku*** - na výběr jsou *ChemoContainer*, *ChemoEmitter* nebo *Chemical*.
8. ***Panel se záložkami bloku výběru akcí*** (*Action selection block*) - přidání a odebrání nové akce se děje pomocí tlačítek *Add ASM* (viz 15) a *Remove ASM* (viz 14). Na záložce akce se vyplňuje jméno akce (*Node name*) shodné s jménem záložky doplněné automaticky a úroveň uzlu v neuronové síti (*Node level*). Tento panel obsahuje další dvě tlačítka *Add input* (9) a *Remove input* (12). Součástí tohoto panelu je také speciální ***panel NonEclusive***, který slouží k zadání akcí, které lze provádět zároveň. Při smazání tohoto panelu se tento panel automaticky přidá při přidání další akce vegetativního bloku.
9. ***Tlačítko Add input*** - přidává dané akci vstup z vegetativního bloku nebo z akcí jiných úrovní.
10. ***Panel se záložkami vstupů vybrané akce*** - na záložce vstupu se vyplňuje jeho jméno (*Input name*) shodné s jménem záložky doplněné automaticky, jeho váha (*Input weight*) a jeho typ (*Input type*) kde jsou na výběr hodnoty vsb – vstup je prvek vegetativního bloku, pl – vstup je prvek bloku vnímání a nebo prázdná hodnota.
11. ***Stavový řádek*** - slouží k zobrazování informací o činnosti programu uživateli.
12. ***Tlačítko Remove input*** - odebírá dané akci vybraný vstup. Proti nechtěnému smazání vstupu je přidán potvrzovací dialog s jménem vstupu.
13. ***Směrové šipky sloužící k navigaci*** mezi záložkami.
14. ***Tlačítko Remove ASM*** - odebere záložku akce z panelu bloku výběru akcí (viz. 8). Proti nechtěnému smazání akce je přidán potvrzovací dialog s jménem akce.
15. ***Tlačítko Add ASM*** - přidá záložku nové akce do panelu bloku výběru akcí (viz. 8).
16. ***Tlačítko Remove VSB*** - odebere záložku prvku z panelu vegetativního bloku (viz. 6). Proti nechtěnému smazání prvku je přidán potvrzovací dialog s jménem prvku.
17. ***Tlačítko Add VSB*** - přidá záložku prvku do panelu vegetativního bloku (viz. 6).
18. ***Zaškrtávací tlačítko Show view*** - při jeho zatrhnutí je zobrazen náhled na vybranou grafickou reprezentaci agenta (viz. 19).
19. ***Výběr grafické reprezentace agenta*** - výběr z několika modelů.
20. ***Grafické okno*** - zobrazující 3D model agenta. Toto okno se zobrazí pouze v případě zatrhnutí zaškrtávacího tlačítka (*Show view* – viz. 18). Inicializace grafické knihovny trvá přibližně 7 sekund. Se zobrazeným modelem lze volně otáčet pomocí stisku levého tlačítka myši a pohybu myší a posouvat pomocí stisku pravého tlačítka myši a pohybu myší.

4.1.5 Popis aplikace pro konfiguraci prostředí

Aplikace konfigurace prostředí se spouští souborem ***startWC.bat***. Její grafickou podobu ukazuje Obrázek č. 27.

1. ***Hlavní nabídka aplikace*** - slouží k načtení (*Open*) a uložení (*Save*, *Save as*) konfiguračního souboru. Také slouží k opuštění aplikace (*Exit*) a zobrazení nápovědy (*Help*) a informací o programu (*About*).



Obrázek č. 27 – Konfigurační utilita prostředí

2. ***Panel základních informací o prostředí*** - vyplňuje se zde jméno světa (*World name*), jeho typ (*World type*) a jeho velikost v osách x (*World x size*), y (*World y size*), z (*World z size*). Také obsahuje tlačítka k přidání vrstvy (*Add layer* viz 9) a odebrání vrstvy (*Remove layer* viz 10).
3. ***Panel se záložkami vrstev*** - záložka vrstvy obsahuje detailní informace o ní, kde se vyplňuje typ vrstvy (*Layer type*), a její krátký popis (*Layer description*). Dále obsahuje tlačítka na přidání agenta do vrstvy (*Add agent* viz 4) a odebrání agenta z vrstvy (*Remove agent* viz 5). Když je agent už jednou zadán a chceme jen toho samého agenta přidat do další vrstvy použijeme zatrhnuté tlačítko *Without position* a u agenta zatrhneme *Link to agent in other Layer*.
4. ***Tlačítko Add agent*** - přidá záložku nového agenta do panelu agentů (viz.6).
5. ***Tlačítko Remove agent*** - odebere vybranou záložku agenta na panelu agentů. Proti nechtěnému smazání agenta je přidán potvrzovací dialog s jeho jménem (viz.6).
6. ***Panel se záložkami agentů*** - záložka agenta obsahuje detailní informace o něm, ty se liší podle toho zda chceme přidat agenta s definicí jeho pozice či nikoliv (Zatrhávací tlačítko *With/Without Position* viz. 8). Při přidání agenta s pozicí lze na kartě agenta

vyplnit jeho identifikátor (*Agent id*) shodný s jménem záložky doplněno automaticky, jméno (*Agent name*) a jeho pozici v x (*Translation x*), y (*Translation y*), z (*Translation z*) souřadnicích a jeho rotaci okolo os x (*Rotation x*), y (*Rotation y*), z (*Rotation z*). Při přidání agenta bez pozice je nutné vyplnit jen identifikátor a jméno.

7. ***Zašktávací tlačítko Link to agent in other Layer*** - propojuje agenta s jeho reprezentací v jiné vrstvě, toto se děje typicky v případě že agenta umístíme do fyzické vrstvy na nějakou pozici a chceme agenta propagovat i do dalších vrstvě, bez tohoto propojení by byl v jiných vrstvách vytvořen nový agent.
8. ***Zaškrtávací tlačítko With/Without Position*** - při přidávání agenta ovlivňuje zda karta agenta bude obsahovat pozici či nikoliv.
9. ***Tlačítko Add layer*** - přidá záložku nové vrstvy do panelu vrstev (viz. 3).
10. ***Tlačítko Remove layer*** - odebere záložku vrstvy na panelu vrstev (viz. 3) a všechny agenty v ní zaregistrované. Proti nechtěnému smazání vrstvy je přidán potvrzovací dialog s jejím jménem.

4.1.6 Postup při spuštění naprogramovaných aplikací

V této kapitole je uveden postup jak spustit všechny navržené a naprogramované aplikace. Ke spuštění všech je nutné mít nainstalovánu podporu jazyka Java. Doporučuji se *JRE 1.4.1* nebo vyšší [P17].

WAL simulátor

Je důležité umístit adresář s aplikací nejlépe do kořenového adresáře, protože aplikace hledá konfigurační soubory a některé cesty (adresáře s mezerami v názvu) by mohli mít za následek nenalezení souborů a pád aplikace. Simulátor se spouští souborem ***start.bat*** v adresáři */WAL*.

VRML vizualizační modul

Ke spuštění tohoto modulu je nutné mít internetový prohlížeč Internet Explorer. Dále je nutné mít nainstalován prohlížeč VRML světů, doporučuji prohlížeč Cortona [P03]. Navíc je nutné mít nainstalovaný a spuštěný webový server z důvodů uvedených v 3.4.4. Doporučuji EasyPHP 1.7 [P15]. Do adresáře */www* tohoto serveru je nutné zkopírovat obsah adresáře ***WAL/VRMLclient***. Následně je možné spustit WAL simulátor viz výše. Poté lze spustit vizualizační modul z prohlížeče Internet Explorer zadáním adresy *http://localhost/vizu/start.htm*.

Konfigurační utility

Nástroj pro konfiguraci prostředí lze spustit souborem ***startWC.bat*** z adresáře ***/WAL***. Nástroj pro konfiguraci agenta je ve dvou verzích. V případě, že je nainstalována Java3D 1.3.1 a vyšší [P18], spolu s Xj3D m8 [P20] a vyšší, lze ho spustit i s 3D zobrazením tvaru agenta souborem ***startAC.bat***. V případě, že nejsou nainstalovány tyto doplňkové knihovny, je přesto možné spustit konfigurační nástroj souborem ***startACNo3D.bat***, nyní již bez podpory 3D grafiky.

