České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Tomáš B l o v s k ý

Studijní program: Softwarové technologie a management

Obor: Inteligentní systémy

Název tématu: Evoluce s otevřeným koncem pro optimalizaci neuronových sítí

Pokyny pro vypracování:

1. Nastudujte problematiku simulace otevřené evoluce.
2. Seznamte se s robotickým simulátorem ViVAE.
3. Navrhněte a implementujte vhodné typy senzorů.
4. Navrhněte pravidla simulovaného světa a naimplementujte odpovídající evoluční algoritmus.
5. Proveďte experimenty a sledujte chování populace. Soustřeďte se zejména na interakci ovlivněnou vzájemnou podobností jedinců.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Ing. Jan Drchal

Platnost zadání: do konce zimního semestru 2011/2012

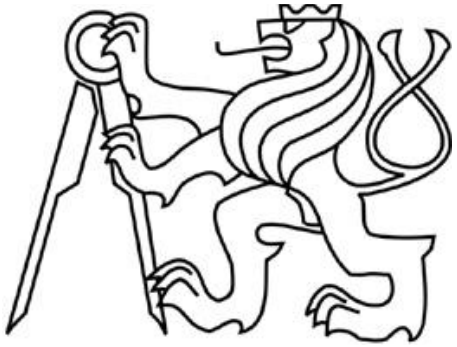
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 2. 2. 2011

České vysoké učení technické v Praze
Fakulta elektrotechnická



Bakalářská práce

Evoluce s otevřeným koncem pro optimalizaci neuronových sítí

Tomáš Blovský

Vedoucí práce: Ing. Jan Drchal

Studijní program: Softwarové technologie a management, strukturovaný, Bakalářský

Obor: Inteligentní systémy

12. května 2011

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne: 27. 5. 2011 ……

……..
Podpis

Abstrakt

Cílem této práce je navrhnout a implementovat otevřenou evoluci neurálních sítí ve virtuálním robotickém simulátoru ViVAE (Visual Vector Agent Environment). Výsledný systém má za úkol demonstrovat schopnost neurálních sítí učit se pouze na základě reálií okolního světa a bez předem stanoveného cíle a propagace kvalitních jedinců pomocí selektivního křížení.

Abstract

The aim of this work is to design and implement an open-ended evolution of neural networks in a virtual robot simulator ViVAE (Visual Vector Agent Environment). The resulting system has the task of demonstrating the ability of neural networks to learn only the realities of the outside world and without pre-defined objectives and promoting quality individuals through selective crossbreeding.

Obsah

Seznam obrázků

Kapitola 1

Úvod

1.1 Motivace

Evoluce umělých neuronových sítí představuje bezesporu velice atraktivní téma pro výzkum v oborech umělé inteligence a to z mnoha důvodů. Za prvé umělé neuronové sítě se snaží simulovat pochody skutečných neuronových sítí, z nichž jsou složeny centra myšlení všech živých tvorů člověka nevyjímaje, a jakožto základní prvek skutečné živé inteligence představují reálný základ pro vytvoření inteligence umělé. A za druhé, učení neuronových sítí pomocí genetických algoritmů následuje mechanismy skutečné evoluce a tato disciplína tedy obsahuje všechny prvky procesu, který vedl od jednobuněčných organismů až k člověku.

Bohužel výpočetní technika v současné době nedovoluje vytvoření umělé neuronové sítě velikostně srovnatelné s mozky většiny vyšších savců, včetně člověka, a je nemožné projít s takovouto sítí stovky miliónů let evoluce tak, jak tomu bylo u živých tvorů. Proto je vytvoření skutečné umělé inteligence touto cestou v nedohlednu. V současné době se umělé neuronové sítě používají často pro úkoly, u kterých je předem známý požadovaný výsledek a učení neuronové probíhá propagací těch aspektů sítě vedoucích k požadovanému výsledku.

V této práci jsem si dal za úkol demonstrovat schopnost umělé neuronové sítě učit se bez vnějšího zvýhodňování určitých znaků a pouze na základě reálií virtuálního světa, do kterého je síť vložena.

1.2 Struktura práce

- Teoretické pozadí
  - Tato část se zabývá rozborem několika metod obecných metod z prostředí neuronových sítí použitých v průběhu práce stejně tak jako prostředí, v němž byla práce implementována.

- Analýza a návrh řešení
  - V další kapitole bude upřesněn cíl práce, předpokládané výsledky a bude proveden rozbor jednotlivých kroků, pomocí nichž bylo cíle dosaženo.

- Implementace
  - Následující část popisuje implementaci jednotlivých kroků nastíněných v předchozí kapitole, případně změny předpokladů na základě vlastností prostředí ViVAE.

- Testování
  - Předposlední kapitola se zabývá testováním implementovaného programu a sumarizaci skutečných výsledků chování agentů.

- Závěr
  - V poslední části jsou konfrontovány předpokládané výsledky, tak jak byly nastíněny analýze problému, a skutečné výsledky dosažené testováním.

Kapitola 2

Teoretické pozadí

V této kapitole se pokusím ve zkratce obsáhnout teoretický základ své práce, který se týká především principů evoluce s otevřeným koncem [8], mechanismů neuronových sítí obecně, architektury neuronových sítí s proměnnou topologií konkrétně, a robotického simulátoru ViVAE [2,3]. Neuronové sítě tvoří poměrně rozsáhlé téma, podklady pro práci jsem čerpal z několika diplomových prací zabývajících se touto tematikou [1,3] a odborných článků na téma neuronových sítí s proměnnou topologií [4,5,6,7].

2.1 Otevřená evoluce

Otevřená evoluce nebo také evoluce s otevřeným koncem je určitou podkategorií genetických algoritmů.

2.1.1 Princip

Na rozdíl od klasického schématu genetického algoritmu, kde algoritmus končí po nalezení optimálního řešení (nebo po přiblížení se k němu) na základě uživatelem zadané ohodnocovací funkce, v otevřené evoluci k ničemu podobnému nedochází. Populace a její jedinci se dál vyvíjejí do nových forem a neexistuje zde nic jako stabilní stav.

2.1.2 Genom

Zatímco u většiny genetických algoritmů má genom neměnnou podobu, danou reprezentací řešeného problému, je tato varianta jaksi kontraproduktivní z pohledu otevřené evoluce protože algoritmus prohledává omezený prostor řešení - genom má většinou jen konečný počet možných kombinací. Proto bývá výhodné povolit genomu měnit v průběhu svou velikost (například při evoluci neuronových sítí použít síť s proměnnou topologií viz dále)

2.2 Vlastní adaptace

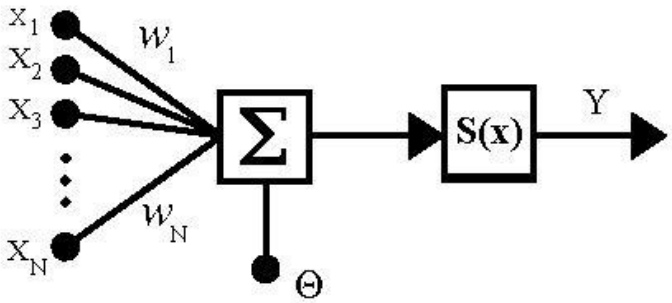
Vlastní adaptace [8] (Intrinsic adaptation) je jedním ze způsobů vývoje v soustavách, které mají co dočinění s evolucí, druhým způsobem je potom vnější adaptace (Extrinsic evolution). Na rozdíl od vnější adaptace, kde evoluce je řízená danou fitness funkcí, v případě vlastní adaptace probíhá evoluce samovolně jako odpověď na změnu prostředí. Jedním ze základních rysů systémů fungujících na principu vlastní adaptace je celková pružnost v oblasti fitness – každý jedinec v populaci ovlivňuje prostředí kolem sebe i ostatní jedince (přímo nebo nepřímo) a tím že mění svět kolem sebe mění i svoji vlastní schopnost v tom světě přežít a mění tak svoji vlastní fitness. Jako příklad vlastní adaptace lze vzít světovou biosféru – ta nemá žádnou předem určenou fitness funkci, a přesto v ní neustále probíhá vývoj.

2.3 Neuronové sítě

Umělé neuronové sítě jsou výpočetní struktury určené pro distribuované paralelní zpracování dat. Vzorem pro jejich vytvoření je chování biologických struktur.

2.3.1 Neuron

Umělá neuronová síť se skládá ze vzájemně propojených umělých neuronů, jejichž předobrazem jsou nervové buňky v tělech živých organismů, spojené synapsemi. Každý neuron má libovolný počet vstupů ale pouze jeden výstup. Existuje množství různých modelů umělého neuronu, nejjednodušším a nejčastěji používaným modelem je perceptronový neuron (Obr. 2.1).



Obr. 2.1: Model perceptronového neuronu, sečtením sumy vážených vstupů a prahové hodnoty Θ je získán vstup pro přenosovou funkci, výsledkem přenosové funkce je už samotný výstup neuronu.

