Umelý život a digitálna evolúcia

Artificial Life And Digital Evolution

Bc. Jozef Fekiač

Diplomová práce
2009

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2008/2009

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jozef FEKIAČ**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Umělý život a digitální evoluce**

Zásady pro vypracování:

Cílem práce je systematicky shrnout současný stav na poli umělého života a vytvoření vlastního prostředí demonstrujícího základní atributy systému s umělým životem.:

1. **Vypracovat přehled problematiky umělého života.**
2. **Navrhnout vlastní prostředí.**
3. **Provést programovou realizaci.**
4. **Vyhodnotit.**

Rozsah práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. LEVY, S., Artificial Life. Vintage, 1993. ISBN 06-797-4389-8.
2. KOMOSINSKI, M., ADAMATZKY, A., Artificial Life Models in Software. Springer, 2005. ISBN 18-523-3945-4.
3. HEATON, J., Introduction to Neural Networks with Java. Heaton Research, 2005. ISBN 09-773-2060-X.
4. ZELINKA, I., Umělá inteligence I. Volume 1. Zlín : Vutium, Brno, 1998. ISBN 80-214-1163-5.
5. KVASNIČKA V., POSPÍCHAL J., TIŇO P., Evolučné algoritmy, STU Bratislava, 2000. ISBN 85-246-2000, 2000.

Vedoucí diplomové práce: **doc. Ing. Ivan Zelinka, Ph.D.**
Ústav aplikované informatiky

Datum zadání diplomové práce: **20. února 2009**

Termín odevzdání diplomové práce: **27. května 2009**

Ve Zlíně dne 13. února 2009

L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Táto práca obsahuje tri základné časti – históriu umelého života, prehľad existujúcich simulačných prostredí v oblasti umelého života a návrh nového simulačného prostredia. V prvej časti sa práca zaoberá dôležitými míľnikmi v oblasti umelého života, v druhej časti vybranými aplikáciami z tejto oblasti. V časti venovanej vývoju popisuje použitie vyvinutého programu a spôsob jeho fungovania. Okrem toho práca obsahuje teoretický základ pre uvedenie do problematiky umelého života a vývoja serverovej aplikácie.

Kľúčové slová: umelý život, digitálna evolúcia, simulácia, umelá inteligencia

ABSTRACT

This thesis contains three basic sections – the history of artificial life, the review of existing simulation environments in the field of artificial life and the development of a new simulation environment. In the first section the thesis deals with important milestones in the area of artificial life, in the second part it discusses selected applications from this area. In the section on development it describes usage of the developed application and its concept of operation. Moreover the thesis contains a theoretical basis of the area of artificial life and developing a server application.

Keywords: artificial life, digital evolution, simulation, artificial intelligence

Chcel by som poďakovať vedúcemu mojej diplomovej práce doc. Ing. Ivanovi Zelinkovi Ph.D. za rady pri práci a pomoc pri zaobstarávaní použitej literatúry, ako aj blízkym, ktorí ma podporili pri tvorbe tejto práce.

Motto:

Každý, kto sa pozrie na žijúce organizmy, vie, že dokážu produkovať ďalšie organizmy také ako oni sami. Toto je ich prirodzený účel, neexistovali by, keby to nerobili a je pravdepodobné, že toto je dôvod, prečo vo svete existujú. Inými slovami, živé organizmy sú veľmi komplikovaným spojením základných častí, a podľa akejkoľvek prijateľnej teórie pravdepodobnosti alebo termodynamiky vysoko nepravdepodobné. To, že sa vôbec môžu vo svete vyskytnúť je zázrak najvyššieho stupňa...

John von Neumann

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.

Ve Zlíně …….

Podpis diplomanta

OBSAH

ÚVOD

Umelý život je zameraný na vytváranie a štúdium organizmov a systémov vytvorených človekom, ktoré majú vlastnosti ako živé organizmy a systémy. Médiom tohto života je neorganický materiál, jeho princípom sú informácie a počítače sú prostriedky, v ktorých sa tieto nové organizmy vyskytujú. Tak, ako sa klasickí biológovia pracujú so živými mechanizmami *in vitro* (v skúmavke), biológovia a počítačoví odborníci zameraní na umelý život sa snažia vytvoriť život *in silico* (na kremíkovom základe mikročipov) s využitím znalostí z ostatných vedných disciplín.

Práve miešanie výpočtovej techniky, filozofie a biológie (v mnohých prípadoch aj chémie, fyziky, mechaniky a iných vied) robí z oblasti umelého života jedno z najširšie zaberajúcich vedeckých odvetví. Táto práca je zameraná na umelý život *in silico* v pravom zmysle slova – na počítačové simulácie, kde počítač tvorí organizmy aj prostredie v ktorom sa pohybujú.

V tomto duchu je zostavený prehľad míľnikov umelého života v minulosti, prehľad viac či menej aktuálnych simulačných prostredí v prítomnosti, aj simulačné prostredie, ktoré by malo týmto prostrediam konkurovať v budúcnosti.

I. TEORETICKÁ ČASŤ

1 ŽIVOT A UMELÝ ŽIVOT

Ako bolo uvedené vyššie, umelý život je snaha o vytvorenie bytostí na neorganickom základe, ktoré sa svojimi vlastnosťami podobajú žijúcim organizmom. Už táto snaha ale vyžaduje definíciu života, aby bolo možné určiť, ktoré vlastnosti a znaky je potrebné napodobniť, aby vytvorenú entitu bolo možné pokladať za organizmus, nie len za stroj alebo predmet. Týchto vlastností je mnoho a podľa rôznych vedcov, vedeckých disciplín či filozofií charakterizujú živú bytosť rôzne podmnožiny vlastností organizmov.

1.1 Vlastnosti života

Aj z dôvodu rôznych názorov na túto problematiku, štúdie umelého života zvyčajne vykresľujú len jeden, alebo niekoľko málo *aspektov života*. Podľa J.Moralesa [3] existuje 6 základných prejavov života, z ktorých väčšina je ale samostatne spochybniteľná.

- **Reprodukcia**
  Všetky organizmy sú schopné reprodukovať sa. Mnohobunkové organizmy sa dokonca reprodukujú na viacerých úrovniach – bunky aj jedinci.
  Bohužiaľ, napríklad aj oheň sa dokáže reprodukovať, napriek tomu ale nie je pokladaný za živý.

- **Metabolizmus**
  Živé organizmy čerpajú materiál z prostredia, spracúvajú ho a vracajú nepoužitý materiál späť do prostredia.
  V tomto prípade je možné opäť použiť oheň na spochybnenie tohto znaku života – oheň sviečky prijíma z prostredia knôt, ktorý tepelne spracuje a do prostredia okrem tepla vráti odpad (napr. oxid uhličitý). Naopak, vírusy, ktoré sú všeobecne pokladané za živé nemajú žiadny metabolizmus.

- **Odozva na prostredie**
  Organizmy reagujú na zmeny v prostredí takým spôsobom, aby podporili svoje pokračovanie (vlastné prežitie alebo prežitie druhu).
  Jednoduchým protikladom je napríklad človek spáchajúci samovraždu. Hoci ide pravdepodobne o poruchu, musel by byť človek, ktorý sa rozhodne spáchať samovraždu, pokladaný podľa tohto tvrdenia za neživého (samozrejme predtým ako sa mu to podarí).

- **Pokračovanie prostredníctvom iných**
  Keďže jednotlivec nemôže žiť večne, vytvára potomstvo podobné sebe, aby tak prežil prostredníctvom neho. Tento bod úzko súvisí z rozmnožovaním, no rozširuje ho o ochranu potomstva aj za cenu straty vlastného života.
  Tento rys sa prejavuje typicky u včiel, ktoré ochraňujú úľ s kráľovnou a mláďatami, ale pritom po prvom bodnutí žihadla umrú. Dokonca je možné toto správanie prirovnať k samovražedným misiám v histórii ľudstva.

- **Homeostáza**
  Okrem toho, že sa organizmus snaží reagovať čo najprínosnejšie na zmeny prostredia, snaží sa podobne reagovať aj na zmeny vo svojom vnútri.

- **Organizácia**
  Organizmus je chápaný ako súbor prvkov (bielkoviny u jednobunečných organizmov, bunky u mnohobunečných), ktoré nemôžu byť náhodne zmiešané, ale majú daný poriadok, podľa ktorého sú zorganizované.

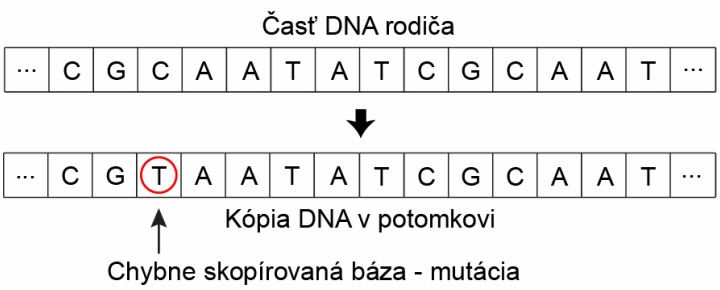
Podľa iných zdrojov ([2]) je vlastnosťou organizmov aj *evolúcia*, resp. javy, ktoré ju spôsobujú. Javy spojené s evolúciou – *mutácia a prirodzený výber* – však nie sú vyvolané priamo organizmami, ale sú to vlastnosti prostredia a chyby z nich vyplývajúce, ktoré podporujú život a vďaka ktorým život môže fungovať.

1.2 Mutácia

Mutácia je chyba spôsobená nedokonalosťou informačného materiálu organizmu (DNA, RNA) alebo reprodukčných orgánov, ktorá spôsobí, že po replikácii tohto materiálu potomok nie je zhodný s rodičom. Vďaka tomu je možný neustály vývoj živých organizmov podľa Darwinovej evolučnej teórie.

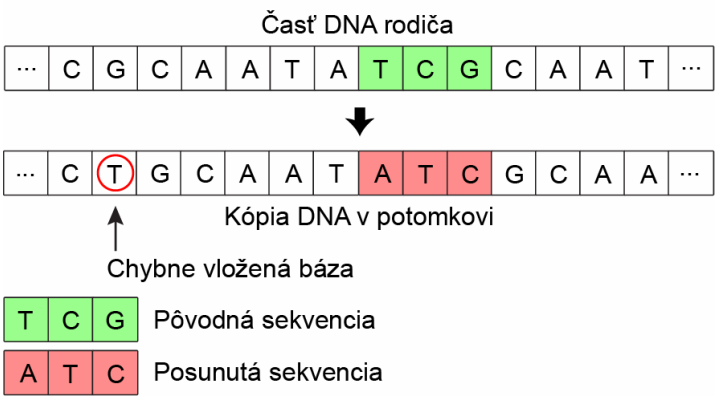
Existujú tri typy mutácie. Najčastejšie sa vyskytuje jednoduché prepísanie jednej informácie (v prípade DNA je to prehodenie jednej bázy). Táto mutácia spôsobuje spravidla malé zmeny v organizme (resp. v jeho fenotype[1]).

[1] Fenotyp = realizácia, prejavenie sa informácie v génoch



Obrázok 1 - Mutácia - jednoduché prepísanie informácie

Dva zvyšné typy mutácie sú pridanie alebo odobranie informácie. Obidve vytvoria tzv. posunutie čítacieho rámca. Informácie v DNA sú závislé jedna na druhej – sekvencie báz teda predstavujú komplexnejšiu informáciu. Vložením bázy do tejto sekvencie alebo jej odstránením sa zmení celá sekvencia, čo môže mať za následok veľké zlepšenie, ale aj zhoršenie fenotypu.



Obrázok 2 - Mutácia - chybne vložená informácia

1.3 Prirodzený výber

Mutácia však nie je jedinou podmienkou evolúcie. Z pôvodných aj chybných (zmutovaných) organizmov vyberá tie najschopnejšie mechanizmus *prirodzeného výberu*. Prirodzený výber nie je žiadne pravidlo určujúce, ktorý jedinec prežije a ktorý nie. Je to abstraktný pojem, ktorý vyjadruje náročnosť a súťažichtivosť prostredia. Netreba zabúdať, že organizmus nemôže fungovať samostatne. Každý živý organizmus žije len v určitom prostredí, ktoré je tvorené neživými objektami, ako aj ďalšími živými organizmami. Toto prostredie a jeho náročnosť určuje v dlhodobom merítku, ktorý jedinec (teda ktorá postupnosť mutácií) je výhodnejšia.

1.4 Životné prostredie

Pre prostredie (keďže je tvorené z veľkej časti živými organizmami) sa dajú odvodiť podobné mechanizmy ako pre samotné organizmy. Analógie *metabolizmu*, *homeostázy* aj *organizácie* sa dajú nájsť v celom životnom prostredí nám známych organizmov – od najmenších ekosystémov až po celú planétu. Táto myšlienka je už dlhé storočia známa v rôznych častiach sveta – pokladá celú planétu za jeden obrovský organizmus (Gaia).

Lynn Margulisová ([4]) prirovnáva planétu a jej jednotlivé ekosystémy k mnohobunečným organizmom a naopak, mnohobunečné organizmy k ekosystémom. Tak ako nemôžu žiť organizmy v jednom ekosystéme v rovnovážnom stave jeden bez druhého, nemôžu ani bunky mnohobunkového organizmu žiť v rovnovážnom stave bez buniek rovnakých, ani odlišných. Margulisová pripisuje už samotný vznik eukaryotických buniek[2] symbióze viacerých proaryotických buniek[3], ktoré tvoria organely eukaryotickej bunky. To by znamenalo, že mnohobunkové organizmy sú zložené z organizmov, ktoré sú zložené z menších organizmov – a tým sa stiera hranica nie len medzi organizmom a prostredím, ale aj medzi samotnými organizmami. Organizmom je teda v tejto práci uvažovaná akákoľvek *skúmaná entita*. Jej okolie je tým pádom považované za životné prostredie.

1.5 Multi-agent systémy

Pri návrhu počítačových simulácií živých organizmov sa často využívajú tzv. *multi-agent systémy*, resp. tieto systémy sú odvodené z pozorovania živých organizmov.

Multi-agent systémy sú systémy zložené z množstva interagujúcich výpočtových prvkov, známych ako *agenti*. Agenti sú počítačové systémy s dvomi dôležitými schopnosťami. V prvom rade sú do určitej miery schopný samostatnej akcie – rozhodovania v svojom záujme, čo je potrebné spraviť pre dosiahnutie ich navrhnutých cieľov. Okrem toho sú schopné interakcie s ďalšími agentami – nie len jednoduchou výmenou dát, ale zúčastňovaním sa podobných aktivít, akých sa v živote zúčastňujeme každý deň: kooperácie, koordinácie, obchodovania apod. [1]

---

[2] Eukarytické bunky = jadrové bunky – okrem iného základ mnohobunkových organizmov

[3] Prokaryotické bunky = bezjadrové, najjednoduchšie bunky

1.6 Umelý život

Ako bolo uvedené vyššie, umelý život sa okrem iného snaží reprodukovať zákony evolúcie. Taktiež sa disciplína umelého života snaží tieto vedomosti využívať za rôznymi účelmi – od optimalizácie matematických problémov, po nové liečebné metódy.

Okrem toho sú využívané rôzne aspekty života, ktoré nepatria priamo k jeho základným vlastnostiam. Jednou z nich je *prekríženie* chromozómov, ktoré sa vyskytuje len u pohlavne sa rozmnožujúcich organizmov. Prekríženie zjednodušene nastáva, keď sa spojí chromozóm jedného partnera s chromozómom druhého partnera, prekrížia sa na náhodnom mieste a opäť sa oddelia. Výsledkom sú dva chromozómy, z ktorých každý má podmnožinu vlastností obidvoch rodičov a dôsledkom je vyššia pestrosť populácie.

Umelý život sa často spája a mýli s umelou inteligenciou. Tieto dve oblasti ale majú spoločného asi toľko, ako skutočný život so skutočnou inteligenciou. Hoci ani jedna z týchto oblastí nie je presne definovateľná, je zrejmé, že inteligencia je len jedným z možných dôsledkov života. Organizmus nemusí byť inteligentný, aby mohol byť živý. No aj napriek tomu sa mnohokrát metódy umelého života pokladajú za umelú inteligenciu, pretože optimalizačné techniky, ktoré sú od nich odvodené dokážu často riešiť zložité problémy, alebo urýchliť vývoj umelej inteligencie.

Tieto metódy a techniky života *in silico* sú popísané v nasledujúcej kapitole spolu s osobnosťami, ktoré sa o ich vznik zaslúžili.

2 HISTÓRIA

V minulosti sa nezdokonaľovalo len niekoľko málo prístupov a techník – každá významná osobnosť na poli umelého života vytvorila nový pohľad na umelý život (a často aj na život ako taký). To dokazuje, že život je takmer nemožné definovať, ale zároveň ponúka množstvo metód využiteľných v *optimalizácii* a *simuláciach* ľubovoľného druhu.

2.1 Bunečné automaty (Cellular Automata)

John Von Neumann je nepochybne jeden z najväčších mysliteľov v oblasti výpočtovej techniky. Okrem prínosu v konštrukcii počítačov bol prvý, kto sa zaoberal *samoreprodukciou* strojov. Snažil sa dokázať, že stroj sa dokáže rozmnožovať, teda vytvárať funkčné kópie seba samého.

Jeho teoretický stroj schopný samoreprodukcie pozostával z niekoľkých symbolických častí. Prvá časť pripomínala fabriku – dokázala *ťažiť materiál z prostredia* a spracovávať ho podľa inštrukcií z druhej časti. Druhá časť čítala *informačné inštrukcie* a *kopírovala* ich. Ďalšou časťou bol kontrolný počítač. Poslednou časťou bola veľmi dlhá *páska* inštrukcií zostavená z trámov poskladaných do tvaru píly. Vrchol píly preťatý kolmým trámom predstavoval hodnotu „1“, v opačnom prípade bola na spracovanom vrchole hodnota „0“. Replikácia bola naštartovaná načítaním a skopírovaním postupnosti trámov. Podľa trámov zostavila fabriková časť nového jedinca, ktorému túto postupnosť trámov predala. Od tejto chvíle bol aj potomok schopný reprodukcie. Tieto stroje sa, žiaľ, nikdy nepodarilo zostrojiť.[4]

Alan Turing, ktorý tak isto do veľkej miery prispel oblasti výpočtovej techniky, sa zaslúžil o vytvorenie mechanizmu *konečných automatov*. Konečný automat je zariadenie, ktoré sa môže nachádzať v jednom z konečného počtu stavov podľa práve spracovaného vstupu. Turing sa ale zameriaval na umelú inteligenciu a aplikáciu konečných automatov na ňu.