Demonstrační programy

Demonstrační programy pro funkci detekce kolizí a funkci genetických operátorů je možné spustit soubory ***startCD.bat*** a ***startGD.bat***.

4.2 Programátorský manuál

Tato kapitola shrnuje implementované programové prostředky. Nesnaží se být kompletním programátorským manuálem. Součástí programátorského manuálu jsou i UML diagramy ilustrující jednotlivé návrhy v části 2 či konkrétní implementaci v části 3. Hlavním zdrojem informací o naprogramovaných třídách je bezpochyby vygenerovaná dokumentace pomocí nástroje `JavaDoc`.

4.2.1 Javadoc

`JavaDoc` je nástroj, sloužící k automatickému generování dokumentace v HTML formátu ze zdrojového kódu. Jeho výsledkem je komplet HTML souborů, které jsou vhodným způsobem vzájemně provázány, tak aby umožnili snadné prohlížení dokumentací ke třídě, skupině tříd, balíku nebo i skupině balíků (viz Obrázek č. 29). Celá API dokumentace, dodávaná k Javě, je generována právě tímto nástrojem.

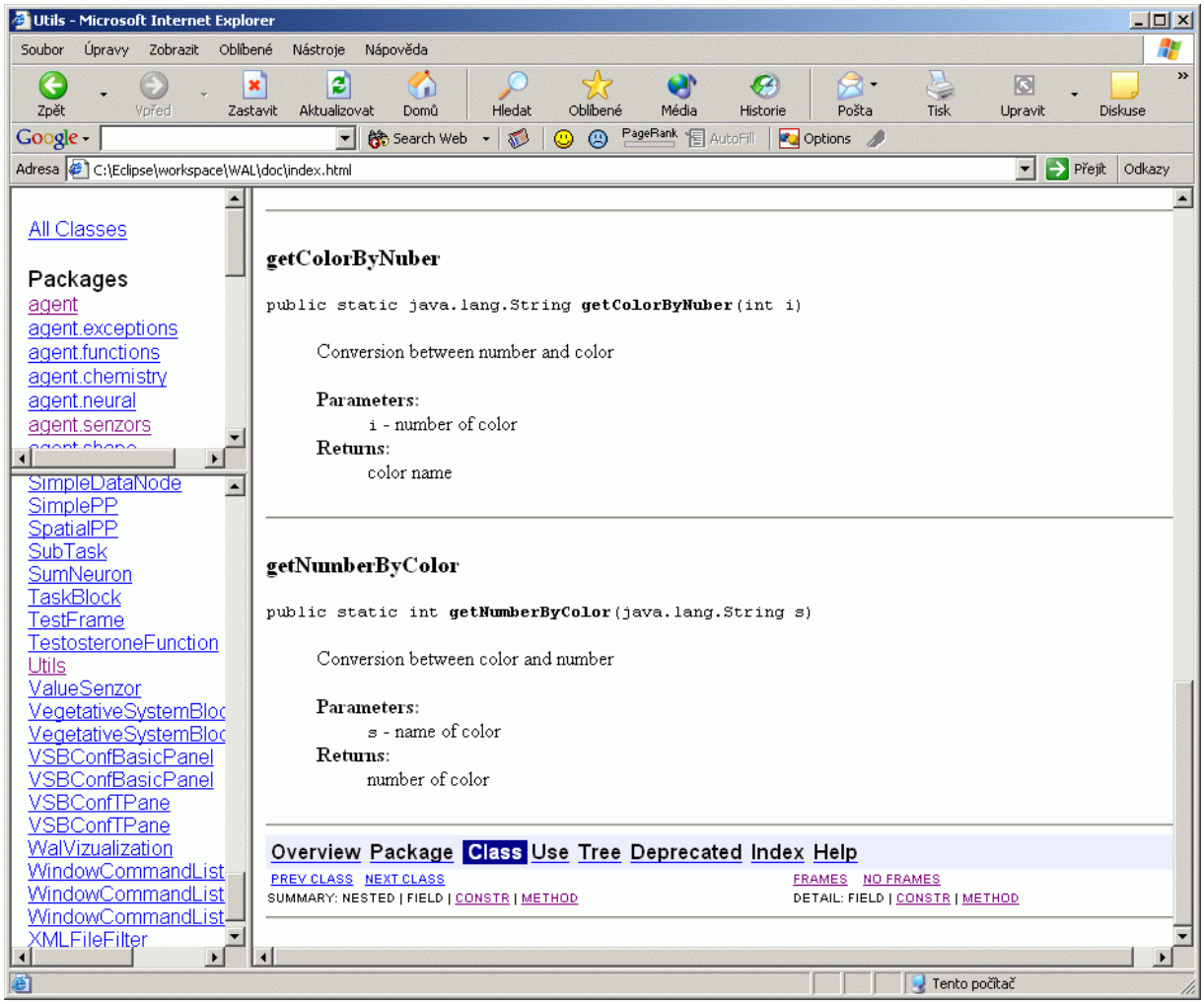
`Javadoc` analyzuje vstupní `.java` soubory a podle nich generuje dokumentaci. Programátor má možnost ve zdrojových souborech používat tzv. dokumentační komentáře (viz Obrázek č. 28), jejichž obsah bude vložen do HTML dokumentace na příslušné místo - popis třídy, popis atributu, popis metody. Dokumentační komentáře začínají znaky `/**` a končí znaky `*/`. Budou vloženy do HTML přesně tak, jak je napíšete (s malými úpravami), tj. můžete v nich používat HTML kód, který bude prohlížečem interpretován. Dokumentační komentáře se vkládají vždy před dokumentovanou entitu.

Do dokumentačních komentářů lze vkládat i speciální značky, které mají zvláštní význam. Značky začínají znakem "@" jsou javadocem interpretovány speciálním způsobem, většinou tučným písmem a speciálním formátováním. Značky mají většinou jeden nebo dva parametry, které se od nich oddělují bílým znakem, většinou tabulátorem. Značky musejí být umístěny hned na začátku řádky (s výjimkou mezer a hvězdiček), jinak

nejsou zpracovány. Značky jsou case-sensitive. I v parametrech značek lze používat HTML kód, lze například vložit odkaz na autorovu osobní stránku [29].

```
/**
 * Some useful functions with general usage
 * @author Karel Kohout
 */
public class Utils {
	/**
	 * Conversion between number and color
	 * @param i number of color
	 * @return color name
	 */
	public static String getColorByNuber(int i) {
		switch (i) {
			case 1: return "red";
			case 2: return "green";
			case 3: return "blue";
			default: return null;
		}
	}
}
```

Obrázek č. 28 – Ukázka zdrojového kódu s komentáři Javadoc



Obrázek č. 29 – Ukázka vygenerované dokumentace v HTML formátu

5 Provedené experimenty

Tato kapitola je věnována testům a pokusům provedených v průběhu vývoje enginu a přidružených aplikací (konfigurační utility, externí moduly). Nejdříve byli provedeny testy konfiguračních utilit, které vznikly nejdříve (5.1.1 a 5.1.2). Posléze byl proveden test komunikace a funkčnosti externího modulu VRML vizualizace (5.2). Posléze následovali komplexnější testy simulací v enginu.

5.1 Ověření funkčnosti konfiguračních aplikací

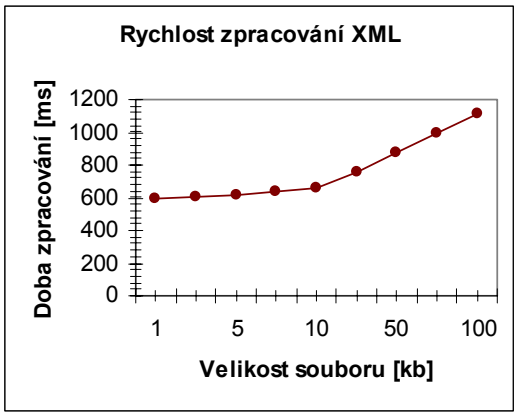
Všechny uvedené simulace a testy byli provedené na počítači s následujícími parametry:

- procesor: Athlon 2500XP+,
- paměť: 512MB DDR, dual mode,
- Grafická karta: GeForce 5600, 128 MB,
- Základní deska: Microstar K7N2 Delta s chipsetem nForce2,
- Operační systém: Windows XP.