Každý neuron je tvořen z následujících částí

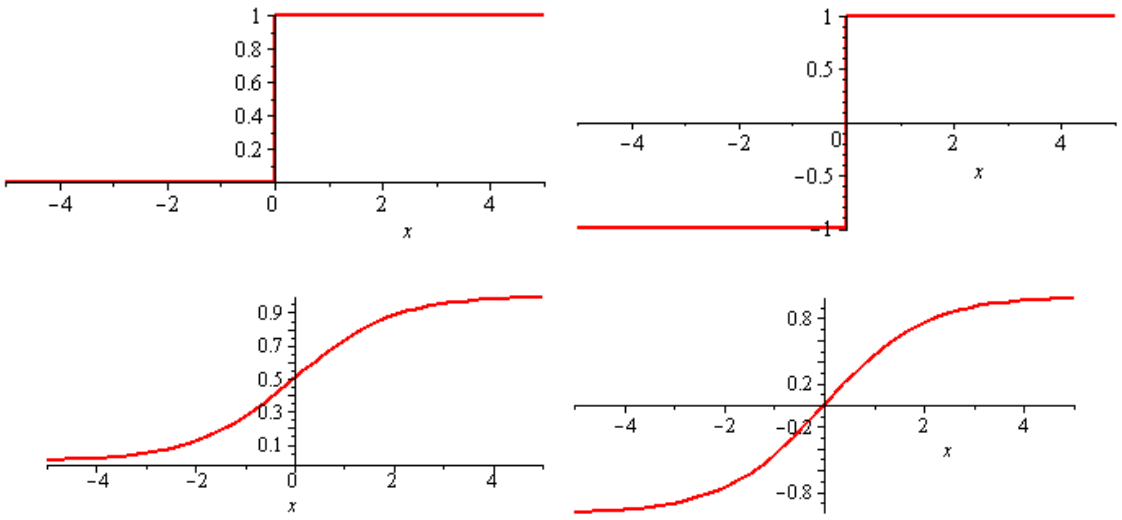
- x_i – i-tý vstup neuronu, v závislosti na vstupních datech jsou vstupy neuronu binární nebo spojité. (i nabývá hodnot od 1 do N, kde N je počet vstupů neuronu)
- w_i – váha i-tého vstupu neuronu. Nabývá zpravidla spojité hodnoty z intervalu kolem nuly (např. <-5,5>)
- Θ – práh neuronu
- S – přenosová (aktivační) funkce neuronu, zpravidla nelineární.
- Y – výstup neuronu

Výstup neuronu s N vstupy pak lze vyjádřit následujícím vztahem:

$$Y = S(\sum_{i=1}^{N}(w_i x_i) + \Theta) \qquad (2.1)$$

2.3.2 Aktivační funkce

Aktivační funkce neuronu převádí vnitřní potenciál neuronu (sumu vážených vstupů + práh) do předem definovaného oboru hodnot. Volba aktivační funkce závisí na tom, zda jako výstup neuronu požadujeme spojitou nebo binární hodnotu. Mezi nejčastěji používané aktivační funkce patří znaménková nebo Heavisidova funkce pro binární výstupní hodnoty a sigmoidální funkce nebo hyperbolická tangenta pro spojité výstupní hodnoty (Obr. 2.3)



Obr. 2.2: Základní typy přenosových funkcí, vlevo nahoře znaménková funkce, vpravo nahoře Heavisidova funkce, vlevo dole sigmoidální funkce, vpravo dole hyperbolická tangenta

2.3.3 Topologie sítě

Z pohledu topologie sítě je možné rozdělit všechny neurony v síti do tří základních kategorií, vstupní neurony, výstupní neurony a skryté neurony. Jako vstupní neurony lze označit ty, jejichž vstupy jsou signály z prostředí, a jako výstupní neurony ty, jejichž výstupy vedou zpět do prostředí. Tyto dvě skupiny lze nalézt v různém množství u každé neuronové sítě. Skryté neurony jsou takové neurony, jejichž vstupy a výstupy je spojují s jinými neurony a které se navenek neprojevují, síť nemusí obsahovat žádné skryté neurony.

2.4 NeuroEvolution of Augmenting Topologies (NEAT)

Jednou z nevýhod většiny typů umělých neuronových sítí je jejich fixní topologie. U takovýchto sítí je třeba zvolit jejich topologii na začátku, a jelikož pro různé úlohy je vhodná jiná topologie sítě, nemusí být zvolený model optimální pro zadanou úlohu a v takovém případě je nemožné najít optimální řešení v krajních případech nelze nalézt řešení žádné – například při modelování funkce XOR pomocí neuronové je nutné, aby v síti byly přítomny skryté vrstvy, a bez nich není možné zadanou funkci vymodelovat.

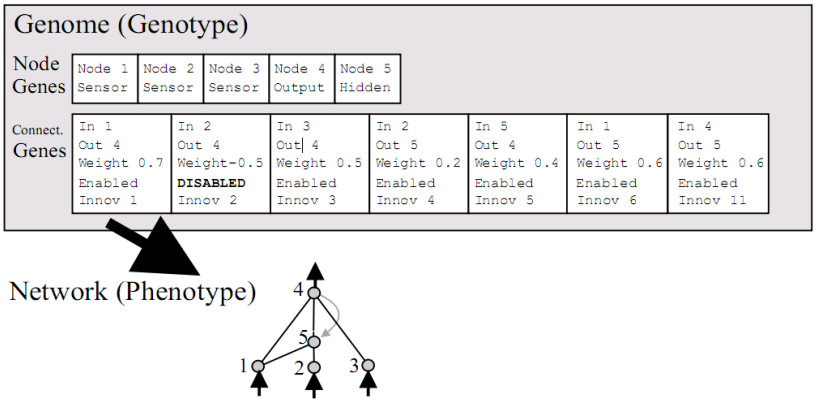
Za účelem eliminace tohoto problému bylo vyvinuto mnoho metod, které optimalizují současně váhy v síti a topologii sítí. Tyto modely jsou souhrnně označovány jako TWEANNs [4] (Topology and Weight Evolving Artificial Neural Networks). Všechny metody patřící mezi TWEANNs se různým způsobem vypořádávají s problémy vyplývajícími z podstaty neuniformní topologie

- Jak křížit sítě s nestejným počtem neuronů a neztrácet při tom informace
- Jak ochránit nové topologické prvky, než se optimalizují
- Jak minimalizovat výslednou strukturu (nenacházet zbytečně složitá řešení)

V následujících odstavcích se budu blíže věnovat modelu NEAT a jeho schopnosti vypořádat se s třemi výše uvedenými problémy TWEANNs.

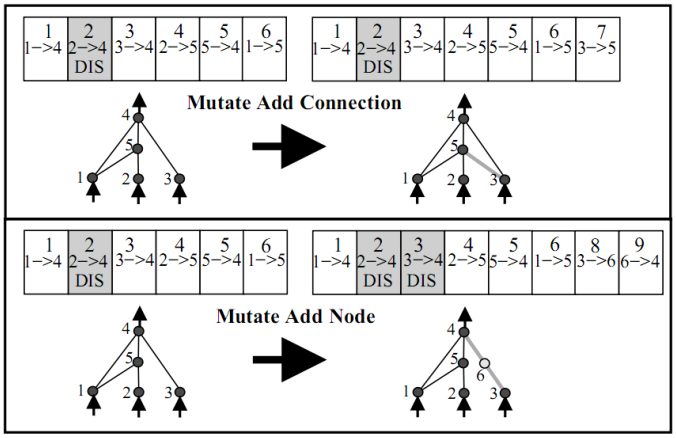
2.4.1 Genetické kódování

NEAT [4,5] je navržen tak, aby křížení dvou genomů bylo možno snadno spárovat odpovídající geny. Genom je reprezentován seznamem vrcholových genů reprezentujících vstupní, výstupní a skryté neurony, a seznamem spojovacích genů reprezentující spojnice mezi jednotlivými neurony (Obr 2.3). Každý gen je pak opatřen inovačním číslem, které umožňuje nalezení shodných genů – geny vytvořené při mutaci dostávají unikátní inovační číslo, které je dále propagováno všem potomkům, takže i po mnoha generacích je snadné zjistit u dvou různých jedinců jejich společné rysy.



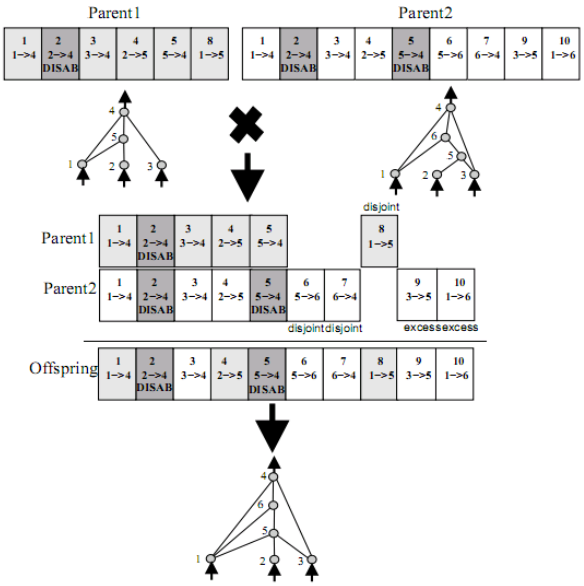
Obr. 2.3: Mapování genu v modelu NEAT [4], genom obsahuje v jednom seznamu uzlové geny definované typem (vstupní, skrytý, výstupní) a ve druhém seznamu synaptické geny definované vstupním a výstupním uzlem, vahou a inovačním číslem.

Mutace na síti může proběhnout několika způsoby. Nejběžnějším způsobem je váhová mutace optimalizující váhy spojovacích genů v genomu. Druhým typem mutace je pak mutace strukturální – ta probíhá buď přidáním spojnice mezi existujícími neurony, nebo rozdělením některé stávající spojnice a přidáním neuronu (Obr 2.4).



Obr. 2.4: Strukturální mutace v modelu NEAT [4], přidání synapse mezi dvěma neurony kde žádná synapse nebyla, nebo nahrazení jedné ze synapsí dvěma novými a novým neuronem, stará synapse je zakázána jedna z nových získá její váhové ohodnocení, zatímco druhá nová začíná s vahou jedna.

Křížení probíhá tak, jak je naznačeno na obrázku 2.5. Nejdříve jsou spárovány všechny geny se souhlasným inovačním číslem – ty jsou následně děděny náhodně – a nesouhlasné (disjoint) a přesahující (excess) geny jsou zděděny od kvalitnějšího z rodičů, nebo náhodně v případě rovnocenné kvality.



Obr. 2.5: Křížení v modelu NEAT [4], geny přítomné u obou rodičů jsou voleny náhodně, nesouhlasné geny jsou voleny buď náhodně, nebo od lepšího z rodičů.