Von Neumann v spolupráci so Stanislawom Ulamom využili teóriu konečných automatov v prospech umelého života. Predpokladali, že svet je diskrétny – že najmenšie stavebné

[4] Podľa R.Lainga by organizmus schopný samoreprodukcie mohol byť evolučným súperom ľudstva a dokonca sa stať jeho prirodzeným nepriateľom. Hollywoodske vedecko-fantastické filmy sa vďaka tomuto tvrdeniu stávajú menej fantastickými.

prvky vesmíru sú oddeliteľné a zmeny nastávajú v diskrétnych časových krokoch (nech sú akokoľvek malé). Prostredie organizmov bolo tvorené poľom konečných automatov (každý sa mohol nachádzať v jednom z dvadsať deväť stavov) a spolu vytvorili tzv. *bunečný automat*. Organizmus bol potom tvorený súborom týchto konečných automatov (*buniek*). Bunečný automat o dvadsiatich deviatich stavoch vznikol len ako matematický model, pretože jeho realizácia bola príliš náročná.

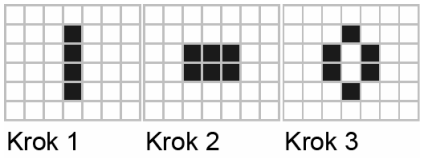
2.2 Hra života (Game of Life)

John H. Conway nadviazal na výskum Von Neumanna a Ulama s tým, že sa snažil zjednodušiť model bunečných automatov tak, aby ho bolo možné realizovať najprv na obrovskej šachovnici, potom aj na počítačovej simulácii halových počítačov.

Preto zredukoval možné stavy buniek automatu na dva (žije – nežije, 1 – 0) a pre zmenu stavu z jedného do druhého platili jednoduché pravidlá. Keďže prostredím je štvorcová sieť (šachovnica), každá bunka môže mať osem susedov.

> *Ak je bunka na šachovnici živá, prežije v ďalšom kroku (generácii), ak sú v okolí živé dve alebo tri ďalšie bunky. Zomrie v dôsledku premnoženia, ak existujú viac ako traja živí susedia, a zomrie v dôsledku vyhynutia, pokiaľ má menej ako dvoch susedov.*
>
> *Ak je bunka na šachovnici mŕtva, zostane v ďalšej generácii mŕtva, pokiaľ nie sú živí presne traja z ôsmich susedov. Pokiaľ sú, bunka sa v ďalšej generácii „narodí“.*[6]

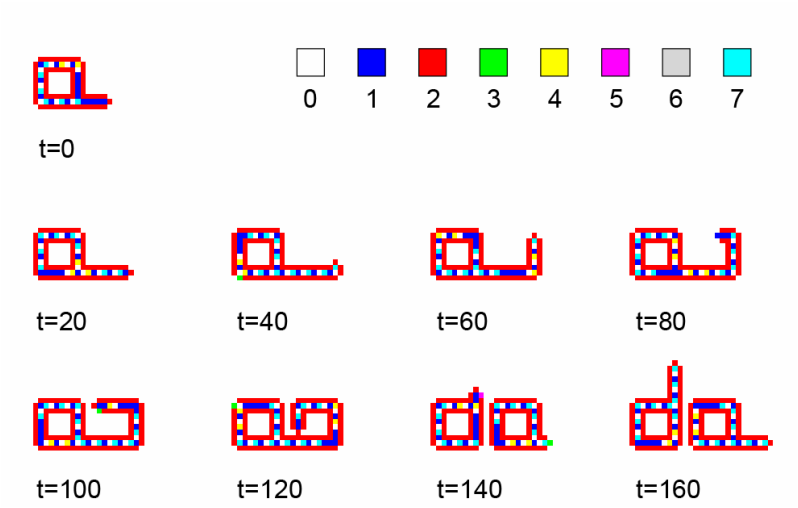
V simulácii vykonávanej ručne na šachovnici v kancelárii (z toho názov *Game of Life*) sa po niekoľkých iteráciách objavili organizované vzory (niektoré statické, niektoré viacfázové, iné pohyblivé), ktoré priamo z pravidiel hry nevyplývali. Jedným z nich je včelí uľ – *beehive* (Obrázok 3).



Obrázok 3 - Ustálenie hry života po troch krokoch vo formácii beehive

2.3 Langtonove slučky (Q-Loops)

Chris Langton opäť zvýšil množstvo stavov polí bunečného automatu (na osem) a vytvoril štruktúru, ktorá bola schopná kopírovať sa podobným spôsobom, ako to robia bunky. Táto štartovacia štruktúra mala v grafickom znázornení tvar písmena Q, preto sa Langtonovej sľučke často hovorí *Q-sľučka*. Jednotlivé polia v tejto sľučke a konfigurácia ich stavov predstavujú membránu sľučky a zakódovanú genetickú informáciu.



Obrázok 4 - Zmena stavov Langtonovho bunečného automatu v čase

Langtonove sľučky ale nie sú jediné – samorozmnožovacích sľučiek bolo vytvorených niekoľko druhov, od veľkých[5] až po minimálne[6], tvorené len niekoľkými políčkami.

Okrem toho vznikli verzie, v ktorých bola do pravidiel bunečného automatu pridaná mutácia (pri zmene stavu automatu je šanca, že nastane chyba). Vďaka tomu sa podľa princípov evolúcie vyvinuli ďalšie formy sľučiek, ktoré sa svojou kvalitou (rozmermi a efektivitou reprodukcie) rovnali pracne navrhnutým sľučkám.

[5] Perrierova sľučka – 158 políčok, 64 stavov automatu

[6] Chou-Reggiova sľučka – 5 políčok, 8 stavov automatu

2.4 Genetické algoritmy

John Holland, prvý americký doktor v odbore počítačovej vedy, dokázal využiť evolúciu k praktickejším cieľom ako optimalizovať veľkosť sľučky na šachovnici. Vďaka jeho *genetickým algoritmom* a algoritmom z nich odvodených je možné optimalizovať obrovskú škálu matematických problémov.

Genetický algoritmus využíva kódovanie informácií do chromozómov podobne ako živé organizmy. Na rozdiel od nich však nerozlišuje medzi genotypom a fenotypom. Reťazec chromozómu v genetickom algoritme predstavuje samotného jedinca – predstavuje inštrukcie programu, parametre výpočtu apod.

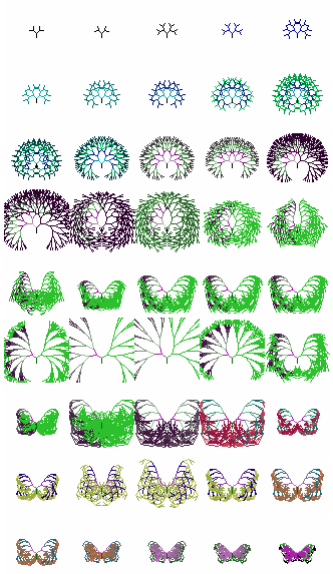
Genetické algoritmy, úspešne používané s mnohými vylepšeniami dodnes, využívajú mutácie a prekríženie chromozómov. Namiesto prirodzeného výberu však využívajú neprirodzený výber – je ním *účelová funkcia* (fitness function), tj. optimalizačná funkcia, pretože priamo určuje, ktorý jedinec ako dobre vyhovuje podmienkam.

2.5 Biomorfy

Evolučný biológ Richard Dawkins využil koncept genetických algoritmov a neprirodzeného výberu, aby vytvoril ilustrovaný priebeh procesu evolúcie. Jeho pôvodný program prekladal gény jedincov (*biomorfov*) do kresliacich inštrukcií a pomocou nich vykresľoval organizmus na obrazovku. Najväčší rozdiel oproti genetickým algoritmom predstavovala účelová funkcia. V prípade biomorfov určoval kvalitu jedinca používateľ. Ten totiž na obrazovke videl celú generáciu, z ktorej vyberal tých najlepších (podľa vlastných estetických kritérií – tzv. *breeding*).

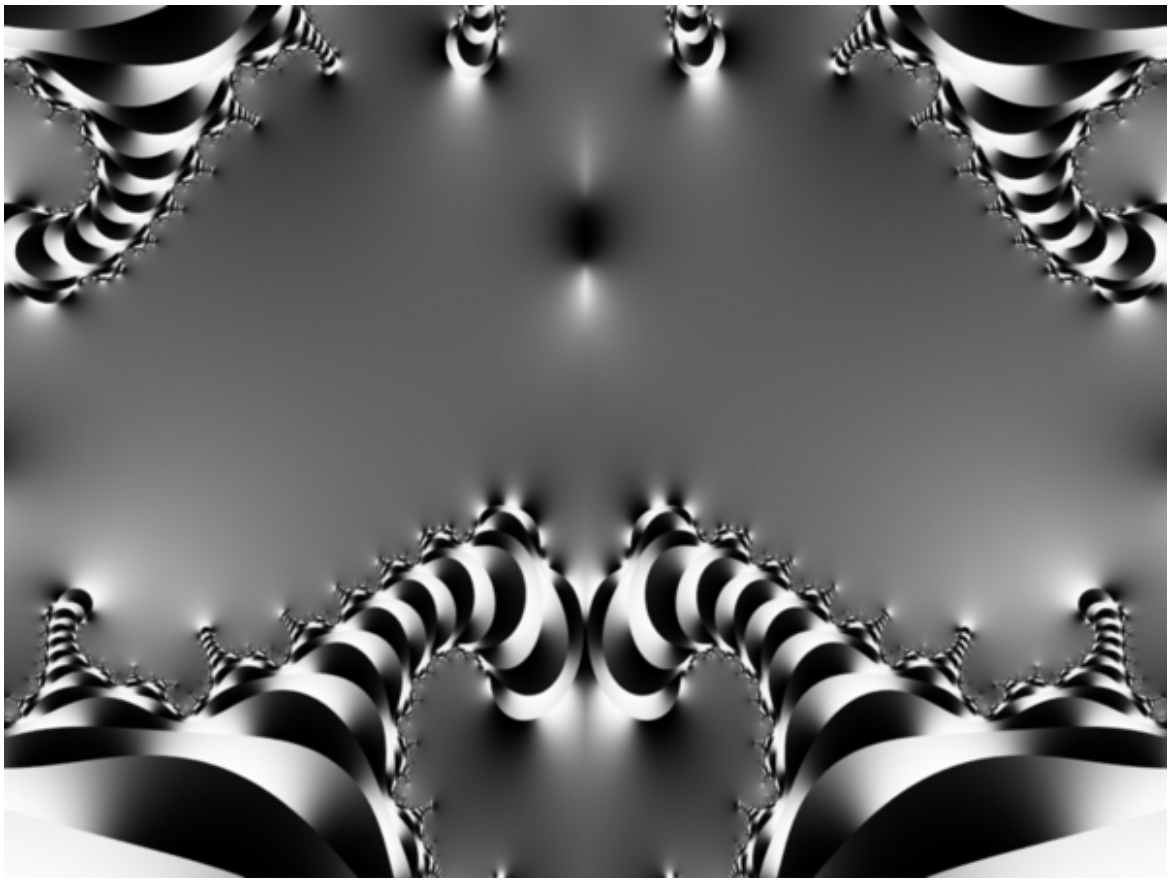
Program navrnutý Dawkinsom mal podľa očakávaní vytvárať len jednoduché stromové konštrukcie, no po menej ako tridsať iteráciách sa na obrazovke objavila konštrukcia podobná hmyzu.

Dawkinsonove biomorfy odštartovali éru *genetického umenia*, ktoré predstavuje úzku spoluprácu umelca s riadeným genetickým algoritmom. Umelec nevytvára priamo svoje dielo, ale prechádza procesom evolúcie kresliacich operácií a efektov, kým nie je s dielom spokojný. V tomto procese sú rovnocenní partneri počítač, aj umelec, pretože vytvorenie diela by nebolo možné ani bez jedného z nich.



Obrázok 5 - Biomorf podobný motýľovi, vyvinutý po 45 iteráciách

[zdroj: http://elegans.uky.edu/jiml/bm/gallery/066868468261422.gif]



Obrázok 6 - Amal – ukážka genetického umenia

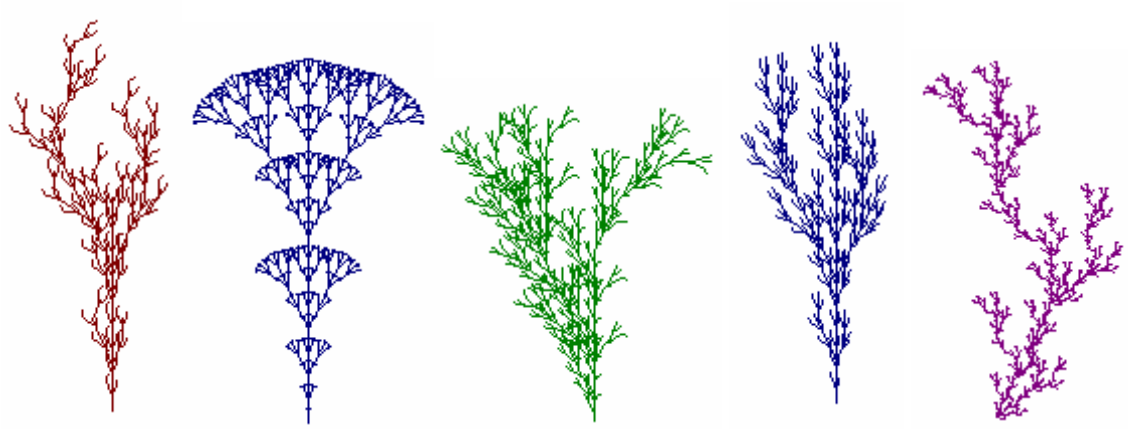
[prevzaté z: http://steike.com/tech/genetic-art, autor: Kyrre Glette]

2.6 L-systémy

O to, že je možné metódy umelého života využiť v umení, sa pričinil aj Aristid Lindenmeyer. *Lindenmeyerov systém* (L-systém) funguje na princípe *formálnej gramatiky* a je prevažne používaný na ilustráciu rastu rastlín, prípadne iných zložitých štruktúr, pre ktoré je potrebný jednoduchý zápis. L-systém teda prepisuje postupne časti reťazca na iné reťazce podľa daných pravidiel, napr.:

> *Pravidlo 1: z **a** v tomto kroku sa stane **ab** v nasledujúcom kroku*
> *Pravidlo 2: z **b** v tomto kroku sa stane **a** v nasledujúcom kroku*
>
> *Proces začína počiatočným reťazcom (axiom), v tomto prípade reťazcom dvoch znakov **ab**. Po prvom časovom kroku bol reťazec transformovaný do **aba** (použitie pravidla 1 na **a** vrátilo **ab**, použitie pravidla 2 na **b** vrátilo **a**). Ďalší krok zmenil **aba** na **abaab**. Nasledujúci krok vrátil **abaababa**, pomerne komplexný výsledok na tri tiky procesoru.* [6]

Nasledovníci Lindenmeyera previedli výsledný reťazec do kesliacich inštrukcií, vďaka čomu je možné jednoduchými zápismi popísať zložité štruktúry, tvarovo pripomínajúce rastliny.



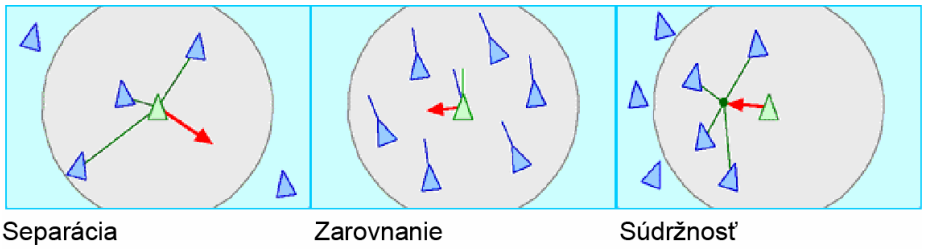
Obrázok 7 - Vizualizácia reťazcov vygenerovaných pomocou L-systému

[prevzaté z: http://www.biologie.uni-hamburg.de/b-online/e28_3/lsys.html]

2.7 Húfy, kŕdle a stáda (Boids)

Animátor Craig Reynolds sa zameral na simuláciu správania spoločenských živočíchov. Neskúmal už tie najzákladnejšie princípy života, ale využil znalosti bunečných automatov a toho, že na základe jednoduchých pravidiel a blízkeho okolia môžu vzniknúť komplexné a organizované vzory.

Po dlhodobom pozorovaní kŕdľov[7] vtákov vyvodil Reynolds tri jednoduché pravidlá, ktoré každý virtuálny vták (*boid* – z anglického birdoid) dodržiaval: *separácia* – boid sa snažil oddialiť od svojich najbližších susedov tak, aby zabránil prehusteniu; *zarovnanie* – boid sa snažil otočiť a prispôsobiť svoju rýchlosť tak, ako okolití boidi; *súdržnosť* – boid sa snažil dostať do stredu svojich susedov (Obrázok 8). Tieto tri jednoduché pravidlá sú realizované základnou vektorovou algebrou a ich výsledkom je komplexné správanie natoľko pripomínajúce skutočné kŕdle a húfy, že sa tieto techniky využívajú vo filmovej animácii[8].



Obrázok 8 - Pravidlá pre pohyb boidov

[prevzaté z http://www.red3d.com/cwr/boids/]

2.8 Vírusy

Do oblasti umelého života bohužiaľ patria aj počítačové *vírusy*. Autorom pravdepodobne prvého samostatne šíriteľného vírusu je Fred Cohen. Ako prvý využil samoreprodukciu programu k tomu, aby dokázal rozšíriť vírus do ďalších a ďalších programov. Tieto programy pre neho predstavovali jedince žijúce vo virtuálnom prostredí. Virtuálnym životným prostredím bol počítač VAX 11-750 s operačným systémom UNIX na Univerzity of South California, ktorý zdieľalo zhruba päťdesiat ľudí.

---

[7] Preklad: kŕdeľ = hejno

[8] Napr. film Batman sa vracia, 1992

Dovtedajšiu podobu vírusu rozšíril o replikačnú časť, vďaka ktorej sa vírus po spustení infikovaného programu dokázal nakopírovať do ďalších programov. Tým využil len malú časť vlastností života, s ktorou sa dnes dokáže každý antivírus vysporiadať.

Neskorší zanietenní tvorcovia vírusov ale pochopili, že evolúcia je cesta, ako vyvinúť vírusy, ktoré sú čím ďalej, tým viac odolné voči antivírovým programom a schopné šíriť sa čo najrýchlejšie. Preto väčšina vírusov okrem štandardnej výbavy (napr. *šifrovací engine*) využíva aj mutáciu – pri kopírovaní do nového programu sa niektoré inštrukcie vírusu náhodne menia. To vytvára obrovské spektrum rôznych verzií vírusu, z ktorých časť dokáže prekonať svojich predchodcov a sťažiť prácu tvorcom antivírového software.

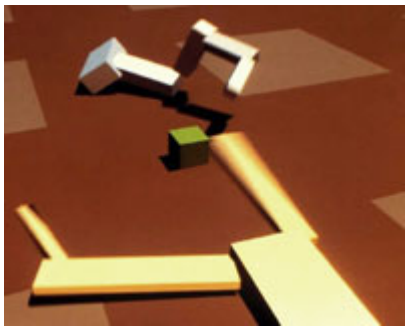
2.9 Panspermia

Jednou z prvých popularizácií umelého života bola simulácia Karla Simsa – Panspermia. Je to animácia vytvorená s použitím princípu biomorphov – všetky druhy rastlín boli vygenerované genetickým algoritmom, ktorého proces ovplyvňoval Sims. Výsledkom je bohatá flóra, ktorá síce nevypadá ako nám známe rastliny, no vygenerované druhy vypadajú uveriteľne a živo, práve vďaka umelej evolúcii (Obrázok 9).