5.1.1 Konfigurace prostředí

Ověření funkčnosti konfigurační aplikace prostředí jsem provedl několika způsoby. Jako první bylo nutné ověření funkčnosti prvků aplikace jako jsou tlačítka, položky menu a jiné prostředky interakce.

Načítání konfigurace prostředí	
velikost souboru 10 051 B	
Parsování XML tvorba GUI	Uložení XML
[ms]	[ms]
890	16
859	31
875	16
860	32



Tabulka č. 2 – Změřené doby (vlevo), rychlost zpracování XML (vpravo)

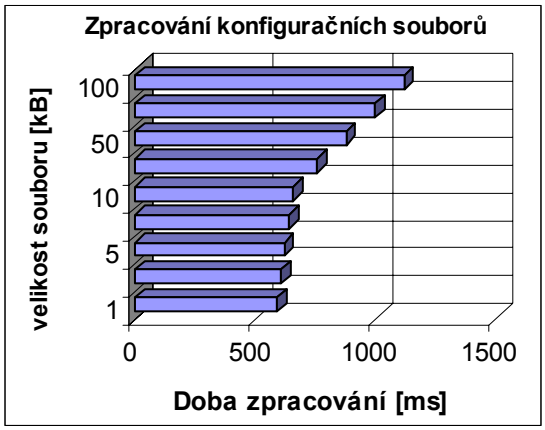
Hlavní funkcí je vytvářet validní (*valid*) a dobře formovaný (*well-formed*) konfigurační soubor ve formátu XML. Pokud aplikace vytvoří validní a správně

formovaný dokument, není ještě zaručeno, že pokud byl původní dokument pouze načten a nezměněn uložen, nedošlo při funkci programu k nějaké úpravě. Proto jsem textově porovnal původní soubor s týž souborem uloženým aplikací. Také jsem změřil doby načítání a ukládání souborů, které obsahuje Tabulka č. 2.

5.1.2 Konfigurace agenta

Obdobným testům jako konfigurační utilitu prostředí jsem podrobil i konfigurační utilitu agenta. Této aplikace existují dvě verze, jedna s podporou Java3D a Xj3D pro náhled na grafické ztvárnění agenta ve vizualizace. Druhá verze je funkčně naprosto totožná, až na zmiňovaný grafický náhled. Tyto dvě verze jsem odlišil proto, aby nebyla po uživateli vyžadována instalace Java3D a Xj3D. Výsledek testu ukazuje Tabulka č. 3.

Načítání konfigurace Agenta		
velikost souboru 10 051 B		
Parsování XML	Uložení XML	Inicializace grafiky
[ms]	[ms]	[ms]
2016	47	7359
1859	31	5968
1891	31	5984
1875	15	5968

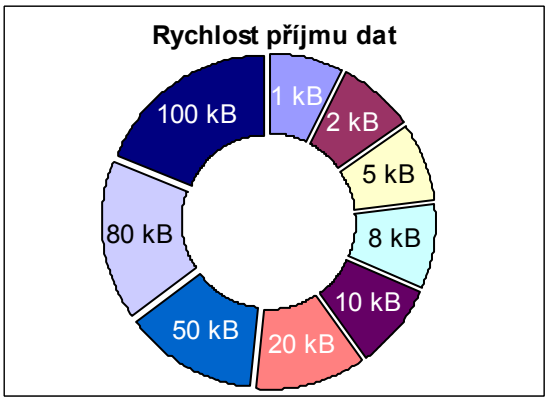


Tabulka č. 3 – Změřené doby (vlevo), rychlost zpracování (vpravo)

5.2 Ověření funkčnosti přenosu dat

Ověření přenosu dat od serveru enginu ke klientovi vizualizačního modulu je ověřeno správným zobrazením situace modulem (viz. 5.4).

Přijímání dat vizualizačním modulem	
Velikos posílaného souboru cca 6kB	
Čas přijímání dat	Čas zpacování dat
[ms]	[ms]
453	63
437	47
422	47
437	47



Tabulka č. 4 – Změřené doby (vlevo), rychlost příjmu dat (vpravo)

Také jsem změřil doby přenosu a zpracování dat. Při přenosu je nutná synchronizace s enginem. Je tedy třeba čekat na data. Tudíž uvedené časy příjmu dat silně závisí na času jednoho kroku simulace. Tedy z hlediska funkčnosti modulu jsou zajímavější časy zpracování přijatých informací. Ty jsou zase úměrné velikosti zasílané informace. Naměřené výsledky uvádí Tabulka č. 4.

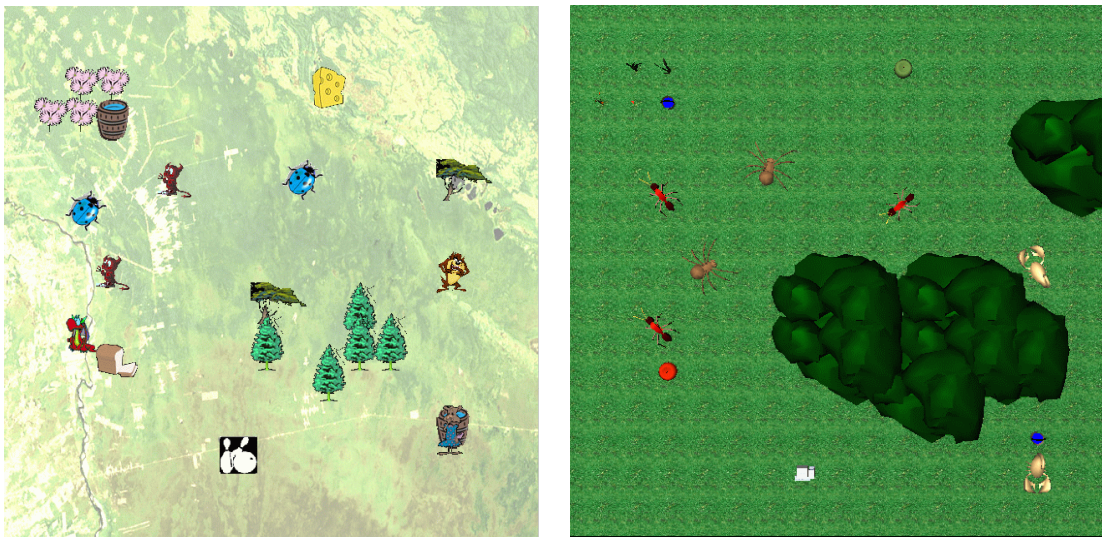
5.3 Ověření funkčnosti algoritmu detekce kolizí

Za účelem demonstrace i ověření správné funkce algoritmu detekce kolizí jsem vytvořil pomocnou aplikaci. Tato aplikace zobrazuje počáteční a koncové polohy dvou agentů. Navíc zobrazuje místo dotyku agentů a místo opětovného rozpojení. Abych ověřil funkčnost výsledku kolizí, navrhl jsem simulační situace takové, kdy jeden z agentů měl trojnásobnou velikost než ostatní. Tento agent při svém pohybu ostatní agenty odtlačil do strany. Pokud navíc kolidující agenti mířili opačnými směry, byl menší (slabší) agent odtlačen větším (silnějším).

5.4 Ověření funkčnosti na simulacích agentů

Tato práce se zabývá návrhem prostředí pro agenty. Nezabývá se architekturou agenta samotného, respektive architekturou řízení (mysli) agenta. Proto pro ověření simulace bylo nutné použít již navrženou architekturu agentů. Za tímto účelem lze vybrat architekturu implementovanou v jazyce Java, aby její začlenění nebylo příliš náročné. Proto jsem zvolil hybridní architekturu *ALS* navrženou Ing. Kadlečkem [7] a modifikovanou kolegou Danem Kolářem [42]. Tento pokus ověřil nejen funkčnost prostředí jako takového, ale i jeho modulárnost a universálnost.