2.4.2 Ochrana inovací

Jednotlivé strukturální mutace by však samy o sobě nepřežily – každé upravení struktury vede nezadržitelně ke zhoršení fitness, dokud se jejich váhy neoptimalizují. Proto používá NEAT rozřazováním druhů. Pomocí následujícího vztahu je měřena vzájemná kompatibilita dvou jedinců

$$\delta = \frac{c1*E}{N} + \frac{c2*D}{N} + c3 * W \qquad (2.2)$$

E je počet přesahujících genů, D je počet nesouhlasných genů a W je průměrný rozdíl vah souhlasných genů. Pomocí koeficientů c1, c2 a c3 je možno nastavit důležitost těchto tří faktorů. Na základě stanoveného prahu podobnosti δ_t probíhá rozřazování do druhů – každý jedinec je testován na kompatibilitu proti vybranému zástupci každého druhu

(náhodný jedinec z předchozí generace patřící k danému druhu) a pokud δ jejich kompatibility nepřekročí δ_t je umístěn do tohoto druhu. V případě že nezapadá do žádného z druhů, vytváří druh vlastní a stává se automaticky jeho zástupcem. Fitness je následně sdílena všemi zástupci druhu a menší druhy jsou zvýhodněny před většími, díky tomu dokáže nová struktura přežít – díky své odlišnosti se dostane do samostatného druhu a má tak do začátku jistou protekci než se dokáže přizpůsobit a rozmnožit.

2.4.3 Minimalizace struktury

Většina modelů neuronových sítí patřící mezi TWEANNs [4] začíná s populací náhodných topologií – kromě toho, že mnohá takto generovaná řešení mohou být kompletně života neschopná (například nemají propojené vstupy s výstupy), se pak ještě potýkají s problémem minimalizace výsledku – často nelze zaručit, že nalezený výsledek je optimální za použití minimálního počtu neuronů. Na druhou stranu NEAT začíná vyhledávání s uniformní populací sítí bez skrytých neuronů, a ke každému výstupu jsou připojeny všechny vstupy. Nové struktury jsou tvořeny pouze strukturálními mutacemi a každá nová struktura se musí nejprve osvědčit, nežli je zaručeno její přežití. To znamená, že všechny strukturální změny, které v NEATu nastanou, jsou vždy ospravedlnitelné. A díky tomu že populace začínají s minimální strukturou, má tento systém dva efekty. Zaprvé je výsledné řešení minimální a zadruhé je minimální i prostor prohledávaných řešení – NEAT vždy hledá řešení přes menší počet dimenzí než modely TWEANNs s náhodnou iniciační populací nebo modely sítí s fixní topologií, což mu dává výhodu lepšího využití operačního času oproti jiným přístupům.

2.4.3 Použití v reálném čase

Varianta pro učení neuronových sítí s proměnnou topologií v reálném čase využívá upravenou verzi modelu NEAT [4]. Tato metoda nazývaná rtNEAT [7] výhody dané reprezentací genomu v normálním NEATu, rozdílem je pak nahrazování jedinců samostatně v průběhu nekonečně běžícího scénáře oproti nahrazování celé generace v původní verzi. To přináší postupné vylepšování chování agentů v průběhu času místo aby docházelo k obrovskému skoku v okamžiku vytvoření nové generace následované obdobím beze změn.

2.5 Visual Vector Agent Environment (ViVAE)

Pro simulaci otevřené evoluce jsem použil prostředí pro simulaci mobilních agentů Visual Vector Agent Environment [2] (ViVAE), navržené a implementované Bc. Petrem Smejkalem v rámci jeho bakalářské práce. Jde o knihovnu jazyka Java, která se zabývá simulací dvourozměrného prostředí, objektů v tomto prostředí a interakcí mezi objekty a prostředím a objekty navzájem.

Prostředí rozeznává čtyři typy objektů, jsou jimi povrchy (surfaces), agenti (actives) překážky (passives) a senzory (sensors). Překážky se pak dále dělí na pohyblivé a

nepohyblivé (movable, fixed). Mezi interakce mezi objekty patří například tření, jakožto vlastnost povrchu – v závislosti na hodnotě tření se mohou agenti pohybovat po daném povrchu různou rychlostí, nebo kolize mezi objekty – agenti nemohou projíždět sebou navzájem ani skrz překážky, za zmínku stojí i vzájemné silové působení – agenti se mohou vzájemně přetlačovat či pohybovat pohyblivými překážkami. Poslední typ objektů (senzory) je z interakcí ve světě jaksi vyjmut – nemají na něj vliv kolize ostatních objektů a senzor sám se pohybuje spolu s agentem, kterému patří. Senzor pak dokáže poskytovat agentovi informace o kolizích s jinými objekty v aréně a zprostředkovává tak agentovi smyslové vnímání (vstupy do neuronové sítě).

Prostředí dovoluje uložení a načtení celé arény včetně všech objektů v podobě souboru vektorové grafiky SVG (Scalable Vector Graphic). Informace jsou v souboru přehledně uloženy jako objekty v několika vrstvách, jejichž typ je určen jmenovkou. Tuto možnost jsem ve svém projektu bohužel nepoužil, protože při své práci jsem potřeboval za běhu dynamicky generovat další objekty a průběžně je odebírat což varianta s načítání arény neumožňuje.

Kapitola 3

Analýza a návrh řešení

3.1 Cíle

Cílem práce je navrhnout a implementovat v prostředí mobilních agentů scénář otevřené evoluce za pomoci vlastní adaptace. Jde o snahu vytvořit v rámci experimentu fungující ekosystém, v němž se budou agenti pohybovat, sbírat potravu, interagovat mezi sebou, rozmnožovat se a umírat – v závislosti na množství nalezené potravy. Agenti budou získávat informace o okolním světě pomocí senzorů a jejich pohyb bude řízen neuronovou sítí s proměnnou topologií [4]. Evoluce neurokontrolerů nebude řízena žádnou fitness funkcí a jediným ukazatelem kvality bude doba, po kterou dokázal agent ve světě přežít [8]. Mezi další cíle patří rozřazení agentů do druhů na základě vzájemné podobnosti, a implementace evoluce tělesné schránky agenta probíhající paralelně a vzájemně se ovlivňující s evolucí chování (neurokontroleru).

3.2 Předpokládané výsledky

Mezi předpokládané výsledky scénáře evoluce náhodně generovaných agentů patří následující:

I. Agenti se postupně naučí vyhledávat potravu, a to buď ve formě rostlinné potravy (primární zdroj energie) nebo konzumací energie ostatních agentů.
II. Agenti se naučí vyhýbat se překážkám v aréně a zdem arény. Možnost použití překážek a stěn jako vodicích prvků při prohledávání prostoru
III. Postupná specializace agentů na jeden z typů potravy (masožravci a býložravci) v rámci jednotlivých druhů
IV. Agenti se naučí rozeznávat příslušníky vlastního druhu a nebudou je identifikovat jako potenciální zdroj potravy
V. Přizpůsobení tělesné schránky robotů vyvinutému chování (bod III.)

3.3 Návrh řešení

V následující podkapitole bude proveden rozbor základních prvků programu, které bude třeba v průběhu práce implementovat.

3.3.1 Svět

Svět představuje prostředí, ve kterém budou agenti operovat, a jako takový bude tvořen dvourozměrnou obdélníkovou arénou. Aréna bude ze všech čtyř stěn uzavřena neprůchodnými detekovatelnými stěnami a bude obsahovat několik fixních detekovatelných překážek. Za běhu programu se bude konstantní rychlostí tvořit na náhodných pozicích jídlo – tj. bude existovat konstantní přísun energie do systému. Podobně bude aréna schopna tvořit náhodně generované agenty (náhodně nastavené váhy neuronové sítě) v případě že počet agentů v aréně klesne pod kritickou hodnotu – zpočátku je předpokládaná vysoká úmrtnost agentů a existuje reálná možnost několikanásobného vymření celé populace hladem, než se bude některý z agentů schopný najít dostatek potravy k přežití a rozmnožování a započne tak proces evoluce. Aréna nebude mít nastavenou žádnou horní hranici počtu agentů, tato hranice by měla vzniknout jako důsledek rostoucího odsunu energie ze systému s rostoucím množstvím agentů a velikost populace by se měla stabilizovat na hodnotě, kdy všichni agenti spotřebují za jeden časový okamžik tolik energie, kolik přibude v podobě generovaného jídla.

3.3.2 Agent

Agent tvoří autonomní jednotku v rámci virtuálního světa. Každý agent je opatřený senzory schopnými detekovat ostatní objekty ve světě, neurální sítí řídící pohyb agenta na základě výstupu senzorů a fyzickými atributy ovlivňujícími některé interakce agenta s okolím (např. velikost, rychlost, dosah senzorů).

Jedním z atributů agenta je energie – hodnota energie ubývá v každém kroku simulace a v případě že její hodnota dosáhne nuly tak agent umírá a jeho tělo je odstraněno z arény – takováto smrt je konečná a veškerý pokrok dosažený agentem je ztracen, v případě že energie dosáhne určeného maxima (tato hodnota bude pravděpodobně dvojnásobkem startovní energie náhodně generovaných robotů) agent se rozmnoží – vytvořený nový agent zdědí fyzické atributy po svém primárním rodiči a bude umístěn do jeho blízkosti. V okamžiku rozmnožení bude náhodně zvoleno, zda neurální síť potomka bude vytvořena pouze mutací sítě primárního rodiče nebo bude ze zbytku populace vybrán sekundární rodič a dojde ke křížení neurálních sítí.

Jediným úkolem agenta je přežít a jediným jeho ohodnocením je doba, po kterou se dokázal udržet na živu. Agent může tohoto cíle dosáhnout navyšováním své energie buďto sbíráním jídla nebo na úkor ostatních agentů (kradením jejich energie).