Obrázok 9 - Scény z animácie Panspermia

[prevzaté z http://www.karlsims.com/panspermia.html]

Sims sa v rámci svojej práce v oblasti počítačovej grafiky zaoberal aj ďalšími aspektami umelého života - súťaživosťou, správaním a pohybom organizmov (Obrázok 10).



Obrázok 10 - Virtuálne organizmy vyvinuté pre získanie koristi (zelená kocka)

[prevzaté z http://www.karlsims.com/evolved-virtual-creatures.html]

2.10 Zhnutie

Tento prehľad bol zameraný na vývoj pohľadov na umelý život. Popísané objavy sú využívané simulačnými prostrediami rozoberanými v nasledujúcej časti. Úmyselne boli vynechané metódy genetických algoritmov a iné optimalizačné techniky využívané aj v oblasti umelej inteligencie (SOMA, evolučné stratégie apod.), pretože ich popis je mimo rozsah tejto práce.

3 SÚČASNÁ SITUÁCIA

Táto časť sa zameriava na popis prostredí vybraných podľa doporučenej literatúry a predchádzajúcich skúseností s niektorými z týchto prostredí.

Označenie prvých dvoch prostredí (Tierra a Avida) ako simulačných, prípadne ako simulátory môže byť sporné. Tie totiž život ani živé organizmy nesimulujú, ale synetizujú. Predstavujú návrat k počiatkom umelého života, a snažia sa vytvoriť život s neživých inštrukcií procesora a tým dokázať, že život nie je závislý na médiu a teda že sa môže objaviť *z ničoho* aj v priestore tvorenom informáciami.

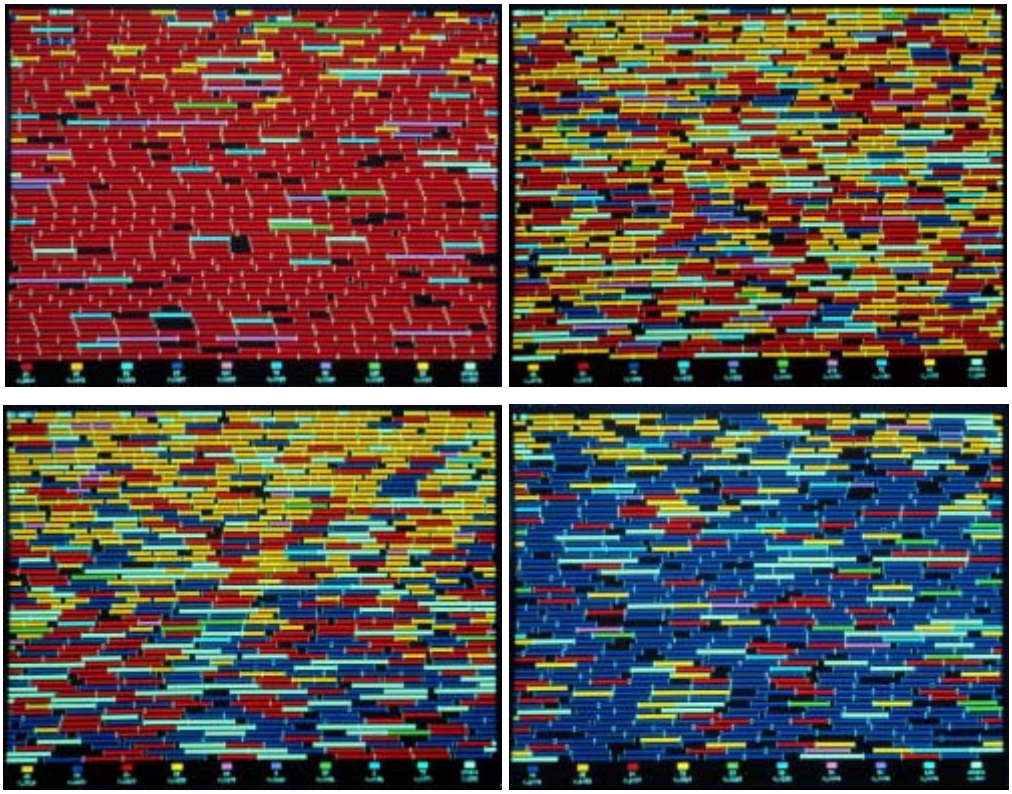
Ostatné prostredia túto myšlienku akceptujú (alebo ignorujú) a sústreďujú sa na simuláciu živých organizmov. Roziel medzi simuláciou a syntézou života objasnia nasledujúce podkapitoly.

3.1 Tierra

Hoci prostredie *Tierra* vzniklo pod rukami Thomasa S. Raya na počiatku 90. rokov, dodnes si zasluhuje uznanie, hlavne z historického hľadiska. Tierra predstavuje virtuálny počítač, v ktorého pamäti súperia programy o zdroje – pamäť a výpočtový čas CPU. Toto prostredie je odvodené od programátorskej hry *Core Wars*.

Na rozdiel od Core Wars, kde celý priebeh života prostredia určujú programátori, v prostredí Tierra je len na počiatku „simulácie“ pamäť naplnená programami, ktoré napísal experimentátor. Tieto programy sú paralelne spúšťané a vďaka zahrnutým samoreplikačným inštrukciám sa dokážu nakopírovať do priestoru ostatných organizmov. To vyvoláva neustály súťaž o čo najefektívnejšie rozmnožovanie. V tejto súťaži je možné využiť inštrukcie iných jedincov, môžu sa teda vyvinúť paraziti závislí na hostiteľoch, ako aj imunita hostiteľov voči týmto parazitom.

V simuláciách zvyčajne dominoval celému virtuálnemu prostrediu jeden najefektívnejší druh a nebolo výnimkov, keď tento druh bol nahradený iným (dá sa povedať že z evolučného hľadiska silnejším) druhom, za „pomoci“ parazita, ktorý pôvodný dominantný druh vyhladil, ale nástupca bol voči nemu imúnny. Na obrázku (Obrázok 11) sú červenou farbou vyznačení pôvodne dominantní jedinci, žltou farbou paraziti a modrou jedinci imúnni voči žltým parazitom.



Obrázok 11 - Tierra - vizualizácia vymretia jedného a objavenia iného druhu vďaka parazitom [prevzaté z: http://life.ou.edu/pubs/images, autor: Marc Cygnus]

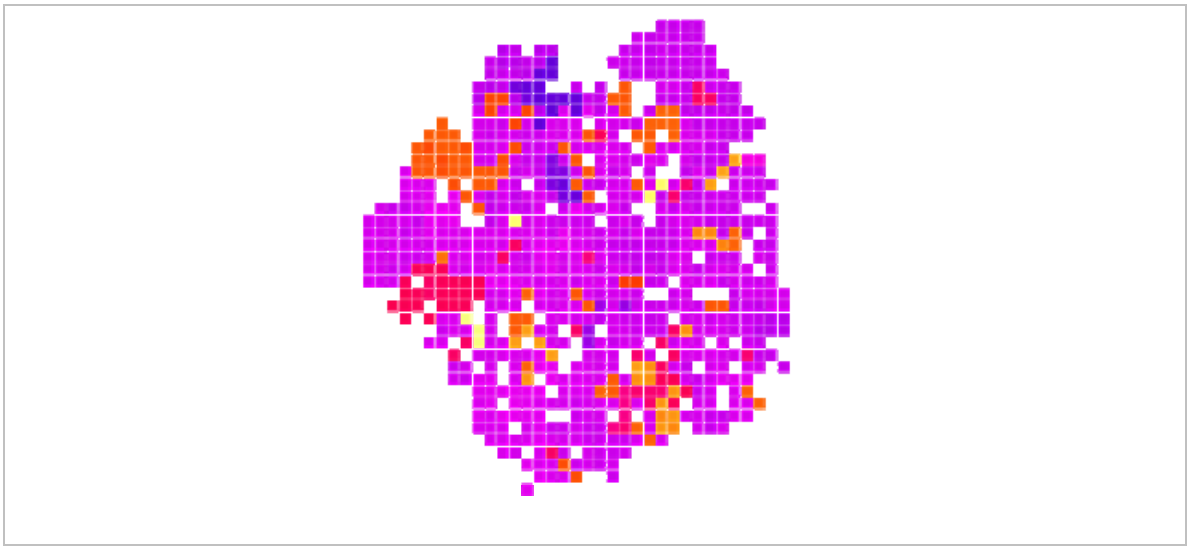
3.2 Avida

Priamy nasledovník prostredia Tierra je *Avida* a v súčasnosti sa aktívne vyvíja pod záštitou zoskupenia Program Error. Princíp fungovania programu je podobný, ako v prípade programu Tierra s rozdielom zvýšenia počtu inštrukcií a rozšírenia zdrojov prostredia (registre, zásobníky virtuálneho počítača).

Avida je okrem komunitne vyvíjanej verzie vyvíjaná aj Michganskou Univerzitou ako verzia Avida-ED. Táto verzia je určená najmä študentom biológie na praktické overenie teoretických znalostí o evolúcii. Táto verzia obsahuje aj plne grafické rozhranie – od znázornenia *petriho misky* (prostredie v ktorom prebieha experiment - Obrázok 12) až po zobrazenie organizmu (Obrázok 13), na ktorom je možné vidieť celý proces replikácie ako animáciu. Organizmus (ktorý je vlastne zhodný s jeho genómom) je zobrazený ako postupnosť inštrukcií usporiadaná do kružnice, keďže inštrukcie sa spúšťajú cyklicky.

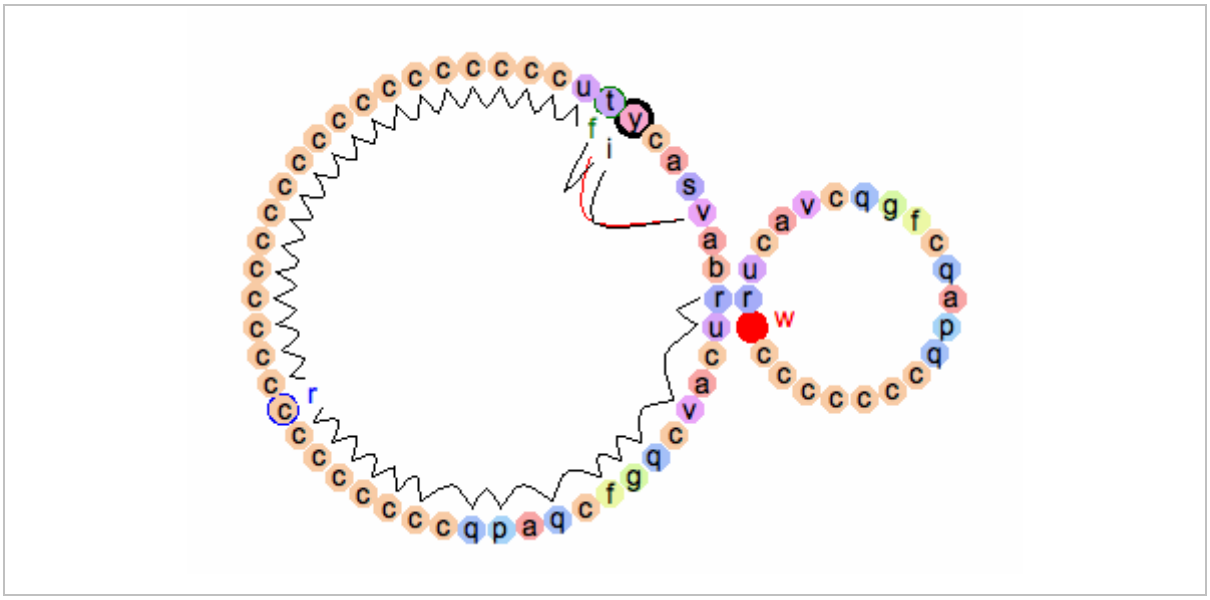
Na obrázku je možné vidieť menšiu kružnicu vpravo, podobnú ľavej kružnici – je to potomok, ktorý je práve vytváraný jedincom vľavo. Na ľavej kružnici je vidieť momentálnu pozíciu programu (čiernou farbou označená inštrukcia). Červený oblúk okolo

tejto inštrukcie znamená skok späť v programe, momentálne sa teda vykonáva inštrukcia vnútri sľučky. Keďže je to jediná sľučka v organizme, je isté že to je replikačná sľučka. Tá má za úlohu prečítať genóm rodiča (inštrukcie kruhu vľavo) a nakopírovať ich do priestoru alokovaného pre potomka. Modrou farbou je označená inštrukcia, ktorá sa práve kopíruje a červenou farbou na potomkovi je označená práve zapisovaná inštrukcia.



Obrázok 12 – Avida-ED - Petriho miska - vizualizácia prostredia

[prevzaté z http://avida-ed.msu.edu/images/PetriDish.png]



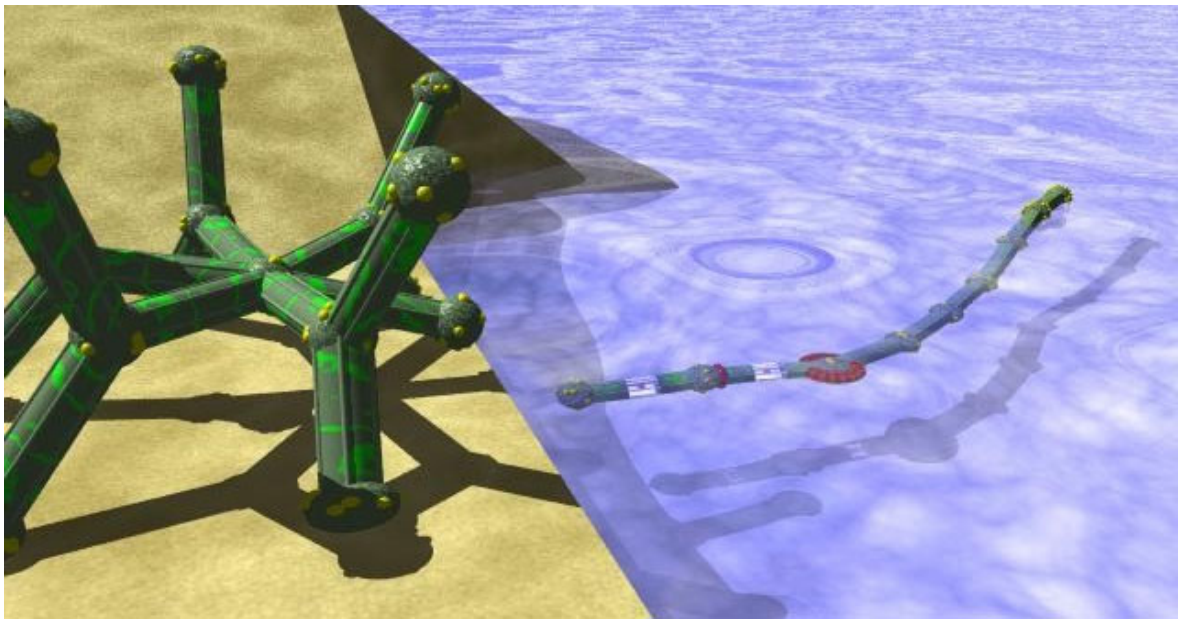
Obrázok 13 - Avida-ED - Detail organizmu - proces rozmnožovania

[prevzaté z: http://avida-ed.msu.edu/images/AvidaED_orgview_replication.png]

3.3 Framsticks

Iný prístup k umelému životu má prostredie Framsticks. Využíva umelú evolúciu k vývoju organizmov k čo najefektívnejšiemu zbieraniu potravy v prostredí. Prostredie tvorí trojrozmerný priestor s plne implementovanou fyzikou. Organizmy sú zložené z niekoľkých častí:

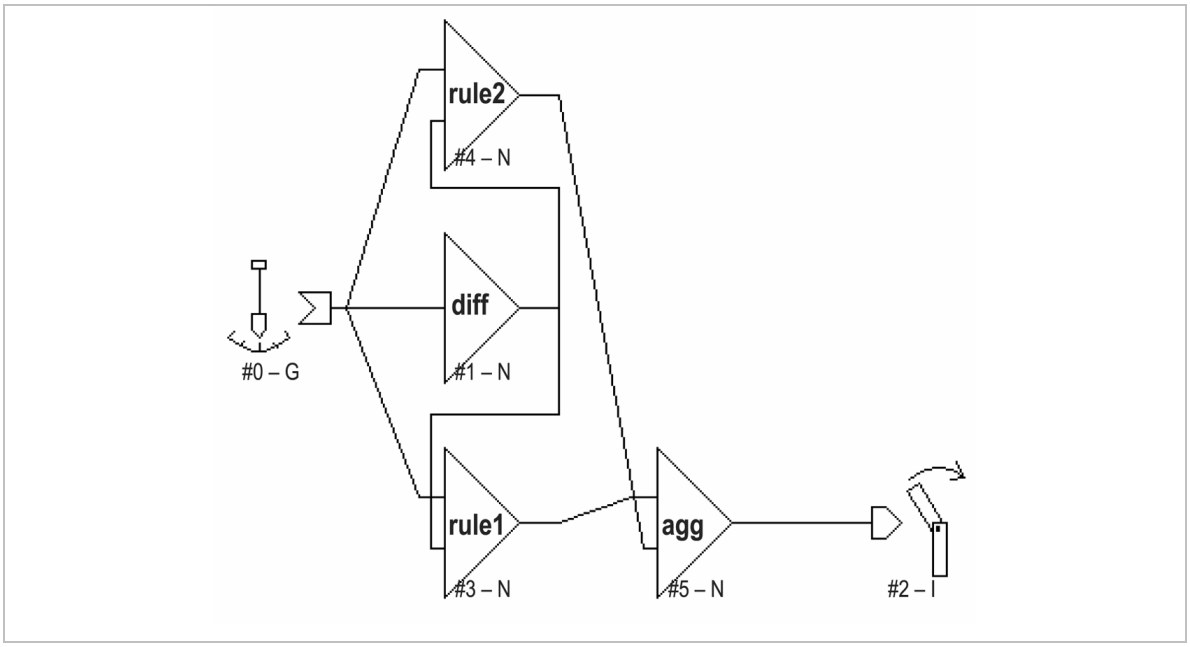
- Pohyblivé tyčinky – *kostra*.
- Motory otáčajúce týmito tyčinkami – *svaly*.
- Senzory, posielajúce signály z prostredia do mozgu – *receptory*.
- Sieť prijímajúca vstupy receptorov a ovládajúca svaly – *mozog* (Obrázok 15).