5.4.1 Porovnání grafických prostředí

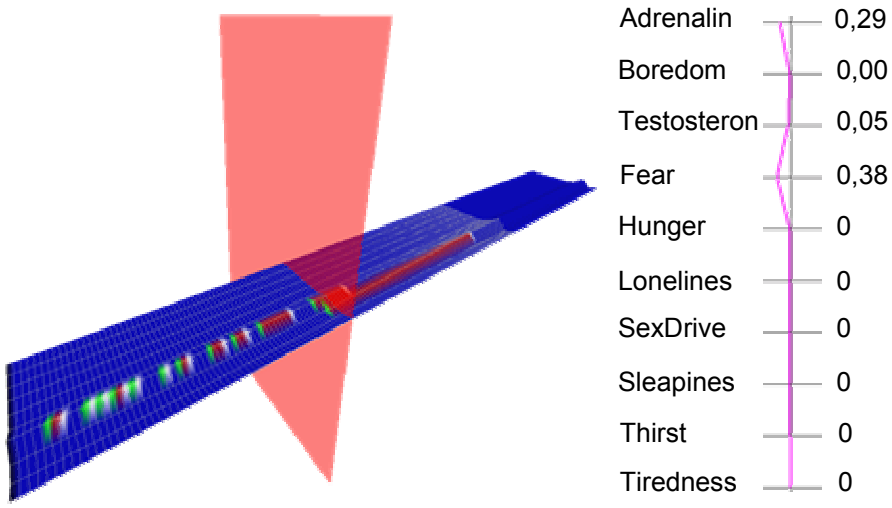


Obrázek č. 30 – Ukázka ALS simulace (vlevo) a WAL simulace (vpravo)

Obrázek č. 30 ukazuje situaci ze simulace původní aplikace ALS a zároveň pohled na tu samou situaci ve vizualizačním modulu v prostředí **WAL**. S jedním malým rozdílem, že ve **WAL** jsou pouze tři modely agentů. Naproti tomu v ALS můžeme vidět pět různých obrázků agentů.

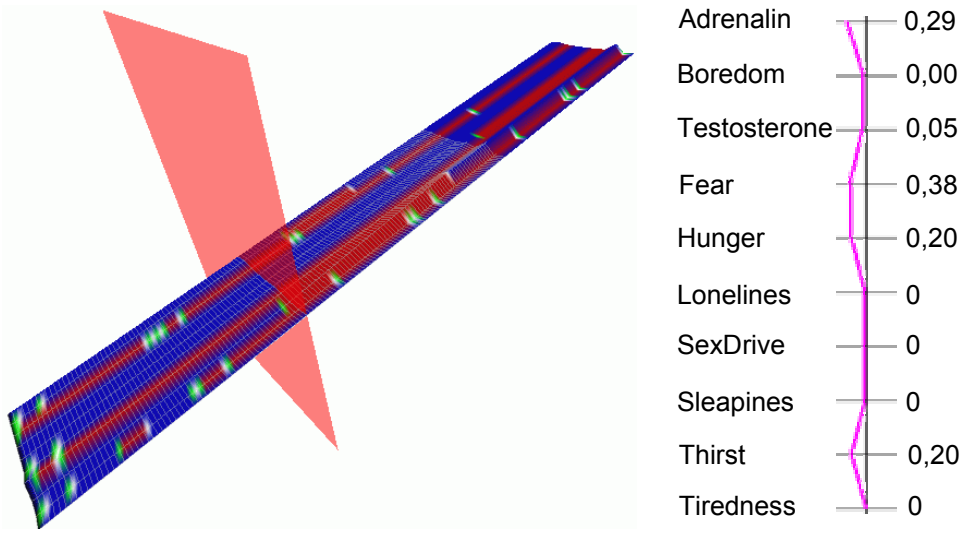
5.4.2 Využití metod vizualizace k ladění simulace

K analýze simulace jsem využil nástroje VAT (viz 3.5.1, [14]). V této kapitole chci demonstrovat jeho použití a přínos pro analýzu chování. Obrázek č. 31 ukazuje data ze simulace. Vlevo je pohled na 3D plochu, vpravo detail hodnot v kroku 60. V této simulaci měl agent za úkol přemisťovat předměty. Měl také k dispozici dostatek potravy a vody. Práh pro plnění daného úkolu byl zvolen tak, aby agent nezemřel hlady při plnění zadaného cíle.



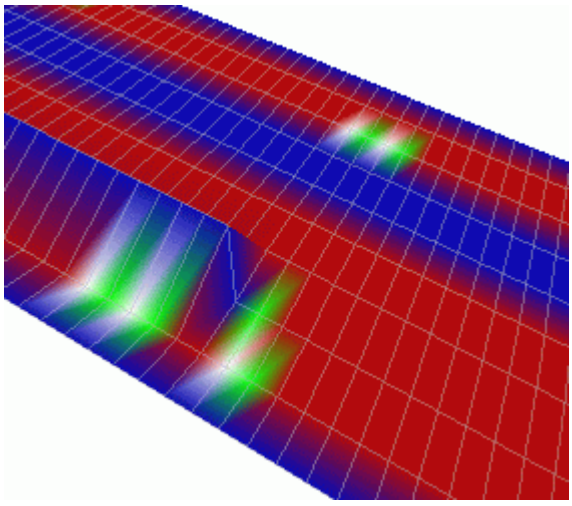
Obrázek č. 31 – Ladění aplikace pomocí analýzy VAT1 - krok 1

Z prvního pohledu na 3D plochu je vidět, že simulaci něco chybí. Plocha je až na dvě výjimky rovná. Pohledem na graf hodnot jsem zjistil, že téměř veškeré atributy agenta jsou nulové. Agent ani jednou nemá hlad, nemá žízeň, není unaven či ospalý. Důvodem může být chyba v exportu dat, chyba ve vegetativním bloku, nebo špatné počáteční nastavení. Export dat je ověřen a funkční, vegetativní blok také funguje bez problémů. Řešení je nutné hledat v konfiguraci agenta. Funkce zvyšující hladiny hormonů ve vegetativním bloku byli špatně nakonfigurovány. Parametr jejich časových funkcí byl příliš malý na to, aby se za čas simulace jejich hladina výrazně zvýšila. Provedl jsem korekci počátečního nastavení a simulaci provedl znovu.



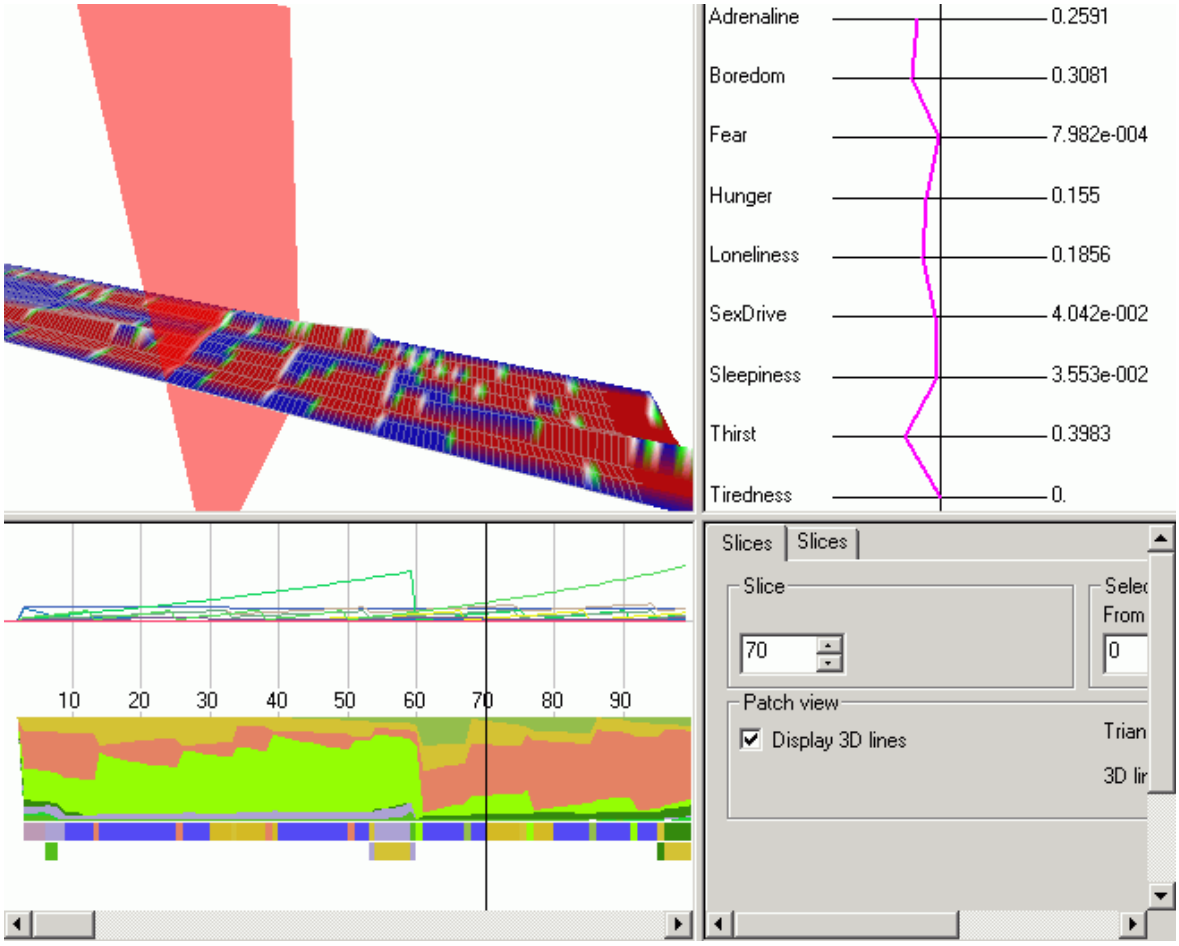
Obrázek č. 32 – Ladění aplikace pomocí analýzy VAT1 – krok 2