3.3.3 Druhování

Agenti budou rozřazováni do druhů na základě podobnosti neurálních sítí. Podrobný princip rozřazování bude založen na principu druhování použitém v případě modelu neurálních sítí rtNEAT [7] – variací na model NEAT použitelnou v reálném čase.

Princip je založen na rozdělení do druhů stejně jako v případě klasického NEATu s tou výjimkou že je předem určen ideální počet druhů které by se měly v populaci vyskytovat a prahová hodnota je v průběhu simulace upravována podle toho zda je populace rozdělena na příliš mnoho nebo příliš málo druhů.

3.3.4 Senzory

Agent bude vybaven dvěma typy senzorů

Smyslové senzory zprostředkovávají agentům smyslové vjemy z nejbližšího okolí virtuálního světa, ve kterém se agent pohybuje. O každém objektu v dosahu senzorů by měly být agentovi dostupné tyto informace:

- Příslušnost objektu k některému typu (zeď, jídlo, agent)
- Příslušnost k vlastnímu nebo cizímu druhu (případě objektu typu agent)
- Vzdálenost objektu od agenta
- Úhel, pod kterým se objekt nachází, vzhledem ke směru agenta

Smyslové senzory by pak měly pokrývat je určitou kruhovou výseč na přední straně agenta se středem v přímém směru před agentem, tudíž nezprostředkovávají agentovi pohled do zadu. Rozpětí senzorů (velikost úhlu kruhové výseče senzory pokryté) by mělo být jedním z parametrů definujících senzory. Druhým parametrem by byla jejich délka, tj. vzdálenost na jakou jsou schopny detekovat objekt. Oba parametry by měly být předmětem náhodných změn v rámci evoluce fyzické schránky agenta, přičemž velikost rozpětí senzorů bude nepřímo úměrná jejich délce – evoluce by měla najít ideální kompromis mezi schopností periferního vidění a schopností vidět na dálku.

Druhým typem senzorů pak budou „ústa". Ústa budou malým senzorem na přední straně robota schopným ohodnocovat jedlost čehokoli, co se dostane do dosahu senzoru – v případě že senzor vyhodnotí, že v dosahu úst je jedlý objekt agent iniciuje akci jezení (přesun energie z cíle k agentovi)

3.3.5 Neurokontroler

Centrálním prvkem každého agenta je neuronová síť založená na modelu s rozšiřující se topologií NEAT. Pro nultou generaci agentů (bez rodičů) je tvořena jako síť bez skrytých neuronů s náhodně inicializovanými váhami. Pro všechny ostatní agenty je pak síť vytvořena buďto mutací rodičovské sítě (jeden rodič) nebo zkřížením dvou sítí (v případě dvou rodičů. Neuronová síť zůstává po celý život agenta neměnná a jediný způsob úpravy vah je mutací a křížením v rámci rozmnožování. Díky modelu topologie NEAT je možné počítat s postupným rozšiřováním neuronové sítě o skryté neurony.

Vstupy do sítě tvoří v první řadě výstupy smyslových senzorů a výstup senzoru úst, v druhé řadě pak několik stavových informací agenta jako například množství zbývající energie a hodnota absolutního natočení v prostoru (informace které by mohla síť použít k zlepšení vyhledávacího vzorce). Počet vstupů sítě bude určen dynamicky podle množství senzorů.

Výstupy sítě budou určovat chování agenta v daném okamžiku. Mezi ně patří rychlost dopředného pohybu a změna směru pohybu.

3.3.6 Vizualizace

Vzhledem k principům otevřené evoluce si program vyžaduje možnost prezentace výsledků algoritmu za běhu, tj. aby bylo kromě sledování chování agentů v aréně možno přímo z obrazovky přečíst i další informace o současném stavu populace a o každém agentovi zvlášť. Mezi tyto informace o jednotlivých agentech napomáhající k vyhodnocení výsledků patří:

- Složení stravy v průběhu života
- Fyzický a genetický věk *
- Okamžitá energie
- Fyzické atributy, které nejsou pozorovatelné pouhým okem
- Druhová příslušnost
- Generace
- Míra vývoje neurální sítě (počet přidaných neuronů a synapsí)

* fyzický věk vyjadřuje stáří agenta od jeho narození, zatímco genetický věk vyjadřuje stáří agentovy linie předků, tj. jak dlouho je daný genom předáván (v upravené formě způsobené mutací a křížením) od svého náhodného vygenerování.

Pro přehledné zobrazení všech těchto informací bude vhodné naimplementovat několik přepínatelných filtrů, kdy každý filtr bude zobrazovat příslušné informace vedle každého agenta, případně podle těchto informací škálovat barevné tónování agenta pro snadné rozpoznání celkových tendencí zobrazených informací.

Kapitola 4

Implementace

Tato kapitola se zabývá samotnou implementací algoritmu navrženého v předchozí kapitole a rozborem jejích jednotlivých kroků.

4.1 Prostředí

Jako základ pro implementaci celého programu bylo použito prostředí pro simulaci mobilních agentů ViVAE [2] jehož autorem je bc. Petr Smejkal a pro neurokontrolery jednotlivých agentů byla použita implementace modelu sítě s proměnnou topologií NEAT v Javě jejímž autorem je ing. Jan Drchal.

Pro modifikaci stávajících dat a implementaci nových funkcí bylo použito open source vývojové prostředí pro jazyk Java NetBeans.

4.2 Součásti programu

Součásti programu jsou následující:

- **ViVae - projekt pro Netbeans**
  - v balících *robot* a *vivae* jsou umístěny třídy tohoto virtuálního prostředí a všech jeho objektů.
  - balík *vivae.blovstom* obsahuje všechny modifikované třídy původního programu (případně třídy poděděné od některé původní), pro lepší přehlednost a odlišení od původního programu začínají jména těchto tříd slovem **Tom-**.
- **Neat – soubor JAR**
  - Obsahuje všechny soubory původní implementace modelu NEAT a HyperNeat pro Javu. Pro tuto práci byly stěžejní balíky *neat* a *common* pokrývající potřebné třídy pro modelování a mutaci jednoduché sítě.
  - Uložen ve formě kompilovaného souboru mezi ostatní knihovny projektu ViVae

4.3 Vlastní implementace

Následující odstavce se zabývají popisem vlastní implementace. Jedná se především o popis jednotlivých implementovaných tříd a jejich funkcí a vzájemných interakcí. Druhá část se pak zabývá rozborem parametrů algoritmu.

4.3.1 Rozbor tříd

První skupinou jsou třídy zajišťující vytvoření a časově neomezené fungování světa.

TomExperiment.java je spouštěcí třídou celého programu. Obsahuje metodu *main*() v jejímž průběhu dochází k iniciaci arény a celého okna programu v metodě *createArena*a() a následné spuštění simulace metodou *startExperiment*().

TomManager.java zajišťuje správu objektů v aréně a na požádání arény přidává nové objekty jídla a náhodně generované agenty metodami *createFood*() a *createRandomRobot*() a zajišťuje tvorbu potomků v případě, že některý agent překročí kritickou hranici energie metodou *createCrossedRobot*(). Poslední funkcí je rozdělování agentů do druhů metodou *distinctSpecies*().

TomArena.java představuje vlastní reprezentaci celého virtuálního světa a je tvoří obsah hlavní okno programu (jedná se o potomka třídy JPanel). V rámci iniciace arény jsou vytvořeny fixní překážky v aréně. Obsahuje seznam všech objektů v aréně a v nekonečné smyčce spouští metodu *step*() ve které jsou postupně vyhodnocovány akce jednotlivých agentů a jejich dopad na ostatní objekty. Ve stejné metodě je pak voláno přidávání nových agentů pokud jejich počet klesne pod kritickou hodnotu a periodické přidávání jídla pokud jeho množství nepřesahuje povolené množství. Na konci každého kroku dochází k překreslení celého obsahu arény metodou *repaint*().

Do druhé skupiny tříd patří třídy jednotlivých objektů v aréně.

TomFood.java představuje objekt jídla které je průběžně přidáváno na náhodné pozice. Tato třída je potomkem třídy Movable a proto je i jídlo v aréně pohyblivým objektem který reaguje na strkání agentů. Při svém vytvoření má fixní hodnotu energie, kterou může poskytnout a po jejím odčerpání je odebráno z arény. Vykreslování je prováděno metodou *paintComponent*() volanou v rámci překreslování arény a barva vykresleného tvaru se v závislosti na zbývající energii mění od zelené po černou.

TomBot.java stěžejní třída celé simulace je potomkem třídy Robot a reprezentuje entitu každého agenta. Pomocí metody *moveComponent*() mění polohu agenta v aréně na základě výstupů neurokontoleru a pozic ostatních objektů v aréně. Metoda *paintComponent*() stejně jako u předchozího případu vykresluje agenta na plochu aréně. Třída obsahuje seznam senzorů daného agenta – pomocí metody *getSensoryData*() čte výstupy ze všech senzorů naráz. Navíc si každý agent uchovává svoje statistiky pomocí instance třídy TomBotStats a .

Poslední skupinou tříd jsou součásti používané třídou TomBot k vytvoření funkčního agenta.

TomBotStats je vnitřní třída v souboru TomBot.java a uchovává seznam fyzických atributů a statistik agenta a metod pro práci s těmito údaji. Mezi atributy patří velikost, metabolismus, délku a rozpětí senzorů a jsou uloženy ve formě poměru od jejich základních hodnot (v intervalu <0.5; 2>). Mezi uchovávané statistiky patří například věk, aktuální energie, generace a množství snědené potravy. V každé iteraci simulátoru (v okamžiku pohybu agenta) je volána metoda *step*() zajišťující aktualizaci statistik a mimo jiné odečtení přiměřeného množství energie. Odečítaná hodnota není uniformní, je vypočítávána pomocí metody *prize*() a je ovlivněna hodnotami atributů velikosti a metabolismu – vyšší hodnoty sice dávají agentovi větší výdrž respektive rychlost ale současně způsobují vyšší energetickou cenu každého kroku.