Obrázok 14 - Framsticks - chodiaci suchozemský jedinec a jedinec vyvinutý na plávanie

[prevzaté z: http://www.vieartificielle.com/nouvelle/index.php?id_nouvelle=847]

V prostredí Framsticks sa vyvinulo niekoľko komplexných organizomov, ktoré boli stavbou tela aj mozgu optimalizované pre prostredie, v ktorom žili (Obrázok 14). Na vyjadrenie takejto komplexnej morfológie, neurónových sietí a vzťahov medzi nimi je používaný *genetické jazyk* podobný L-systémom.

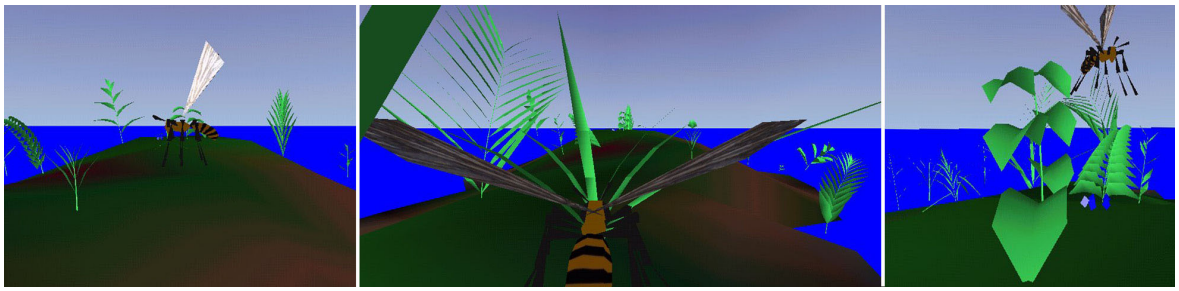


Obrázok 15 - Framsticks - evolúciou vyvinutý „mozog“ jedinca

[zdroj: http://www.emeraldinsight.com/fig/0670320107008.png]

3.4 Nerve Garden

Nerve Garden je multi-užívateľský trojrozmerný *virtuálny svet* inšpirovaný biológiou a dostupný širokému internetovému publiku. Projekt kombinuje množstvo metód a technológií, vrátane L-systémov, Javy, bunečných automatov a VRML. Svet Nerve Garden je navrhnutý tak, aby poskytol zaujímavý zážitok virtuálneho terária, ktoré preukazuje známky rastu, zániku a prenosov energie, pripomínajúc jednoduchý ekosystém (Obrázok 16). Cieľom projektu Nerve Garden je vytvoriť on-line „laboratórium umelého života založené na spolupráci“, ktoré môže byť rozšírené veľkým počtom používateľov za účelom výuky a výskumu. [7]



Obrázok 16 - Nerve Garden – rôzne pohľady na prostredie

[prevzaté z: http://www.biota.org/nervegarden/publish.html]

3.5 Gene Pool

GenePool je simulácia umelého života navrhnutá tak, aby osvetlila niekoľko základných princípov evolúcie zábavným a zaujímavým spôsobom. Najdôležitejší je aspekt sexuálneho výberu – výber partnera je dôležitý faktor v evolúcii morfológie a ovládania svalov vo fyzikálne založených animovaných organizmoch. [7]

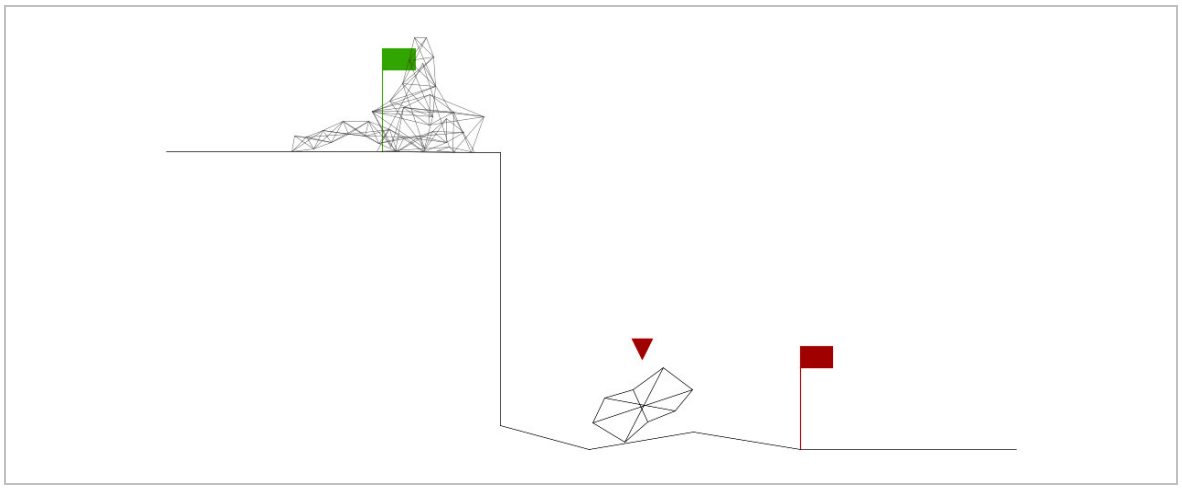
Obrázok 17 - Gene Pool - pohľad na prostredie

V prostredí GenePool, ktoré predstavuje veľké akvárium, sa pohybujú virtuálni plávajúci roboti – tzv. *swimboti* (Obrázok 17). Zmyslom života swimbotov je hľadať a konzumovať potravu v prostredí a páriť sa s ďalšími swimbotmi podľa nastavených preferencií. V prostredí je možné nastaviť niekoľko druhov sexuálnych preferencií – organizmy môžu preferovať partnerov s podobnou farbou, opačnou farbou, čo najrovnejších, čo najrozvetvenejších, dlhých alebo rýchlych. Sexuálne preferencie ovplyvňujú schopnosti nových swimbotov, keďže nepriamo určujú aké typy predkov bude mať.

Swimbot je zložený z 2 až 10 častí. Časti sú navzájom spojené pevnými kĺbmi a vzájomne sa pohybujú kyvadlovým spôsobom. Gény pre morfológiu určujú dĺžku, hrúbku, farbu a prirodzený uhol každej časti. Gény pre pohyb určujú fázy a amplitúdy ovládacích sínusových funkcií pre každú časť. [7]

3.6 Sodarace

Tento projekt bol pôvodne vyvinutý ako online Olympiáda, porovnávajúca ľudskú inteligenciu s inteligenciou počítačov. Projekt *Sodarace* je rozšírením softwaru *Sodaconstructor* – online konštrukčnej sady pracujúcej s hmotou, strunami a svalmi... [7]



Obrázok 18 - Sodarace - pohľad na simuláciu

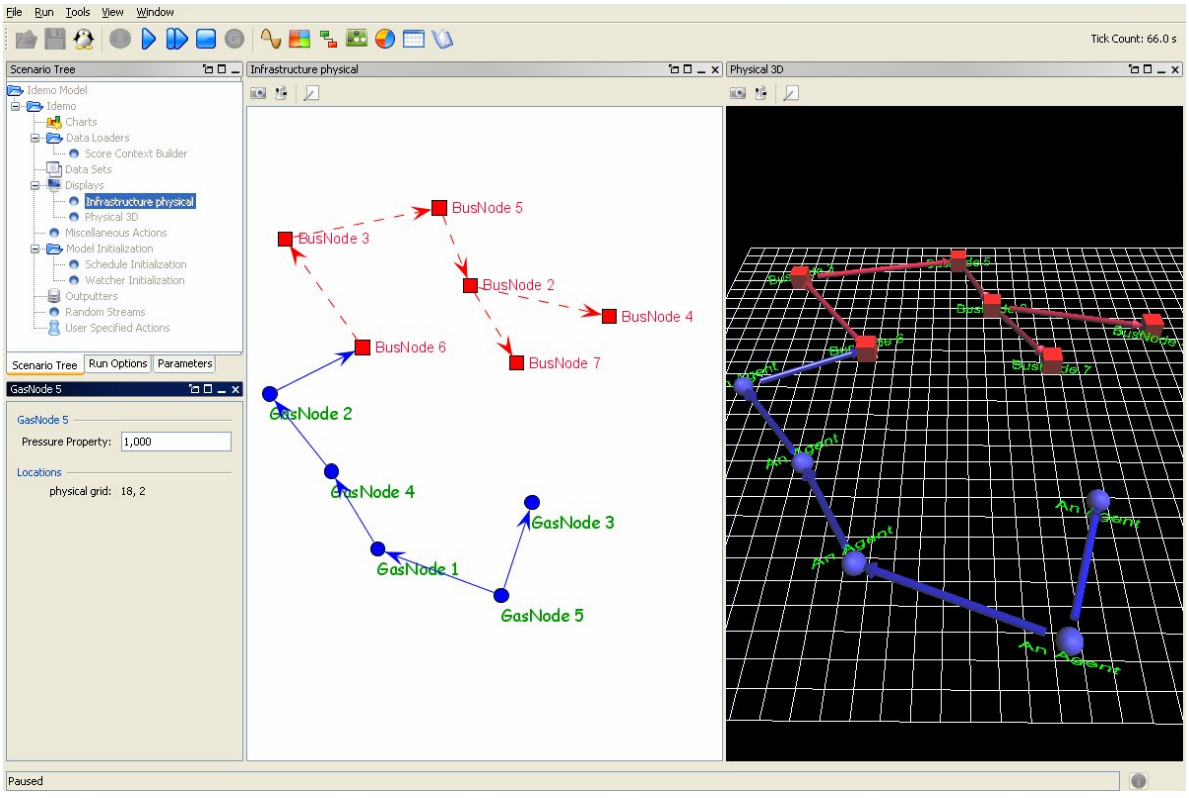
Táto „hra“ umožňuje používateľovi skonštruovať zo základných stavebných prvkov pohyblivých robotov, ktorí súťažia s evolučne vyvinutými robotmi (Obrázok 18). Každého robota je možné ďalej geneticky šľachtiť a tým optimalizovať pre väčšiu rýchlosť. Používateľom navrhnutý robot je spravidla pomalší.

Na stránke projektu existuje rozsiahle aktívne fórum, kde zanietení používatelia skúmajú výsledky simulácií a nové konštrukcie robotov.

3.7 Repast Simphony

Repast Simphony je priamym nasledovníkom simulačného prostredia *Repast*[9]. Repast je sada knižníc pre jazyky Python (*RepastPy*), Java (*RepastJ*) a C# (*Repast.NET*). Simulačné prostredie Repast Simphony predstavuje prepojenie knižníc RepastJ s rôznymi matematickými a štatistickými nástrojmi, a je postavené na viacúčelovom vývojovom prostredí Eclipse.

Simulačné prostredie Repast Simphony je zamerané na simuláciu multi-agent systémov a poskytuje dve možnosti definovania agenta a prostredia: programovanie v jazyku Java, alebo *blokové programovanie* – grafická tvorba objektov spájaním podobjektov (napr. torba agenta spájaním častí správania).



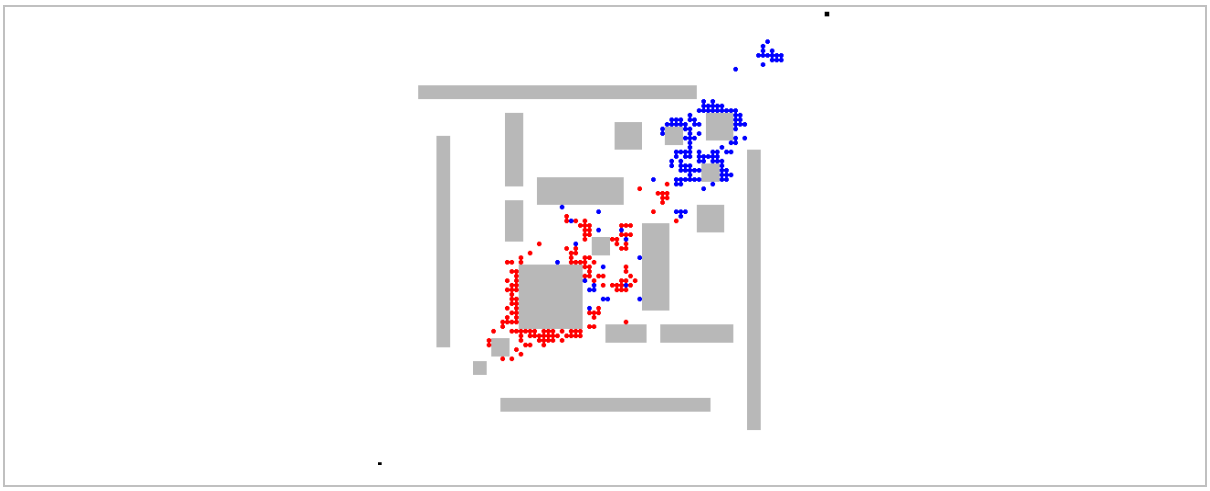
Obrázok 19 - Repast Simphony - grafová simulácia

[zdroj: http://portal.ncess.ac.uk/access/content/group/mass/SimphonyTutorialImages/repast_runtime_gui.jpg]

[9] Recursive Porous Agent Simulation Toolkit – sada na simulovanie rekurzívnych priepustných agentov

3.8 EINSTein

EINSTein je navrhnutý na načrtnutie pohľadu na pozemný súboj ako na samoorganizovaný, emergentný fenomén, vyplývajúci z dynamickej siete interakcií medzi abstraktnými bojovníkmi. [7]



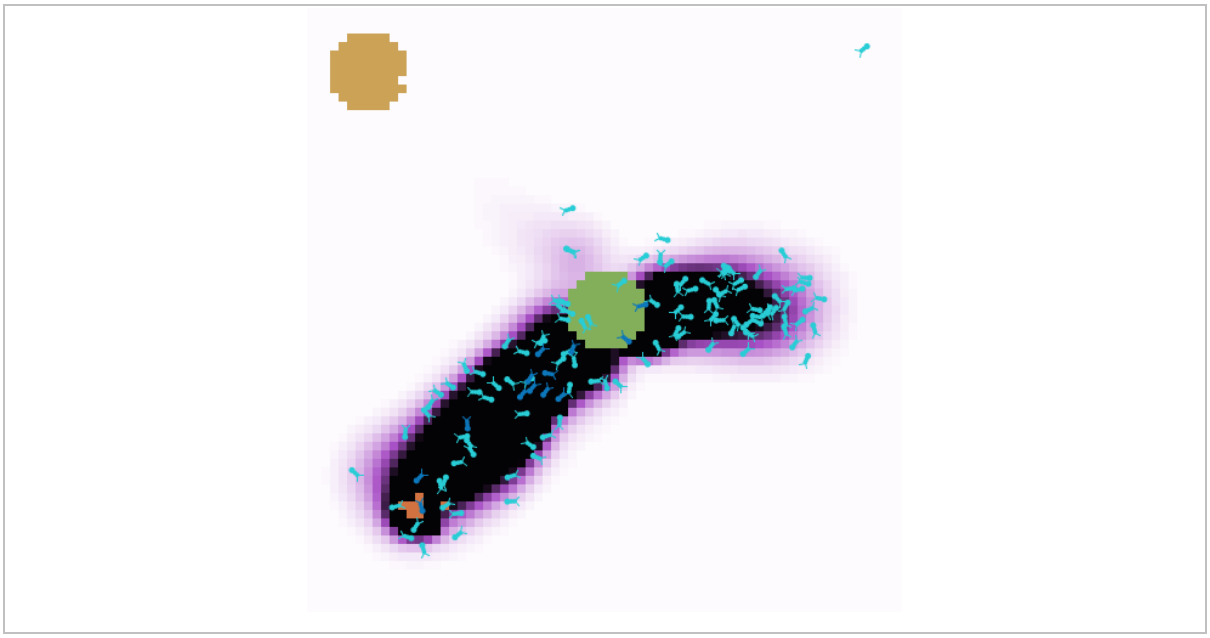
Obrázok 20 - EINSTein - pohľad na simuláciu boja

Bojovníkov reprezentujú *autonómni agenti*, ktorí môžu byť v troch stavoch – živí, zranení alebo zabití. Vďaka tomu môžu mať zranení agenti iné správanie ako nezranení. Agenti používajú senzory (zrak) a komunikáciu ako vstupy a ich výstupom je pohyb alebo paľba na nepriateľa. Pri štarte simulácie sú nastavené vlastnosti všetkých agentov – dosahy, vzory správania a príkazy.

Prostredie – bojisko – je reprezentované dvojrozmerným poľom diskrétnych pozícií. Každá pozícia môže byť obsadená červeným alebo modrým agentom, alebo byť zabraná prekážkou, ktorými je možné vytvárať komplexné bojové prostredia (Obrázok 20).

3.9 NetLogo

Program *NetLogo* je ďalší univerzálny nástroj na multi-agent simulácie. Logiku agentov je možné písať vo vlastnom jednoduchom jazyku s rozšírenými príkazmi dialektu *Logo* – otočenie agenta, pohyb agenta, získanie vstupov apod.



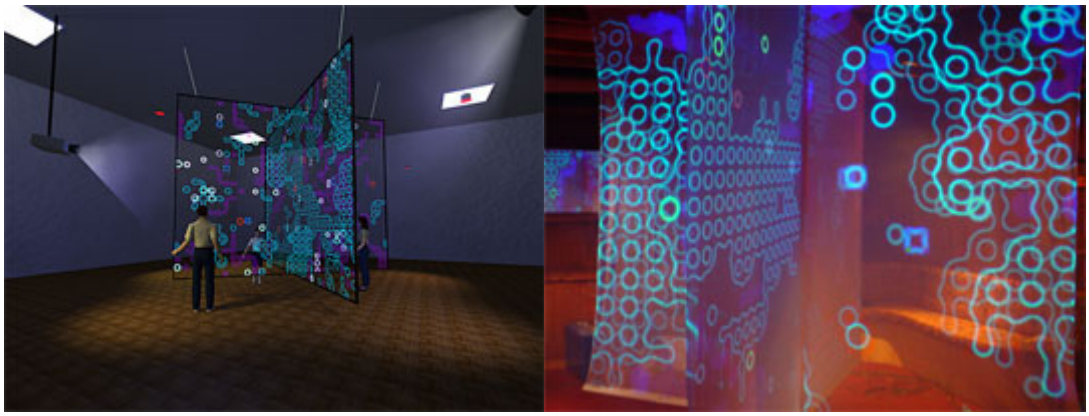
Obrázok 21 - NetLogo - pohľad na simuláciu správania mravcov

V prostredí NetLogo je možné vytvárať simulácie s vlastným používateľským rozhraním. Je možné využiť tlačidlá a prepínače rôzneho druhu, a na výstup použiť okrem vykresľovacieho plátna rôzne druhy grafov.

Tento program je využiteľný aj v ekonomike, biológii, fyzike, chémii, psychológii a medicíne na modelovanie komplexných systémov, ktoré sa v čase vyvíjajú (napr. ekonomické procesy, fyzikálne modely apod.).

3.10 Eden

Inštalácia *Eden* je interaktívne umelecké dielo využívajúce metódy umelého života. Prvotne šlo o jedno prostredie premietané na obrazovku monitora, s ktorým používateľ interagoval pomocou myši. V súčasnosti prebieha simulácia súčasne v niekoľkých prepojených prostrediach (premietané na obe strany projekčnej plochy – Obrázok 22). Interakcia návštevníka s prostredím prebieha vďaka senzorom rozmiestneným v okolí projekcie.