Jak ukazuje Obrázek č. 32 je 3D plocha i simulace pestřejší. Dalším artefaktem, kterého jsem si při analýze povšiml, je prudký skok hormonu testosteronu z hodnoty 1 na hodnotu 0. Tento skok zachycený na ploše zobrazuje Obrázek č. 33. Tento pokles je však zcela v pořádku a nejedná se o žádnou chybu v programu ani v návrhu, nýbrž o záměr. Agent totiž v tomto bodě právě zestárl a není tedy schopen dále se rozmnožovat. Proto je jeho hladina hormonu testosteronu vynulována.



Obrázek č. 33 – Prudký pokles hormonu testosteronu

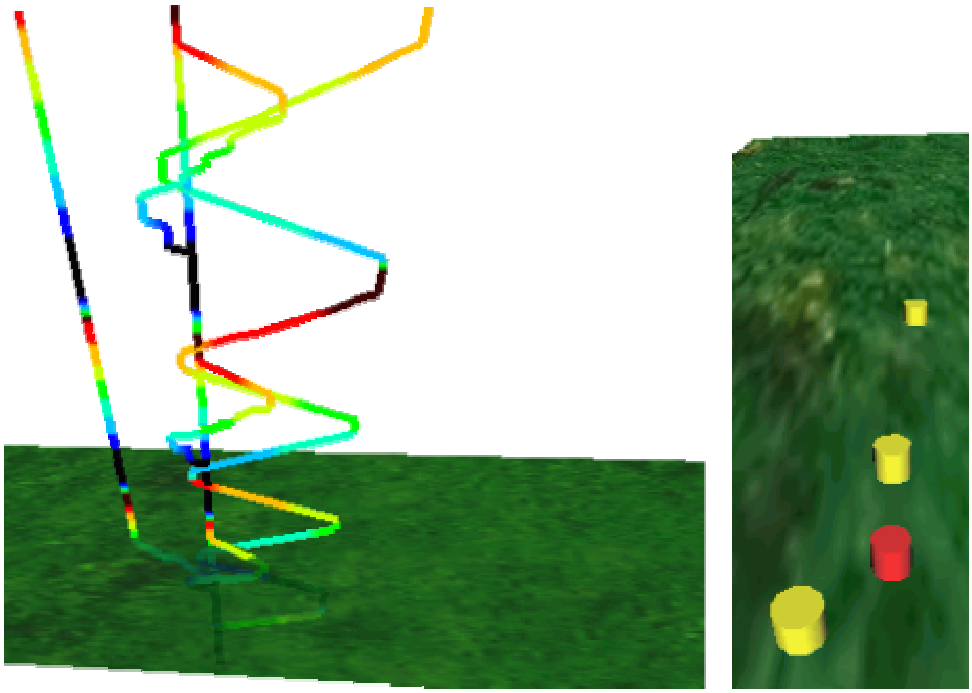
Těmito dvěma situacemi jsem demonstroval použití analytického nástroje VAT pro odladění běhu simulace. Obrázek č. 34 ukazuje v plném kontextu VAT úrovně jedna, jak vypadá celý život agenta plný nejrůznějších podnětů a situací, jež agent musí řešit. Na vzory chování agenta nelze usuzovat jen na základě průběhu jeho parametrů v čase. Je nutné uvážit i polohy a parametry jiných agentů a stav prostředí. Pro tento účel slouží VAT úrovně 2 a 3.



Obrázek č. 34 – Průběh života agenta

5.4.3 Analýza simulace s podporou moderních metod vizualizace

Pro analýzu interakce mezi agenty a hodnocení chování agentů ve společenství slouží nástroj VAT úrovně 3. Přínos a užitečnost tohoto nástroje jak k analýze chování, tak zajisté také k odlaďování programu zde budu demonstrovat na dvou příkladech simulací. V první z nich se vyskytují 4 agenti. Dva z nich mají za úkol přenášet objekty mezi dvěma místy. Dále je v simulaci jeden predátor a jeden agent, který se stará jen o své přežití. Obrázek č. 35 ukazuje výsledek VAT úrovně 3 při načtení této simulace. Bez znalosti zadání simulace si lze okamžitě všimnout periodicky se opakující struktury v levé části obrázku. Ta odpovídá pohybům agentů z jednoho místa do druhého a zpět při přenášení předmětů. Barva na této křivce znázorňuje ospalost agenta (*sleapiness*). Škálu barev, kdy černá odpovídá nejnižší intenzitě a červená nejvyšší ukazuje Obrázek č. 36. Je tedy vidět, že žádný z agentů simulace není nočním tvorem. Všichni agenti se periodicky s denní dobou stávají více ospalejšími (největší ospalost je reprezentována červenou barvou). Po tom co se agent vyspí, klesne opět hodnota ospalosti na minimální úroveň (zobrazeno černou barvou). Na tyto křivky lze však mapovat i jiný z parametrů agentů a obdobným způsobem analyzovat jeho průběh.



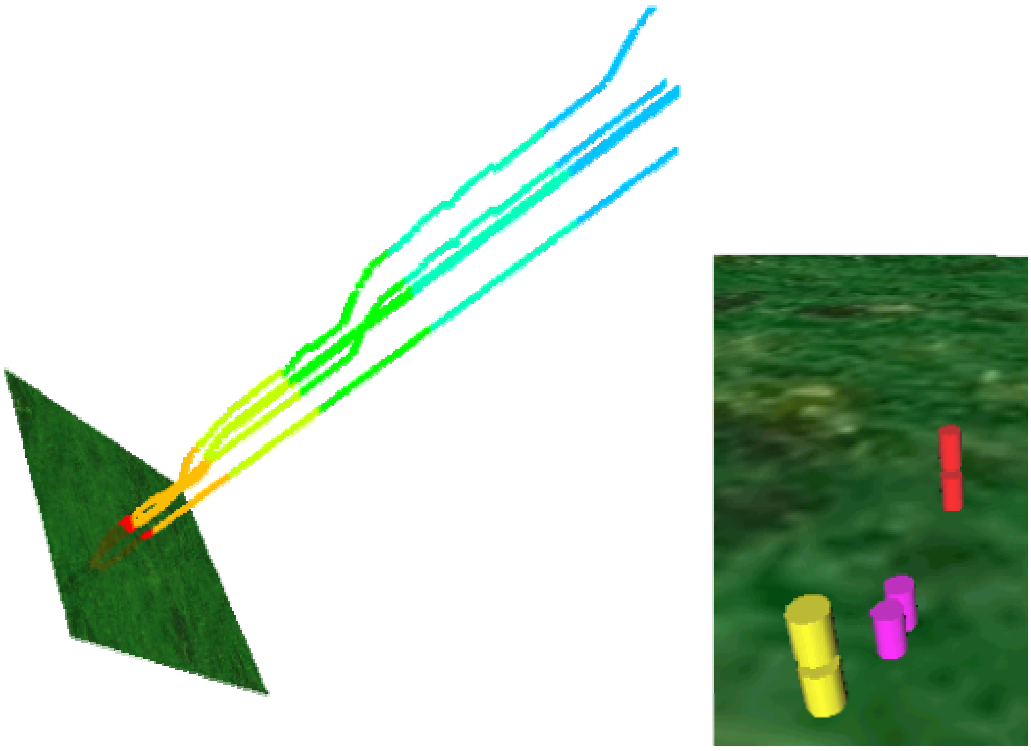
Obrázek č. 35 –Analýza simulace pomocí VAT3