TomNeatController.java je třída využívající třídy Net a Genom z projektu Neat pro vytvoření a uchovávání neuronové sítě. Instance třídy jsou vytvářeny metodami TomManageru a následně jsou uloženy v příslušných instancích TomBotů. V závislosti na způsobu vzniku je síť iniciována jednou ze tří metod *intitRandom*(), *intitOneParent*() a nebo *intitTwoParent*(). Hlavní funkcí této třídy je metoda *moveControlledObject*() která si od svého agenta vyžádá výstupy ze senzorů a vrátí mu rychlost, kterou se bude agent v příštím okamžiku pohybovat a úhel o který se pootočí. Poslední funkcí je dopočítávání odlišnosti od jiné instance neurokontroleru pro účely rozdělení do druhů viz odstavec ochrana inovací v o modelu NEAT v kapitole Teoretický úvod.

TomController.java starší verze výše zmíněného TomNeatControlleru. Nepoužívá neuronovou síť typu NEAT ale plně propojenou rekurentní neuronovou síť FRNN (Full Recurrent Neuron Network). Jinak implementuje stejné metody jako výše zmíněná třída a při použití v programu jsou zaměnitelné. Není použita ve finální verzi programu.

TomMultiSensor.java třída senzoru napsaná po vzoru třídy DistanceSensor. Kromě metody *getDistance*() která ze seznamu vivae objektů vypočítá vzdálenost nejbližšího z nich k počátku senzoru (vrací číslo v rozsahu <0;1>) implementuje i metodu *getCloseVivaes*() používanou k redukci seznamu objektů s ohledem na vzdálenost od agenta. Protože metoda *getDistance*() je výpočetně velmi náročná, dochází zmenšením velikosti prohledávaného seznamu k výraznému urychlení celého programu. Mezi další implementované metody patří *getCloseFood*(), *getCloseRobots*() a *getCloseWalls*() které redukují seznam objektů i s ohledem na typ díky čemuž může jeden senzor detekovat současně vzdálenost k několika objektům různého typu.

TomMouthSensor.java představuje zjednodušenou formu výše uvedeného vzdálenostního senzoru. Vrací pouze seznam objektů zasahujících do zóny senzoru.

4.3.2 Nastavení parametrů

Vzhledem k povaze implementovaného algoritmu sestávala značná část implementace v nalezení vhodných hodnot parametrů, protože nebylo možné předem odhadnout, jak bude algoritmus na tu kterou hodnotu reagovat a v případě nevhodného nastavení buď agenti nebyli schopni evolučně postoupit (přežití bylo příliš obtížné) nebo se algoritmus stával nevyváženým a docházelo ke zhroucení celého systému. Proto byly parametry nastavovány převážně experimentálně na základě pozorovaných výsledků.

V následném výčtu jsou jmenovány všechny hlavní parametry použité při simulaci. Ke každé skupince je připojen odstavec popisující mechaniky v rámci daného parametru.

Svět

Minimální počet agentů v aréně = 40
Maximální počet jídla v aréně = 200
Rychlost generování potravy = 0.1
Hodnota jídla v jednotkách energie = 1000

Při rychlosti generování potravy jednou za deset kroků přibude v každém kroku 100 jednotek energie. Při průměrné spotřebě jedné jednotky energie na krok je takovýto přísun schopen uživit 100 agentů. Nastavení maxima je důležité v prvotních fázích simulace kdy agenti nejsou schopni jídlo najít a aréna by se brzy zahltila. Minimální počet 40 agentů představuje číslo, při kterém netrvá extrémně dlouho, než je nalezen vzorec chování potřebný k přežití.

Agent

Počáteční energie = 750
Maximální energie = 1500

Hodnoty počáteční a maximální energie platí pouze pro první generaci agentů – agent první generace má 750 kroků, aby našel zdroj jídla – to dává, při úrovni schopností náhodně generovaného agenta, dostatek času pro slibné jedince a naopak dostatečně rychle likviduje nevhodné kandidáty. Při dosažení energetické hladiny maximální energie dochází k rozmnožení. V dalších generacích představuje počáteční energie polovinu maximální energie rodiče a hodnota maximální energie se lehce mění s mutací atributů (obecně agenti s vyšší spotřebou energie potřebují vyšší zásobárnu energie, aby přežili)

Délka senzorů = 50
Počet senzorů = 5

Větší množství senzorů výrazně snižuje rychlost provádění simulace (výpočet objektů v dosahu senzorů je nejsložitější výpočet programu) na druhou stranu menší počet senzorů nedává agentovi dostatečné informace o okolí. Základní délka senzorů 50 dává agentovi možnost přehlédnout přiměřený povrch arény.

Velikost, metabolismus = <0.5 ; 2>
Délka senzorům rozsah senzorů = <0 ; 2>

V případě hodnot atributů šlo hlavně o určení rozsahu, v kterém se mohou pohybovat. Základní hodnotou všech atributů je 1 a v rámci mutací po rozmnožování se mohou pohybovat v zadaném rozsahu. Velikost dává výhodu při požírání ostatních agentů (rychlejší přesun energie podle poměru velikostí útočník oběť) a přímo zvětšuje dosah senzoru úst a zvětšuje maximální energii, metabolismus přímo ovlivňuje rychlost pohybu agenta. Tyto dva atributy současně zvyšují cenu přežití v jednotkách energie a za krok jako kompenzaci výhod získaných jejich zvýšením a nevýhod získaných snížením. Déla a rozpětí senzorů není nijak energeticky penalizovaná, ale oba atributy jsou navzájem doplňkové – mutace jsou uzpůsobeny tak že součet obou hodnot je vždy 2.

Rychlost jedení agent-potrava = 10
Rychlost jedení agent-agent = 40

Ty to parametry přímo ovlivňují rychlost transferu energie ze zdroje energie k příjemci. Postupné požírání potravy vytváří nutnost zůstat u ní stát (případně ji tlačit před sebou) než je pozřena má velký dopad na vývoj chování agentů. Vyšší rychlost požírání ostatních agentů se ukázalo jako jeden ze způsobů jak podpořit vznik masožravců – v případě stejné rychlosti požírání neexistuje dostatek motivace k lovení pohyblivých cílů místo statických.

Křížení a rozdělení do druhů

Šance křížení = 75%
Šance strukturální mutace – přidání neuronu = 2%
Šance strukturální mutace – přidání synapse = 10%

V případě vzniku nového agenta je 75% šance, že bude použito křížení dvou agentů, ve zbylých případech jde pouze o mutaci. Ze všech mutací je pak celkem 12% šance na strukturální mutaci a ve zbylých případech dojde pouze k mutaci vah. V případě že je zvolena metoda rozmnožování se dvěma rodiči je jako druhý rodič zvolen nejstarší přislušník stejného druhu (agenti jsou uchováváni v seznamu a noví agenti jsou přidáváni na konec – proto nejstarší je vždy na prvním místě seznamu.

Počet druhů = 5-10

K přerozdělování do druhů bude docházet v okamžiku změny populace (přidáním nového agenta). Rozdělování bude probíhat podle principu rozdělování do druhů u modelu rtNEAT, tedy udržováním přibližně konstantního počtu druhů a pružnou změnou prahu pro zařazení v případě zmenšení nebo zvětšení přes hranici. Počet druhů v rozmezí 5-10 dává při velikosti populace 100 přibližně 10-20 jedinců na druh.

4.4 Vizualizace a ovládání

Principem filtrů je vytvoření globální proměnné, která může nabývat několika hodnot, a v závislosti na aktuální hodnotě této proměnné pak vybrat jeden ze způsobů vykreslování objektů v aréně. Dalším rozšířením ovládání je pak možnost pozastavit simulaci.

4.4.1 Ovládání

Při vytváření okna v metodě *createArena*a() třídou TomExperiment je do okna (JFrame) vloženo horizontální lišta (JMenuBar) obsahující dvě rozbalovací menu (JMenu) naplněná jednotlivými položkami (JMenuItem). Každé položce je přiřazena akce pomocí interfacu ActionListener. Pro snazší ovládání je každé položce přiřazena také klávesová zkratka pomocí metody *setAccelerator*() třídy JMenuItem.

První menu s popiskem „Pauza“ umožňuje pozastavení a opětovné spuštění simulace – buď přímo vybráním položek menu, nebo klávesovými zkratkami ‘p’ pro pozastavení a ‘P’ pro spuštění. Pozastavení je realizováno nastavením logického parametru třídy TomArena, který v případě hodnoty *true* ukončuje metodu *step*() stejné třídy hned po její iniciaci díky čemuž je čas v aréně efektivně zmražen.

Druhé menu s popiskem „Typ vizualizace“ se stará o samotné přepínání filtrů. Jednotlivé položky nastavují hodnotu statické proměnné colorDisplaied třídy TomBotStats. Během vykreslování agenta (metoda *paintComponent*() třídy TomBot) je pak volána metoda *filter*() v třídě TomBotStats která na základě nastaveného filtru vypíše v okolí pozice agenta vybrané statistiky a nastaví barvu štětce kterou je pak agent vykreslen.

4.4.2 Druhy filtrů

Program obsahuje 8 různých filtrů zobrazujících jednotlivé skupiny statistik. Na následujících řádcích je popsán každý z filtrů včetně příslušné klávesové zkratky. V rámci zobrazených statistik jsou údaje zobrazované o jeden řád nižší – vzhledem k počtu kroků za sekundu je řád jednotek nečitelný a kratší výpis zlepšuje viditelnost v aréně (týká se i zobrazeného počtu kroků v rohu obrazovky). Příklad jednotlivých filtrů je zobrazen na obrázku 4.1.