Obrázok 22 - Eden - pohľad na interaktívnu inštaláciu

[prevzaté z: http://www.csse.monash.edu.au/~jonmc/projects/eden/edenHistory.html]

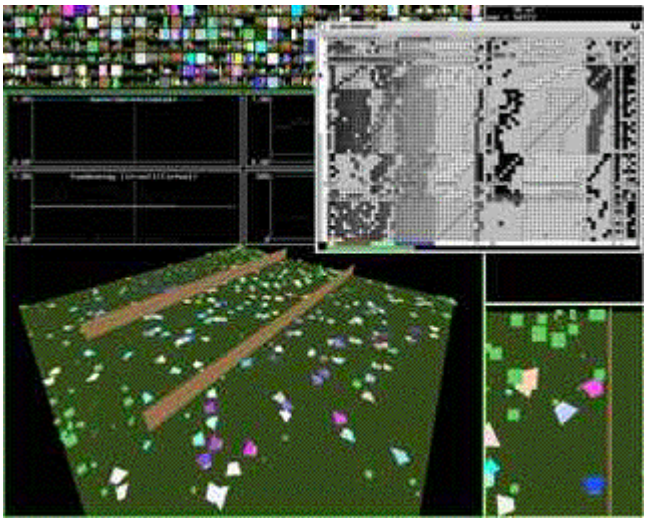
Prostredie je založené na princípe bunečného automatu. Každá bunka môže byť obsadená jedným z nasledujúcich objektov:

- Kameň – prekážka v prostredí
- Biomasa – zdroj potravy
- Organizmus – mobilný agent

Agenti medzi sebou komunikujú akusticky, teda vydávaním a prijímaním zvukov, ktoré sú cez reproduktory sprostredkované aj návštevníkovi. Návštevník naopak ovplyvňuje prostredie svojou prítomnosťou na určitom mieste. Prítomnosť návštevníka vytvára pozitívnu väzbu pre organizmy na danom mieste, napr. zvýšeným rastom biomasy. Výsledkom je nepriamo riadená estetická evolúcia zvuku – jedinci, pri ktorých sa návštevníci zdržiavali častejšie, majú väčšiu šancu na prežitie.

3.11 Polyworld

Prostredie Polyworld je zjednodušenou simuláciou *ekosystému*. Simulované organizmy sa sexuálne rozmnožujú, bojujú, zabíjajú a požierajú sa navzájom, konzumujú potravu, ktorá rastie v prostredí. Buď vyvinú úspešné stratégie na prežitie, alebo umrú. Celá paleta správania organizmu (pohyb, otáčanie, útok, jedlo, párenie, svietenie) je kontrolovaná jeho „mozgom“, tvoreným neurónovou sieťou. [10]



Obrázok 23 - Polyworld - pohľad na prostredie

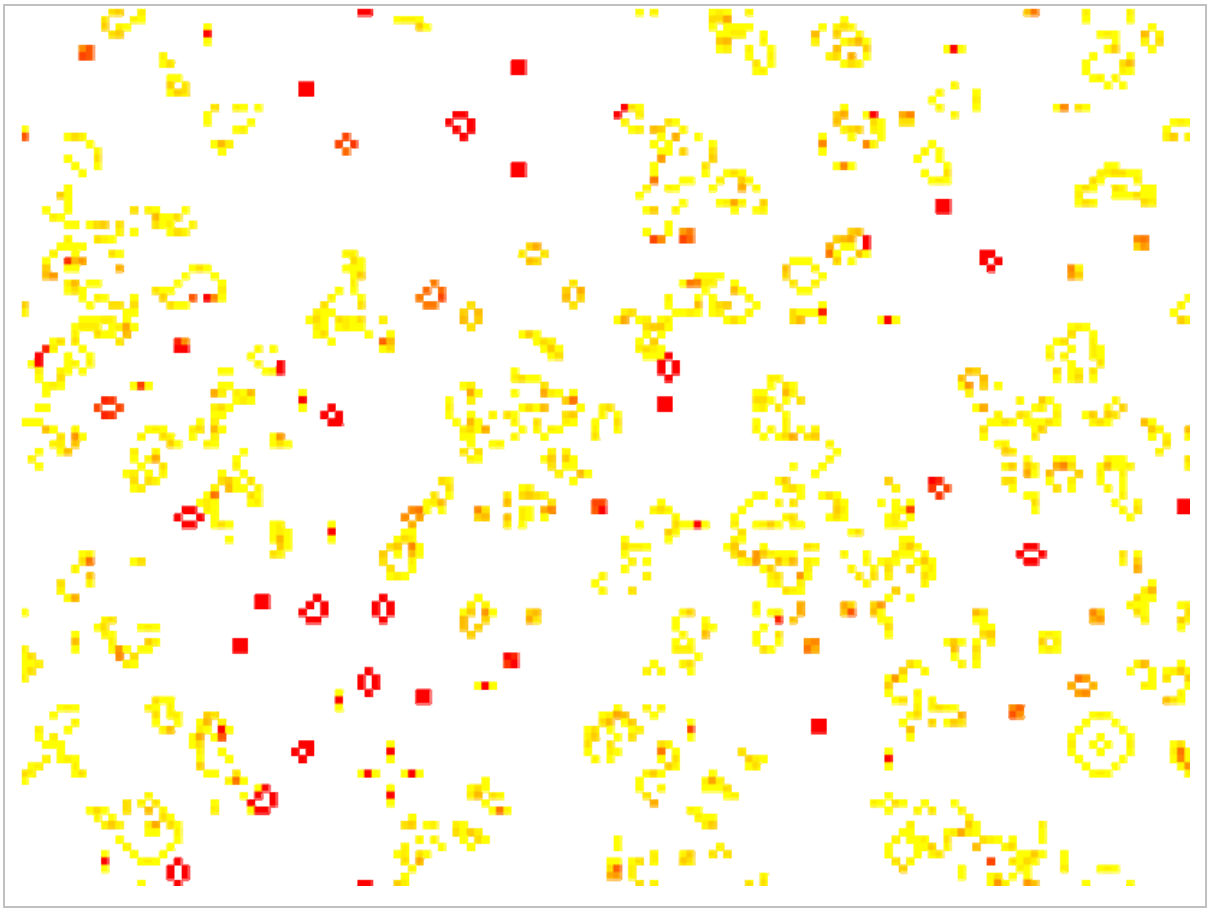
[prevzaté z: http://www.beanblossom.in.us/larryy/polyworld.html]

Organizmy využívajú tzv. Hebbovské učenie – v priebehu života sa učia na základe pozitívnej a negatívnej odozvy a bez poskytnutia trénovacej množiny. Svet vnímajú vďaka jednorozmernej projekcii okolia na jedinca. Všetky vlastnosti jedinca sú dané geneticky, vďaka čomu sa v prostredí vyvíja súbežne niekoľko druhov, ktoré si vybudovali odlišné stratégie na prežitie. V niektorých simuláciách sa objavili rôzne komplexné správania, napr. združovanie do húfov, kanibalizmus alebo vyhýbanie sa útoku.

Program obsahuje štatistické a analytické nástroje, trojrozmerný pohľad na prostredie i pohľady organizmov (Obrázok 23).

3.12 MJCell

MJCell je Java applet, odvodený od programu *MCell* (Mirek's Cellebration). Tento program je vytvorený na zobrazenie veľkého množstva jednorozmerných a dvojrozmerných bunečných automatov. Obsahuje 340 vstavaných bunečných automatov, ktoré sú zotriedené do 15 skupín.

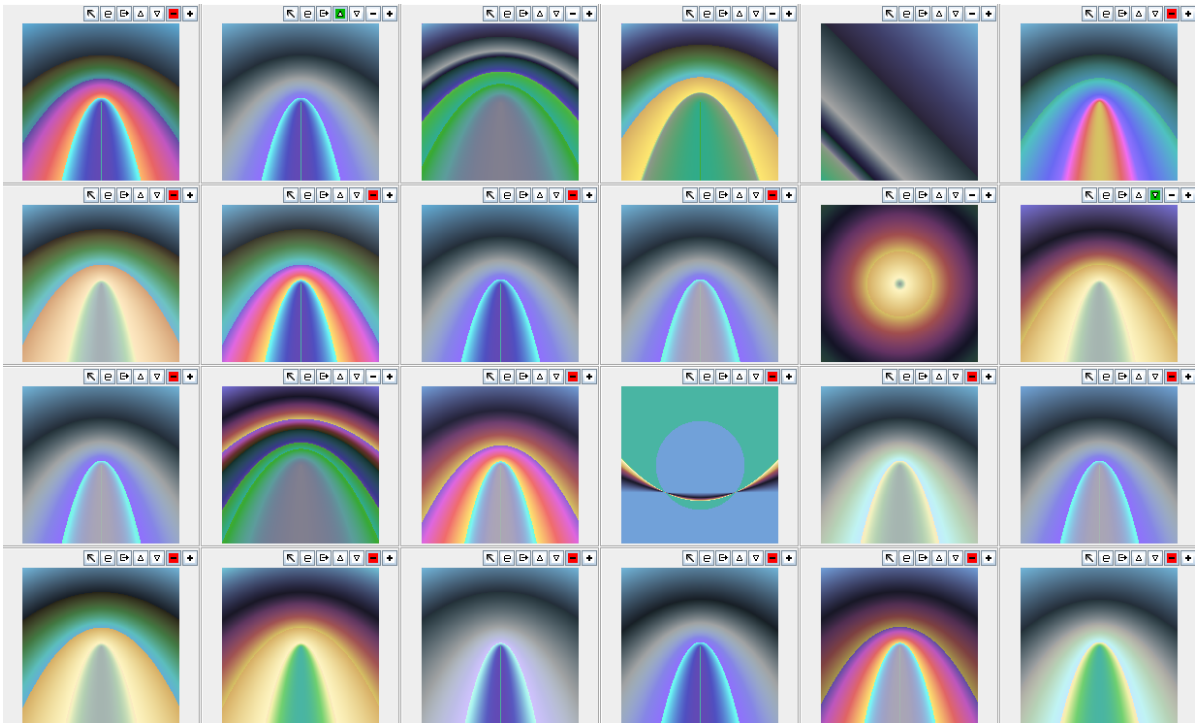


Obrázok 24 - MJCell - pohľad na simuláciu Conwayovej hry života

Simulácie sa spúšťajú jednoduchým vybraním typu bunečného automatu, vygenerovaním počiatočnej generácie (tj. nastavenie počiatočných stavov buniek) a stlačením tlačidla Start. V priebehu simulácie je možné ovládať rýchlosť, alebo krokovať animáciu. K dispozícii sú bunečné automaty od Conwayovej Hry života (Obrázok 24), cez Von Neumannove automaty až po jednoduché fyzikálne simulácie.

3.13 Kandid

Program *Kandid* je založený na myšlienke Dawkinsových biomorfov. Slúži na generovanie genetického umenia takmer akéhokoľvek druhu (Obrázok 25). V programe je možné zvoliť si metódu vykresľovania grafiky – výrazy jazyka LISP, fraktály, bunečné automaty, Voronoiove diagramy, L-systémy apod. Tiež je možné si zvoliť farebnú schému pre generovaný obrázok – čiernobiele, odtiene šedej, paletové farby, farebné prechody, priehľadnosť.



Obrázok 25 - Kandid - jedna generácia obrázkov

Kandid využíva *sexuálnu reprodukciu* obrázkov a používateľ kontroluje niekoľko parametrov populácie. Každý jedinec je implicitne označený na zmazanie, čo môžme zmeniť vybraním ho za jedného z rodičov, alebo jednoducho zrušiť prepínač. Každého jedinca je možné vyexportovať ako obrázok, priblížiť, alebo dokonca upraviť jeho chromozóm.

3.14 Porovnanie

V závere tejto časti je vypracované porovnanie základných vlastností jednotlivých systémov a stručné zhodnotenie kladov, označených znakom „+“, a záporov, označených znakom „-“ (Tabuľka 1). Stĺpec s označením *GUI* informuje o dostupnosti grafického používateľského rozhrania, stĺpec *OS* o dostupnosti aplikácie pre rôzne operačné systémy a stĺpec *SRC* o dostupnosti zdrojových kódov. V stĺpci *OS* označenie *Java* znamená dostupnosť na všetkých platformách podporujúcich beh Java aplikácií, označenie *web* znamená, že aplikácia je dostupná on-line – údaj v zátvorke potom znamená použitú technológiu. Hviezdička vedľa údaju v tabuľke znamená „čiastočne“ – údaj teda platí len pre podmnožinu súčastí softwaru.

Tabuľka 1 - Porovnanie aplikácií umelého života

Názov aplikácie	GUI	OS	SRC	Zhodnotenie
Tierra	nie	OS X Lin Win	áno	+ Jedno z prvých prostredí + Syntéza organizmov - *Extrémne náročné ovládanie aj inštalácia*
Avida	áno*	OS X Win	áno*	+ Názorné zobrazenie organizmu a jeho procesov + Na výber verzie podľa požiadaviek
Framsticks	áno	Win Lin* Java*	áno*	+ Pekná a rozšíriteľná grafika + Vznikajú pomerne komplexné organizmy - *Dokumentácia prevažne v Poľskom jazyku*
Nerve Garden	áno	Web (VRML)	nie	+ Verejne dostupný virtuálny svet - *Použitie neštandardnej technológie VRML* - *Takmer nedostupná dokumentácia*
Gene Pool	áno	OS X Win	nie	+ Prehľadné rozhranie + Zaujímavé organizmy + Interakcia používateľa s prostredím - *Nedopracovaná fyzika*
Sodarace	áno	Web (Java)	nie	+ Prezentácia umelého života pomocou hier + Dobre spracovaná fyzika - *Absencia dlhodobých simulácií*
Repast Simphony	áno*	Java	áno	+ Kompletný framework na vývoj multi-agent simulácii + Možnosť blokového vývoja agentov
EINSTein	áno	Win	áno	- Jednoúčelný zastaralý program
NetLogo	áno	Win	nie	+ Vhodné pre akékoľvek multi-agent simulácie + Množstvo predpripravených simulácií
Eden	-	-	-	+ Audiovizuálna inštalácia + Rozšírená myšlienka breedingu - *Mimo výstavy nedostupné*
Polyworld	áno	OS X Lin Win	áno	+ Zdokumentované výsledky simulácií + Štatistické nástroje - *Problémy so spustením aj kompiláciou vo Win*
MJCell	áno	Java	áno	+ Obrovské spektrum bunečných automatov + Dobre spracovaná grafika aj GUI + Dostupnosť z webu, aj samostatná aplikácia
Kandid	áno	Java	áno	+ Prehľadné ovládanie + Pokrýva celé spektrum genetického umenia

4 TECHNOLÓGIE

Táto kapitola stručne predstavuje technológie použité pri vývoji vlastného prostredia, ktoré je popísané v poslednej časti práce. Ide o stručný vedomostný základ pre čitateľa, ktorý sa s týmito technológiami nestretol.

4.1 Java

Java je objektový programovací jazyk a jeho autorom je spoločnosť Sun Microsystems. Syntaxou je tento jazyk podobný jazyku C++, a jazyku C# od firmy Microsoft je dokonca podobný aj spôsobom programovania.

To, že je jazyk Java objektový neznamená len, že umožňuje tvorbu tried a objektov ako jazyk C++, ale tvorbu tried a objektov vyžaduje. V Jave nie je možné vytvoriť funkciu, ktorá by nebola členskou funkciou triedy (tzv. metóda), tak isto nie je možné definovať premennú mimo triedu alebo jej metódy. Táto vlastnosť jazyka sa môže zdať negatívna, no v skutočnosti vynucuje kultivovaný zdrojový kód, ktorý je abstraktnejší a teda flexibilnejší ako pri programovaní bez použitia objektového programovania.

Ďalším rysom spojeným s objektovým programovaním je rozdelenie tried do balíkov (tzv. package), ktorých štruktúra je určená adresárovou štruktúrou zdrojových kódov jednotlivých tried. Táto vlastnosť umožňuje prehľadnejšiu štruktúru zdrojových súborov, ako aj tvorbu navzájom nezávislých a teda znovupoužiteľných zoskupení tried v jednotlivých balíkoch.

Skompilovaný program v jazyku Java je multiplatformý, čiže spustiteľný na ľubovoľnej platforme, pre ktorú existuje Java Runtime Environment (JRE). JRE je prostredie pre beh javovských aplikácií napísané pre všetky väčšinové operačné systémy. Toto prostredie transformuje inštrukcie skompilovaného programu (tzv. bytecode) na natívny kód danej platformy.

Existujú tri hlavné vývojárske verzie Javy – Java ME (mobile edition - pre mobilné aplikácie), Java SE (standard edition - pre desktop aplikácie) a Java EE (enterprise edition - pre serverové aplikácie). Tieto verzie sa odlišujú množstvom dostupných balíkov a programovým príslušenstvom.

Jednou z výhod jazyka Java je tzv. perzistencia (JPA), ktorá umožňuje ukladanie/načítanie celých objektov priamo do/z databázy.

4.1.1 Technológie JSP a Servlet

Technológia JSP (Java Server Pages) umožňuje webovým vývojárom a dizajnérom rapídne vyvíjať a jednoducho udržovať dynamické a na informácie bohaté webové stránky, ktoré využívajú existujúce systémy. Ako súčasť rodiny Java technológií, JSP umožňuje vývoj webových aplikácií, ktoré sú nezávislé na platforme. Technológia JSP oddeľuje používateľské rozhrannie od generovania obsahu, čo dovoľuje designérom zmeniť výzor stránky bez zásahov do dynamického obsahu.

JSP používa tagy podobné XML, ktoré zapúzdrujú logiku generujúcu obsah stránky. Aplikačná logika môže byť umiestnená do rôznych serverových zdrojov, ku ktorým stránka pristupuje pomocou týchto tagov.