Obrázek č. 36 – Škála barev od nejnižší po nejvyšší intenzitu

Zaměřme nyní pozornost na obrázek vpravo. Zde jsou agenti zobrazeni jako válce. Barva agentů odpovídá skupině, do které byli agenti zařazeni. Zde jsem zvolil dvě skupiny. Červeně je na obrázku označen predátor, žlutě ostatní agenti. Na výšku agenta jsem namapoval jeho hlad. Z toho lze usoudit, že predátor společně s agentem nad ním mají největší hlad. Na průměr válce je mapován strach agentů. Jak je vidět, predátor téměř strach nemá, zato agent nacházející se v obrázku nejníže, se bojí nejvíce ze všech agentů. Je to z důvodu přítomnosti predátora v jeho blízkosti. Agent v obrázku umístěný nahoře má strach nejmenší, protože je od predátora dostatečně vzdálen, nebo jeho senzory přítomnost predátora ani nezaznamenaly. Pomocí analytického nástroje VAT3, lze vyčíst mnohem více informací, použitím citlivostní analýzy i na jiné parametry, než jsem zde vybral já.

Rád bych demonstroval použití tohoto analytického nástroje ještě na průběhu jedné simulace. Zde bych ukázal jeho použití při jejím odladění. V této simulaci se vyskytují dva predátoři, dva agenti jednoho druhu a dva agenti jiného druhu. Agenti se starají o své přežití, jedná se tedy o simulaci systému lovec-oběť. Obrázek č. 37 ukazuje výsledek analýzy pomocí VAT3. Barva křivek v levé části obrázku ukazuje strach agentů. Na první pohled je vidět, že zde je něco v nepořádku. Agenti na počátku mají vysokou hodnotu strachu a i když se vyskytují v blízkosti predátoři, strach jim lineárně a navíc všem stejně klesá. To indikuje, že simulace špatně nebo vůbec modifikuje strach agenta, nebo že agenti

nejsou schopni rozpoznat predátora a nebo nejsou schopni ho zachytit na senzorech. Toto bychom použitím VAT1 nepoznali, protože by nám scházel kontext, tím je přítomnost predátora.



Obrázek č. 37 – Odladění simulace pomocí VAT3

Přesuňme pozornost na pravou část obrázku. V tomto případě jsem výšce válce přiřadil atribut adrenalinu. S adrenalinem je také něco v nepořádku, všichni agenti mají hladinu adrenalinu přibližně stejnou. Tato situace by mohla teoreticky nastat. Proto jsem zaměřil pozornost na hladinu adrenalinu a nechal ji namapovat na časový průběh. Zjistil jsem, že adrenalin stejně jako výše zmíněný strach má stejnou hodnotu pro všechny agenty a klesá lineárně s časem. Tuto skutečnost bychom pomocí VAT1 zjistili, ale jen u jednotlivých agentů. Příčiny mohou být podobné jako u strachu. Avšak již dva z hormonů nejsou při simulaci měněny, z toho lze usuzovat na špatnou funkci vegetativního bloku. Průměr válců reflektuje hlad agenta. Ten je zde pro každého z agentů jiný, tedy není důvodu k podezření, že by i tímto parametrem bylo něco v nepořádku.

Na těchto dvou příkladech jsem se snažil demonstrovat jednak postup při analýze simulací a jednak užitečnost analytických nástrojů VAT. Pro obě uvedené úrovně VAT jsem uvedl i negativní případy, kdy bylo možno s jejich pomocí nalézt nějakou chybu v simulaci.

Závěr

V předložené diplomové práci se zabývám návrhem prostředí pro simulaci agentů-animátů a navazuji na předchozí práce výzkumné skupiny MRG vedené doc. Ing. Pavlem Nahodilem, CSc. na katedře kybernetiky FEL ČVUT v Praze.

V prvním bodu řešení zadání práce jsem se seznámil s přístupy MAS a ALife a to takovým způsobem, abych nabité znalosti mohl využít při simulacích chování agentů-animátů. Dále jsem se seznámil s možnými metodami analýzy simulací jak jednotlivců, tak celých společenství. Získané znalosti jsem krátce shrnul v kapitole 1 této práce.

Dalším bodem byl návrh prostředí pro modely společenství robotů-animátů. Součástí tohoto bodu bylo i odzkoušení vybraného jedince i vybraných společenství v situacích převzatých z přírody. Po analýze dostupných prostředí pro simulace společenství agentů-animátů a po dohodě s vedoucím diplomové práce jsem odstartoval projekt, který jsem nazval *World of Artificial Life* zkráceně **WAL**. Jeho abstraktní architektura nazvaná ***WALA*^{2}** je popsána v kapitole 2. V kapitole 3 popisuji její konkrétní implementaci, tak jak jsem ji realizoval. V navrhnutém prostředí lze simulovat různé fenomény z oblasti ALife. Příkladem je simulace boidů, naprogramovaná na základě abstraktní architektury ***WALA*^{2}** kolegou Lukášem Foltýnem [43]. Složitějších simulace jsou provedené jednak v této práci, jednak v práci kolegy Dana Koláře [42]. Export dat ze simulace byl v této práci rozpracován do detailů. Poskytuje možnost posílání dat po síti internet, jejich ukládání pro zpětný chod simulace a také výpis do binární nebo XML podoby. Modulárnost realizovaného prostředí jsem dokumentoval na vizualizačním modulu (viz 3.4), i když zvolené řešení nebylo nejvhodnější, bylo plně funkční a pro demonstraci principu plně postačilo.

Posledním bodem zadání této práce bylo vyhodnocení výsledků s důrazem na využití metod počítačové vizualizace. Za tímto účelem jsem využil nástroje VAT. S jeho pomocí jsem v kapitole 5 demonstroval analýzu a odladění chování jak vybraného jedince tak i celého společenství.

Na základě uvedeného se proto domnívám, že jsem všechny body zadání diplomové práce splnil. Na pokyn vedoucího práce jsem se soustředil zejména na návrh architektury a implementaci simulačního prostředí a jejich popis včetně podrobného uživatelského a programátorského manuálu.

Nad rámec zadání jsem navrhl vizualizační modul využívající technologii VRML a implementoval konfigurační utility agenta a prostředí sloužící k ulehčení práce. V neposlední řadě jsem také specifikoval podobu binárního stromu a jeho posílání externím modulům. Rovněž jsem vytvořil pomocnou aplikaci sloužící k demonstraci algoritmu kolizí. Při řešení výsledků kolizí jsem jí často využíval.

Při řešení práce jsem se snažil dodržet citační normu ISO 690 z roku 2002 [40], [41].