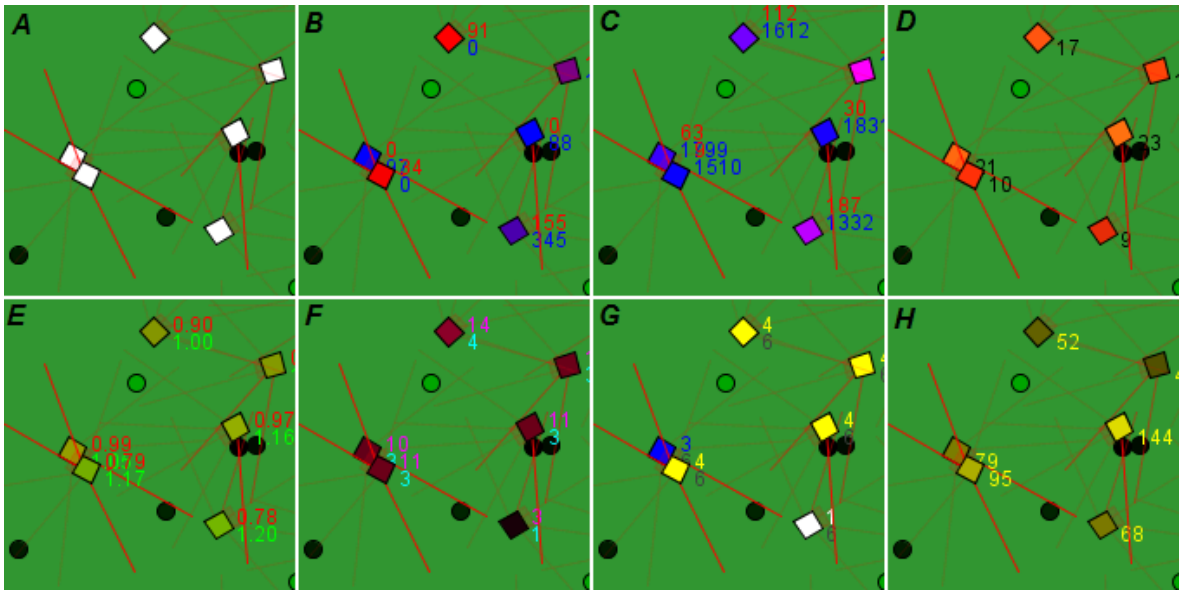
Nic – jedná se o základní filtr bez jakýchkoli zobrazovaných informací. Je možné ho použít v případě, že situace v aréně se stává málo přehlednou. Agenti mají bez výjimek základní bílou barvu. Zapnutí filtru je pomocí klávesy ‘q’.

Dieta – filtr ukazuje množství přijaté potravy za život agenta. Zobrazeny jsou dvě hodnoty: množství energie přijaté z potravy (modře) a množství energie ukořistěné ostatním agentům (červeně). Podle poměru obou hodnot je agent zabarven do červena nebo do modra v případě že jeden zdroj převažuje a do fialova v případě vyrovnanosti. Agent, který nepřijal žádnou potravu, je černý. Aktivace filtru klávesou ‘w’.

Věk – tento filtr zobrazuje věk agenta (červeně) a jeho genetický věk, tj. časovou délku jeho rodokmenu (modře). Barva agenta pak odráží poměr obou hodnot – starý agent je červený, agent s dlouhým rodokmenem modrý a starý agent s dlouhým rodokmenem fialový. Filtr se aktivuje klávesou ‘e’.

Generace – v tomto módu je každý agent očíslován svojí vzdáleností v generacích od náhodně generovaného jedince. Ten má hodnotu 0, jeho potomci 1 atd., barevně začínají agenti na černé barvě a s přibývajícími generacemi se postupně propracovávají barevným spektrem k bílé. Klávesou pro tento filtr je ‘r’.

Atributy – v režimu zobrazení atributů je možno přečíst aktuální hodnotu metabolismu (červeně) a velikosti (zeleně). Barva agenta je určena jako odstín žluté se zabarvením do zelena nebo do oranžova v závislosti na poměru obou atributů. Filtr používá klávesovou zkratku ‘t’.

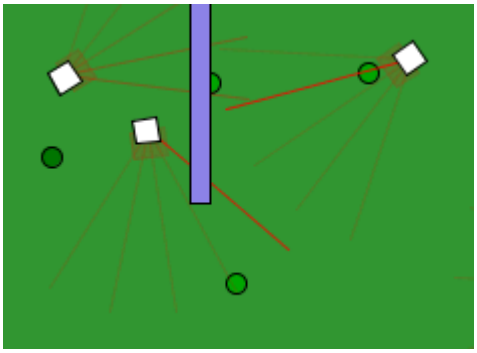


Obr. 4.1: Ukázky filtrů. A) bez filtru, B) Dieta, C) Věk,
D) Generace E) Atributy F) Genom G) Druh H) Energie

Genom – podává informace o přidaném počtu skrytých neuronů (tyrkysová) a synapsí (purpurová) díky čemuž lze sledovat stupeň vývoje topologie. Tato informace je obzvláště důležitá v případě že chceme sledovat, zda dochází k pozastavení vývoje topologie. Klávesa ‘z’.

Druh – tento filtr podává informaci o druhové příslušnosti agenta a to formou čísla druhu, ke kterému agent náleží a barevným odlišením jednotlivých druhý pomocí základních barev, číslo druhu je vyvedeno v barvě příslušného druhu. Dalším zobrazeným údajem je pak práh rozdělení (tmavě šedá), což je globální atribut, na základě něhož jsou agenti rozřazováni do druhů. Aktivace filtru klávesou ‘u’.

Energie – posledním filtrem je zobrazení aktuálního množství energie (žlutá) tj. počet kroků simulace, které agent přežije, pokud nenajde jídlo. Barva agenta je pak poměr energie proti maximální energii, jasně žlutá pro plnou kapacitu a černá pro okamžik před smrtí hladem.



Obr. 4.2: Ukázka zobrazení senzorů, senzory detekující překážku jsou výraznější v závislosti na vzdálenosti překážky

4.4.3 Senzory

Senzory jsou zobrazeny jako průhledná červená linka směřující od středu agenta ve směru vnímání senzoru do jeho vzdálenosti. Základní úroveň průhlednosti $\alpha = 0.2$. Pro přehled o aktivitě senzorů je vhodné, aby senzor dával najevo, že detekuje danou překážku. Proto si každý senzor v průběhu kroku programu pamatuje nejmenší vzdálenost, na kterou detekoval objekt v aréně a při vykreslování je pak úroveň průhlednosti α nastavena v rozmezí $<0.2, 1>$ na základě této nejmenší vzdálenosti. Čím je detekovaný objekt blíž tím je senzor výraznější (Obr. 4.2).

4.4.4 Souhrnné informace

V každé iteraci programu se v levém horním rohu zobrazují následující informace o stavu simulace:

- Počet kroků simulace, které již uběhly od startu
- Čas od startu v hodinách, minutách a vteřinách
- Aktuální počet agentů v aréně pro rychlý přehled o velikosti populace

Kapitola 5

Testování a experimenty

Cílem této kapitoly je popsat výsledky pozorování experimentů, které byly průběžně prováděny při implementaci jednotlivých částí algoritmu popsaného výše.

Každý z následujících experimentů je rozdělen do tří částí. V první části jsou sumarizovány všechny funkce implementované v algoritmu, druhá část se zabývá několika různými variacemi na počáteční podmínky simulace a jako poslední následuje rozbor výsledků experimentu.

Vzhledem tomu že průběh algoritmu je vysoce stochastickým prohledáváním bylo u každé varianty provedeno více pokusů a při vytváření závěru jsou sledovány spíše obecné tendence.

5.1 Hledání jídla

V první fázi byly implementovány zaštiťující třídy řídící běh simulace, aréna je uzavřená ze všech stran ale prozatím nejsou umístěny překážky uvnitř arény. Byly vytvořeny třídy reprezentující jídlo a agenta. Je použit pouze neurokontroler s fixní topologií (plně propojená rekurentní síť). Agenti dokáží jíst jídlo ale ostatní agenty prozatím ne. Senzory detekují jídlo, agenty a zdi. Fyzické atributy nejsou implementovány. Všichni agenti patří k jednomu druhu.

Bez omezení množství jídla

Jídlo je generováno neustále bez ohledu na množství jídla v aréně. V první fázi nejsou agenti schopni jídlo vyhledávat a po určité době začíná být aréna zahlcena jídlem a dochází ke zpomalení programu. V dalších fázích simulace dochází k cyklům populačních explozí v okamžiku, kdy aréna obsahuje tolik jídla, že pro náhodně se pohybující agenti do něj naráží na každém kroku a vytváří množství potomků s minimálním dopadem na vývoj chování. Následuje vymření v okamžiku, kdy populace zkonzumuje všechno jídlo. Vytvořením stropu pro nárůst jídla byl problém z větší části odstraněn.

Fixní pozice jídla

Jídlo není generováno na naprosto náhodných pozicích ale na pevně zadaných místech po aréně. Už po několika málo minutách se ukazuje, z hlediska agentů je nejvýhodnější stát na místě těsně u zdroje a čekat kdy se jídlo objeví. Ačkoli logické vyústění, je tato varianta nevhodná pro další práci.

Rovnoměrné rozdělení jídla

Pokus s generováním jídla sice na náhodných pozicích ale tak aby bylo rozloženo po aréně rovnoměrně. Protože se jídlo generuje v sektoru, kde chybí, nepotřebuje agent přílišnou sofistikovanost a brzy dochází malé populační explozi v daném sektoru. Ve většině případů následuje hromadná sebevražda. Agenti se například neumí vypořádat s detekcí zdi a většina z nich u ní jednoduše zůstane stát a zahyne hlady.

Výsledek: Rozložení jídla náhodnou metodou se ukazuje nejen jako nejjednodušší co do implementace ale současně i jako nejefektivnější v ohledu vývoje agentů. V případě že agent začne být úspěšný v jedné oblasti je tato oblast vyjedena jeho potomstvem a v případě že si s tím agenti neporadí, rychle zahynou a jsou nahrazeni novým pokusem. Nevhodní agenti nejsou tolik protěžováni náhodou v případě, že získají dobré postavení.