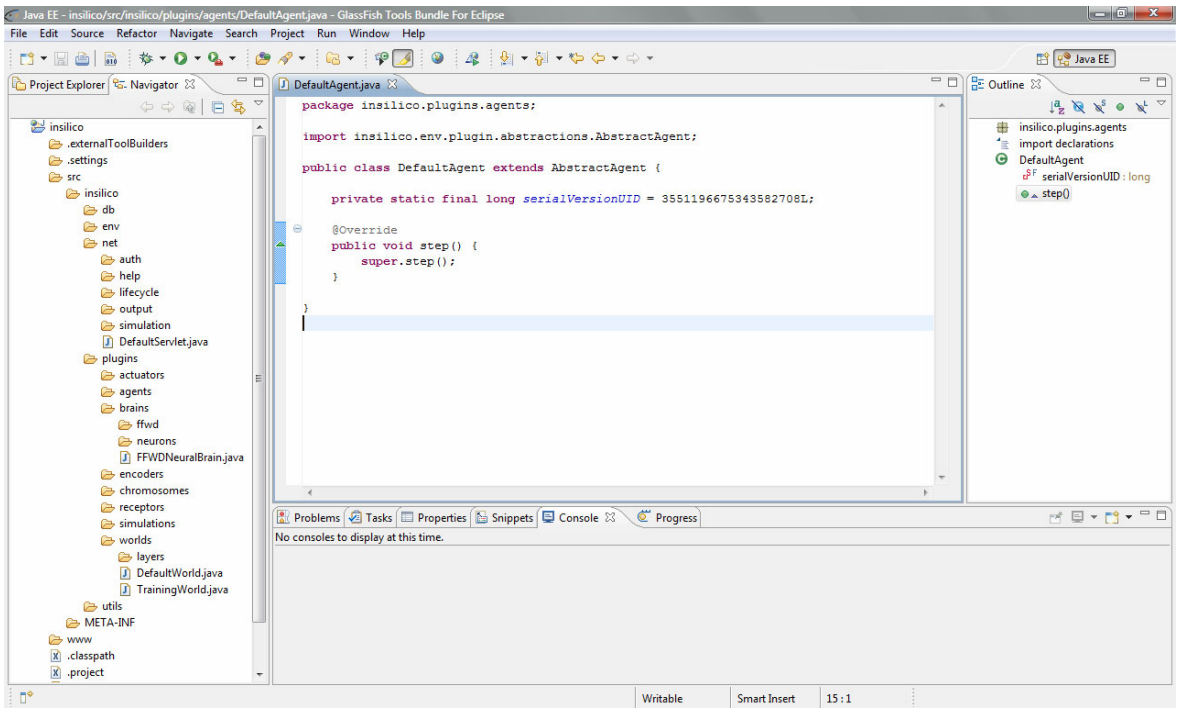
Technológia JSP je rozšírením technológie Java Servlet. Servlety sú platformovo nezávislé, serverové moduly, koré zapadajú do serverového frameworku a sú používané na rozšírenie schopností webového servera s minimálnou nutnou réžiou a údržbou. [12]

Servlet obsahuje logiku vykonanú po dotaze na webový server, JSP obsahuje pravidlá pre zobrazenie výstupu servletu, spravidla v HTML formáte.

4.2 Eclipse

Eclipse je open source komunita podporovaná neziskovou organizáciou Eclipse Foundation. Cieľom tejto komunity je tvorba rozšíriteľných vývojových prostredí a nástrojov poktývajúcich potreby celého vývoja softwaru.

Z projektov Eclipse bol použitý GlassFish Tools Bundle For Eclipse, prostredie zostavené špeciálne na vývoj webových aplikácií s použitím technológií JSP a Java Servlet. Toto prostredie je rozšírením štandardného prostredia pre vývoj v jazyku Java o vstavaný webový server, nové typy projektov a ďalšie rozšírenia, napr. sprievodcovia na prácu s JPA, priamy prístup do databázy apod.



Obrázok 26 - Pohľad na prostredie Eclipse

4.3 Glassfish

Sun Glassfish Enterprise Server je open source server (resp. serverový kontainer), do ktorého je možné umiestňovať serverové aplikácie využívajúce technológie JSP a Servlet. Tento server je vyvíjaný komunitou Glassfish a dlhodobo podporovaný spoločnosťou Sun.

4.3.1 Quartz

Server Glassfish samostatne nepodporuje viacvláknové aplikácie a tak isto neposkytuje dostatočné možnosti na spracovanie úloh na pozadí, preto bola ako riešenie zvolená knižnica Quartz, korá umožňuje servletom po spracovaní dotazu spustiť proces napozadí a ukončiť svoju činnosť.

Jeden proces sa nazýva úloha (job), ktorý môže byť časovačom (scheduler) spustený v ľubovoľnom čase. Zároveň prostredie dostáva informácie o stave úlohy.

4.4 PostgreSQL

PostgreSQL je multiplatformná open source objektovo-relačná databáza. Mámodulárnu architektúru, vďaka ktorej je možné server ľubovoľne prispôsobiť požiadavkam aplikácie. Je možné použiť existujúce moduly, prípadne dopísať vlastné.

4.4.1 Navicat Lite

Keďže ani PostgreSQL server, ani prostredie Eclipse neposkytujú vyhovujúci databázový editor, bol na návrh použitý správca databáz Navicat Lite.

Tento editor umožňuje prehľadné spravovanie databáz, štruktúr ich tabuliek a zároveň obsahuje tabuľkový editor, v ktorom je možné tabuľky prezerať a priamo editovať príslušné polia.

4.5 XML

Extensible Markup Language (XML – rozšíriteľný značkovací jazyk) je jednoduchý, veľmi flexibilný formát odvodený od formátu SGML[10]. Pôvodne bol navrhnutý aby vyhovel požiadavkam veľkorozmerného elektornického publikovania, ale XML tiež hrá dôležitú úlohu pri výmene širokého spektra dát na webe i na iných miestach.[13]

4.6 Open Flash Chart

Open Flash Chart je flashová open source komponenta, ktorá umožňuje jednoduché zobrazovanie grafov. Táto komponenta sa vloží do HTML stránky a ako parameter sa použije URL adresa súboru s dátami. Na tejto adrese je možné umiestniť serverový skript, ktorý vygeneruje dáta dynamicky.

[10] Standard Generalized Markup Language (ISO 8879) – univerzálny značkovací metajazyk – jeho podmnožinou sú napríklad formáty XML, HTML a DocBook.

II. PRAKTICKÁ ČASŤ

5 TVORBA SIMULAČNÉHO PROSTREDIA

Jedným z cieľov tejto práce bolo nájsť simulačný software, ktorý by spĺňal všetky požiadavky, alebo navrhnúť vlastný. Základnými požiadavkami na software boli:

- **Architektúra klient/server**
  Simulátor by mal bežať na *serveri*, ktorý môže poskytnúť väčšiu výpočtovú kapacitu. Simulácie sa teda musia spúšťať na diaľku, najlepšie prostredníctvom tzv. *ľahkého klienta*[11].
- **Multiplatformnosť**
  Prostredie by sa malo dať spustiť na operačných systémoch Windows, Linux a MAC OS.
- **Rozšíriteľnosť**
  Malo by byť možné do simulátora doprogramovať aj iné typy simulácie podľa požiadaviek, a zároveň možnosť meniť parametre simulácií bez zásahu do zdrojového kódu.
- **Použitie za účelom optimalizácie**
  Simulačné prostredie teda musí obsahovať štatistické výstupy zo simulácií.

Keďže žiadne z popisovaných prostredí nespĺňalo podmienky, bol navrhnutý základ vlastného univerzálneho simulačného systému s jednoduchou testovacou simuláciou – vývoj neurónovej siete vhodnej na riešenie XOR problému.

5.1 Vnútorný popis systému

Táto časť popisuje návrh systému. Mala by slúžiť ako návod pre programátora, ktorý rozširuje funkcie prostredia.

5.1.1 Voľba prostriedkov

Na vývoj aplikácie bol zvolený programovací jazyk Java, pretože je multiplatformný a vďaka technológiám, ktoré sú dostupné v tomto jazyku, je možné napísať aj samotný

[11] Ľahký klient = aplikácia (alebo HTML stránka) zobraziteľná vo webovom prehliadači a spolieha sa veľkou časťou aplikačnej logiky na sevrer

simulátor, aj webové rozhranie. Java EE obsahuje serverový kontainer Glassfish, ktorý sa stará o všetky aktivity potrebné k behu serverovej aplikácie, takže na webové rozhranie je potrebné napísať servlety vykonávajúce potrebnú „business logic“ (v tomto prípade prácu s databázou) a JSP stránky, ktoré zobrazia výstupy týchto servletov a grafy. Na zobrazenie grafov bola zvolená flashová komponenta Open Flash Chart kvôli minimálnej konkurencii.

V priebehu vývoja sa objavili problémy s vláknami a paralelizáciou úloh na Glassfish serveri, preto bola na jednoduchú paralelizáciu a časovanie úloh využitá knižnica Quartz.

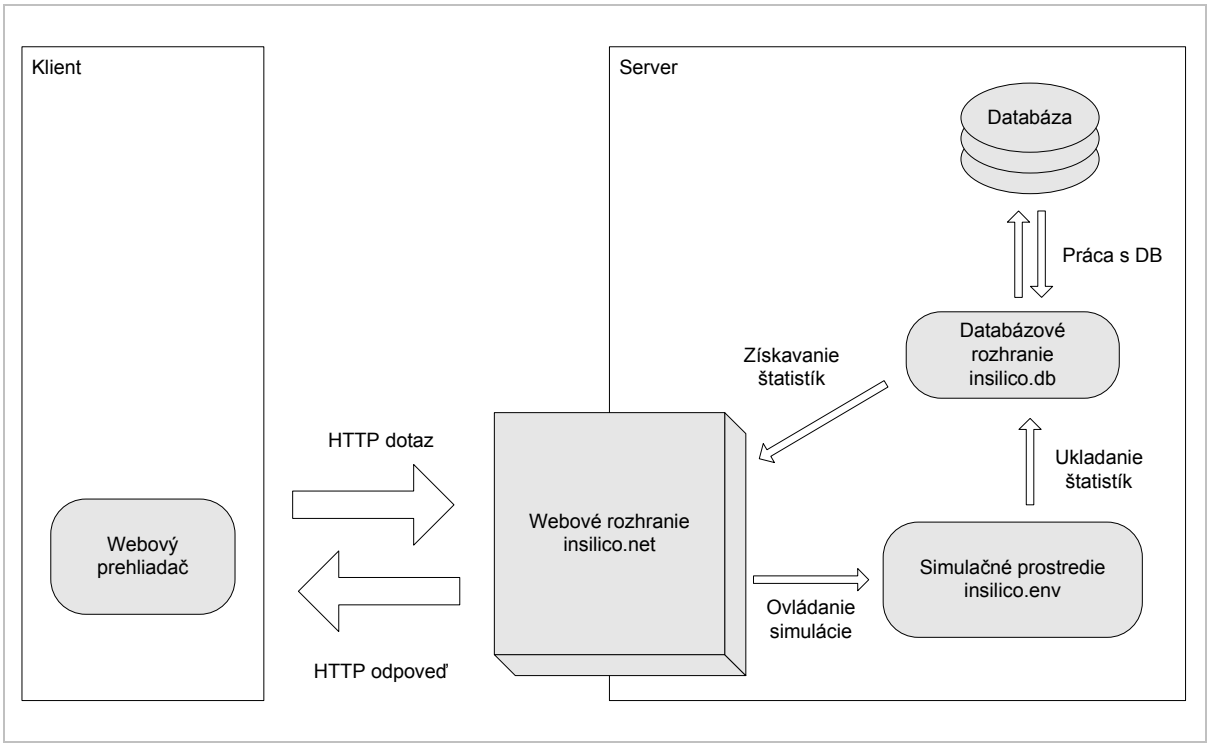
Ako databázový server bol vybraný server PostgreSQL kvôli niekoľkoročným skúsenostiam s ním. Vývojové prostredie Eclipse bolo vybrané tak isto z dôvodu dlhodobého používania a spokojnosti s týmto prostredím a vďaka jeho integrácii so serverom Glassfish.

5.1.2 Štruktúra

Celé prostredie je rozdelené do logických častí, ako naznačuje štruktúra balíkov zdrojového kódu:

- Balík **insilico.db** – spoločné funkcie na prácu s databázou a perzistenciou objektov.
- Balík **insilico.env** – simulačné prostredie, ktoré je možné v prípade potreby oddeliť do samostatnej aplikácie s iným rozhraním.
- Balík **insilico.net** – servlety webového rozhrania.
- Balík **insilico.plugins** – rozšíriteľné časti simulačného prostredia.
- Balík **insilico.utils** – spoločné pomocné funkcie.

Servlety v balíku **insilico.net** spracúvajú HTTP dotazy webového klienta a ako výsledok dynamicky generujú HTML stránky, ktoré toto rozhranie tvoria. Podľa prijatého dotazu vykonávajú dva druhy akcií – výpisy a ovládanie simulácií. Pokiaľ je klientom požadovaný výpis (napr. zoznam bežiacich simulácií), servlety pristupujú k databáze pomocou databázového rozhrania (**insilico.db**). Pokiaľ je dotaz kontrolného typu (napr. štart simulácie), je zavolaná jedna z funkcií triedy **insilico.env.SimulationManager** s príslušnými parametrami, ktorá sa ďalej stará o prácu so simuláciami. Trieda SimulationManager a samotné simulácie potom pristupujú k databáze opäť prostredníctvom databázového rozhrania.

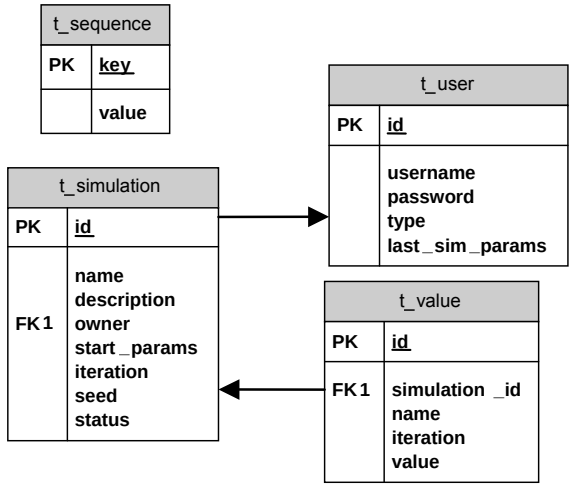


Obrázok 27 - Tok dát medzi prvkami simulačného prostredia

5.1.3 Databáza

Výsledky a súčasný stav simulácií sú po každej iterácii zapísané do databázy (Obrázok 28). Tabuľka t_simulation obsahuje informácie o danej simulácii a do tabuľky t_value sú zapisované výsledky simulácie pre každú iteráciu.

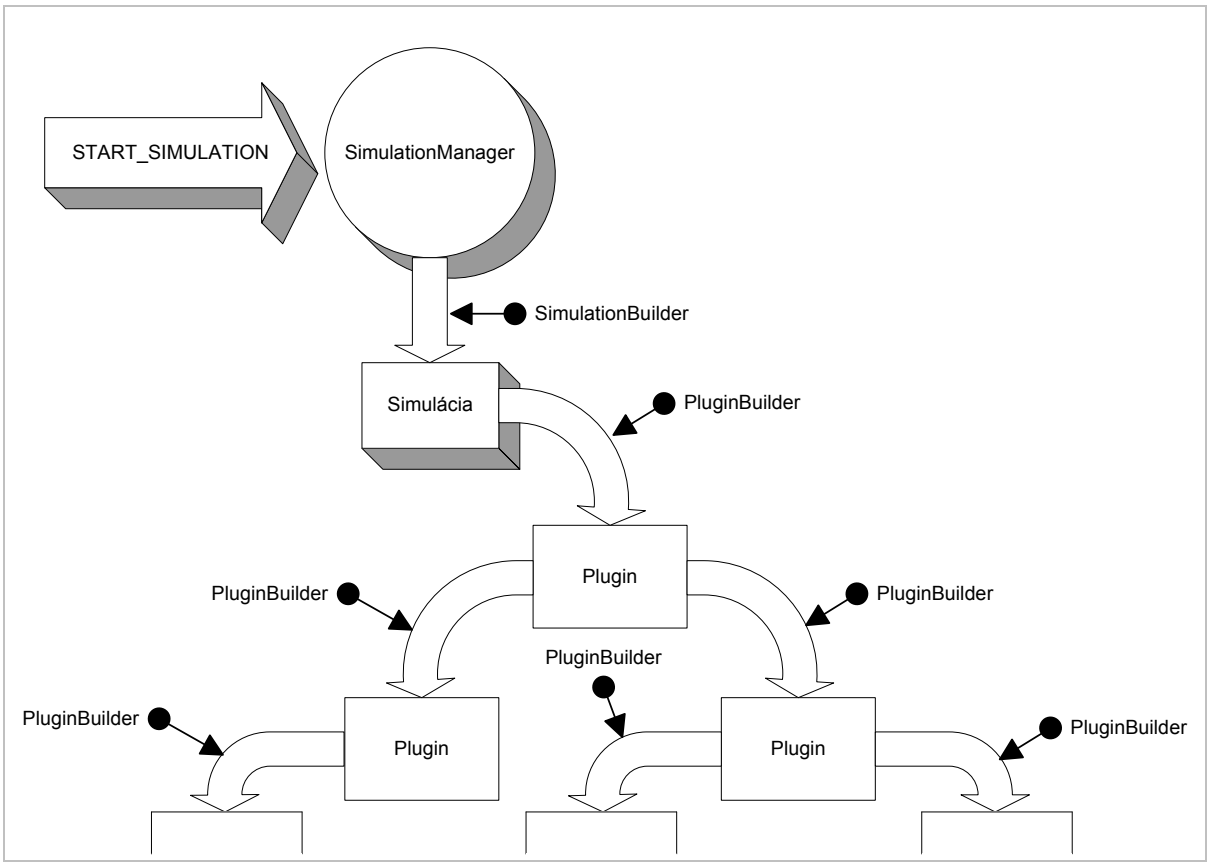
Okrem toho sú v databáze uložení aj používatelia systému v tabuľke t_user kvôli autorizácii prístupov k simuláciám.



Obrázok 28 - Štruktúra databázy

5.1.4 Vytvorenie simulácie

Vytvorenie simulácie je inicializované triedou SimulationManager, keď dostane od webového rozhrania podnet k štartu simulácie. Pri štarte aplikácie používateľ posiela systému XML dokument s nastavením simulácie (viac informácií v časti 5.2.2 - Spustenie simulácie). Štruktúra tohto dokumentu určuje, ako bude simulácia zostavená. SimulationBuilder podľa neho vytvorí inštanciu triedy odvodenej od rozhrania[12] ISimulation. Táto simulácia potom podľa vnorenej časti XML dokumentu zostaví strom zásuvných modulov pomocou triedy PluginBuilder (Obrázok 29). Každému zásuvnému modulu sú predané jeho inicializačné parametre ako podmnožina celého XML dokumentu.



Obrázok 29 - Spôsob inicializácie zásuvných modulov

[12] Rozhranie v jazyku Java podobný typ ako napríklad abstraktná trieda v jazyku C++. Udáva funkcie, ktoré musia odvodené triedy implementovať. Vďaka tomu je možné navrhnúť modulárny systém, kde sa môžu objekty navzájom zamieňať a meniť funkcionalitu bez zmeny konzistencie programu.

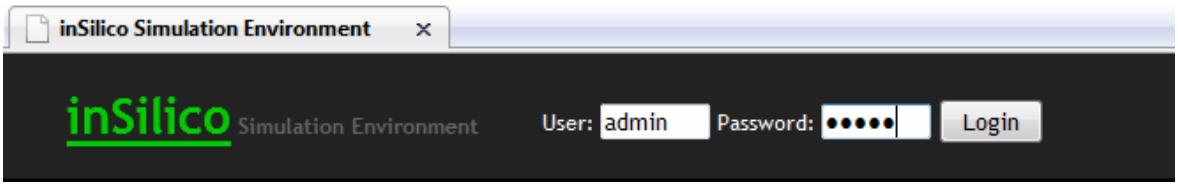
5.2 Práca s prostredím

Táto časť popisuje ovládanie prostredia z pohľadu používateľa-experimentátora. Predpokladá, že existujú napísané simulačné triedy (zásuvné moduly), ktoré môže experimentátor využívať.