Přínos práce

Jádrem této diplomové práce jsou kapitoly 2 a 3, které popisují navržené prostředí jak z hlediska teoretického, tak i implementačního. Za svůj přínos považuji původní myšlenku **WAL** abstraktní architektury neboli **WALA²** a její rozpracování až po úroveň konkrétních aplikací. V této architektuře jsem autorem myšlenky vrstev prostředí (viz 2.3), myšlenky modulárnosti, komunikace s externími moduly a datové reprezentace používané pro komunikaci. Dalším mým přínosem je pokus o vizualizaci světa agentů za pomocí technologie VRML. Tento přístup se však v průběhu řešení ukázal jako nepříliš vhodný. V práci jsem ho ponechal ze dvou důvodů. Jednak je zástupcem externích modulů, které komunikují s vlastní aplikací jen pomocí přenosu dat po síti, a jednak abych upozornil své nástupce na možná úskalí tohoto řešení. Další body abstraktní architektury jsou výsledkem kolektivní práce skupiny vedené Doc. Ing. Pavlem Nahodilem, CSc. [42], [43], [44], [45], [46], [47], [P04], [P05], [P06], [P07], [P08].

Jím jsem byl pověřen organizací a koordinací dílčích úkolů jednotlivých členů uvnitř skupiny, která byla myšlenkově a zejména časově náročná. Věřím, že navržená architektura prokáže své kvality nejen v pracích mých kolegů ze skupiny MRG, ale i v jiných výzkumných pracovištích u nás i ve světě.

Na doporučení vedoucího práce jsem i z těchto důvodů věnoval zvýšenou pozornost tvorbě uživatelského a programátorského manuálu.

Možné směry dalšího vývoje WAL

Úplné řešení simulace společenství agentů-animátů svým rozsahem značně překračuje rámec této práce. Každá z identifikovaných oblastí zde byla zpracována tak, aby byla funkční a použitelná. Často jsem však musel přistoupit k zjednodušení řešeného problému. Chování i jednoduchých živočichů je složité a matematicky špatně uchopitelné, proto je nutné dělat ústupky a soustředit se jen na důležité aspekty chování. Prohloubení algoritmů použitých v této práci bude, jak doufám, náplní prací mých nástupců.

Prohloubení a vylepšení jsou možná především v oblastech:

- Detekce kolizí ve vrstvách prostředí, především propracování a navržení důmyslnějšího mechanismu a implementace alespoň jednoduchého fyzikálního modelu.
- Implementace vyšších verzí datového stromu udržujícího informaci o prostředí a rozšíření o možnost udržovat více položek pod jedním klíčem a adresovat je.
- Implementace binární datové representace, která byla v této práci popsána.
- Rozšíření parametrizace simulace dle návrhu, který jsem v této práci provedl.

Reference

[1] Lorenz, K. *Základy etologie*, Praha: Academia, 1993. ISBN 80-200-0477-7.

[2] Herout, P. *Učebnice jazyka Java*. 1. vydání. České Budějovice: Koop, 2003. ISBN 80-7232-115- 3.

[3] Mařík, V., Štěpánková, O., Lažanský, J. a kol. *Umělá inteligence 3*. Praha: Academia, 2001. ISBN 80-200-0472-6.

[4] Herout, P. *JAVA - grafické uživatelské rozhraní a čeština*. České Budějovice: Koop, 2001. ISBN 80-7232-150-1.

[5] Nahodil, P., Kadleček, D., Kohout, K., Svrček, A. Theory and Robotics Artificial Life Applications for Mobile Robots Control. In *Proceedings of WORKSHOP 2004, 22.-26. 3. 2004, Praha, Česká Republika.* Praha: ČVUT v Praze, 2004. s. 368-369. ISBN 80-01-02945-X.

[6] Arkin, R. C. *Behavior-based robotics: Intelligent robots and autonomous agents*. Cambridge: The MIT Press, 1999.

[7] Kadleček, D. *Simulace agenta-mobota ve virtuálním prostředí*. Diplomová práce. Praha: České vysoké učení technické v Praze, fakulta elektrotechnická, katedra kybernetiky, 2001. 75 s.

[8] Kočí, J. *Motivace akcí mobotů*. Diplomová práce. Praha : České vysoké učení technické v Praze, fakulta elektrotechnická, katedra kybernetiky, 2000. 59 s.

[9] Nahodil, P., Kadleček, D., Kohout, K., Svrček, A. Artificial Life Simulation. In *Proceedings of WORKSHOP 2003, 10.-12. 2. 2003, Praha, Česká Republika.* Praha: ČVUT v Praze, 2003. s. 364-365. ISBN 80-01-02708-2.

[10] Nahodil. P., Kadleček, D., Kohout, K., Svrček A. *Teorie a robotické aplikace umělého života*. Skripta ČVUT. Praha: ČVUT v Praze. 2004.

[11] Nahodil, P. Mobilní roboty s prvky umělého života. *Automatizace*. Říjen 2003, vol. 46, no. 10, s. 660 – 663.

[12] Nahodil, P. Petrus, M., Kadleček, D. Od inteligentních robotů k umělému životu. *Pražská technika*. Praha: ČVUT v Praze. 2002, číslo 1, s. 40 – 42. ISSN 1213-5348.

[13] Kadleček, D., Řehoř. D., Nahodil, P., Slavik, P., Kohout, K. Transparent visualization of multi-agent systems. In *Proceedings of 4th International Carpathian Control Conference*, *26.-29. 5. 2003, Vysoké Tatry*. s. 723 – 726. ISBN 80-7099-509-2.

[14] Kadleček, D., Řehoř, D., Nahodil, P., Slavik, P. Analysis of Virtual Agent Communities by Means of AI Techniques and Visualization. In *Lecture Notes in Artificial Inteligence – LNAI 2792*. Berlin: Springer Verlag, s. 274 – 282, ISBN 3-540-20003-7.

[15] Řehoř, D., Kadleček, D., Slavik, P., Nahodil, P. VAT - A New Approach for Multi-Agent Visualization. In *3rd IASTED International Conference on Visualization, Imaging and Image Processing, 8.-10. 9. 2003, Benalmadena, Spain*. Spain: ACME Press, s. 849 – 854. ISBN 0-88986-382-2.

[16] Kadleček, D., Nahodil, P. New Hybrid Architecture in Artificial Life Simulation. In *Lecture Notes in Artificial Intelligence No. 2159*, Berlin: Springer Verlag, 2001. s. 143-146.

[17] Parker, L. E. Current Research in Multi-Robot Systems, In *Journal of Artificial Life and Robotics*, vol. 7.

[18] Winfield, A. Distributed sensing and data collection via broken ad hoc wireless connected networks of mobile robots. In *Proceedings of Fifth International Symposium on Distributed Autonomous Robotic Systems, 2000*.

[19] Stone, P., Veloso, M. A layered approach to learning client behaviors in the robocup soccer server. *Applied Artificial intelligence, 1998*. s. 165 – 188.

[20] Weiss, G., Sen, S. *Adaptation and Learning in Multi-Agent Systems*. Springer, 1996.

[21] Balch, T., Arkin, R. C. Communication in reactive multiagent robotic systems. *Autonomous Robots*. 1994, s. 1-25.

[22] Stilwell, D., Bay, J. Toward the development of a material transport system using swarms of ant like robots. In *Proceedings of IEEE International Conference on Robotics and Automation, 1993*. s. 766 – 771.

[23] Arkin, R.C. Integrating behavioral, perceptual and world knowledge in reactive navigation. *Robotics and autonomous systems, 1990*. s. 105-122.

[24] Beni, G. The concept of cellular robot. In *Proceedings of Third IEEE Symposium on Intelligent Control, Arlington, Virginia, 1988*. s 57 – 61.

[25] Brooks, R. A. A robust layered control system for a mobile robot. In *IEEE Journal of robotics and automation*, *1986*. s. 12-23.

[26] *World Wide Web Consortium* [online]. Poslední revize 2004-5-5 [cit. 2004-3-15]. <http://www.w3c.org>.

[27] *XML Tutorial* [online]. Poslední revize 2004 [cit. 2004-3-15]. <http://www.w3schools.com/xml/default.asp>

[28] *Mobile Robots Group, ČVUT v Praze* [online]. Poslední revize 2004-1-21 [cit. 2004-4-20]. <http://cyber.felk.cvut.cz/gerstner/eth>.

[29] *Javadoc - The Java API Documentation Generator* [online]. Poslední revize 2002 [cit. 2004-4-16]. <java.sun.com/j2se/1.4/docs/tooldocs/solaris/javadoc.html>.

[30] *Web3D Consortium - Open Standards for Real-Time 3D Communication* [online]. Poslední revize 2004-5-4 [cit. 2004-3-20]. <http://www.web3d.org>.