5.2 Masožravost a býložravost

Stále je používána plně propojená rekurentní síť. Agenti dokáží nyní konzumovat i ostatní agenty, ale stále nejsou rozdělováni do druhů.

Stejná rychlost konzumace

Ani po několika generacích, kdy už většina populace dokáže bez problémů vyhledávat jídlo, nejeví agenti známky jakékoli snahy o pronásledování jiných agentů za účelem obživy. Lov se jeví jako neefektivní, pravděpodobně protože lovící agent musí svou kořist udržet dostatečně dlouho v hledáčku ústního senzoru, aby ji celou zkonzumoval. Ačkoli převážnou část stravy stále tvoří rostlinná potrava, agenti se přestávají vyhýbat kolizím s ostatními agenty. Začínají vznikat bojůvky, když se dva agenti zaklesnou do sebe předními stranami a vzájemně se pojídat. Oba umírají vysílením, protože suma energie soustavně klesá a agenti se pojídají stejnou rychlostí.

Zvýšená rychlost konzumace agentů

V případě že lovec dokáže svou oběť pozřít rychleji, stává se lov podstatně efektivnějším a z agentů se po nějaké době vyvíjejí všežravci aktivně vyhledávající jak jídlo, tak ostatní agenty. Bojůvky popsané v předchozí variantě se stávají častějším úkazem – nezáleží na rychlosti konzumace, protože energie se přelévá stejnou rychlostí oběma směry. Bojující agenti ale zůstávají statičtí a tvoří snadný cíl. Nezřídka se proto objeví třetí strana, která konflikt rozhodne. Dalším efektem vzniku všežravců je vzájemné požírání rodičů se svým potomstvem. Agent často pozře jídlo, vytvoří potomka před sebou, aby ho okamžitě zkonzumoval a vytvořil dalšího.

Žádná potrava

V tomto pokusu se v aréně neobjevuje žádná potrava, místo toho je zvýšena minimální populace a mírně zvýšena rychlost vytváření nových agentů. Evoluční postup agentů je mnohem rychlejší v důsledku zvýšené populace. Ačkoli agenti nejsou nijak formálně rozděleni, brzy lze sledovat vznik dvou skupin – náhodně generovaní agenti, kteří se jen tak motají arénou, suplují potravu, zatímco vyvinutí agenti projíždějí napříč arénou a cíleně vyhledávají ostatní. Populace vyvinutých agentů zůstává víceméně stabilní – ve větším množství na sebe začínají příliš často narážet a bojovat. To funguje jako regulace jejich množství.

Výsledek: Výsledky zavedení masožravosti jsou uspokojující – agenti se dobře adaptují do role predátorů. Nastaly dokonce situace, kdy vzájemně se pronásledující agenti vytvořili „vláček“ což svědčí o tom, že jsou schopni sledovat pohyblivý cíl. Prozatím nebyla sledována žádná specializace – vzhledem k tomu že všichni agenti patří technicky k jednomu druhu pravděpodobně ani není možná.

5.3 Zavedení atributů

Poslední experiment používající neuronovou síť s fixní topologií. Novým prvkem je zavedení fyzických atributů agentů a náhodná mutace při rozmnožování. Zavedenými atributy je délka a rozpětí senzorů, velikost a metabolismus. Velikost a metabolismus ovlivňují výsledný úbytek energie za krok.

Neomezené hodnoty

Zavedené atributy mají počáteční hodnotu 1, jediným omezením je že nesmějí nabývat nuly záporných hodnot. Ačkoli hodnoty velikosti a metabolismu se vzájemně neovlivňuj (v kódu), vykazují tendence nepřímé závislosti. V průběhu několika simulací se začaly vyskytovat dva druhy extrémů. Zaprvé to byl značný nárůst velikosti v populaci a současně s ním snížení metabolismu – výsledkem byl jedinec s velkou kapacitou energie pohybující se velmi pomalu, jehož energetické nároky byly zhruba stejné jako při původních hodnotách. Vysoká kapacita energie znamenala menší míru rozmnožování a agent uchoval více energie pro vlastní potřebu. Tato varianta se nakonec sama omezila v růstu, protože dostatečně pomalý agent nebyl schopen dojít k potravě včas nehledě na své schopnosti. Zadruhé docházelo k opačnému extrému - velikost se snížila na minimum a metabolismus se v odpovědi na to zvyšoval. Energetická náročnost se stále udržovala na původních hodnotách. Problém nastal v okamžiku, kdy se hodnota velikosti přiblížila nule. Z důvodu velmi malé energetické kapacity se měl každý agent okamžitě po narození dostatek energie pro rozmnožování a agenti se začali množit geometrickou řadou a simulace zkolabovala. Příčinou problému je pravděpodobně chyba dělení při hodnotách blízkých nule.

Omezené hodnoty

Omezením hodnot shora i zdola bylo dosaženo určitého vyvážení atributů. Agenti se dále vyvíjejí jedním z výše popsaných směrů, ale je zabráněno extrémnímu přemnožení nebo zastavení vývoje (když agent našel energii pro přežití, ale nikdy nebyl chopen naplnit celou svou kapacitu a rozmnožit se). Dalším efektem omezení hodnot je omezení tendencí nepřímé závislosti mezi atributy – protože jeden z atributů neroste do nekonečna, není jedinou možností pro přežití snižování druhého atributu.

Senzory

Pokusy s úpravou délky a rozpětí senzorů byly převážně uspokojivé. Tyto atributy jsou implicitně nataveny jako doplňkové (zvyšující se délka snižuje rozpětí) takže technicky se jedná o jeden atribut. Při pozorováních byla sledována velká závislost k druhým dvěma atributům – v případě že v daném běhu dali agenti přednost velikosti, bylo preferováno velké rozpětí senzorů, aby agent zachytil všechny známky potravy (při malé rychlosti si nemohl dovolit nějakou minout). V opačném případě – při vysoké rychlosti – bylo úspěšným rozložení senzorů s malým, téměř nulovým rozpětím. V důsledku toho sice agent minul většinu jídla, ale v případě spatřené potravy mu husté senzory pomohly lépe se zaměřit (při najetí na potravu pod úhlem a při větší rychlosti se stávalo, že potrava byla odstrčena z cesty a pro rychlého agenta ztracena).

Výsledek: Zavedení fyzických atributů dává agentům další rozměr možného vývoje, a při rozumné parametrizaci se vylučuje možnost poškození rovnováhy simulovaného světa. Jedním z dopadů zavedení atributů bylo téměř kompletní vymizení bojůvek, kdy se dva agenti zaklesnou do sebe a za vzájemné konzumace vyčerpají svoje zásoby energie. Důvodem je zrychlení konzumace energie podle poměru velikosti útočníka a oběti – souboje jsou poměrně rychle vyřešeny ve prospěch většího z agentů.

5.4 NEAT

Na počátku tohoto experimentu stála implementace kostry třídy TomNeatController a nahrazení sítě s fixní topologií sítí proměnnou. V průběhu testů pak byly zpřístupňovány jednotlivé funkce této implementace a nastavení pravděpodobností jednotlivých typů mutací.

Bez mutací topologie

Zavedení sítě modelu NEAT [4] s vypnutou funkcí mutací topologie vedlo ve fázi náhodných agentů k téměř nerozeznatelným výsledkům od fixní sítě, v průběhu simulace ale bylo zřejmé, že nepřítomnost skrytých neuronů je pro kontroler nevýhodou.

Bloating

V případě zapnutých možností strukturálních mutací je výsledný efekt značný. Jako optimální se ukázala neurální síť s cca 7 skrytými neurony, což koresponduje se zkušenostmi s použitím fixní sítě – tam bylo experimentálně dosaženo nejlepších výsledků u plně propojené rekurentní sítě s 10 neurony. Po dosažení tohoto množství neuronů dochází k výraznému snižování úmrtnosti a zvyšování efektivity vyhledávání. Protože evoluce probíhá stále dál, dochází v průběhu dalšího běhu simulace nevyhnutelně k přidávání dalších neuronů do topologie sítě. Tyto změny už výrazně nevylepšují výkon agentů v aréně, ale neuronová síť se tak stává odolnější vůči mutacím a snižuje se pravděpodobnost degenerace potomstva. Tento jev se nazývá bloating.

Výsledek: Výsledky tohoto pokusu nejsou na první pohled příliš zřejmé. Proměnná síť dosahuje v zásadě stejných výsledků jako síť fixní. Přesto lze tuto variantu považovat za úspěšnější, protože na rozdíl od sítě s fixní topologií dosahuje těchto výsledů sama bez nutnosti vnějšího hledání ideálního počtu neuronů.

5.5 Rozdělení do druhů

Kritickou částí implementace se stalo rozdělování do druhů – konečně se dostáváme k testování interakcí agentů založených na vzájemné podobnosti. Pro testování těchto vztahů bylo použito rozdělení do druhů podle vzájemné podobnosti neuronových sítí, tak jak je popsáno v návrhu modelu rtNEAT [7] a v této práci v kapitole 2.4.2. Senzory robotů byly upraveny tak aby poskytovaly oddělené informace pro agenty stejného a pro ostatní agenty.

Pevný práh

Při použití pevné hodnoty prahu nedocházelo v první fázi k žádnému rozdělení do druhů – rozdíly ve váhách náhodně generovaných jedinců nebyly dostatečné, aby došlo k odlišení. V pozdějších fázích pak naopak vznikalo velké množství druhů po přidání některých strukturálních mutací.