5.2.1 Prihlásenie do systému

Pre prácu so simuláciami je potrebné prihlásiť sa do systému. Každý používateľ má právo spúšťať nové simulácie a sledovať simulácie, ktoré sám spustil. Používateľské účty sa vytvárajú podľa žiadosti na správcu servera.

Pre prihlásenie je potrebné vyplniť používateľské meno do poľa *User*, používateľské heslo do poľa *Password* a odoslať formulár stlačením tlačidla *Login* (Obrázok 30).

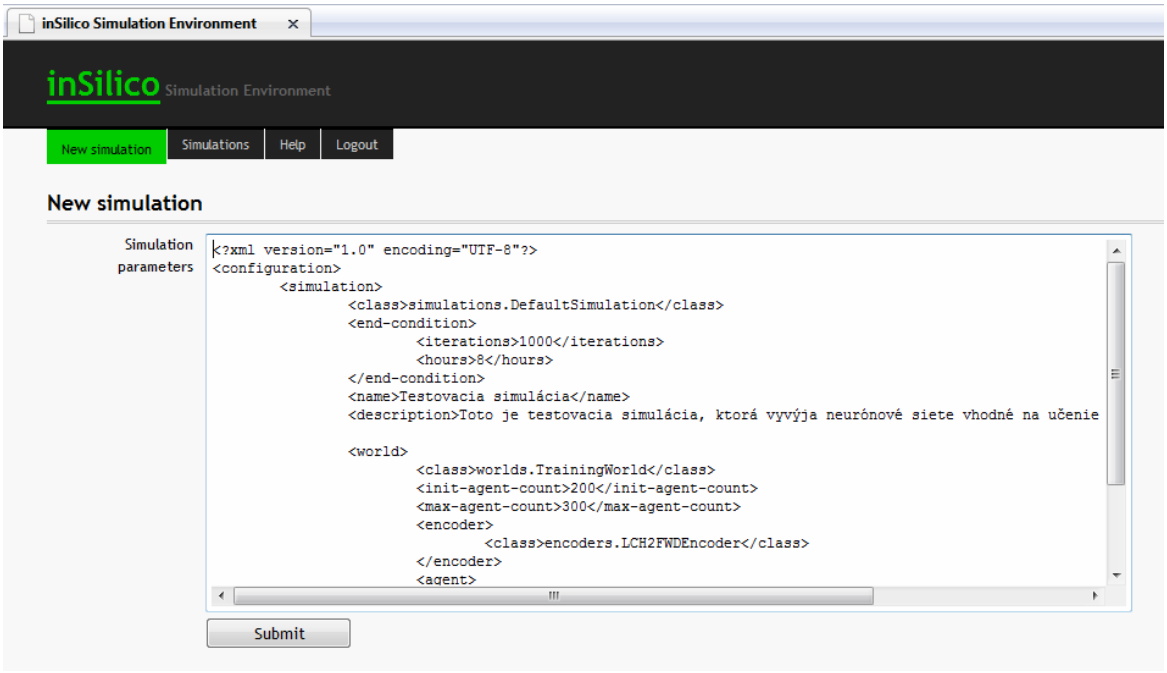


Obrázok 30 - Prihlásenie do systému

Po prihlásení vidí používateľ hlavné menu a zoznam simulácií, ktoré spustil, s možnosťami popísanými v časti 5.2.3 - Správa simulácie. Poslednou položkou menu je *Logout*, ktorá slúži na odhlásenie systému po ukončení práce. Simulácie, ktoré nie sú ukončené, budú bežať aj po odhlásení zo systému.

5.2.2 Spustenie simulácie

Novú simuláciu je možné spustiť v záložke *New Simulation* (Obrázok 31). Parametre simulácie sa nastavujú v textovom poli Simulation parameters a odosielajú tlačidlom *Submit*.



Obrázok 31 - Spustenie novej simulácie

Parametre sa zadávajú vo formáte XML. Štruktúra tohto XML dokumentu nie je pevne daná, jedinou podmienkou je tvar:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
      <simulation>
            <class>trieda_simulacie</class>

            vnorene_tagy

      </simulation>
</configuration>
```

Všetky ďalšie vnorené tagy závisia na vybranej triede simulácie. Trieda simulácie (tak ako aj triedy vnorených zásuvných modulov) sa udáva v tagu <class> aj s názvom balíka v ktorom sa nachádza (s dôrazom na malé a veľké písmená). Koreňovým adresárom pre zásuvné moduly je balík insilico.plugins, takže názov balíka modulu sa uvádza len od tejto cesty ďalej.

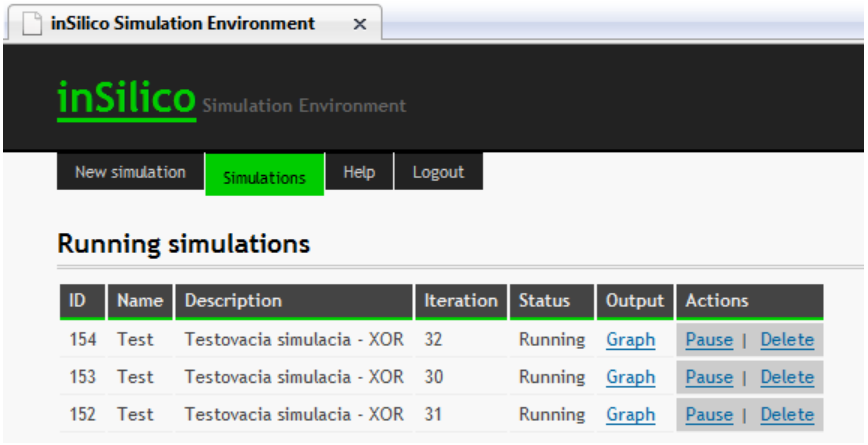
Napríklad, pri spustení novej simulácie, ktorá je implementovaná v triede insilico.plugins.simulations.DefaultSimulation a s použitím zásuvného modulu sveta insilico.plugins.worlds.DefaultWorld s paramterom max-agent-count (maximálny počet agentov), bude spúšťacie XML vypadať nasledovne:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
      <simulation>
            <class>simulations.DefaultSimulation</class>
            <world>
                  <class>worlds.DefaultWorld</class>
                  <max-agent-count>100</max-agent-count>
            </world>
      </simulation>
</configuration>
```

Zoznam zásuvných modulov simulácie sa líši podľa použitej triedy, tak ako aj následne závisia napríklad použiteľné zásuvné moduly v module <world> od zvolenej triedy reprezentujúcej svet. Preto je, žiaľ, pri spúšťaní simulácie, nutné poznať požadované moduly a submoduly simulácie a zoznam potrebných parametrov (napríklad zo zdrojového kódu).

5.2.3 Správa simulácie

Po kliknutí na položku menu *Simulations* je možné prezerať práve bežiace simulácie, ovládať ich a sledovať ich výstupy (Obrázok 32).

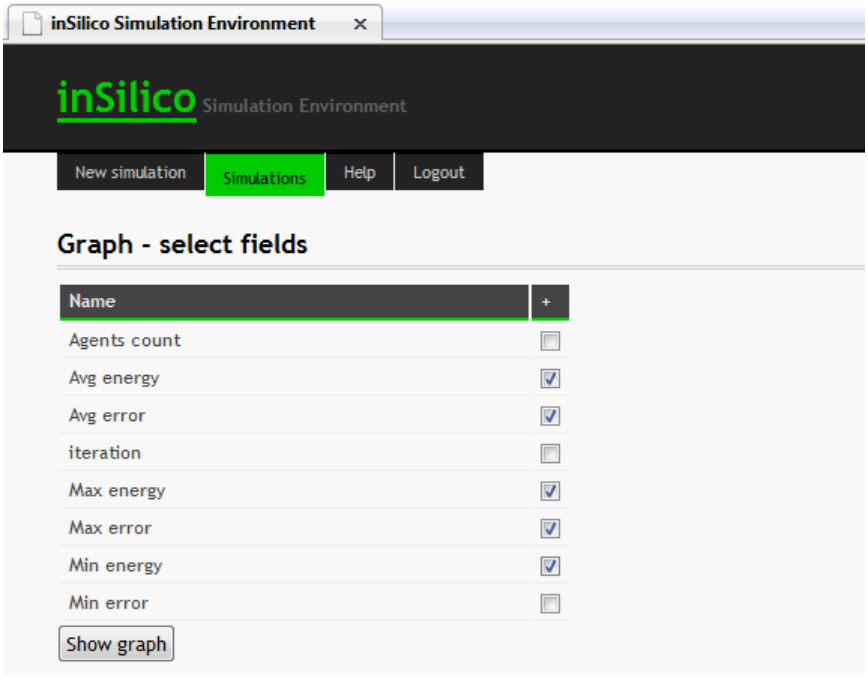


Obrázok 32 - Výpis bežiacich simulácií prihláseného používateľa

Simuláciu je možné pozastaviť kliknutím na odkaz *Pause* a znovu obnoviť pomocou odkazu *Resume*. Pozastavená simulácia sa uloží na pevný disk a nespotrbúva žiadne systémové zdroje.

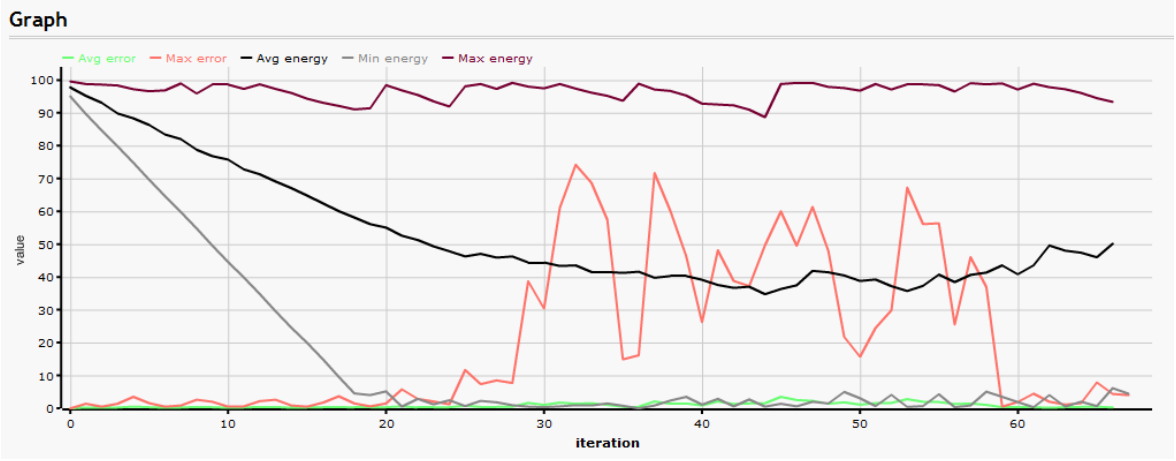
Kliknutím na odkaz *Delete* bude simulácia zmazaná, no jej výstupné dáta zostanú uložené v databáze, kvôli ošetreniu nechceného zmazania.

Po kliknutí na odkaz *Graph* sa zobrazí zoznam premenných, ktoré simulácia počas svojho behu vygenerovala. V tomto zozname je potrebné zaškrtnúť hodnoty, ktoré sa majú zobraziť súčasne na grafe (Obrázok 33).



Obrázok 33 - Výber premenných, ktoré sa majú zobraziť v grafe

Graf sa zobrazí po kliknutí na tlačidlo *Show graph* (Obrázok 34).



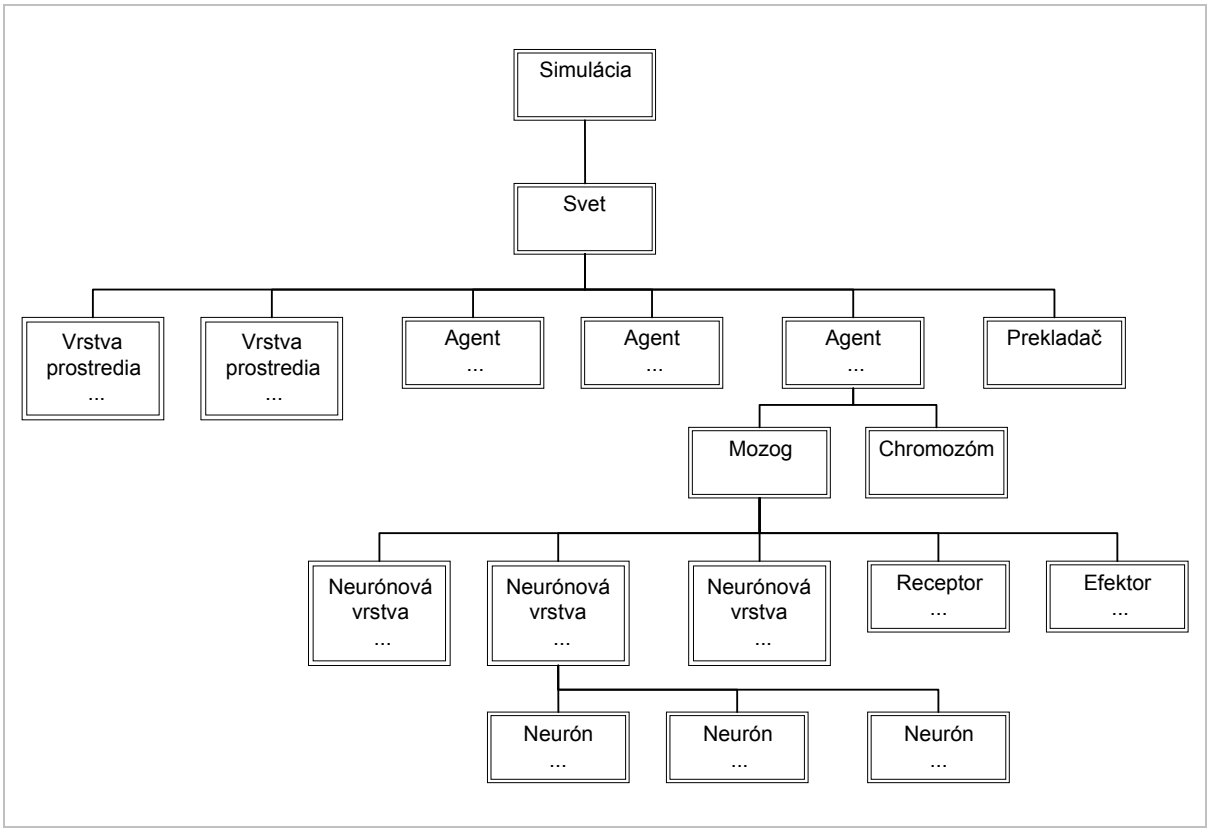
Obrázok 34 - Ukážka grafu

5.3 Testovacia simulácia

Keďže vytvorené prostredie samé o sebe nedokáže spustiť žiadnu simuláciu, ktorá by nebola naprogramovaná ako zásuvný model, bolo nutné na otestovanie jeho funkčnosti vytvoriť testovaciu simuláciu aj so všetkými potrebnými zásuvnými modulmi.

5.3.1 Návrh

Bola navrhnutá simulácia, ktorej úlohou je vyvinúť viacvrstvé neurónové siete s dopredným šírením chyby schopné naučiť sa funkciu XOR pomocou metódy Backpropagation. Preto bola navrhnutá štruktúra zásuvných modulov tak, aby ju bolo možné využiť aj pre iné podobné úlohy (Obrázok 35). Navrhnutú štruktúru je možné využiť pre takmer všetky viac či menej abstraktné multi-agent problémy.



Obrázok 35 – Návrh štruktúry simulácie

Základom simulácie je svet, ktorý má tri základné zložky – vrstvy, agentov a prekladač. Najjednoduchšou časťou je prekladač, ktorý dokáže podľa chromozómu vytvoriť agenta. Ide v podstate o pomocnú triedu, ktorá musí byť samostatne napísaná pre každú použitú kombináciu Agentov a Chromozómov.

Vrstvy si je možné predstaviť ako oddelené problémy, ktoré agenti riešia (čo nevylučuje ich vzájomnú závislosť). V simulácii robotov pohybujúcich sa v priestore, by mohli existovať vrstvy: fyzikálna (implementácia hmoty a fyzikálnych zákonov), akustická (šírenie zvukových vĺn), svetelná (šírenie optických signálov). Každá vrstva má priradené vlastné receptory a efektory agentov, ktorým predáva informácie, alebo ich naopak spracúva. V horeuvedenej fiktívnej fyzikálnej vrstve by receptormi mohli byť reprezentácie hmatového orgánu a gyroskopu, efektormi svaly spôsobujúce pohyb.

Poslednou časťou sveta sú agenti. Tí sa skladajú z dvoch častí – mozgu a chromozómu. Chromozóm je dedičná informácia, podľa ktorej budú zostavení prípadní potomkovia. Mozog je kontrolný orgán agenta (neurónová sieť), ktorý obsahuje vstupy receptorov, vrstvy neurónov a výstupy efektorov. Každá vrstva neurónovej siete môže obsahovať ľubovoľné neuróny (resp. kontrolné jednotky, ktoré nemusia byť ani simuláciou skutočného neurónu, ani matematickým modelom neurónu).

5.3.2 Implementácia

Ako bolo uvedené vyššie, boli implementované zásuvné moduly tak, aby sa v prostredí mohli vyvinúť agenti schopní efektívneho učenia XOR problému pomocou metódy Backpropagation. Štruktúru tried tejto simulácie zobrazuje Obrázok 36.

Za zmienku stoja hlavne triedy TrainingWorld, TrainingLayer, TrainingReceptor, TrainingActuator, FFWDNeuralBrain, LinearChromosome a LCH2FFWDEncoder.