[31] *ISO/IEC 14772-1:1997 Information technology -- Computer graphics and image processing -- The Virtual Reality Modeling Language -- Part 1: Functional specification and UTF-8 encoding* [online]. Poslední revize 2004-5-4 [cit. 2004-3-20]. <http://www.web3d.org/x3d/specifications/vrml/>.

[32] *ISO/IEC 14772-2:2004 Information technology -- Computer graphics and image processing -- The Virtual Reality Modeling Language (VRML) -- Part 2: External authoring interface (EAI)* [online]. Poslední revize 2004-5-4 [cit. 2004-3-20]. <http://www.web3d.org/x3d/specifications/vrml/>.

[33] *BS Contact VRML* [online]. Poslední revize 2004 [cit. 2004-12-5]. <http://www.bitmanagement.de/>.

[34] *Blaxxun technologies - Blaxxun Contact* [online]. Poslední revize 2004 [cit. 2004-12-5]. <http://www.blaxxun.com/en/products/contact/index.html>.

[35] *StarLogo on the Web* [online]. Poslední revize 2002-9-27 [cit. 2003-12-5]. <http://www.media.mit.edu/starlogo/>.

[36] *The Swarm Development Group* [online]. Poslední revize 2004-4-8 [cit. 2003-12-5]. <http://wiki.swarm.org>.

[37] *Natural Resources and Multi-Agent Simulations* [online]. Poslední revize 2003-11-20 [cit. 2003-12-5]. <http://cormas.cirad.fr/indexeng.htm>.

[38] *XRaptor: Simulation Environment for Multi-Agent Systems* [online]. Poslední revize 2003-4-13 [cit. 2003-12-5]. <http://www.xraptor-simulator.de.vu/>.

[39] Komosinski, M., Ulatowski, S., *Framsticks – Artificial Life – 3D evolution and simulation* [online]. Poslední revize 2004 [cit. 2004-12-5]. <http://www.frams.alife.pl/index.html>.

[40] Boldiš, P. *Bibliografické citace dokumentů podle ČSN ISO 690 a ČSN ISO 690-2 (01 0197): Část 1 – Citace: metodika a obecná pravidla* [online]. Verze 3.2. c 1999–2002, poslední aktualizace 3.9. 2002 [cit. 2004-5-5]. <http://www.boldis.cz/citace/citace1.pdf>.

[41] Boldiš, P. *Bibliografické citace dokumentů podle ČSN ISO 690 a ČSN ISO 690-2 (01 0197): Část 2 – Modely a příklady citací u jednotlivých typů dokumentů* [online]. Verze 2.5 (2002). 1999–2002, poslední aktualizace 3. 9. 2002 [cit. 2004-5-5]. <http://www.boldis.cz/citace/citace2.pdf>.

[42] Kolář, D. *Skupina robotů-animátů se společným cílem*. Diplomová práce. Praha: České vysoké učení technické v Praze, fakulta elektrotechnická, katedra kybernetiky, 2004. Vedoucí diplomové práce Doc. Ing. Nahodil Pavel, CSc. Rozpracovaný rukopis – stav duben 2004.

[43] Foltýn, L. *Realizace architektury inteligentních agentů s prvky umělého života*. Diplomová práce. Praha: České vysoké učení technické v Praze, fakulta elektrotechnická, katedra kybernetiky, 2005. Vedoucí diplomové práce Doc. Ing. Nahodil Pavel, CSc. Rozpracovaný rukopis – stav březen 2004.

[44] Altmann, J. *Porovnání kvality chování agenta s prvky umělého života s jinými typy agentů*. Diplomová práce. Praha: České vysoké učení technické v Praze, fakulta elektrotechnická, katedra kybernetiky, 2005. Vedoucí diplomové práce Doc. Ing. Nahodil Pavel, CSc. Rozpracovaný rukopis – stav březen 2004.

[45] Svoboda, J. *Adaptivní chování agentů-animátů s prvky umělého života*. Diplomová práce. Praha: České vysoké učení technické v Praze, fakulta elektrotechnická, katedra kybernetiky, 2005. Vedoucí diplomové práce Doc. Ing. Nahodil Pavel, CSc. Rozpracovaný rukopis – stav duben 2004.

[46] Mach, M. *Mechanismus dobývání znalostí o prostředí na základě chování a funkčnosti jeho dílčích objektů*. Diplomová práce. Praha: České vysoké učení technické v Praze, fakulta elektrotechnická, katedra kybernetiky, 2005. Vedoucí diplomové práce Doc. Ing. Nahodil Pavel, CSc. Rozpracovaný rukopis – stav duben 2004.

[47] Duda, F. *Roboti-animáti v jednoduchém prostředí společného cíle*. Diplomová práce. Praha: České vysoké učení technické v Praze, fakulta elektrotechnická, katedra kybernetiky, 2005. Vedoucí diplomové práce Doc. Ing. Nahodil Pavel, CSc. Rozpracovaný rukopis – únor 2004.

[48] *Foundation for Intelligent Physical Agents* [online]. Poslední revize 2003 [cit 2004-2-20]. <http://www.fipa.org>.

[49] Eclipse Foundation [online]. Poslední revize 2003-4-27 [cit 2004-4-17]. <http://www.eclipse.org/>.

[50] *Java Technology* [online]. Poslední revize 2004 [cit. 2004-4-20]. <http://java.sun.com/>.

[51] *Unified Modeling Language* [online]. Poslední revize 2004-3-29 [cit. 2004-2-13]. <http://www.uml.org/>.

Dodatek A - Seznam příloh na CD

Veškeré zdrojové kódy, konfigurace, grafická knihovna i dokumentace kódu se nacházejí v adresáři ***WAL***. Programátorská dokumentace v podobě `JavaDoc` se nachází v adresáři ***WAL/doc***. Zdrojové kódy jsou umístěny v adresáři ***WAL/src***.

Tuto práci jsem naprogramoval v prostředí ***Eclipse*** volně šiřitelným pod Eclipse Public License v 1.0 (EPL) [49],[P16] v jazyce Java [50], [P17].

Text této práce ve formátu Microsoft Word (*.doc*) a Adobe Acrobat Reader (*.pdf*) se nacházejí v adresáři ***Text***.

Následující seznam popisuje aplikace a nástroje, které jsem použil při tvorbě této práce.

Příloha č. 1 – [P01] Other/FIPA/SC00001L.pdf
Příloha č. 2 – [P02] .....Other/Alife/Alife aplikace.doc
Příloha č. 3 – [P03] .....Other/Cortona/cortvrml.exe
Příloha č. 4 – [P04] Other/ZS/04.03.2004.wav
Příloha č. 5 – [P05] Other/ZS/24.2.2004.wav
Příloha č. 6 – [P06] Other/ZS/04.03.2004.doc
Příloha č. 7 – [P07] Other/ZS/10.2.2004.doc
Příloha č. 8 – [P08] Other/ZS/24.2.2004.doc
Příloha č. 9 – [P09] .....Other/VRML ISO/ISO_IEC_14772-All.zip
Příloha č. 10 – [P10] Other/VAT/Level1/vizualizer.exe
Příloha č. 11 – [P11] Other/VAT/Level1-1/vizualizer.exe
Příloha č. 12 – [P12] Other/VAT/Level2/vizualizer.exe
Příloha č. 13 – [P13] Other/VAT/Level3/vizualizer.exe
Příloha č. 14 – [P14] Other/VAT/Level3/vizualizer.exe
Příloha č. 15 – [P15] .....Other/EasyPHP/easyphp1-7_setup.exe
Příloha č. 16 – [P16] Other/EclipseSDK/eclipse-SDK-2.1.2.zip
Příloha č. 17 – [P17] .....Other/Java/j2sdk-1_4_1_02.exe
Příloha č. 18 – [P18] .....Other/Java/java3d-1_3_1.exe
Příloha č. 19 – [P19] .....Other/UML/UML Specification.pdf
Příloha č. 20 – [P20] Other/Xj3D/Xj3D-M8-full.exe
Příloha č. 21 – [P21] .....Other/XML/jdom.jar

Dodatek B - Seznam obrázků

Dodatek C - Seznam tabulek