Pohyblivý práh

Druhou variantou bylo použití rozdělení do druhů na základě pohyblivého prahu podle modelu rtNEATU. V případě že druhý bylo méně, než určený počet byl práh snižován a naopak. Problémem se stávaly situace, kdy nebylo možno nastavit práh tak aby počet druhů přesně odpovídal požadovanému číslu – hodnota prahu oscilovala kolem určité hodnoty a průběh rozdělování nemohl být ukončen. Řešením problému se ukázalo rozsahu, do kterého se měl počet druhů vejít na místo jedné hodnoty.

Neurčený vztah ke svému druhu

Agenti nemají explicitně zakázáno konzumovat příslušníky vlastního druhu. V průběhu mnohanásobně opakovaných experimentů se ukazuje, že agenti nejeví sebemenší sklony k odlišnému chování vůči vlastnímu druhu. Z podstaty simulovaného světa jde pravděpodobně o další ze skrytých nástrojů evoluce – pokud agent například narazil na shluk potravy (a byl dostatečně sofistikovaný, aby ho celý zkonzumoval) vytvořil ve svém blízkém okolí několik svých potomků. V některých případech se agenti nespatří a vyrazí dál vlastními cestami. V případě opačném se rozpoutá potyčka vzájemného požírání a tvorby dalších potomků, ve které průběžně vyhrávají silnější. Potyčka je skončena když někteří účastníci vyrazí pryč, nebo v dané oblasti už není dostatek energie pro další rozmnožování a zůstává jeden vítěz, který pokračuje ve své cestě.

Protekce vlastních

V případě že svět striktně zakazuje kanibalismus a konzumace příslušníků není možná, zůstává situace překvapivě podobná předchozímu příkladu. Na vině je pravděpodobně systém rozřazování do druhů – protože v úvodních fázích jsou si jednotliví agenti vzájemně dosti podobní a v průběhu několika rozřazování v řadě se často stane, že agenti jednou přiřazení do stejného druhu jsou o chvilku později rozděleni do druhů odlišných. Často proto lze pozorovat agenta zavěšeného v závěsu za některým svým soukmenovcem čekajícího, až už tento jeho soukmenovcem nebude.

Specializace diety

Dalším předpokladem založeným na rozřazování do druhů byl vývoj specializovaných druhů. Pokud je ale agentům dána volnost v konzumaci čehokoliv konzumovatelného žádná specializace se nekoná. Z hlediska agenta by bylo kontraproduktivní odmítat jakýkoli zdroj potravy byť by to byli vlastní potomci.

Předem určená dieta

V tomto pokusu dostal každý agent na začátku náhodně zvolenou kombinaci logických hodnot *carnivorous* a *herbivorous* na jejichž základě byl schopen konzumovat jiné agenty respektive rostliny. Někteří agenti nedostali ani jednu ze schopností a byli odsouzeni k rychlé smrti. V případě ostatních tří variant vždy potomek podědil tyto schopnosti po svém primárním rodiči. V době kdy se populace naučila obstojně prohledávat arénu a nacházet potravu sestávala celá populace z všežravců – až takovou výhodu dávalo agentům vlastnictví obou schopností.

Předem určená dieta, bez všežravců

Variace na předchozí experiment, na místo náhodné kombinace obou schopností získává každý agent právě jednu z nich náhodně zvolenou. Zatímco býložraví agenti začali po několika minutách tápání vyhledávat potravu po aréně, v případě masožravých se nekonal žádný vývoj kupředu.

Výsledek: Rozdělení do druhů mělo očekávaný průběh – v z počátku je populace náhodná a proto je i přiřazování převážně chaotické a příslušnost ke druhu neměla dlouhodobé trvání. V pozdějších fázích se pak jev částečně stabilizuje a přesto že jednotlivý agent je v průběhu času přiřazován do různých druhů (rozdělování probíhá podle pozice v seznamu agentů a tato se v případě každého jedince často mění), stává se pravidlem spíše příslušnost příbuzných jedinců ke stejnému druhu, než naopak.

Z pohledu chování nemělo nesupervizované rozdělení do druhů očekávaný efekt na populaci. Výsledné chování bylo zhruba totožné u všech jedinců napříč populací, navíc chování nebylo, ve většině případů, ovlivněno vzájemnou podobností. Druhová solidarita je neexistující pojem a agenti se řídí pravidlem „Každý sám za sebe“. V případě nastavených možností chování se agenti bez problémů přizpůsobují stávajícím pravidlům, sami od sebe podobná omezení netvoří.

5.6 Další experimenty

V poslední fázi bylo provedeno několik dodatečných experimentů s přidáním překážek do prostoru arény a s vyzkoušením několika variant výběru sekundárního rodiče při křížení.

Varianty křížení

Základní variantou bylo výběr nejdéle žijícího příslušníka stejného druhu za sekundárního rodiče při křížení. Pro další testování bylo vyzkoušeno náhodný výběr partnera (v rámci populace) a výběr založený na souhrnu fyzického věku a věku genomu.

Výsledkem náhodného výběru bylo očekávané zhoršení schopnosti agentů učit se v průběhu času. Zajímavým faktorem je, že schopnost učit se byla pouze zhoršena, ale k učení stále dochází. V druhém případě došlo k mírnému zlepšení díky tomu, že agent například v desáté generaci je vyhodnocen jako kvalitnější než agent generace páté přesto že je fyzicky mladší díky přičtenému věku genomu.

Zdi a překážky

Vztah agentů ke zdem arény se v průběhu všech experimentů jeví jako značně problematický. Téměř pravidlem se stává, že agent několikáté generace, který již prokázal schopnost sběru potravy, ale jehož předci se nikdy nesetkali se zdí, přijede při prvním kontaktu ke stěně a jeho život zde po čase skončí. V experimentu bylo přidáno několik fixních překážek do prostoru arény.

V průběhu simulace se často stává, že mladší agenti umírají k nově přidaným překážkám. Výhodou však je, že při častějším kontaktu s překážkou, kterou je třeba objet, předávají tuto znalost svým potomkům, v případě přežití, a v pokročilejších stádiích simulace je schopnost vyhýbání na výrazně vyšší úrovni.

Kapitola 6

Závěr

V průběhu psaní této práce se podařilo dosáhnout cílů stanovených v zadání práce. Podařilo se implementovat algoritmus otevřené evoluce a navrhnout taková pravidla simulovaného světa, která umožňují evoluci náhodně generovaných agentů řízených neuronovými sítěmi a současně vytvářejí ztížení podmínek pro pokročilejší agenty ve formě konkurenčního boje, což podporuje průběžný vývoj. Díky navrženým senzorům získávají agenti veškeré potřebné informace ze svého okolí, na základě nichž se rozhodují o dalším postupu.

Hlavním cílem bylo demonstrovat schopnost samostatného učení neuronových sítí, na základě vnitřní adaptace. Tento cíl byl beze zbytku splněn, protože evoluce probíhala i bez jednoznačné fitness funkce a dokonce i v případě, že partner pro křížení byl vybírán čistě náhodně, docházelo k vývoji byť velmi pomalému. Neuronové sítě se ukazují dostatečně flexibilní a díky tomu agenti pružně reagují na spuštění simulace s odlišnými podmínkami změnou chování.

Při testování bylo porovnáváno chování agentů s předpokládanými výsledky stanovenými v rámci určení cílů práce. Některé z předpokladů se naplnili, jako vyhledávání potravy, vyhýbání se překážkám a souběžná evoluce fyzické schránky, jiné se ukázaly jako mylné – mezi ně patří předpoklad specializace agentů na jeden typ potravy a odlišné chování k na základě vzájemné podobnosti. Ukázalo se, že reprezentace světa není dostatečně přesná, aby se tyto znaky (obecně se vyskytující při evoluci ve skutečném světě) představovaly vývojovou výhodu a proto se při experimentech nepodařilo je reprodukovat.

V budoucnosti této aplikace bych se zaměřil zejména na zpřesnění modelu světa pro dosažení věrohodnějších výsledků v porovnání s reálným světem, a upravení simulace pro práci s podstatně rozměrnější arénou s možností interakcí většího počtu objektů, protože nízká velikost populace je jedním z faktorů zkreslujících výsledek.

Literatura

[1] Jan Drchal. Evolution of Recurrent Neural Networks. Diplomová práce, ČVUT, 2006.

[2] Petr Smejkal. Výpočetní geometrie pro simulaci koevoluce. Bakalářská práce, ČVUT, 2008.

[3] Ondrej Kapraľ. Evoluce neuronových sítí pro řízení mobilních agentů. Diplomová práce, ČVUT, 2009.

[4] Kenneth O. Stanley, Risto Miikkulainen. Evolving Neural Networks through Augmenting Topologies. Department of Computer Sciences, The University of Texas at Austin, 2002.

[5] Kenneth O. Stanley, Risto Miikkulainen. Effcient Reinforcement Learning through Evolving Neural Network Topologies. Department of Computer Sciences, The University of Texas at Austin, 2002.

[6] Kenneth O. Stanley, Risto Miikkulainen. Competitive Coevolution through Evolutionary Complexification. Department of Computer Sciences, The University of Texas at Austin, 2004.

[7] Kenneth O. Stanley, Bobby D. Bryant, Risto Miikkulainen. Evolving Neural Network Agents in the NERO Video Game. Department of Computer Sciences, The University of Texas at Austin, 2005.

[8] Norman H. Packard. Intrinsic Adaptation in a Simple Model for Evolution. Center for Complex Systems Research and the Physics Department, The University of Illinois, 1988.

Dodatek A

Obsah přiloženého CD

Kořenový adresář CD obsahuje následující podadresáře a soubory

ViVae/	Obsahuje data celého projektu.
doc/manual/	manuál prostředí ViVAE ve formátu TEX
lib/	knihovny používané projektem
nbprojekt/	soubory týkající se Netbeans projektu
src/	zdrojové kódy projektu
build.xml	soubor nástroje ant pro kompilaci dat
info.txt	soubor obsahující postup spuštění programu
Text/	Obsahuje vlastní text bakalářské práce.
Blovstom_2011bach.pdf	soubor bakalářské práce