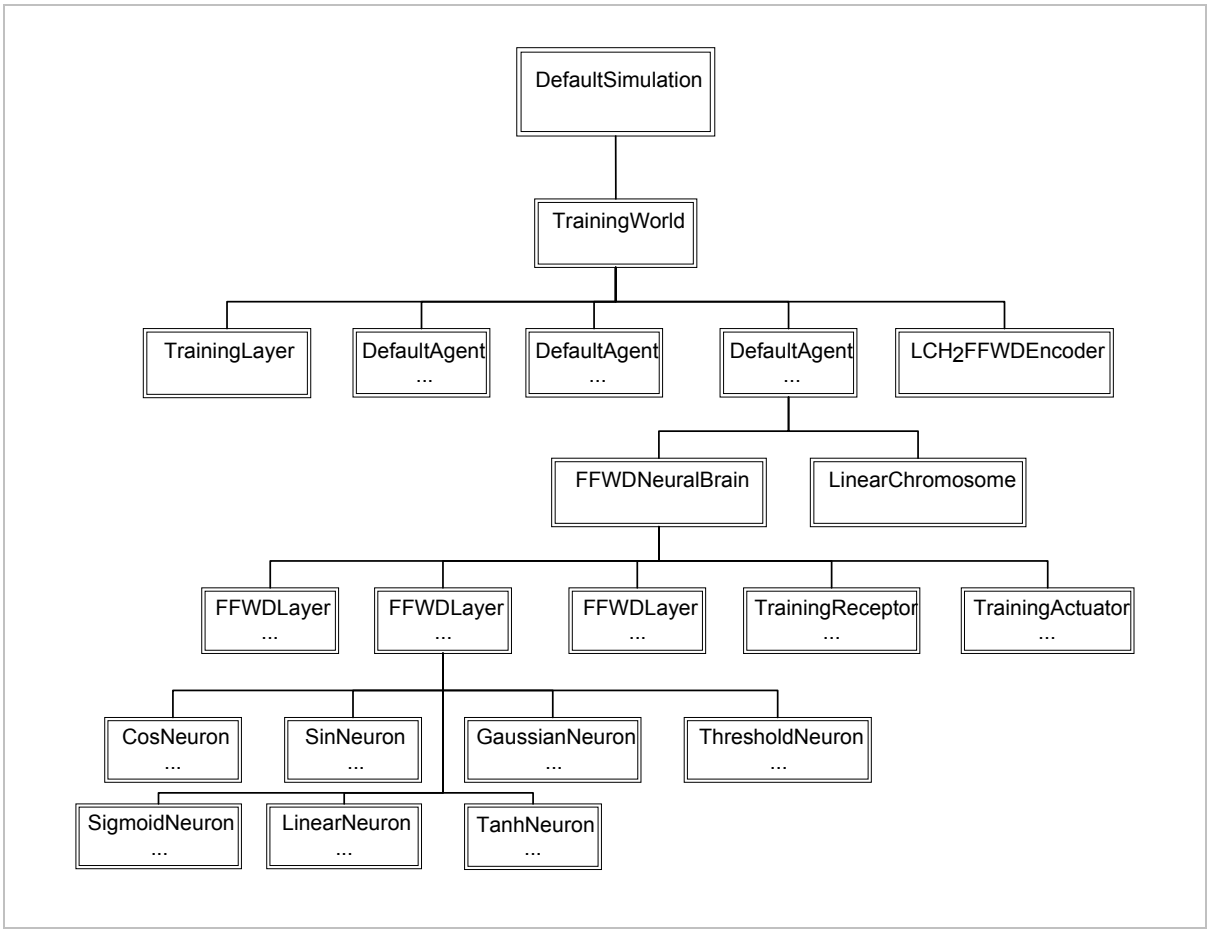
TrainingWorld je trieda, ktorá okrem zložiek popísaných v predchádzajúcej časti obsahuje tránovaciu množinu funkcie XOR. Táto množina je nastavená aj vrstve TrainingLayer, ktorá má na starosti cyklické predávanie týchto hodnôt na receptory mozgu, ktoré sú typu TrainingReceptor, a získavanie hodnôt z efektorov typu TrainingActuator. Tieto hodnoty sú potom v objekte triedy TrainingWorld porovnávané s testovacími a výsledná chyba je predaná mozgu (FFWDNeuralBrain).

Mozog prispôsobí váhy medzi neurónmi metódou Backpropagation a svet TrainingWorld daného agenta odmení pridaním energie za zmenšenie chyby oproti predchádzajúcej iterácii, a potrestá odobraním energie podľa veľkosti momentálnej chyby. Okrem toho je organizmu odobraná energia za každý neurón v mozgu.

V prípade, že agent dosiahol najmenšiu chybu, môže sa skrížiť s náhodným agentom. Vzniknú vždy dvaja potomkovia, pretože chromozómy (LinearChromosome) obidvoch

partnerov sa prekrížia na náhodnom mieste a z obidvoch nových chromozómov sú vytvorení noví agenti. Predtým, samozrejme, prejdú obidva chromozómy procesom mutácie. Keďže trieda LinearChromosome kóduje jedinca do sekvencie celých čísel, mutácia pozostáva zo zmeny náhodného čísla (génu), pridania náhodného génu, alebo odobrania náhodného génu.

Potomkovia vzniknú vďaka inštancii triedy LCH2FFWDEncoder, ktorá číta gény z objektu typu LinearChromosome a na ich základe vytvára agentov typu DefaultAgent s mozgom FFWDNeuralBain. Tieto celočíselné gény predstavujú počet vrstiev neurónovej siete, počet a typ neurónov v jednotlivých vrstvách.



Obrázok 36 - Štruktúra implementovaných tried testovacej simulácie

5.3.3 Spustenie

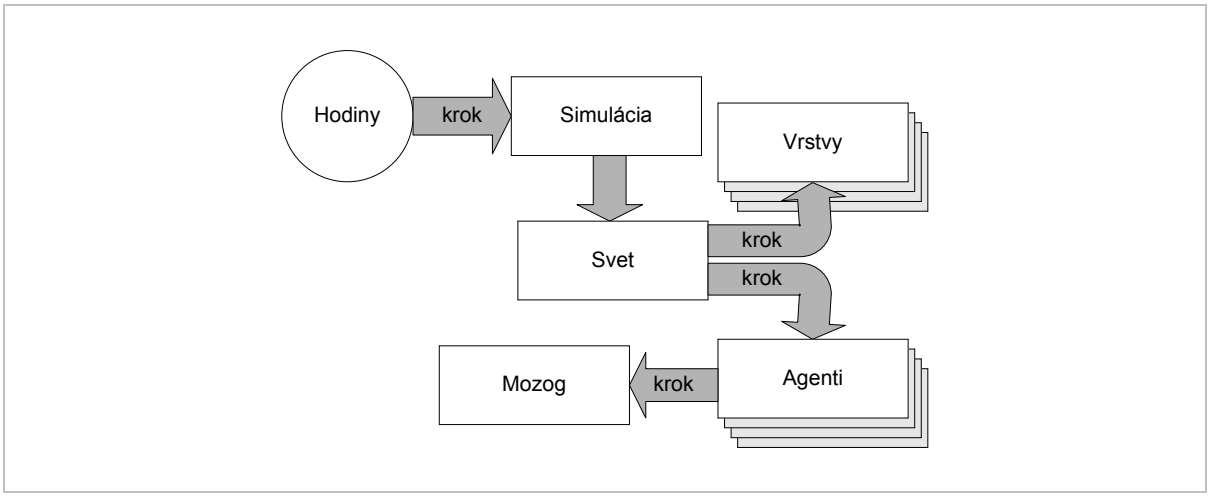
Simuláciu s využitím horeuvedených tried je možné spustiť odoslaním jednoduchým XML dokumenty pomocou spúštiaceho formulára popísaného vyššie.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
      <simulation>
            <class>simulations.DefaultSimulation</class>

            <name>Test</name>
            <description>
                  Testovacia simulacia – XOR
            </description>

            <world>
                  <class>worlds.TrainingWorld</class>
                  <init-agent-count>100</init-agent-count>
                  <max-agent-count>400</max-agent-count>
                  <encoder>
                        <class>encoders.LCH2FWDEncoder</class>
                  </encoder>
                  <agent>
                        <class>agents.DefaultAgent</class>
                  </agent>
            </world>
      </simulation>
</configuration>
```

Zásuvné moduly, ktoré tu nie sú uvedené (a zobrazuje ich Obrázok 36), sú pevne dané implementáciou simulácie. Po spustení tejto simulácie sa inicializujú moduly podľa horeuvedenej schémy (Obrázok 29). Po inicializácii sú nastavené hodiny, ktoré spúšťajú kroky simulácie s rovnomernými časovými rozostupmi. Simulácia spúšťa kroky sveta, svet spúšťa kroky agentov a vrstiev, agenti spúšťajú kroky svojich mozgov (Obrázok 37)



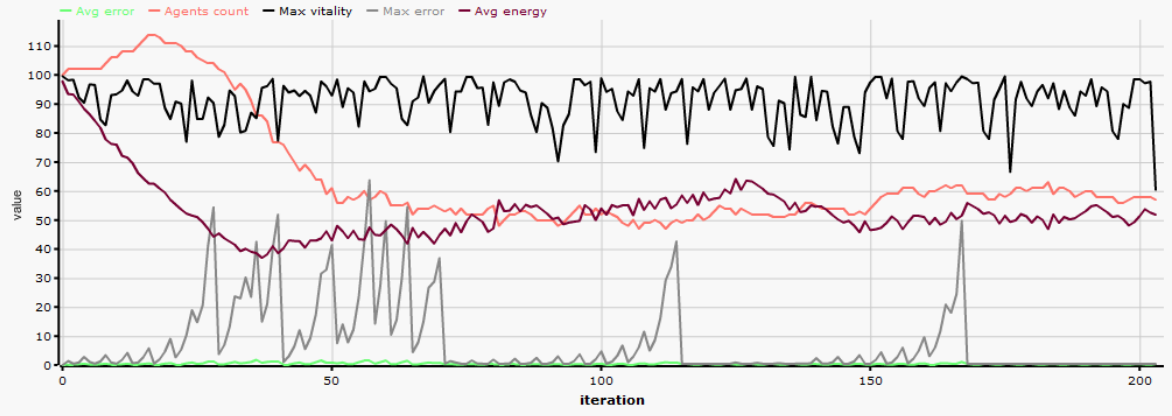
Obrázok 37 - Šírenie diskrétnych krokov medzi modulmi simulácie

5.3.4 Vyhodnotenie

Bolo spustených niekoľko simulácií, pričom všetky výsledky dopadli podobne. Sledované premenné boli počet agentov, chyba agentov (minimum, maximum, priemer), energia agentov (minimum, maximum, priemer) a maximálna vitalita. Vitalita predstavuje relatívne merítko kvality jedinca:

$$vitalita = \frac{energia}{e^{chyba}}$$

Podstatné charakteristiky vývoja populácie sú však hlavne počet agentov, maximálna chyba, priemerná chyba a priemerná energia. Práve tieto veličiny majú vo všetkých simuláciách rozoznateľný priebeh (Obrázok 38).



Obrázok 38 - Typický priebeh simulácie

Najtypickejší priebeh má krivka zobrazujúca počet agentov (na grafe svetlo červená – Agents count). V prvých zhruba 20 iteráciách rastie počet agentov, pretože všetci agenti sú na počiatku inicializovaní s maximálnou energiou, takže žiadny neumiera, ale rozmnožujú sa najlepší. Potom nastáva fáza „čistenia“, kedy agenti s rozsiahlými sieťami alebo nevhodnými typmi neurónov nedokážu hromadiť dostatok energie a ich réžia na neuróny a chyby je príliš veľká. Dôkazom toho je pokles priemernej energie (bordová – Avg energy). Približne po 100 iteráciách boli obidve hodnoty ustálené takmer vo všetkých simuláciách.

Ďalším typickým znakom sú kmity maximálnej chyby (šedá – Max error). Takmer v každej simulácii sa vyskytli obrovské kmity po približne 20 iteráciách a trvali 50 – 60 iterácií. Vo väčšine simulácií sa potom maximálna chyba ustálila na hodnote 0,5, no v niektorých simuláciách sa objavili menšie kmity (viď obrázok), ktoré sú spôsobené príchodom nového (nevhodného) genotypu do populácie vďaka mutácii.

Priemerná chyba (zelená – Avg error) sa približne po 100 iteráciách ustáli a osciluje okolo hodnoty 0,2. To nie je uspokojivý výsledok, no je to vysvetliteľné nedokonalosťou upravenej verzie algoritmu Backpropagation, ktorá bola použitá – neodlišuje sa adaptačná od aktivačnej fázy, a jedna položka trénovacej množiny znamená pre mozog agenta jednu iteráciu učenia.

I vzhľadom na chybu vyvinutých sietí je však zrejmé, že v simulácií pracuje komplexný systém navzájom interagujúcich agentov (hoci zatiaľ iba vďaka chromozómom), ktorý je i napriek prekmitom schopný usporiadať sa do rovnovážneho stavu.

5.4 Budúcnosť

Takmer všetky použité technológie sa ukázali pre vývoj tohto prostredia ako neoptimálne. Server Glassfish má príliš veľkú réžiu a zároveň je paralelizácia v ňom veľmi obtiažna. Preto bude v budúcnosti nevyhnutné webové rozhranie systému nahradiť vlastným serverom, a simulačné prostredie spúšťať ako samostatný aplikáciu.

Ďalšou nevýhodnou voľbou bolo použitie Java Persitence API, ktoré síce uľahčuje programovanie a sprehľadňuje kód, no jeho réžia pri častých zápisoch je príliš veľká. Prácu s databázou bude preto nutné presunúť na úroveň SQL dotazov a zápisy sledovaných hodnôt vykonávať jednorázovo po niekoľkých iteráciách.

Tak isto by bolo vhodné v rámci webového klienta využiť Java applet, ktorý by plnil úlohu grafického rozhrania (napr. s použitím frameworku Processing) a umožnil by sledovanie procesov simulácie v reálnom čase.

Prostredie bolo navrhnuté ako modulárne, čo umožňuje tvorbu ďalších typov simulácií v budúcnosti. V prvom rade by bolo vhodné implementovať fyzikálnu vrstvu, k nej náležiace receptory a efektory, a agentov schopných pohybu v nej. Potrebná je aj implementácia chromozómu využívajúceho L-systémy, vďaka ktorým bude možné z neho vygenerovať komplexnú nepravidelnú neurónovú sieť. Posledným z plánov do budúcnosti je aplikácia tzv. Hebbovského učenia na túto sieť.

ZÁVER

V tejto práci bol vypracovaný prehľad problematiky umelého života. V prvej časti tohto prehľadu bol popísaný vývoj umelého života a techniky, ktoré tento vývoj priniesol, pretože väčšina z nich je použitá v aplikáciách popisovaných ďalej. V druhej časti boli popísané aplikácie a simulačné prostredia z tejto oblasti. V závere boli stručne zhodnotené vlastnosti a dojmy z týchto prostredí.

Po analýze prostredí bolo navrhnuté vlastné simulačné prostredie. Toto prostredie bolo navrhnuté ako rozšíriteľná klient-server aplikácia. Stavebnými prvkami tohto prostredia sú moduly, ktoré môže používateľ spúšťajúci simuláciu do určitej miery kombinovať.

Prostredie bolo naprogramované ako serverová aplikácia pre server Glassfish. Ako úložisko dát generovaných prostredím bola použitá databáza PostgreSQL. Používateľ komunikuje s prostredím pomocou webového rozhrania.

V závere práce bol vytvorený a spustený test prostredia, ktorý potvrdil funkčnosť prostredia i evolúcie v ňom, ale i poukázal na nedostatky učiaceho algoritmu použitého v tomto teste.

Tento dokument slúži ako obecný teoretický základ k ďalšej práci v tejto oblasti, a aplikáciu, ktorá vznikla súbežne s týmto dokumentom je možné využiť ako základ robustného serverovo orientovaného simulačného prostredia.

CONCLUSION

In this thesis there is outlined an overview of artificial life issues. In the first part of this overview was described the evolution of artificial life and techniques brought by this evolution, since most of them are used by applications described later. In the second part applications and simulation environments in this area were described. In the end of this part, the characteristics and experience of the environments were evaluated.

After analyzing the environments there was designed own simulation environtment. This environment was designed as extendable client-server application. Buildin blocks of this environment are the modules, that can be combined by the user starting the simulation.

The environment was programmed as a server application for the Glassfish server. As a repository of data generated by the environment was used PostrgeSQL database. Users interact with the environment using a web interface.

In the end of the thesis, there was creater and launched a test of the environment, which confirmed the functionality of the environment an evolution in it, but also pointed to shortcomings of the learning algorithm used in this test.

This document serves as a theoretical basis for further work in this area, and the application, which was created along with this document can be used as a basis for a robust server-based simulation environment.

ZOZNAM POUŽITEJ LITERATÚRY

Život a umelý život

[1] WOOLDRIDGE, M. *An Introduction to Multiagent Systems*. [s.l.] : [s.n.], 2002. 365 s.

[2] DAVISON, Paul G. *How to Define Life* [online]. [2004] [cit. 2009-05-19]. Dostupný z WWW: <http://www.una.edu/faculty/pgdavison/BI%20101/Overview%20Fall%202004.htm>.

[3] MORALES, J. *The Definition of Life* [online]. 1998 [cit. 2009-05-19]. Dostupný z WWW: <http://baharna.com/philos/life.htm>.

[4] MARGULISOVÁ, L. *Symbiotická planeta*. [s.l.] : [s.n.], 2004. 150 s.

[5] WHITELAW, Mitchell. *Metacreation : Art and Artificial Life*. [s.l.] : [s.n.], 2004. s. 281.

História

[6] LEVY, S. Artificial Life. [s.l.] : [s.n.], 1992. 390 s.

Súčasný stav

[7] KOMOSINSKI, M., ADAMATZKY, A. Artificial Life Models in Software. [s.l.] : [s.n.], 2005. 344 s.

[8] COLLINS, R. J., JEFFERSON, D. R. *AntFarm: Towards Simulated Evolution*. [s.l.] : [s.n.], 1991. 23 s.

[9] RAY, T.. *Documentation for the Tierra Simulator* [online]. 1998 [cit. 2009-05-22]. Dostupný z WWW: <http://life.ou.edu/pubs/doc/index.html>.

[10] YAEGER, L.. *Polyworld* [online]. [2008] [cit. 2009-05-25]. Dostupný z WWW: <http://www.beanblossom.in.us/larryy/polyworld.html>.

Vývoj software

[11] ECLIPSE FOUNDATION. *About the Eclipse Foundation* [online]. [2009] [cit. 2009-05-16]. Dostupný z WWW: <http://www.eclipse.org/org/>.

[12] *JavaServer Pages Overview* [online]. c1994-2009 [cit. 2009-05-16]. Dostupný z WWW: <http://java.sun.com/products/jsp/overview.html>.

[13] W3C. *Extensible Markup Language (XML)* [online]. 1996-2003 [cit. 2009-05-22]. Dostupný z WWW: <http://www.w3.org/XML/>.

[14] HEATON, J. *Introduction to Neural Networks with Java*. [s.l.] : [s.n.], 2005. 376 s.

[15] *Sun GlassFish Enterprise Server 2.1 Quick Start Guide* [online]. c2009 [cit. 2009-05-25]. Dostupný z WWW: <http://docs.sun.com/app/docs/doc/820-4334>.

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

DNA	Deoxyribonukleová kyselina.
RNA	Ribonukleová kyselina.
SOMA	Self Organizing Migrating Algorithm.
CPU	Central Processing Unit
VRML	Virtual Reality Modeling Language
GUI	Graphical User Interface
OS	Operačný systém
JRE	Java Runtime Environment
ME	Java ME – Java Mobile Edition
SE	Java SE – Java Standard Edition
EE	Java EE – Java Enterprise Edition
JPA	Java Persistence API
API	Application Programming Interface
JSP	Java Server Pages
XML	Extensible Markup Language
HTML	Hypertext Markup Language
SGML	Standard Generalized Markup Language
URL	Uniform Resource Locator
HTTP	Hypertext Transfer Protocol
XOR	Exclusive Or
SQL	Structured Query Language

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK