Masarykova univerzita
Fakulta informatiky

Srovnání učících algoritmů pro hledání dobré strategie v multiagentním prostředí

Diplomová práce

Jiří Ohnheiser

Brno, jaro 2013

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: doc. RNDr. Aleš Horák, Ph.D.

Poděkování

Za podporu bych rád poděkoval mé rodině, přítelkyni a všem přátelům. Také bych rád velmi poděkoval mému vedoucímu za poskytnuté cenné rady.

Shrnutí

Tato diplomová práce se zabývá programováním univerzálního systému simulujícího virtuální svět pro učení agentů řízených umělou inteligencí. Tento virtuální svět má dva rozměry a respektuje základní fyzikální zákony.

Učící proces agentů je možné pozorovat jak přímo, tak prostřednictvím tabulek a grafů. Celý systém je modulární a je rozdělen na server a klienta. Umělá inteligence se rozhoduje pomocí neuronových sítí a evolučních algoritmů.

Klíčová slova

Umělá inteligence, Agentní systémy, Genetické algoritmy, Neuronové sítě, Virtuální svět.

Obsah

1 **Předmluva** 1
2 **Související práce** 3
  2.1 *Polyworld* 3
  2.2 *Evolved Virtual Creatures* 5
  2.3 *Rozdíly mezi projekty* 6
3 **Analýza** 7
  3.1 *Neuronové sítě* 7
    3.1.1 Aktivační funkce 9
    3.1.2 Předzpracování vstupu 10
    3.1.3 Topologie 10
    3.1.4 Učení 11
  3.2 *Genetické algoritmy* 12
    3.2.1 Křížení a mutace 14
    3.2.2 Funkce zdatnosti 15
    3.2.3 Generace 15
  3.3 *Stanovení cílů* 16
4 **Návrh** 18
  4.1 *Architektura* 18
    4.1.1 Registr 19
    4.1.2 Server 19
    4.1.3 Klient 20
  4.2 *Architektura virtuálního světa* 21
    4.2.1 Plemena 23
    4.2.2 Objekt 25
    4.2.3 Agent 27
    4.2.4 Návrh umělé inteligence 29
    4.2.5 Fyzika 31
5 **Implementace** 33
  5.1 *Umělá inteligence* 33
    5.1.1 Náhodná 33

5.1.2 Vícevrstvá neuronová síť 34
5.2 *Plemena a jejich komponenty* 36
5.2.1 Generace 36
5.2.2 Výběr rodičů 37
5.2.3 Logování 38
5.2.4 Výpočet zdatnosti 39
5.2.5 Modifikace agentů 39
5.2.6 Různé 40
5.3 *Fyzika* 41
5.3.1 Akcelerace detekce kolizí 41
5.3.2 Senzory umělé inteligence 42
5.4 *HTML statistiky* 43
5.5 *Konfigurační soubory* 45
5.6 *Použité technologie* 46
5.6.1 Programovací jazyk C++ 46
5.6.2 SDL 47
5.6.3 ENet 47
5.6.4 CEGUI 48
5.6.5 RapidXml 48
5.6.6 jqPlot 48
5.6.7 Ostatní 49
6 **Testování** 50
6.1 *Jednoduché experimenty* 51
6.1.1 Test zastavení 51
6.1.2 Test přežití 51
6.2 *Netriviální experimenty* 53
6.2.1 Test vzdálenosti 53
6.2.2 Test komplexní vzdálenosti 54
6.3 *Experiment se zadaným cílem* 56
7 **Závěr** 58
7.1 *Možnost rozšíření* 59
A **Popis konfiguračních souborů** 63
A.1 *Konfigurace světa* 63
A.2 *Konfigurace plemen* 65
A.2.1 Komplexní funkce zdatnosti 66
A.3 *Konfigurace modelů* 66
A.4 *Konfigurace agentů* 67
A.5 *Konfigurace fyzické schránky agentů* 69

A.6 *Konfigurace komponent agentů* 69
B **Elektronické přílohy** 72

Kapitola 1

Předmluva

Žijeme na exponenciále – každým rokem se zrychluje technologický pokrok a už mnoho let není v silách jediného člověka mít alespoň letmý přehled o tom, co se událo v posledním roce. Rok co rok jsme svědky tak nepředstavitelného pokroku, že si jej mnohdy ani neuvědomujeme.

Na počátku tohoto století trvalo přečíst lidský genom roky, v současné době jsme schopni přečíst celou lidskou DNA během jednoho dne a pokrok se zdaleka nezastavil, již brzy bude pravděpodobně možné číst genom v řádu minut. Je jen otázkou času, než se analýza DNA stane běžnou součástí vstupní prohlídky u lékaře.

Rozvoj v oblasti informačních technologií je ještě znatelnější. Dnes běžně nosíme po kapsách mobilní telefony s výkonem superpočítačů předminulého desetiletí. Téměř každý obor lidské činnosti dnes využívá informačních technologií: archeologií počínaje, zoologií konče. Už si ani nedokážeme představit život bez počítačů a internetu, na který jsme připojeni dvacet čtyři hodin sedm dní v týdnu.

Jedním ze zajímavých oborů informačních technologií je umělá inteligence. Díky strmě rostoucímu výkonu počítačů tento obor nabývá na významu.

Když roku 1997 porazil Deep Blue [1] Garri Kasparova[2] výsledkem dvě výhry a tři remízy, byla to světová senzace. Dnes umělá inteligence zasahuje do mnoha oblastí lidské činnosti. Používáme ji při focení (detekce tváří, úsměvu), pokaždé, když zadáváme dotaz do internetového vyhledávače, pomáhá udržovat auta v jízdních pruzích,

1. Superpočítač vyvinutý společností IBM speciálně pro šachy.
2. Gari Kasparov byl tehdejší světový šampión v šachu.

brzdí je za nás, když se nedíváme, a nezapomeňme na to, že se používá při navrhování mikročipů.

Není pochybností, že tento obor bude naše životy ovlivňovat stále více, proto je důležité mu věnovat pozornost.

Cílem této diplomové práce je naprogramovat univerzální systém simulující virtuální svět pro agenty řízené umělou inteligencí. Tento virtuální svět bude mít pro jednoduchost dva rozměry a bude respektovat základní fyzikální zákony. Učící proces agentů v něm bude možné pozorovat jak přímo, tak prostřednictvím tabulek a grafů. Celý systém bude velice modulární a jeho základní komponenty budou klient a server. Umělá inteligence bude rozhodovat pomocí neuronových sítí a evolučních algoritmů.

Součástí práce jsou také příklady různých strategií a informace o jejich průběhu v podobě grafů.

Kapitola 2

Související práce

V této kapitole se budeme zabývat podobnými projekty. Jedná se o Polyworld a Evolved Virtual Creatures.

2.1 Polyworld

Polyworld je program tvořící virtuální prostředí pro evoluci umělé inteligence za pomocí evolučních algoritmů[Yaeger(1994)]. Je to nástroj pro zkoumání jevů důležitých v evoluční biologii, behaviorální ekologii, etologii a informatice.

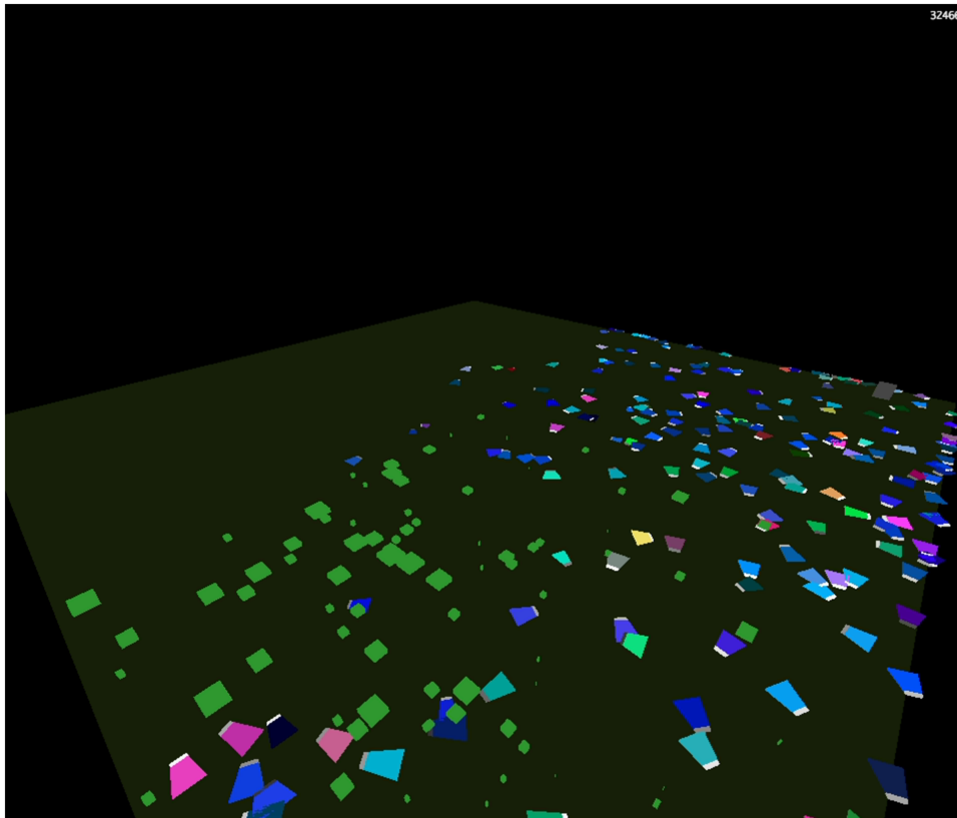
Polyworld se snaží vytvořit virtuální prostředí fungující na podobných principech jako náš vlastní svět. Virtuální svět má tvar čtverce a obsahuje agenty, překážky a potraviny. Tito agenti mají simulovaný jednoduchý metabolismus a k rozhodování používají neuronové sítě.

Tyto neuronové sítě jsou při „zrození“ nového agenta vytvořeny na základě jeho DNA, v průběhu života se pak používá Hebbovo učení[Kempter et al.(1999)Kempter, Gerstner, and Van Hemmen].
Ke svému přežití potřebují energii, kterou mohou načerpat z jídla.

Jakákoliv činnost agenta pak čerpá jeho energii: pohyb, množení, útok, přemýšlení, dokonce i nečinnost spotřebovává malé množství energie. Život agenta je omezen tvrdým limitem podobně, jako v reálném světě (např. zkracování telomer).

Velikost mozku (neuronové sítě kontrolující agenta) je omezena vyšší energetickou náročností. Při ranných pokusech bez tohoto omezení měli výslední agenti obrovské neuronové sítě, které z velké části neprováděly žádnou činnost.

Naopak po zavedení tohoto omezení dochází k zajímavému jevu, kdy komplexita neuronových sítí agentů roste postupem času s tím,



Obrázek 2.1: Grafické zobrazení virtuálního světa Polyworld[Yaeger(2010)]

jak se učí přežívat v prostředí do doby, než najdou ideální strategii (dostatečně stabilní lokální minimum). Poté komplexita v dalších generacích mírně poklesne.

Autoři si tento závěrečný pokles vysvětlují tím, že po „vyřešení problému virtuálního světa“ se evoluce soustředí na zvýšení efektivity tohoto řešení.

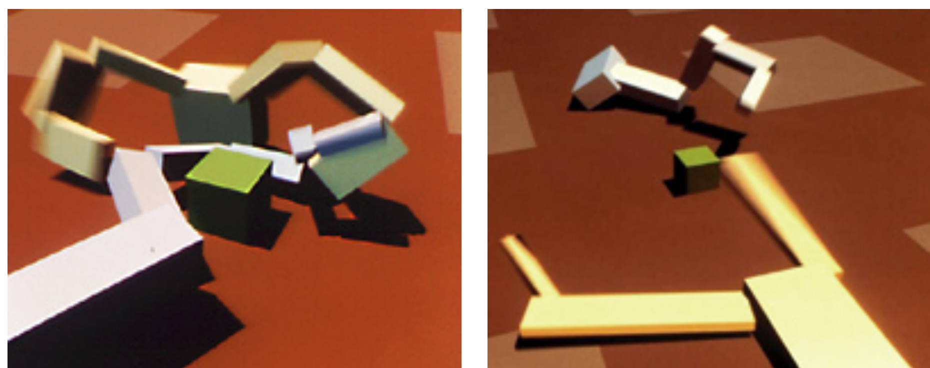
Zajímavé je, že v Polyworldu není přítomna žádná funkce zdatnosti. Noví agenti se tvoří při setkání dvou agentů, kteří jsou ochotni se rozmnožit (ochota se množit je jeden z výstupů neuronové sítě agentů). Pokud ano, tak ztratí část své energie, a na základě jejich DNA se vytvoří nový agent.

Důležitým parametrem zde je to, že agenti musí obětovat část

vlastní energie. V pokusech, kde tato vlastnost nebyla implementována, evoluce našla snadnou cestu k úspěchu – rozmnožit se, zabít potomka, sníst mršinu. Ačkoliv je evoluce slepým hodinářem[Dawkins(1986)], dokáže velice dobře najít úspěšnou strategii a neodpustí virtuálnímu světu jedinou mezeru v zákonitostech světa.

Pro zobrazení virtuálního prostředí Polyworld využívá knihoven Qt a OpenGL.

2.2 Evolved Virtual Creatures



Obrázek 2.2: Ukázka jedinců soupeřících o zelenou kostku[Sims(1994a)]

Evolved Virtual Creatures (EVC)[Sims(1994b)] je prostředí pro simulovanou evoluci jedinců složených z virtuálních bloků. EVC se soustředí hodně na vývoj tělesné schránky jedinců, během evoluce tedy vznikají jedinci velice různorodých tvarů. Jedinci se pohybují ve virtuálním 3D prostředí a sami jsou složeni z 3D bloků. Tyto bloky jsou spolu spojeny klouby.

Prostředí simuluje kolize, gravitaci a tření. Umožňuje i jednoduchou simulaci plavání jedinců ve vodě nebo létání ve vzduchu. Fyzická schránka jedince i jeho mozek (neuronová síť) je určena geneticky.

Mezi úkoly, zadávané virtuálním jedincům, patří například

schopnost dobře plavat, chodit, vysoko skákat, sledovat bod v prostoru nebo soupeřit s jiným jedincem o virtuální zelenou kostku.

Na začátku každého experimentu se vytvoří populace několika stovek jedinců. Ti jsou poté podrobeni testu schopnosti plnit určené úkoly. Z nich se vyberou ti, kteří splňují nejlépe požadavky, a na základě jejich genů se vytvoří nová generace. Tato nová generace opět otestuje své schopnosti při plnění daného úkolu a z nejlepších jedinců se vytvoří další generace. Tímto způsobem pak s postupujícím časem vytváříme agenty, kteří už úspěšně zvládají plnit své úkoly.

V průběhu simulací se vynořují zajímavé techniky řešící daný problém. Například při souboji o zelenou kostku se pokoušejí někteří agenti odtlačit soupeře pryč, jiní zase zkouší přitáhnout kostku k sobě. V průběhu tohoto experimentu se v některých případech vyvinou i jedinci, používající hokejkám podobné končetiny pro přisunutí kostky do své blízkosti.

2.3 Rozdíly mezi projekty

V této kapitole si popíšeme odlišnosti této diplomové práce od zmíněných projektů. Ačkoliv jsou si zaměřením podobné, každá má jiný cíl.

Polyworld neobsahuje žádnou funkci zdatnosti, agenti se množí na základě dostupné energie a vlastní volby. Agenti v Polyworldu také disponují schopností učit se v průběhu svého života. Tato funkcionalita není v této diplomové práci implementována, ale díky modulárnímu designu ji lze také implementovat.

Evolved Virtual Creatures se zabývá více simulací fyzického těla agentů. Naopak ale obsahuje funkci zdatnosti, která řídí evoluci.

Ani jeden z těchto projektů nevyužívá klient-server přístup takovým způsobem, jak je popsáno v této diplomové práci. Nebyly tedy vytvářeny s výhledem na možnost přesunout výpočty umělé inteligence na více počítačů a tím umožnit simulaci opravdu velkých virtuálních světů s velkými populacemi agentů.

Kapitola 3

Analýza

Tato kapitola se věnuje analýze důležitých teoretických oblastí, týkajících se diplomové práce. Je třeba uvést varování, že informace zde uváděné nemusí platit pro všechny existující implementace. Kapitola je zaměřena jen na ty nejrozšířenější implementace.

V podkapitole, zabývající se neuronovými sítěmi, lze najít popis základních principů, na kterých neuronové sítě pracují, popisy způsobů učení neuronových sítí a jejich různé topologie.

Další kapitola se věnuje genetickým algoritmům. Ty se snaží aplikovat evoluční principy na řešení problémů v informatice. Popsány jsou důležité principy genetických algoritmů: dědičnost, křížení, mutace a funkce zdatnosti.

V podkapitole stanovení cílů si pak podrobněji popíšeme cíle celého projektu.

3.1 Neuronové sítě

Neuronové sítě jsou jedním z přístupů používaných v umělé inteligenci. Vycházejí z pozorování funkce biologického mozku a snaží se jej napodobit. Stejně jako biologický mozek se neuronová síť skládá z neuronů, které jsou mezi sebou propojeny synapsemi.

Každý neuron může mít neomezeně vstupů, ale má jen jeden výstup. Každému vstupu je přiřazena takzvaná váha. Podle hodnot na vstupech, jejich vah a na základě aktivační funkce (viz strana 9) nastaví neuron odpovídající hodnotu na svém výstupu. Ten může být připojen na vstupy dalších neuronů nebo může jít o výstup neuronové sítě (výstupní neuron).

Mezi výhody neuronových sítí patří velmi dobrá paralelizace[1], neurony je možné vyhodnocovat současně, což v době OpenCL, CUDA a více-jádrových procesorů přináší své výhody. V dnešní době pro výkonné a velké neuronové sítě není třeba drahý speciální hardware, vývojář si může vystačit s grafickou kartou za řádově menší cenu.

U mnoho problémů je mnohem snadnější naučit je řešit neuronové sítě, než pracovat na složitých algoritmech, které by nakonec mohly mít horší schopnosti, než dobře naučená neuronová síť. Neuronové sítě je vhodné zkusit v případech, kdy máme k dispozici soubor dat, na kterých je možné neuronovou sít naučit, jaké výsledky po ní budeme požadovat.

Neuronové sítě mají velmi dobrou schopnost generalizace (zobecňování), jsou schopny v průběhu učení do sebe vstřebat mnoho závislostí, mezi nimi i ty, které nejsou na první pohled zřejmé. Díky tomu, že jsou informace v neuronové síti uloženy pouze přibližně, jsou také velmi odolné vůči nepřesnosti vstupů.

Kromě toho jsou také neuronové sítě odolné proti poškození, to se projevuje ztrátou přesnosti výsledku.

Mezi nevýhody neuronových sítí patří jistá nepředvídatelnost. Pokud totiž máme dostatečně složitou neuronovou síť s tisíci neurony a desítkami tisíc synapsí, nemůžeme dostatečně dobře říci, jak síť vlastně uvnitř funguje. Není zaručená garance, že při určité nešťastné kombinace hodnot na vstupech nám neuronová sít dá opačný výsledek, než by měla.

V mnoha oborech toto nemusí být problém, ale například v lékařství nebo v leteckém průmyslu, by to být problém mohl, nemluvě o armádním využití. Na obranu neuronových sítí je třeba říci, že pravděpodobnost kritické chyby může být mnohem menší u neuronové sítě než u člověkem vyvinutého algoritmu, případně než u živého člověka konajícího rozhodnutí místo neuronové sítě.

Problém je tak spíše zajímavý z morálního hlediska (kdo by byl zodpovědný za takovou chybu? Máme vůbec právo na to svěřovat svůj/cizí život umělé inteligenci?), než z hlediska matematického, například při využití umělé inteligence při řízení automobilů bychom mohli říci, že nižší počet lidských obětí na životech

1. Neplatí obecně pro všechny typy a jejich implementace.

vyváží tuto obavu. Ale minimálně zajímavá otázka to je.

3.1.1 Aktivační funkce

Existuje celá řada různých aktivačních funkcí, od těch nejjednodušších lineárních až po složité funkce, popisující detailně chování neuronů u živých organismů.

V této diplomové práci je použita jedna z jednodušších variant neuronu. Použitý neuron má *n+1* vstupů (vstupy budou mít indexy 0 - *n*), vstup s indexem 0 je využit pro práh (prahový vstup bude vždy mít hodnotu 1).

Výpočet výstupní hodnoty neuronu probíhá tak, že si nejdříve vypočteme takzvaný vnitřní potenciál ξ. Ten se vypočítá podle následujícího vzorce (w_i je váha na vstupu i a $input_i$ označuje hodnotu na vstupu i):

$$\xi = \sum_{i=0}^{n} (w_i * input_i)$$

Poté na tento vnitřní potenciál aplikujeme aktivační funkci σ a získáme výstupní hodnotu neuronu y.

$$y = \sigma(\xi)$$

Aktivační funkcí mohou být různé funkce, mezi nejznámější patří:

- Skoková funkce $f(x) = \begin{cases} 1, & \text{pokud } x \geq 1 \\ 0, & \text{pokud } x < 0 \end{cases}$
- Lineární funkce $f(x) = x$
- Omezená lineární $f(x) = \begin{cases} 1, & \text{pokud } x \geq 1 \\ x, & \text{pokud } -1 < x < 1 \\ -1, & \text{pokud } x \leq -1 \end{cases}$
- Sigmoida $f(x) = \dfrac{1}{1 + e^{-x}}$
- Hyperbolický tangens $f(x) = \dfrac{1 - e^{-2x}}{1 + e^{-2x}}$

3.1.2 Předzpracování vstupu

U neuronových sítí je také důležité dobře nastavit rozsah vstupních a výstupních hodnot podle typu řešené úlohy. Ačkoliv si neuronové sítě v některých případech mohou poradit i se špatnými rozsahy hodnot, pokud jim to ulehčíme, tak nám nabídnou přinejmenším rychlejší proces učení.

Rozsah hodnot by se u vstupů neměl příliš lišit a měl by korespondovat s aktivační funkcí (tím je myšleno: Pokud aktivační funkce pracuje od 0 do 1, vstupní hodnoty by se měly pohybovat také v tomto rozsahu).

V některých případech je vhodné použít pro předzpracování vstupů logaritmickou funkci. Pokud bychom například chtěli pomocí neuronové sítě klasifikovat velikost pozemku do čtyř kategorií, náš vstup – velikost pozemku – se bude pohybovat od velice malých pozemků (zahrady) až po ty velmi velké (lesy). Velikost těchto pozemků (obsah plochy) se tak liší řádově.

Zde je vhodné velikost pozemku nejdříve zpracovat logaritmickou funkcí a upravit rozsah tak, abychom po zpracování získali číslo, které bude vyhovovat aktivační funkci. Například pro omezenou lineární funkci (viz příklady aktivačních funkcí výše) by to byl rozsah $(-1, +1)$.

3.1.3 Topologie

Neuronové sítě mohou mít různé topologie. Na počátcích jejich využití se používali prakticky pouze jednovrstvé sítě - to znamená, že každý neuron může být napojený na všechny vstupy a každý neuron je zároveň výstupním neuronem.

Toto uspořádání bylo jediné použitelné, protože v té době ještě nebyl známý žádný učící algoritmus pro vícevrstvé neuronové sítě. Nevýhodou jednovrstvé neuronové sítě je, že dokáže pouze rozdělit prostor vstupů na dvě části, nedokáže v něm identifikovat složitější problémy. Jednovrstvá síť je tedy velice omezená co se výpočetních schopností týče (například nedokáže provádět XOR [2]).

Po vyčerpání možností jednovrstvých sítí nebyly dále vyvíjeny další významné aktivity a jejich rozvoj. Jedním z důvodů byla

2. Exkluzivní disjunkce.

nízká možnost využití z důvodu neexistence algoritmu pro učení vícevrstvých sítí. Až když byl publikován materiál, který popisoval algoritmus pro učení vícevrstvých sítí – zpětnou propagaci, začaly zažívat neuronové sítě opět rozkvět.

Vícevrstvé sítě už mají mnohem větší vyjadřovací schopnosti než jednovrstvé neuronové sítě. Tyto neuronové sítě již zvládají všechny logické operace. Také zvládají aproximovat libovolné spojité funkce.

Dalším stupněm neuronových sítí jsou cyklické neuronové sítě. To jsou takové sítě, které obsahují zpětnou vazbu - nejsou tedy pouze dopředné. Jejich výhodou je ještě větší vyjadřovací schopnost, než v případě dopředných vícevrstvých sítí. Nevýhodou je, že výpočet se nemusí nikdy zastavit. Díky zpětné vazbě je totiž výsledek výpočtu neuronové sítě závislý nejen na vstupech, ale také na jejím vnitřním stavu. Lze říci, že tyto neuronové sítě se zpětnou vazbou si dokáží „pamatovat" informace. Cyklické neuronové sítě mají také alespoň tak silnou vyjadřovací schopnost jako Turingovy stroje.

3.1.4 Učení

Učení je postupný proces změn neuronové sítě směrem k lepším výsledkům. Konečným výsledkem učení by měla být neuronová síť, dávající dostatečně přesné výsledky. Učení neuronových sítí se dá rozdělit na dva hlavní přístupy – učení s učitelem a učení bez učitele. Každý přístup má své použití a své výhody a nevýhody.

Při učení s učitelem máme k dispozici množinu tréninkových vzorů, každý z nich se skládá z množiny vstupů a množiny požadovaných výstupy. Pokud tedy například chceme neuronovou síť naučit rozpoznávat, jestli pacient trpí určitou chorobou, tak se jeden tréninkový vzor bude skládat z množiny vstupů (zde mohou být příznaky, stáří pacienta, pohlaví, rizikové zaměstnání, atd.) a množiny výstupů (v tomto případě informací, jestli pacient trpěl danou nemocí).

Velmi laicky lze postup popsat takto: Neuronové síti jsou předložena vstupní data za vzoru. Na základě odchylky od požadovaného výsledku je spočítána chyba. Na základě této chyby spočítáme potřebnou korekci, podle které poté upravíme váhy v neuronové síti. Postup opakujeme pro další vzory.

Celé opakujeme až do dosažení námi stanovené minimální

chyby, kdy prohlásíme neuronovou síť za naučenou. Stanovení minimální chyby také není zcela triviální záležitostí. Její pomocí chceme zabránit takzvanému přeučení neuronové sítě. To je stav, kdy je neuronová síť příliš přizpůsobena tréninkovým datům a ztrácí schopnost generalizace. Příliš dlouhé učení je ale jen jedna z příčin tohoto stavu.

Metod, jak zabránit přeučení, je mnoho – od změny parametrů neuronové sítě, přes zavedení náhodného šumu do trénovacích dat až po stanovení vhodné minimální chyby. Mnohdy je zapotřebí kombinace různých metod.

Těchto tréninkových vzorů je potřeba mít rozumné množství, aby se na jejich základě dokázala neuronová síť naučit generalizovat daný problém. Je také vhodné část dat vyčlenit jako validační množinu, na které můžeme následně testovat schopnost generalizace a případné přeučení neuronové sítě. Ovšem existují i postupy, jak pracovat i s malou množinou tréninkových dat.

Při učení bez učitele nemáme k dispozici požadovaný výstup (učitele). Síť při tomto učení může například třídit množiny hodnot vstupů do skupin. To se používá například při pokusu najít skrytou strukturu v datech. Toto učení se také úspěšně používá v systémech dobývání dat, kde pomáhá odlišit validní data od nevalidních[Pedregosa et al.(2011)].

3.2 Genetické algoritmy

Zvláštní skupinou učících algoritmů jsou genetické algoritmy, ty jsou inspirované biologickou genetikou a Darwinovou teorií evoluce. Evoluční algoritmy používají tyto hlavní principy:

- Dědičnost – Nový jedinec (v normálním případě kromě počáteční generace) vzniká s použitím DNA jedinců z předchozích generací.

- Mutace – Stejně jako v reálném světě, přenos genetické informace není stoprocentní a bez šumu. Je tedy třeba zavést do procesu obdobu biologické genetické mutace. Bez nich bychom pouze recyklovali vzory rodičů. Více na straně 14.

- Křížení – Nový jedinec může mít více než jednoho rodiče. Poté je třeba DNA rodičů zkombinovat do nové DNA, která bude použita pro nového jedince. Více na straně 14.
- Přirozený výběr – Stejně jako v reálném světě, je třeba mít mechanismus, který zajistí, že k rozšíření genů mezi populací se dostanou nejčastěji ti úspěšnější jedinci. Toho se nejčastěji dosahuje pomocí takzvané funkce zdatnosti (viz strana 15). Ovšem je také možné pracovat i bez ní, příkladem může být již dříve zmíněný Polyworld (viz strana 4).

Výhodou genetického učení je, že je méně složité na implementaci a využití, než například mechanismus zpětné propagace (neplatí pro všechny případy). Srozumitelně popsat rozdílné principy mezi genetickým přístupem a dříve zmíněnými principy učení neuronových sítí lze takto:

- Učení s učitelem – Učitel ukazuje neuronové síti, jak to dělat lépe (chybová funkce).
- Učení bez učitele – Neuronová síť se sama rozhoduje (na základě nastavení algoritmu), co je řešení.
- Genetické algoritmy – Přirozený výběr vybírá do další generace neuronové sítě, které řešily problém lépe než ostatní (přirozeným výběrem může být funkce zdatnosti nebo svět samotný).

Například při použití v mém projektu bylo výhodou, že není třeba učitele. Jeho roli zde zastupuje funkce zdatnosti. Agent prožije svůj život ve virtuálním světě a na konci je mu přidělena jeho úspěšnost dle funkce zdatnosti. Ta je konfigurovatelná a může využít řadu parametrů, popisujících dosažené úspěchy agenta.

V podobných případech je netriviální problém určit chybovou funkci, nutnou pro metodu zpětné propagace. V těchto případech totiž netušíme, jaké by měli být výstupy. Kdybychom to věděli, tak bychom mnohdy nepotřebovali neuronovou síť.

Na genetických algoritmech je zvláštní i to, že se zde překrývá informatika a biologie. Pro informatiku je zajímavé čerpat z poznatků a hypotéz moderní biologie, ale výhodnost je oboustranná.

Informatika totiž může otestovat některé hypotézy evoluční biologie (výsledky je samozřejmě třeba brát s rezervou), popřípadě může ukazovat na zajímavé zákonitosti genetiky.

3.2.1 Křížení a mutace

Křížení a mutace jsou základním kamenem genetických algoritmů. Základním kamenem křížení a mutací je potom DNA. Ačkoliv zkratka DNA[3] se u genetických algoritmů může jevit poněkud nepatřičně, protože s žádnou kyselinou neoperujeme, zajisté si pod ní každý programátor představí jakousi obdobu zdrojového kódu živých bytostí.

DNA v oblasti genetických algoritmů aplikovaných na neuronové sítě můžeme definovat jako posloupnost hodnot (genů), podle kterých lze sestavit neuronovou síť[4]. Tato DNA může mít velikost pevně danou nebo proměnlivou.

Podle typu genetického algoritmu je třeba uvažovat o křížení. Křížení je proces, kdy z různých DNA více rodičů sestrojíme DNA pro potomka, podle které jej poté zkonstruujeme. Výhodou více rodičů je větší robustnost evoluce (vyšší polymorfismus v populaci díky kombinaci genů více rodičů [Jaroslav(2008)]), což může být výhodou.

Naopak výhodou pouze jednoho rodiče je typicky rychlejší evoluce. Ovšem je zde větší nebezpečí uvíznutí v lokálním minimu v důsledku nižší genové diversity v populaci.

Křížení může probíhat mnoha způsoby. Jedním z nich může být ten, že budoucí DNA potomka rozdělíme do několika částí, a u každé z nich náhodně určíme, čí odpovídající část DNA použijeme na tuto část.

Důležitou součástí evoluce jsou také mutace. To je způsob, jak zavádět nové vlastnosti do systému za pomocí náhody. Tyto náhodné změny pak spolu s přirozeným výběrem (řízeného také zčásti náhodou) a těžko představitelným množstvím generací a času mohou nakonec z opic stvořit o něco chytřejší opice.

3. Deoxyribonucleic acid – Deoxyribonukleová kyselina
4. A nejen ji, ale vzhledem k srozumitelnosti textu se dále bude psát jenom o neuronových sítí. Ale je třeba poznamenat, že genetické algoritmy lze aplikovat mnoha způsoby na mnoho různých problémů.

3.2.2 Funkce zdatnosti

Pokud používáme variantu genetického algoritmu, kde sami musíme rozhodovat, kteří jedinci byli úspěšní, je vhodné mít nějaký nástroj na výpočet jejich kvality. Tímto nástrojem je funkce zdatnosti.

Funkce zdatnosti je speciální funkce, která na základě úspěchů a statistik ze života jedince vypočítá jeho zdatnost. Zdatnost udává, jak byl jedinec ve svém životě úspěšný.

Je tedy zřejmé, že správně nastavená funkce zdatnosti je nutným (ale bohužel nikoliv postačujícím) předpokladem úspěšné evoluce. Pokud si při sestavování funkce zdatnosti neuvědomíme všechny souvislosti, evoluce se může vyvíjet jiným směrem, než jsme zamýšleli.

Důvodem je, že evoluce hledá nejlepší řešení problému, a když tento problém nepřesně definujeme, evoluce může najít řešení pro jiný problém, který ale neodpovídá původnímu úmyslu a zadání. Příkladem může být Polyworld (strana 4) a kapitola 6.2.1 na straně 53.

Příkladem použití funkce zdatnosti může být Evolved Virtual Creatures popsaný na straně 5 a také tato diplomová práce.

3.2.3 Generace

Každá evoluce musí mít svůj počátek. V případě evoluce našeho druhu to byly pravděpodobně sebe-replikující shluky molekul, v případě genetických algoritmů bývá první generace náhodně vygenerovaná.

Ovšem i při náhodném generování první generace je třeba zohlednit způsob generování. V případech neuronových sítí bývá obecně vhodné, aby generovaná neuronová síť měla své váhy relativně blízko nuly, abychom předešli extrémním výsledkům prvních generací. Evoluce si sama najde, které váhy jsou důležité a následně je sama zvýší.

Když už máme vytvořenou první generaci, tak ji, v případu agentů ve virtuálním prostředí, do tohoto prostředí umístíme a spustíme simulaci. Po skončení časového limitu nebo neúspěchu[5] všech agentů pro všechny agenty spočítáme jejich zdatnosti pomocí

5. Například smrt.

funkce zdatnosti.

Poté pomocí zvolené metody vybereme z populace několik agentů (například 10% s nejvyšší zdatností) a jejich DNA použijeme na vytvoření nové generace (popis v projektu implementovaných způsobů výběru rodičů lze najít na straně 37).

Pokud používáme metodu s více rodiči, DNA rodičů nejdříve zkřížíme a poté na výsledné DNA provedeme mutace (malé změny v DNA). Pomocí tohoto DNA pak vytvoříme nového agenta. Takto pokračujeme pro zbývající agenty.

Toto ovšem není jediná cesta, jak implementovat genetické algoritmy. Na příkladu Polyworldu (viz 2.1) je vidět, že není nutné rozdělit evoluci do generací, ale je možné použít i mnohem více přírodním zákonům podobný model. Dokonce ani není nutné mít funkci zdatnosti, jak již bylo podotknuto.

3.3 Stanovení cílů

V této kapitole se věnuji bližšímu popisu požadavků a cílů projektu.

Hlavní částí celého projektu je serverová část. Je to vlastně jeden běžící virtuální svět se svým vlastním nastavením daným konfiguračními soubory. Důraz je také kladen na konfigurační možnosti, v konfiguraci je možné zcela nastavit virtuální svět a cíle umělé inteligence.

V jednom virtuálním světě je také požadována možnost mít v jednom časovém úseku více různých skupin agentů s odlišnou konfigurací. Tyto skupiny jsou dále v textu nazývány „plemeny“. Mezi těmito „plemeny“ neprobíhá mísení DNA, jsou tedy druhově oddělené. Tato vlastnost umožní srovnávat různé strategie (různé typy umělých inteligencí nebo odlišně nastavené stejné umělé inteligence) při soupeření. Využití této vlastnosti je zajímavé pro pokusy typu holubi a jestřábi[6]. Více o serverové části lze najít v kapitole 4.1.2 na straně 19.

Další oblastí projektu je klientská část, která umožní uživateli připojit se na běžící virtuální svět (server) a pozorovat jej, popřípadě ovlivňovat jeho běh.

Tato klientská část bude graficky zobrazovat stav světa v reálném

6. Jeden z modelů teorie her

čase. Mezi serverovou a klientskou částí bude probíhat oboustranná komunikace. Teoreticky tedy bude možné řídit agenty z klientské části.

Další myšlenkou, která se skrývá za zvolením architektury klient-server částí, je totiž možnost rozšíření projektu takovým způsobem, aby bylo možné na serverové části provádět pouze fyziku a umělá inteligence by se mohla počítat na jiných počítačích. Tímto způsobem by se značně posunul limit výpočetní síly na jeden probíhající pokus a umožnilo by se tím použití mnohem komplexnějších umělých inteligencí.

Aplikace by měla být uzpůsobena pro běh ve více instancích, aby bylo snadné mít spuštěno několik pokusů v jednu chvíli. Počet těchto souběžných instancí se může pohybovat i v desítkách. Bude třeba tedy implementovat speciální server, který si bude udržovat přehled o probíhajících pokusech a bude umožňovat klientské části připojit se na kteroukoliv z nich (Tato část bude nazývána „Registr", více o ní v kapitole 4.1.1 na straně 19).

Finální verzí bude projekt implementovaný způsobem, který umožní budoucí rozšiřitelnost. Základním kamenem programu tedy bude modularita a možnost relativně snadno rozšířit či změnit funkcionalitu programu. Jedná se tedy o jakýsi framework. Z tohoto důvodu byl také zvolen objektově orientovaný programovací jazyk (viz kapitola 5.6.1), který dosažení těchto cílů usnadňuje.

Dalším důležitým požadavkem bude rychlost a optimalizace aplikace. Protože simulace světa, kolizí a samotných neuronových sítí je výpočetně náročná, je třeba při návrhu myslet také na výkon.

Součástí zadání je možnost přehledně zobrazovat stav prostředí a výkonnost agentů. Tento požadavek je implementován za pomocí webové stránky. Tato stránka bude mít za úkol přehledně zobrazovat grafy zdatnosti agentů, stav běžících serverů a jejich konfigurace. Uživateli bude také umožněna základní práce se zobrazením grafů. Více na straně 43.

Kapitola 4

Návrh

Tato kapitola se věnuje návrhu projektu. Jelikož metoda použitá k vývoji práce byla blízká rapidnímu prototypování[1], analýza a implementace v mnoha případech splývá.

Konkrétní implementace některých zajímavých částí je pak k nalezení v kapitole 5.6 začínající na straně 46.

4.1 Architektura

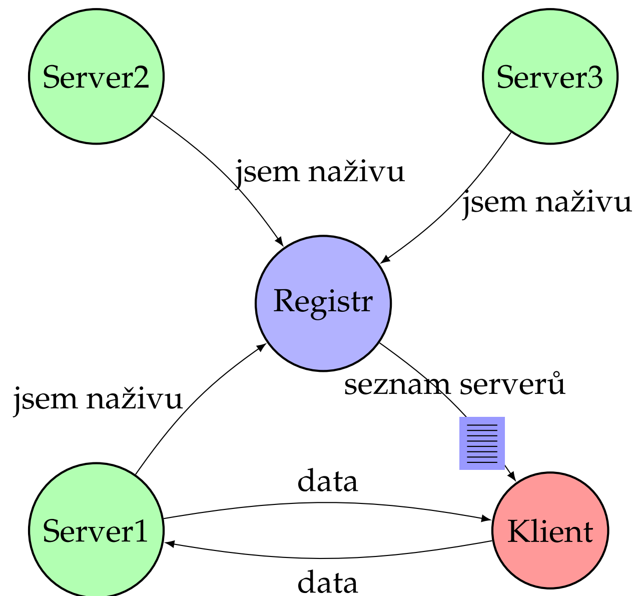
Jak již bylo naznačeno v kapitole 3.3, projekt má tři hlavní komponenty, které mezi sebou komunikují. Těmito komponentami jsou:

- Registr – udržuje seznam běžících serverů, předává jej klientovi.
- Server – Stará se o simulaci jednoho virtuálního světa.
- Klient – Stáhne si z registru seznam běžících serverů, dokáže se připojit na server a umožňuje sledovat dění na serveru v reálném čase, také jej dokáže ovlivňovat.

Přehledně zobrazuje komunikaci mezi hlavními komponentami obrázek 4.1. Komunikace mezi servery a registrem se odehrává jednou za několik minut. Server pošle registru pouze svůj název a krátký popis experimentu.

---

1. Metoda vývoje softwaru patřící mezi agilní metody.



Obrázek 4.1: Pohled na architektura projektu

4.1.1 Registr

Registr spravuje seznam běžících serverů. Registr z vlastní iniciativy nikoho nekontaktuje, servery se samy musí ohlašovat. Pokud se server delší dobu neohlásí, je vyřazen z tabulky běžících serverů.

Další logiku registr neobsahuje. Důraz je kladen na spolehlivost a nenáročnost na systémové prostředky. Součástí komunikačního protokolu je handshake, který by měl zaručit, že se připojuje validní klient nebo server.

Adresu registru je možné u klienta i serveru nastavit.

4.1.2 Server

Při spuštění serveru je jako parametr uvedena cesta ke konfiguračnímu souboru. Ten obsahuje potřebné nastavení virtuálního světa a agentů. Kromě toho server na začátku přečte další konfigurační soubory, obsahující definice typů objektů a jejich komponent (více kapitola 5.5 na straně 45 zabývající se konfiguračním soubory).

Po načtení všech konfiguračních souborů server vytvoří svět

s danými parametry a zaplní jej v konfiguraci definovanými objekty.

Server simuluje pro virtuální svět také reálný čas. Jsou podporovány dvě rychlosti: Reálný čas (server se snaží, aby jedna sekunda ve virtuálním světě byla stejně dlouhá jako v reálném světě) a maximální rychlost (server ignoruje reálný čas a provádí simulaci tak rychle, jak to výkonnost procesoru dovolí).

Je vhodné mít stanovenou nejmenší časovou jednotku – délku jednoho tiku simulace. Ten je stanoven na $1/50$ sekundy a v současné implementaci nejde konfigurovat (ale je snadné tuto možnost přidat).

Tento časový interval byl vybrán ze zkušenosti z herního prostředí. Při počtu snímků pod 20 za sekundu se stává simulace obtížně ovladatelnou i pro člověka. $1/50$ se jeví jako vhodná hodnota. Jistě by bylo zajímavé provést měření i na toto téma a zjistit, jak se umělá inteligence vyrovná s různými délkami tiku.

Kromě provádění tiku virtuálního světa server také periodicky provádí následující úkoly:

- Kontrola nových spojení a reakce na ně, realizace připojovací komunikace (handshake).
- Jednou za několik minut se server spojí s registrem.
- Odesílání změn pozic objektů klientům – odesílají se pouze malé části objektů (update) obsahující často měněné fyzikální vlastnosti (poloha, rychlost, natočení, apod.).
- Odesílání nových objektů – nově vzniklé objekty se posílají celé. Popřípadě si může klient vyžádat celý objekt, pokud jej nemá celý a přijde mu jen jeho malý update.
- Odesílání zničených objektů – klient sám nemůže prohlásit objekt za zničený, toto právo má pouze server. Existuje ale výjimka, když více updatů po sobě nepřijde informace o daném objektu, klient jej smaže ze své databáze.

4.1.3 Klient

Po spuštění se klient nejdříve pokusí připojit k registru. Pokud se to podaří, tak si z něj stáhne seznam běžících serverů. Ten poté zobrazí

uživateli. Uživatel si může vybrat server, ke kterému se připojí nebo může zadat přímo IP adresu a port.

Při připojení klienta na server obdrží klient ze serveru datovou strukturu, obsahující konfiguraci virtuálního světa (velikost světa apod.). Dále klientovi server začne posílat informace o objektech, které jsou ve virtuálním světě.

Klient okamžitě po úspěšném připojení začne ukazovat stav virtuálního světa (viz obrázek na straně 57). Změny poloh objektů v něm dostává periodicky od serveru v balících. Tyto balíky nejsou doručovány ve spolehlivém režimu. Není totiž třeba dostat je ve správném pořadí, protože klienta zajímá pouze nejaktuálnější stav. Ty staré se ignorují.

Toto je implementováno pomocí zvyšujícího se čítače balíku. Pokud klientovi přijde balík s nižším číslem, je zahozen, má už totiž novější verzi (samozřejmě je myšleno i na ochranu před přetečením). Podobný způsob se používá v herním průmyslu pro udržení nízkých latencí při online akčních hrách, kde je důležitý reakční čas soupeřů.

Klient provádí vlastní simulaci virtuálního světa (má k dispozici vektory rychlosti objektů), ale údaje ze serveru mají přednost. Tímto se kompenzují výpadky ve spojení i nižší četnost updatů ze serveru (klient obdrží přibližně 5 updatů za sekundu). Na plynulost pohybu agentů řízených na serveru tento fakt nemá postřehnutelný vliv (díky simulaci na straně klienta).

Klient uživateli umožňuje přístup do konzole, přes kterou je možné serverové části posílat textové příkazy. Těmito příkazy může ovlivňovat dění ve virtuálním světě. Dále může komunikovat pomocí těchto příkazů přímo s komponentami plemen (více na straně 23 a 36).

4.2 Architektura virtuálního světa

Hlavní třídou, která se stará o běh virtuálního světa a uchovávání informací o něm, je třída CWorld. Tato třída v sobě obsahuje všechny datové struktury a používá se jak v klientské, tak i v serverové části.

Třída CWorld obsahuje následující důležité datové objekty:

- Konfigurace serveru – tato datová struktura obsahuje načtenou konfiguraci serveru (velikost světa, port, na kterém

má server běžet apod.).

- Tabulka objektů – v této tabulce o dynamické velikosti jsou uloženy všechny aktivní objekty. Objektem je vše, co se ve virtuálním světě pohybuje nebo se pro to počítá kolize. Objektem je tedy jak agent, tak překážka nebo projektil. Více o objektech na straně 25.

- Tabulka plemen – plemeno agentů je struktura starající se o vše ohledně svých agentů. Plemeno se stará o cyklus generací, vytváření statistik, uchovávání kyblíku DNA. Více o plemenech v kapitole 4.2.1 na straně 23.

- Struktura pro uchovávání vzorů pro agenty (více na straně 27).

- Struktura pro uchovávání vzorů pro fyzickou schránku agentů (více na straně 27).

- Struktura pro uchovávání vzorů pro komponenty agentů (více na straně 27).

- Strukturu pro akceleraci detekce kolizí (více o akceleraci kolizí v kapitole 5.3.1 na straně 41).

Třída CWorld také obsahuje metody potřebné pro vytváření nových objektů a jejich správu (například přesuny na náhodné místo v definované ploše apod.). Dále také obsahuje funkce pro načítání konfiguračních souborů. Tyto soubory využívají XML formát a jsou parsovány pomocí RapidXML (více na straně 48). Různé konfigurační soubory a jejich možnosti jsou popsány v kapitole 5.5 na straně 45.

CWorld se také stará o provedení tiku virtuálního světa. Tento tik se vykonává na serverové části 50 krát za jednu sekundu. Důvody pro tuto hodnotu jsou k nalezení v předchozí kapitole. Jeden tik virtuální světa sestává z těchto úkonů:

- Naplnění struktury pro akceleraci detekce kolizí (vice na straně 41).

- Naplnění vstupních struktur agentů (senzory, čidla rychlosti, vzdálenost k cíli, apod.).
- Úprava statistik agentů – periodicky se počítá celková uražená vzdálenost, čas života, maximální vzdálenost od startovní pozice[2], nejnižší vzdálenost od cíle, apod.
- Krok objektů – u každého objektu se volá funkce zpracovávající jeho vnitřní logiku, pro agenty se v tomto bodě počítá umělá inteligence.
- Fyzikální simulace – obnáší detekci kolizí, reakci na ně a předávání parametrů o kolizích objektům.

4.2.1 Plemena

Aby bylo možné pozorovat interakci různých druhů umělé inteligence, nebo naopak dvou stejných druhů umělých inteligencí s různým nastavením v jednom virtuálním světě, je třeba je od sebe logicky a implementačně oddělit.

Toho je dosáhnuto pomocí třídy, jejíž instance se budou starat každá o svůj vlastní druh – o své vlastní „plemeno". Ve zdrojovém kódu je tato třída pojmenována CBreed, v diplomové práci ji pak označuji jako plemeno.

Třída CBreed je hlavním stavebním kamenem každého experimentu. Aby byla zachována maximální rozšiřitelnost a konfigurovatelnost, třída CBreed se skládá z různých komponent. Sama o sobě toho moc neumí. Tyto komponenty se nastavují v XML konfiguračním souboru. Plemeno si vytváří jejich instance při načítání konfigurace.

Nová komponenta vzniká (většinou) odvozením od základní prázdné komponenty přepsáním virtuálních metod třídy. Každá komponenta může svou funkcionalitu provádět v několika předem stanovených funkcích, které jsou v případě potřeby volány přímo z plemene:

- Konstruktor a destruktor.

2. Tento parametr se ukázal býti velice důležitým. Viz strana 50.

- Funkce zpracovávající tik světa.
- Funkce volaná v případě obdržení zprávy pro dané plemeno – argumentem je textová zpráva.
- Funkce volaná při smrti agenta patřícího do plemene – využívají komponenty starající se o funkci zdatnost. Parametrem je ukazatel na agenta.
- Funkce pro konstrukci komponenty z XML dat (odkaz do struktury RapidXML).
- Funkce pro reakci na začátek nové generace.
- Funkce požadující vytvoření nového agenta – komponenta implementující tuto funkci se stará o výběr vhodných rodičů pro nového agenta.
- Funkce dávající možnost modifikovat právě vytvořeného agenta (například přesun, nastavení cíle) – volá se po vytvoření agenta a dávají všem komponentám možnost jej upravit.

Jak již bylo dříve uvedeno, plemeno samo o sobě téměř nic nedělá, jeho úkolem je zapouzdřit sadu komponent a dát jim přístup ke svým datovým strukturám. Instance CBreed obsahuje tyto hlavní struktury:

- Kyblík s DNA – zde se ukládají DNA agentů po smrti. Každý objekt v kyblíku se skládá z DNA, statistických údajů a dalších pomocných údajů. Plemeno se samo o sobě nestará o mazání DNA z kyblíku. Tuto funkcionalitu je třeba implementovat v některé komponentě.
- Seznam „živých“ agentů – například při ukončení generace časovým limitem je nutné ukončit zbylé žijící agenty.
- Vektor komponent – zde jsou instance komponent, z nichž je konkrétní plemeno tvořeno.
- Statistické data plemene.

CBreed také obsahuje tyto hlavní funkce:

- Konstruktor a destruktor – inicializace plemena a deinicializace, při deinicializaci je nutné zavolat destruktory komponent, které, pokud je to jejich úkol, budou reagovat na konec experimentu.
- Funkce pro zpracování tiku virtuálního světa – volá funkce pro zpracování tiku všech svých komponent.
- Funkce reagující na zničení agenta patřícího do plemene – zpracovává DNA, volá odpovídající funkce komponent.
- Funkce pomáhající při tvorbě nové generace.
- Další funkce pro vnitřní funkci plemene.

Velký důraz celého návrhu plemen a jejich komponent byl kladen na rozšiřitelnost. Určitě není podchycen každý případ, ale díky modulárnímu designu není příliš těžké přidávat novou funkcionalitu. Díky předávání konfiguračních parametrů ve formě XML struktury pomocí RapidXML, je snadné si zpracování konfigurace přizpůsobit.

4.2.2 Objekt

Třída CObjekt je základní stavební jednotkou virtuálního světa. Vše co má nějaké fyzikální vlastnosti nebo má být zobrazeno, je objektem nebo je od něj odvozeno. Tato základní třída obsahuje dvě fyzikální struktury, jednu pro rychlé updaty, které posílá server klientovi, ta obsahuje pouze často měněné údaje o objektu:

- ID objektu – jednoznačný identifikátor objektu (ten se sice často nemění, ale slouží k jednoznačné identifikaci objektu, ke kterému se váže zbytek updatu).
- Pozici objektu – 2D vektor obsahující pozici objektu ve virtuálním světě.
- Úhel natočení objektu v radiánech.
- Změnu natočení za 1 sekundu.

- Vektor pohybu objektu.
- Relativní hodnotu zbývajícího života objektu – nabývá hodnot $0-1$. Typicky se snižuje po nárazu do jiného objektu. Klesne-li pod nulu, objekt je zničen.

Další struktura obsahuje údaje, jež není potřeba periodicky obnovovat, protože se u nich nepředpokládají změny v průběhu životnosti objektu. Tato struktura obsahuje následující parametry:

- ID objektu – jednoznačný identifikátor objektu (pro jistotu je ID zduplikováno i v této struktuře).
- Hmotnost objektu.
- Kolizní poloměr – velikost objektu (v současné implementaci je každý objekt kruhem).
- Maximální hodnota života – aktuální hodnota života (az) se vypočítá následovně: $az = rz * mz$, kde rz je relativní hodnota života a mz je maximální hodnota života.
- Odolnost objektu – koeficient jímž se násobí udělené poškození
- Statický objekt – přepínač, který určuje, jestli je možné objektem pohnout nárazem jiného objektu (je vhodné využít pro situace, kde je žádané, aby svět zůstal nezměněný po celou dobu trvání experimentu).
- Přepínače pro detekce kolizí.
- Dvě hodnoty pro tuhost: r a s. Používají se při výpočtu, jestli bude objektu uděleno vypočtené poškození. Hlavní myšlenkou je, že malý náraz by neměl způsobit žádnou ztrátu života. Pokud $p < (s + r * az)$ není poškození p uděleno.
- ID agenta, který stvořil tento objekt. Používá se pro připsání bodu za zničení jiného agenta projektilem.
- Typ – obsahuje informaci o typu instance (objekt, agent, asteroid, projektil).

- ID modelu – který model se má použít při vykreslení objektu.

CObjekt je základní třída, od které všechny ostatní implementované třídy dědí. Dalšími implementovanými třídami jsou třídy pro projektily, agenty a asteroidy. Každá odvozená třída má vlastní přidanou funkcionalitu, která se řeší přepsáním virtuálních funkcí třídy CObjekt. Těmito virtuálními funkcemi jsou:

- Konstruktor objektu.
- Destruktor objektu.
- Funkce volaná při tiku světa (zde se například zpracovává umělá inteligence u agentů).
- Funkce volaná při udělení poškození objektu.
- Funkce informující objekt, že byl zničen (například agent zde ohlašuje své zničení svému nadřazenému plemenu, to pak zpracuje jeho DNA).
- Funkce pro zabalení celého objektu do surových dat pro přenesení mezi serverem a klientem.
- Funkce pro rozbalení celého objektu ze surových dat pro přenesení mezi serverem a klientem.

Tento návrh je vhodný pro pozdější možné rozšíření funkcionality vytvořením nového typu objektů.

4.2.3 Agent

Agent je implementován rozšířením třídy CObjekt. Třídu CObjekt rozšiřuje o funkcionalitu a datové struktury, potřebné pro vytvoření agenta. Důležité je, že tato nová třída obsahuje instanci umělé inteligence.

Objekt agenta obsahuje tyto datové struktury a třídy (také obsahuje všechny struktury obsažené v třídě objekt):

- Instanci své umělé inteligence – více na straně 29.

- Statistiky – tato struktura obsahuje informace o dosavadních úspěších agenta. Tato struktura se kopíruje v okamžiku smrti agenta spolu s DNA agenta do kyblíku DNA v mateřském plemenu agenta. K této struktuře přistupuje funkce při výpočtu funkce zdatnosti.
  - Celková uražená vzdálenost.
  - Startovací a konečná pozice.
  - Nejvyšší vzdálenost od startovního bodu.
  - Vzdálenost od startovního bodu v okamžiku skončení života agenta.
  - Čas života agenta.
  - Průměrná rychlost.
  - Minimální vzdálenost od cíle.
  - Vzdálenost od cíle v době smrti.
  - Počet zničených agentů.
  - Počet vypuštěných projektilů.
  - Udělené poškození jiným agentům.
  - Hodnota zdatnosti.
- Instance fyzické schránky.
- Instance jednotlivých komponent agenta.
- Odkaz na plemeno, do kterého agent patří.
- Strukturu obsahující nastavení cíle – agentu může být určen cíl, k dispozici má pak vzdálenost a odchylku od svého směru směrem k cíli. Tímto cílem může být oblast, úsečka, nebo jiný agent.

Novou instanci agenta zpravidla tvoří plemeno při začátku nové generace. K vytvoření nové instance agenta potřebujeme znát:

- název konfigurace agenta – konfigurace pak obsahuje:

  - název fyzické schránky – ta obsahuje informace o fyzických vlastnostech agenta (velikosti, život, počet slotů a rozmístění slotů apod.
  - seznam použitých komponent – komponentou může být pohon nebo zbraň (tento způsob práce je zvolen kvůli rozšiřitelnosti a konfigurovatelnosti).
  - Textový popis dané konfigurace.

- Typ umělé inteligence – abychom mohli vytvořit odpovídající instanci umělé inteligence.
- DNA pro umělou inteligenci – Novou instanci umělé inteligence lze stvořit buď za pomoci DNA, nebo odkazu na část XML struktury obsahující konfiguraci umělé inteligence.
- Mateřské plemeno – každý agent patří do nějakého plemena (plemena viz strana 4.2.1). Agent potřebuje znát to své, protože v okamžik zničení ohlásí plemenu svůj konec.

V průběhu života agenta se každý tik světa volá i funkce zpracovávající tik v agentovi, tato funkce poté zavolá výpočet umělé inteligence a také funkce zpracovávající tik pro komponenty agenta (více o komponentách agentů lze najít na straně 69).

4.2.4 Návrh umělé inteligence

Základním kamenem umělé inteligence je prázdný objekt obsahující rozhraní, přes které je s umělou inteligencí komunikováno. Od tohoto prázdného objektu pak dědí jednotlivé implementace umělé inteligence.

Implementovat nové typy umělé inteligence je tedy pouze otázkou vytvoření nového objektu a implementování základních funkcí. Cílem návrhu je učinit návrh nové inteligence co nejjednodušší při zachování tvůrčí svobody.

V jednom plemenu může být použit pouze jeden typ umělé inteligence s pouze jedním nastavením. Tím je zaručena kompatibilita mezi všemi agenty v plemenu. Tvůrce dané implementace umělé inteligence má kontrolu nad strukturou DNA a může s ní libovolně pracovat. Z pohledu zbytku programu jsou DNA jen hrubá data.

Tento objekt obsahuje tyto funkce a datové struktury:

- Vstupní objekt pro UI – obsahuje veškeré senzorické vstupy pro umělou inteligenci. Vstupy jsou znormalizovány do rozmezí $[0, 1]$ nebo $[-1, 1]$.
  - Senzory vzdálenosti – v současné době je pevně nastaveno 9 senzorů o úhlech $0, 10, 25, 45, 80$ (symetricky) a délkách $17, 15, 12, 9, 9$ (od čelní po krajní). Více o senzorech viz kapitola 5.3.2 na straně 42.
  - Život.
  - Bolest.
  - Rychlost.
  - Rozdíl úhlu – rozdíl mezi vektorem pohledu agenta a jeho pohybem (tato informace pomáhá při pohybu se zapnutou setrvačností).
  - Vzdálenost k cíli – normalizovaná a popřípadě ořezaná, pokud tak bylo nastaveno.
  - Úhel k cíli – rozdíl mezi vektorem pohledu agenta a vektorem ukazujícím směrem k cíli.
- Výstupní objekt pro UI – obsahuje možnosti umělé inteligence interagovat s okolním světem.
  - Výkon motoru vpřed (v případě záporné hodnoty vzad, nula znamená žádný výkon).
  - Výkon motoru do strany (úkrok).
  - Otáčení vpravo a vlevo.
  - Aktivace komponent – použitá například pro aktivaci zbraní.
- Konstruktor a destruktor.
- Rozhodovací funkce – ta se volá každý tik, na základě vstupní struktury nastaví strukturu výstupní, podle které se pak řídí komponenty (například pohon, nebo zbraně).

- Funkce inicializující umělou inteligenci z XML – tato funkce dostává jako argument ukazatel do XML struktury (využívá se knihovny RapidXML) na část obsahující nastavení konkrétní umělé inteligence. Každá umělá inteligence má tak možnost načíst konfigurační data specifické pro svůj typ. Tato funkce nemusí být optimalizovaná, protože se volá pouze při tvoření první generace. Další generace jsou už tvořeny pouze pomocí DNA rodičů. To znamená, že pokud síť umožňuje konfiguraci, tak ta musí být součástí DNA.

- Funkce pro odevzdání svého DNA – volá se typicky při smrti agenta, umělá inteligence vytvoří DNA a to je uloženo do kyblíku DNA plemene, kde počká na využití při tvorbě potomků.

- Funkce pro vložení DNA jednoho rodiče – umělé inteligenci je předložena DNA ve formě surových dat.

- Funkce pro vložení DNA dvou rodičů – umělá inteligence zde musí provést vlastní křížení z DNA dvou rodičů.

- Funkce mutace DNA – každá umělá inteligence má implementován svůj vlastní způsob mutace. Funkce je volána po vložení DNA rodičů. Funkce jako parametr dostává velikost mutace uvedený v konfiguraci.

- Funkce pro dokončení vytváření DNA – jejím úkolem je vytvořit funkční instanci umělé inteligence, pokud již se tak neděje ve funkcích, které obstarávají vkládání DNA. Tato funkce je volána až po mutaci. V případě, že se jedná o první generaci je volána po funkci inicializující UI z XML.

4.2.5 Fyzika

V projektu je implementován jednoduchý fyzikální model s kolizemi založenými na kruhových objektech. Tímto se značně ulehčí detekce kolizí a reakce na ně. Více o implementaci konkrétních algoritmů na straně 41.

Protože náročnost detekce kolizí při použití naivního přístupu (detekce každý s každým) roste exponenciálně s počtem objektů,

byla by tedy nevhodná pro virtuální světy s více objekty. Je implementována alespoň základní akcelerace kolizí pomocí rozdělení objektů do mřížky (více na straně 41).

Kapitola 5

Implementace

5.1 Umělá inteligence

Jak již to bylo popsáno v na straně 29 v kapitole popisující návrh fungování umělé inteligence v tomto projektu, vzniká umělá inteligence děděním od prázdné třídy umělých inteligencí. V třídě nové umělé inteligence se implementují virtuální třídy rodiče.

Je úkolem vývojáře, aby správně implementoval vše potřebné. Díky možnosti napsat novou umělou inteligenci přímo v programovacím jazyce projektu, může být tato implementace velmi rychlá. Zároveň má vývojář obrovské možnosti díky svobodě, kterou mu tento přístup poskytuje.

Třída umělé inteligence má k dispozici datové struktury pro obousměrnou komunikaci s okolím. Tyto datové struktury slouží jako jednoduché rozhraní mezi umělou inteligencí a virtuálním světem. Byly navrhnuty s ohledem na jednoduchost a rychlost.

Jak již bylo zmíněno v minulých kapitolách, každá implementace umělé inteligence má při generování nové generace přístup k části XML struktury, ve které se nachází konkrétní konfigurace pro daný typ umělé inteligence. Formát, parametry a strukturu tohoto popisu určuje tvůrce dané implementace. Díky tomu je dosaženo požadavku na dobrou konfigurovatelnost.

5.1.1 Náhodná

Velice jednoduchá umělá inteligence, jejím úkolem je poskytnout možnost srovnání úspěšnosti testované umělé inteligence oproti náhodnému algoritmu. Výsledky sofistikované umělé inteligence totiž nejsou vždy zaručeně lepší než u obyčejného náhodného algo-

ritmu.

Náhodný algoritmus zde pracuje tak, že jednou za nastavený čas (pokud není nastaveno jinak, tak je tento čas 1 sekunda) vybere náhodná čísla a ta vloží do výstupní struktury umělé inteligence (viz strana 30). Tyto výstupní hodnoty pak zůstávají stejné až do další změny.

U této umělé inteligence je jedinou možností nastavení v konfiguračním souboru možnost nastavit povolené výstupy a časový okamžik, za který se změní hodnoty na výstupu.

5.1.2 Vícevrstvá neuronová síť

Jako příklad složitější umělé inteligence byla implementována konfigurovatelná vícevrstvá neuronová síť. Tato neuronová síť využívá pro učení evoluční algoritmy. Její DNA se skládá z konfigurace (počet vrstev, neuronů, aktivované vstupy a výstupy, atd.) a vah. Mutace a dědění se provádějí pouze na vahách.

Tato neuronová síť je dopředná, acyklická vícevrstvá neuronová sít, kde každý neuron vnitřní a výstupní vrstvy je propojen se všemi neurony vrstvy předcházející.

Výpočet probíhá v několika krocích, kdy postupně spočítáme výsledky všech vrstev, než vypočteme výsledek na vrstvě výstupní. Díky tomu, že tato neuronová síť je acyklická, se výpočet zastaví. Celý tento výpočet probíhá v každém jednotlivém tiku virtuálního světa (tedy 50 krát za sekundu).

Váhy neuronové sítě zůstávají stejné po celou dobu života agenta. Neuronová síť je acyklická, nemůže si tedy nic pamatovat (v cyklech lze uchovávat informace). Pro konkrétní vstupní hodnoty bude tedy mít neuronová síť po celou dobu života stejné výstupní hodnoty.

Nastavení této neuronové sítě umožňuje měnit povolené vstupy a výstupy. Další nastavení se pak týká samotné umělé inteligence:

- Typ aktivační funkce – implementované jsou následující:
  - Lineární funkce $f(x) = x$
  - Omezená lin. $f(x) = \begin{cases} 1.5, & \text{pokud } x \geq 1.5 \\ x, & \text{pokud } -1.5 < x < 1.5 \\ -1.5, & \text{pokud } x \leq 1.5 \end{cases}$

  - Hyperbolický tangens[1] $f(x) = 1.7159 * \tanh(\frac{2}{3} * x)$

- Způsob provádění mutace – lze zvolit tyto druhy mutace:
  - Fixní mutace: $x_\Delta = x + m * r$, kde r je náhodné číslo z intervalu $[-1, 1]$, kde pravděpodobnost lineárně klesá od 0 směrem ke krajním hodnotám. Kde m je síla mutace definovaná v konfiguraci.
  - Relativní mutace: $x_\Delta = x * (1 + m * r) + m * r$, kde r je náhodné číslo z intervalu $[-1, 1]$, kde pravděpodobnost lineárně klesá od 0 směrem ke krajním hodnotám. Kde m je síla mutace definovaná v konfiguraci.
- Způsob pro křížení v případě použití více rodičů. Jsou implementovány tyto tři způsoby:
  - Každá váha náhodně – u každé váhy se náhodně rozhoduje, od kterého rodiče bude pocházet.
  - Poloviny – DNA se rozdělí na dvě poloviny a náhodně se rozhodne, která polovina bude pocházet od kterého rodiče.
  - Náhodný – algoritmus náhodně rozdělí DNA na náhodný počet (průměrně 6) různě velkých dílů a u těch potom náhodně vybere rodiče, od kterého bude pocházet daná část DNA.
- Počet vnitřní vrstev – je v současné implementaci programově omezen konstantou na 8.
- Počet neuronů v každé vnitřní (skryté) vrstvě.
- Počet neuronů ve vstupní a výstupní vrstvě je dán počtem vstupů a výstupů.
- Iniciální rozpětí vah – rozpětí ve kterém budou váhy v první[2] generaci.

1. Se zvolenými koeficienty je funkce téměř lineární v rozsahu $(-1, 1)$ a mezní hodnoty se blíží ± 1.7159 [LeCun et al.(1998)LeCun, Bottou, Orr, and Müller]).
2. První generace má váhy vytvořené náhodně.

- Maximální hodnota váhy – hodnoty vah budou ořezány, pokud překročí tyto hodnoty.

Možnosti nastavení jsou komplexní, díky tomu je možno porovnávat různé přístupy k neuronovým sítím a genetickým algoritmům.

5.2 Plemena a jejich komponenty

Jak již bylo popsáno v kapitole zabývající se návrhem funkce plemen (viz strana 23), samo o sobě plemeno moc velkou funkcionalitu nemá. To, co z něj dělá funkční objekt, jsou komponenty, které obsahuje. Tyto komponenty se pak starají o veškerou přidanou funkcionalitu.

Komponenty jsou do plemene přidávány na základě XML konfigurace. Díky tomu je možné implementovat mnoho různých způsobů funkce komponenty plně konfigurovatelných uživatelem. Zároveň je snadné přidávat nové komponenty a tím rozšiřovat funkcionalitu.

V následujících podkapitolách jsou popsány některé implementované komponenty a jejich použití.

5.2.1 Generace

Důležitou třídou komponent jsou komponenty, rozhodující kdy a kolik agentů je třeba vytvořit.

Pro použití modelu s oddělenými generacemi bylo potřeba vytvořit komponentu, která by tyto generace obstarávala. Tato komponenta pracuje tím způsobem, že hlídá počet agentů plemene ve virtuálním světě a čas trvání generace. Nová generace začíná po uplynutí času (přeživší agenti jsou zničeni před začátkem nové generace, jejich DNA je tedy přidána do kyblíku DNA), nebo při poklesu počtu živých agentů na nulu.

Tato komponenta má následující konfigurační možnosti:

- Počet agentů v jedné generaci.

- Maximální čas jedné generace.

- Typ umělé inteligence.
- Přepínač určující, má-li se počkat na konec časového limitu i když není ve virtuálním světě už žádný živý agent plemene (je vhodné při porovnávání dvou různých plemen v jednom virtuálním světě, pokud není použito mohou se generace rozcházet).

5.2.2 Výběr rodičů

Další otázkou při použití evolučních algoritmu je výběr vhodných rodičů. Stejně jako v reálném světě je tato volba velmi důležitá. Jako ke všem důležitým otázkám existuje i k této mnoho odpovědí, z nichž několik bylo vybráno pro implementaci v této diplomové práci.

Tento typ komponenty používá k rozhodování statistické informace patřící ke každé jednotlivé DNA v kyblíku DNA. Je pak na konkrétní komponentě, jak bude problém výběru rodičů řešit.

První implementovaná komponenta, zabývající se výběrem rodičů, je inspirována reálným světem: Motivace pro tuto metodu je, že v reálném světě potkáme pouze část možných partnerů a z těch si vybereme toho nejlepšího (v ideálním případě).

Tento způsob je implementován tak, že pro každého nového agenta náhodně vybereme z kyblíku DNA n DNA a z nich následně vybereme tu jednu z nejlepších DNA. To probíhá tak, že první přijmeme automaticky a pro každou další, pokud je lepší, než ta současná zvolená DNA, rozhodne o přijetí náhodné číslo. Pokud je toto náhodné číslo větší než parametr, který může nabývat hodnot $[0, 1]$. Pro další nové agenty opakujeme.

V konfiguraci lze nastavit:

- Počet DNA, které algoritmus projde (absolutně, nebo relativně ve vztahu k počtu DNA v kyblíku DNA).
- Pravděpodobnost přijetí nového DNA.
- Velikost mutace po křížení.
- Název konfigurace agenta.

Jiná komponenta vybere X nejlepších DNA z kyblíku DNA a vytvoří z nich „elitní“ kyblík, z toho pak náhodně vybírá vhodné rodiče. U této komponenty lze nastavit tyto parametry:

- Velikost elitního kyblík DNA (absolutně, nebo relativně ve vztahu k počtu DNA v kyblíku DNA).
- Velikost mutace po křížení.
- Název konfigurace agenta.

Motivací za tímto způsobem je například způsob šlechtění rostlin, cíleně vybíráme nejlepší jedince k dalšímu rozmnožení na základě jejich schopností (zde podle jejich zdatnosti).

5.2.3 Logování

V základním provedení plemena neprovádějí žádným způsobem ukládání statistik. Jediným způsobem, jak sledovat pokrok experimentu, je tedy sledovat chování umělé inteligence pomocí připojeného klienta.

Jelikož tohle řešení není ideální, jsou implementovány základní logovací komponenty. Tyto komponenty využívají funkci pro zpracování nové generace, která se volá pro všechny komponenty při generování nové generace. V této funkci pak provádějí výpis dat na statistiku nebo do souboru.

První komponenta vypisuje po skončení generace její aktuální statistiku do konzole. To jest průměrné hodnoty všech statistických údajů uchovávaných u agentů z poslední generace (obsah statistiky je popsán na straně 28).

Další komponenta umožňuje zápis statistiky do csv databázového souboru. Nový záznam je přidán s každou další generací. Je možné dokonce nechat zapisovat více plemen do jednoho csv souboru, každý záznam je totiž opatřen identifikátorem plemena.

Další komponenta pro logování se zabývá splněním požadavku na přehledný způsob zobrazení průběhu experimentu. Jedná se o komponentu generující HTML statistiky. Více o návrhu HTML statistiky se lze dočíst na straně 43.

U této komponenty lze nastavit cestu k HTML statistice a index souborů, do kterých se má zapisovat. Kvůli omezení popsaném

v kapitole 5.4 nemohou být data dynamicky načítány, ale musí být načteny najednou spolu se HTML stránkou. Problém je vyřešen tak, že stránka se pokouší vložit javascriptové soubory předem definovaných jmen („breedX.js“ a „breeddatX.js“, kde X je z předem daného rozsahu), data z těchto souborů pak jsou zobrazeny ve statistice.

5.2.4 Výpočet zdatnosti

Pro zachování rozšiřitelnosti byla vytvořena také funkce počítání zdatnosti jako komponentu. Tento přístup má tu výhodu, že lze vytvořit jinou komponentu s odlišnou funkcí, pokud je třeba.

První implementovaná komponenta pro počítání funkce zdatnosti je velice jednoduchá a jde pouze o součet všech údajů ze statistiky agenta násobenými nastavitelnými koeficienty.

Kvůli potřebě složitějších funkcí zdatnosti byla implementována také komplexnější komponenta, která umožňuje funkci zdatnosti definovat textově popsaným vzorcem, který je při načítání komponenty parsován a vzniká podle něj stromová struktura, jejímiž listy jsou buď konstanty nebo proměnné reprezentující hodnoty ze statistik konkrétních agentů. Díky tomuto přístupu je výpočetní náročnost i složitých funkcí malá, neboť parsování textu probíhá jen při načítání komponenty z XML.

Komponenta také dovoluje používat kromě základních matematických operací i složitější matematické funkce, jako například $\log(x), \sin(x), \cos(x), x^y, \max(x, y), \min(x, y)$. Komponenta rovněž obsahuje i funkce pro generování náhodných čísel.

5.2.5 Modifikace agentů

Pro modifikace agentů ihned po vytvoření bylo implementováno několik komponent schopných měnit pozici a nastavení agenta:

- Komponenta pro změnu pozice agenta.
- Komponenta pro změnu natočení agenta.
- Komponenta pro nastavení cíle agenta.

Komponenta pro změnu pozice agenta slouží k přesunutí agenta na náhodné místo ve čtvercové oblasti vymezené konfiguračním souborem. Komponenta kontroluje kolize s objekty již v oblasti přítomnými.

Komponenta pro změnu natočení agenta nastaví směr pohledu agenta na číslo uvedené v konfiguračním souboru.

Komponenta pro nastavení cíle agenta umožní nastavit strukturu agenta definující cíl agenta. Agent má k potom k dispozici informace o vzdálenosti (normalizované) a odchylce svého pohledu od cíle. Tímto cílem může být:

- Úsečka – V konfiguraci se určí dva body v prostoru, minimální a maximální vzdálenost (pokud jsou tyto hodnoty překročeny, tak je vzdálenost ořezána).
- Kruhová oblast – V konfiguraci se zadá bod v prostoru, poloměr a maximální vzdálenost pro ořez.
- Nejbližší agent – lze nastavit, má-li se jednat o agenta ze stejného plemene, z jiného plemene nebo libovolného agenta.

5.2.6 Různé

Další důležitou komponentou je komponenta starající se o úklid starých DNA z kyblíku DNA. DNA se totiž z kyblíku neuvolňují automaticky. To je vhodné pro pokusy s větší genetickou základnou. Tato komponenta má jediný parametr, který určuje, s jakou pravděpodobností bude staré DNA smazáno.

Další implementovanou komponentou je komponenta starající se o ukládání nejlepších generací do souborů. Pracuje tak, že se ukládá jak DNA, tak i jednotlivé statistiky příslušných DNA. Zároveň se ukládají i datové struktury plemene. To spolu s dalšími provedenými úpravami umožňuje experiment přerušit a pokračovat v něm později.

Vybírání generací probíhá tak, že aktuální generace je uložena do struktury pouze tehdy, když splňuje následující tři podmínky:

- Má vyšší průměrnou zdatnost větší než nejlepší z uložených generací.

- Počet generací mezi poslední uloženou generací je vyšší, než je definováno v konfiguraci.
- Poměr zdatností nové a dosavadní nejlepší je lepší, než je definováno v konfiguraci.

  Také je třeba experiment ukončit. Pro tento účel byla implementována komponenta starající se o ukončení experimentu po uplynutí nastaveného času nebo nastaveného počtu generací.

5.3 Fyzika

Fyzika pracuje se všemi instancemi objektů ve virtuálním světě. Všechny potřebné fyzikální parametry jsou nedílnou součástí datové struktury objektů (viz strana 25).

Všechny objekty mají kruhovou kolizní zónu. Při detekci kolize se vypočítají nové pozice objektů (aby nenastala kolize znovu v příštím tiku), vypočítají se nové pohybové vektory (v potaz se bere hmotnost, vzájemná poloha při nárazu), dále se vypočítá poškození pro oba objekty a zavolají se příslušné funkce obsluhující náraz objektu.

Matematikou, která se ukrývá za reakcí na kolize, se podrobněji zabývat nebudeme, neboť není stěžejním bodem této diplomové práce.

5.3.1 Akcelerace detekce kolizí

Protože detekce kolizí metodu testování každý s každým má exponenciální složitost (což začal být problém při vyšším počtu objektů ve virtuálním světě), bylo nutné implementovat alespoň jednoduchou akceleraci detekce kolizí.

Zvolena byla metoda akcelerace pomocí mřížky. Tato metoda používá dvourozměrné pole listů (čtverců) o rozumné dimenzi[3].

Akcelerace pomocí mřížky pak v jednom tiku vypadá následovně (pro výpočty složitosti platí, že n je počet objektů a g je počet listů

3. Tak aby počet objektů v jednom listu byl průměrně rozumný počet objektů. To, kolik rozumný počet znamená, záleží na konkrétní implementaci.

v mřížce):

- Vyčištění struktury – složitost $\mathcal{O}(g)$.
- Naplnění struktury čerstvými daty – složitost $\mathcal{O}(n)$.
- Testování kolizí – složitost[4] $\mathcal{O}(n * \frac{n}{g})$. V praxi se osvědčila hodnota n/g kolem 3 (to znamená v průměru tři objekty na jeden list), pokud tedy fixujeme poměr n/g, dostáváme průměrnou složitost ve třídě $\mathcal{O}(n)$.
- Grid se používá i pro zjištění blízkých objektů, které by mohly být zachyceny senzory.

Samozřejmě se kvůli zvýšené režii akcelerace vyplatí pouze pro netriviální světy s vyšším počtem objektů. Ale pro virtuální světy, které jsou uvedeny jako příklady (stovky objektů) v této diplomové práci, bylo zrychlení celého výpočtu přibližně desetinásobné. Při větších světech by rozdíl byl řádově vyšší.

5.3.2 Senzory umělé inteligence

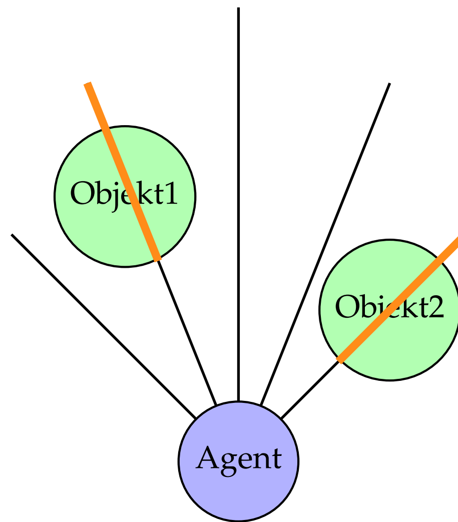
Aby agenti (jejich umělé inteligence) mohli reagovat na virtuální svět, potřebují prostředky pro jeho vnímání. Tímto prostředkem je několik čidel vzdálenosti ukazujících vpřed.

Současná implementace obsahuje 9 pevně definovaných senzorů o úhlech $0, 10, 25, 45, 80$ od směru pohledu (symetricky doprava a doleva) a délkách $17, 15, 12, 9, 9$ (od čelního po krajní).

Tyto senzory pracují tak, že pokud v jejich dosahu není žádný objekt, pro umělou inteligenci jde o vstup 0. Pokud senzor detekuje objekt, umělé inteligenci je předána hodnota v normalizovaném intervalu $[0, 1]$ podle toho, jak moc blízko se objekt nachází.

Pro lepší pochopení se můžeme podívat na obrázek 5.1. Na něm je vyznačeno pět senzorů, z nichž dva zachycují nějaké objekty. Oranžovou barvou je označena informace od senzoru, že je nablízku objekt. Čím by pak byla oranžová čára delší, tím více by se hodnota senzoru blížila 1.

4. Při rovnoměrném rozložení objektů po ploše virtuálního světa.



Obrázek 5.1: Senzory vzdálenosti agenta

Dá se tak říci, že senzor předává umělé inteligenci velikost hrozby nárazu. Kromě těchto senzorů na vzdálenost má umělá inteligence k dispozici i další senzory, které jsou popsány na straně 30.

5.4 HTML statistiky

Součástí diplomové práce je také HTML statistika. Ta je implementována jako jedna z komponent, ze kterých se skládají plemena (viz strana 23). Stránky využívají HTML, JavaScript a kaskádové styly. Důležité bylo, aby k jejich prohlížení nebyl třeba webový server.

Kvůli požadavku na nevyžadování webového serveru bylo třeba učinit kompromis v oblasti datových souborů. Některé prohlížeče totiž nepodporují dynamické nahrávání souborů z lokálního adresáře z důvodu bezpečnosti. Z tohoto důvodu je nutné, aby data byly součástí stránky. Tohoto je docíleno tak, že soubory mají koncovku js a mají formát JavaScriptové funkce, která naplňuje JavaScriptové proměnné a pole hodnotami potřebnými k zobrazení statistiky. Tyto soubory jsou načteny spolu se stránkou.

Server tedy s HTML stránkou komunikuje tak, že prakticky mění její zdrojové soubory. Ačkoliv se tento způsob nedá označit za ele-

gantní, svůj úkol plní.

Kvůli tomu bylo nutné implementovat funkce pro práci s JavaScriptovými datovými soubory s možností append[5] na již vytvořený soubor.

Vzhledem k nepoužití webového serveru[6], je uživatelské rozhraní webové stránky implementované pomocí javacriptu. Uživatel má možnost zobrazit množství informací o probíhajících experimentech:

- Název serveru, název plemene.
- Doba běhu experimentu – v čase virtuálního světa.
- Doba běhu experimentu – v čase reálného světa.
- Datum a čas startu experimentu.
- Velikost virtuálního světa.
- Verzi serveru.
- Stav serveru – běžící, vypnutý, běžící maximální rychlostí[7]
- Počet DNA v kyblíku DNA.
- Číslo generace.
- Poslední zdatnost – průměrná zdatnost poslední generace.
- Rychlost simulace – při zapnuté maximální rychlosti udává, kolikrát je simulace rychlejší, než reálný čas.
- Nastavení serveru – XML konfigurační soubor serveru.
- Nastavení plemene – část XML konfiguračního serveru s nastavením vybraného plemene (informace je duplikována pro lepší přehlednost).

5. Připojení dat na konec již naplněného souboru.
6. Například Apache, Nginx, apod.
7. Speciální režim, kdy server nechává simulaci probíhat maximální rychlostí. Jedna hodina ve virtuálním světě pak může v závislosti na hardwaru a konfiguraci virtuálního světa trvat méně než minutu ve světě reálném.

- Popisy vybraného plemene a jeho serveru.
- Grafy vývoje statistických údajů pokusů.

Grafy jsou zobrazovány pomocí knihovny jqPlot(viz strana 48). Při zobrazování grafů s mnoha záznamy je problém s výkonem. Proto bylo nutné implementovat funkci, která nejdříve provede kompresi dat (sloučení n záznamů do jednoho za pomocí průměrování).

Uživatel může provádět některé základní operace s grafy, aby měl možnost rychle porovnat výsledky různých pokusů:

- Vybírání plemen k zobrazení.
- Volba porovnávaného atributu (zdatnost, doba života, uražená vzdálenost atd.).
- Možnost přiblížení určitého místa.
- Možnost vyhlazení křivky (kvůli lepší čitelnosti grafu).

HTML statistiky jsou implementovány tak, aby bylo zobrazitelné v moderních prohlížečích (Google Chrome, Firefox, Opera). HTML statistika byla vyvíjena a testována převážně na webovém prohlížeči Chromium[8].

5.5 Konfigurační soubory

Jedním z hlavních bodů zadání diplomové práce je možnost specifikovat maximum parametrů pomocí konfiguračních souborů. Pro dosáhnutí tohoto úkolu je vytvořen systém s několika různými konfiguračními soubory, obsahujících různá data. Tento systém v této kapitole popíši.

Všechny tyto konfigurační soubory využívají formát XML a parsování probíhá pomocí RapidXML parseru (viz strana 48).

Je implementována podpora pro tyto konfigurační soubory:

- Konfigurace světa – zde se nachází popis konkrétního experimentu (strana 63) a konkrétních plemen (strana 65).

8. Chromium je open source projekt stojící za Google Chrome

- Konfigurace modelů – zde se vytváření modely (v konkrétní implementaci se jedná pouze o textury), které pak lze přiřadit objektům (strana 66).
- Soubor s konfiguracemi agentů (strana 67).
- Konfigurace fyzické schránky agentů (strana 69).
- Konfigurační soubor komponent agentů (strana 69).

5.6 Použité technologie

V této kapitole se budeme zabývat použitými technologiemi a důvody pro zvolení konkrétní technologie.

5.6.1 Programovací jazyk C++

Programovací jazyk C++ je objektově orientovaný imperativní jazyk. Byl vyvinut v Bellových laboratořích AT&T. Jeho vývoj započal v 80. letech a první oficiální norma byla přijata v roce 1998. Jazyk C++ je rozšířením jazyka C o objekty a mnohé další funkce.

Jazyk C++ obsahuje mnoho vymožeností moderních jazyků[9], ale přesto umožňuje tvorbu velice rychlých aplikací (což je značný problém s vyššími programovacími jazyky).

Nevýhodou je, že se od programátora očekává jistá znalost jazyka a architektury (ve složitějších programech se prakticky nelze vyhnout ukazatelům a dynamické alokaci paměti). Tvorba aplikace také zabere více času, než při použití skriptovacích jazyků (například Python).

Jazyk C++ byl vybrán z důvodu jeho rychlosti a zároveň snadnosti použití (oproti jazyku C nebo Assembleru). Jelikož aplikace má být schopna co nejrychleji simulovat virtuální svět, požadavek na výkon je kritický.

Do budoucna by bylo možné některé výpočetně náročné části aplikace přesunout na grafický akcelerátor za použití jazyka

9. Především v posledních verzích standartu C++11x.

OpenCL či CUDA (například velké neuronové sítě velmi dobře škálují na SIMD architekturách[10], stejně tak i výpočty fyziky).

5.6.2 SDL

SDL – Simple DirectMedia Layer je multiplatformní multimediální knihovna pro práci se zvukem, 3D grafikou pomocí OpenGL a zpracování vstupů ze vstupních zařízení (klávesnice, myš, joystick). SDL podporuje operační systémy Linux, Windows, MacOS, FreeBSD a další. Knihovna SDL je napsána v jazyce C, ale lze jej použít v mnoha jazycích. Knihovna SDL je dostupná pod licencí GNU LGPL version 2.

SDL bylo zvoleno kvůli ulehčení vývoje pro více platforem. Knihovna SDL je použita pro klientskou část programu, která zobrazuje uživateli aktuální stav virtuálního světa a umožňuje mu tak pozorovat chování agentů v reálném čase. Požadavky na grafickou kvalitu byly pouze základní, proto je v klientské aplikaci použita pouze jednoduchá 2D grafika využívající textury.

5.6.3 ENet

ENet[11] je velice lehká knihovna pro síťovou komunikaci. Sestává pouze z několika zdrojových souborů a je velice jednoduché včlenit ji do projektu. Knihovna ENet je postavena nad protokolem UDP, do kterého přidává několik vlastností TCP (spolehlivost, zaručenost pořadí příjmu paketů).

Vývojář může kombinovat výhody obou protokolů podle dat, která potřebuje posílat mezi jednotlivými instancemi programu (například některá data není třeba poslat v zaručeném pořadí, ale zato je třeba je doručit co nejdříve, u jiných to může být zase naopak).

Knihovna ENet je uvolněna pod licencí MIT.

10. Single instruction, multiple data.
11. http://enet.bespin.org/

5.6.4 CEGUI

Crazy Eddie's GUI System [12] je otevřená knihovna, umožňující tvorbu GUI. Navzdory poněkud netypickému názvu se jedná o kvalitní knihovnu poskytující celou škálu dobře zpracovaných widgetů.

CEGUI je multiplatformní projekt s renderery do celé řady grafických enginů a knihoven: OpenGL, DirectX 9, DirectX 10, DirectX 11, Irrlicht, Ogre3D, Crystal Space a Open Scene Graph.

Knihovna je napsána v jazyce C++ a plně využívá možností objektově orientovaného jazyka. Umožňuje GUI tvořit buď přímo v programu nebo pomocí XML konfiguračních souborů. K dispozici je také WYSIWYG editor, který sice nelze považovat za vzor uživatelské příjemnosti, ale pracovat v něm lze.

V případě problémů s vývojem je tady CEGUI komunita, která víceméně ochotně poradí s mnoha problémy. Knihovna CEGUI je uvolněna pod licencí MIT.

5.6.5 RapidXml

RapidXml je parser XML psaný v jazyce C++. Je naprogramován s důrazem na rychlost (parsování se rychlostí blíží funkci strlen[13] na stejných datech). Kromě toho je parser jednoduchý na používání a na zapracování do zdrojového kódu projektu, což se nabízí s ohledem na jeho velikost a velmi mírnou MIT licenci.

Kromě parsování umožňuje RapidXML také úpravu stromu a jeho zpětné uložení do souboru.

5.6.6 jqPlot

Vzhledem k požadavku na přehledné prostředí k zobrazení stavu prostředí bylo třeba naleznout způsob, jak uživateli přehledně prezentovat v jeho webovém prohlížeči pokroky umělé inteligence. Jako vhodný nástroj byl vybrán jqPlot.

jqPlot[14] je javascriptový framework pro vykreslování grafů v prohlížeči. Vzhledově je víc než vyhovující a problémy s výkonem

12. http://www.cegui.org.uk/
13. Funkce měřící délku řetězce.
14. http://www.jqplot.com/

při jeho používání nebyly zaznamenány. Framework je rozšiřitelný pomocí pluginů a oplývá množstvím nastavení. jqPlot je je k dispozici jak pod licencí MIT, tak i GPLv2.

5.6.7 Ostatní

Další použité nástroje, které stojí za zmínění:

- gcc – kompilátor.
- git – verzovací nástroj.
- Linux Mint (různé verze) – operační systém, na kterém převážně probíhal vývoj.
- Sublime text 2 - vývojové prostředí.
- Google-code-prettify - javascriptový modul pro zvýraznění syntaxe (použit pro zobrazení XML částí konfiguračních souborů v HTML statistice).
- JQuery.

Kapitola 6

Testování

Pro účely testování bylo navrhnuto několik testovacích konfigurací, kterým se věnují následující kapitoly. Nejdříve jsou popsány výsledky jednoduchých testovacích konfigurací, postupně se pak přechází ke složitějším.

Testování probíhalo na několika počítačích s různou hardwarovou konfigurací pod operačním systémem Linux Mint.

Součástí popisu každého experimentu je také použitá funkce zdatnosti, ta používá proměnné a funkce popsané v tabulce A.1 na straně 67.

Pokud není popsáno jinak, experiment se odehrává ve virtuálním světě o tvaru čtverce se stranou 280 metrů obsahujícím 600 pevných objektů o poloměru 1.6 metrů (pro testování schopnosti agentů vyhýbat se překážkám, agent přežije náraz pouze ve velmi malé rychlosti). Doba trvání jedné generace je ohraničena 60 sekundami.

Samotní agenti mají poloměr kolize $1m$, maximální rychlost $7ms^{-1}$ a zrychlení dopředu a do stran $2ms^{-2}$. Agenti jsou nakonfigurováni tak, aby pro ně platila setrvačnost pohybu. Pohybují se tedy podobně jako na ledě. Tím je dosáhnuto zvýšené složitosti přežití ve virtuálním světě (agent musí počítat s tím, že se nepohybuje vždy směrem, kterým se dívá).

Veškerá statistická data zde uvedená včetně příslušných konfigurací lze najít v elektronické příloze pod příslušným označujícím kódem. Tento kód je vždy uveden v popisu grafu. Některé experimenty byly prováděny vícekrát, výsledky těchto testů jsou také součástí elektronické přílohy. Výsledky jsou uloženy ve formě HTML statistik obsahujících kromě zdatnosti i další měřené veličiny.

6.1 Jednoduché experimenty

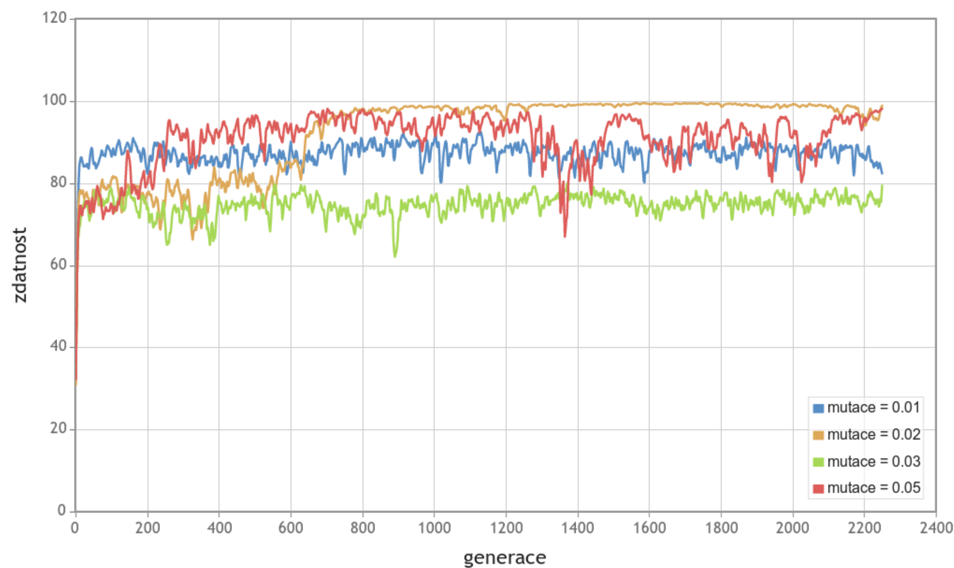
Testy obsažené v této podkapitole byly navrhnuty pro otestování základní funkcionality programu.

6.1.1 Test zastavení

Cílem agentů v tomto testu je naučit se zůstat stát na místě. Funkce zdatnosti je definována takto:

$$fitness = 100 - distance$$

Výsledek experimentu lze vidět v grafu na obrázku 6.1. V tomto experimentu evoluce velmi rychle dosáhla dobré strategie.



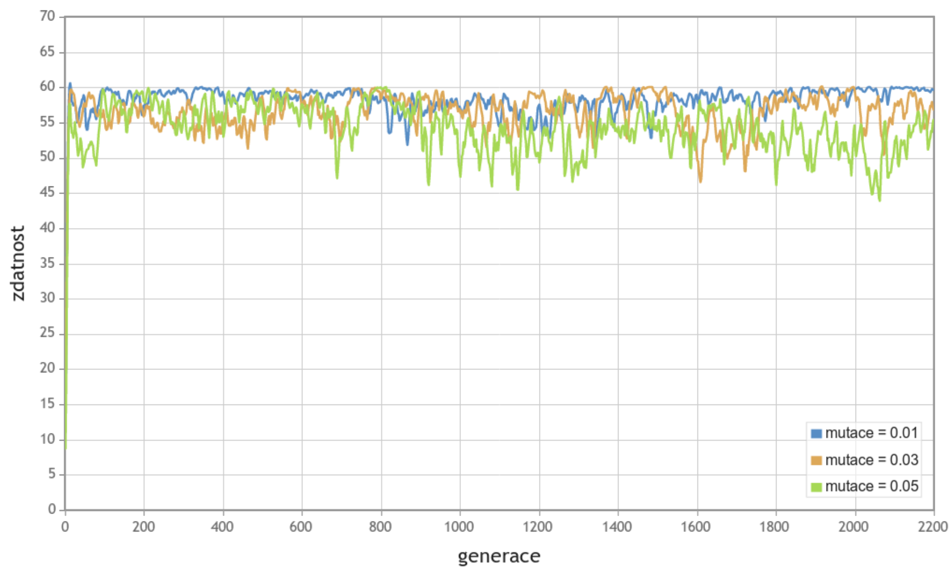
Obrázek 6.1: Experiment S-3-htan-fl-1p – test zastavení.

6.1.2 Test přežití

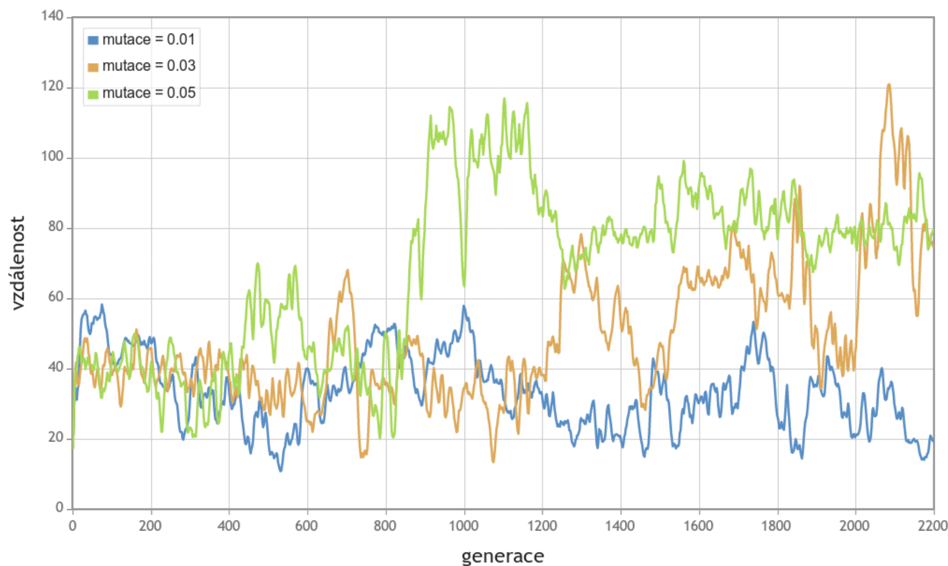
Cílem agentů v tomto testu je přežít ve virtuálním světě co nejdéle. Průběh zdatnosti v závislosti na generacích lze vidět na obrázku 6.2. Funkce zdatnosti je definována takto:

$$fitness = time$$

Na obrázku 6.3 lze vidět, že vzdálenost, kterou agenti během života urazí, se drží v nízkých hodnotách (během svého života agent může urazit přibližně $7 * 60 = 400m$). Také ale vzdálenost není příliš nízká, protože to funkce zdatnosti nevyžaduje.



Obrázek 6.2: Experiment L-3-htan-fl-1p – test přežití.



Obrázek 6.3: Experiment L-3-htan-fl-1p – test přežití, graf uražené vzdálenosti.

6.2 Netriviální experimenty

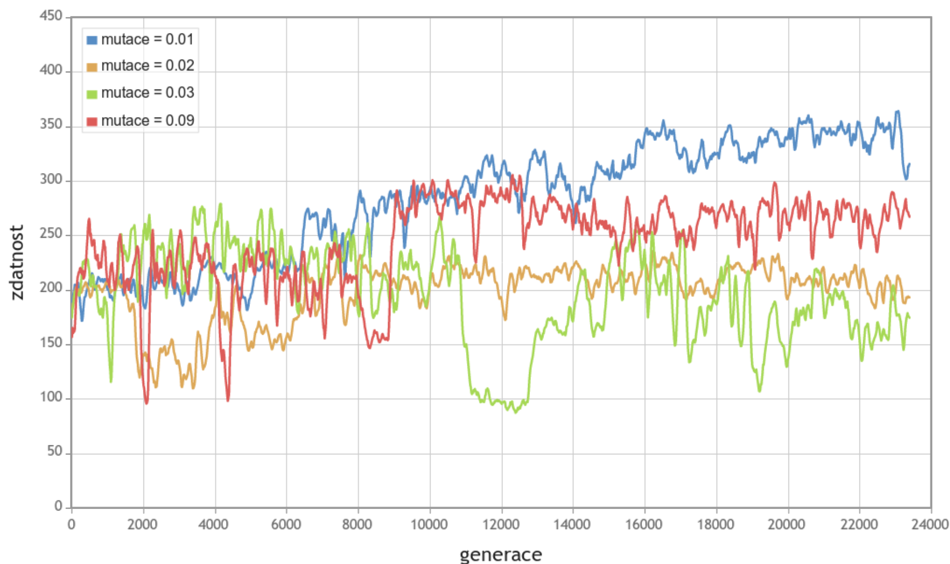
Tato podkapitola obsahuje složitější experimenty. Pro nalezení dobrých řešení už genetické algoritmy vyžadují řádově vyšší počet generací a tím i času. Provádění experimentů z této kapitoly si vyžádalo desítky hodin výpočetního času.

6.2.1 Test vzdálenosti

Tento experiment byl původně navržen pro naučení agentů pohybovat se rozumným způsobem mezi překážkami. Cílem bylo, aby se pohybovali vpřed a naučili se vyhýbat překážkám. Funkce zdatnosti byla definována takto:

$$fitness = distance + (3 * time)$$

Výsledek lze vidět na grafu 6.4. Na první pohled sice vše vypadá v pořádku, ale po vizuální kontrole chování agentů se ukázalo, že evoluce si našla jednodušší cestu, jak splnit úkol. Agenti se naučili kroužit v plné rychlosti kolem jednoho bodu. Tím, že kroužili



Obrázek 6.4: Experiment D-3-htan-fl-1p - test vzdálenosti.

v bezpečné vzdálenosti od překážek, dosáhli vysoké doby života a velké uražené vzdálenosti.

Je tedy důležité zmínit, že v případě evolučních algoritmů je třeba věnovat více času tvorbě funkce zdatnosti a virtuálního světa.

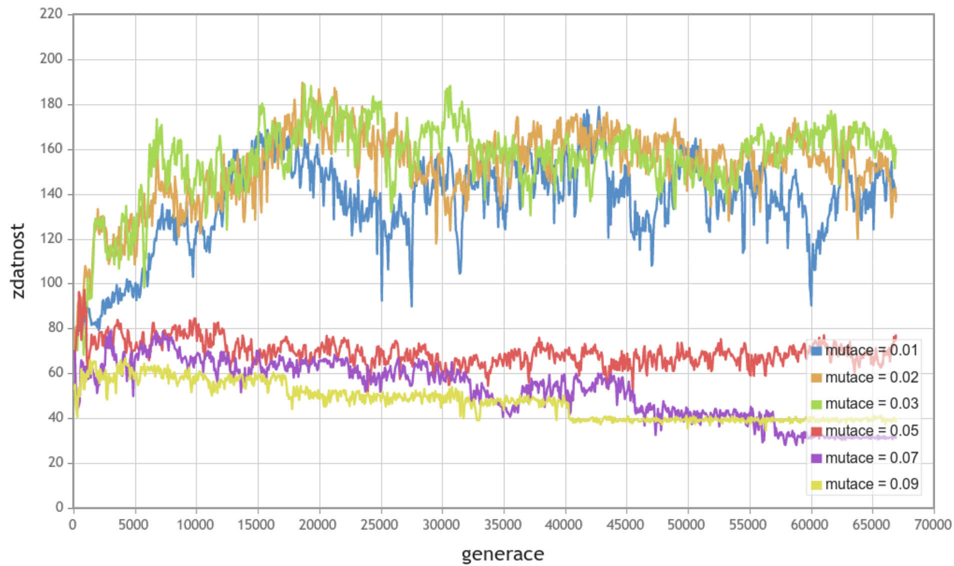
6.2.2 Test komplexní vzdálenosti

Kvůli problémům v předchozím testu byla v programu implementována možnost měřit vzdálenost od startovacího bodu. Použitím tohoto parametru ve funkci zdatnosti lze navést evoluci ke správnému výsledku. Grafy 6.5 a 6.6 byly počítány s touto funkcí zdatnosti:

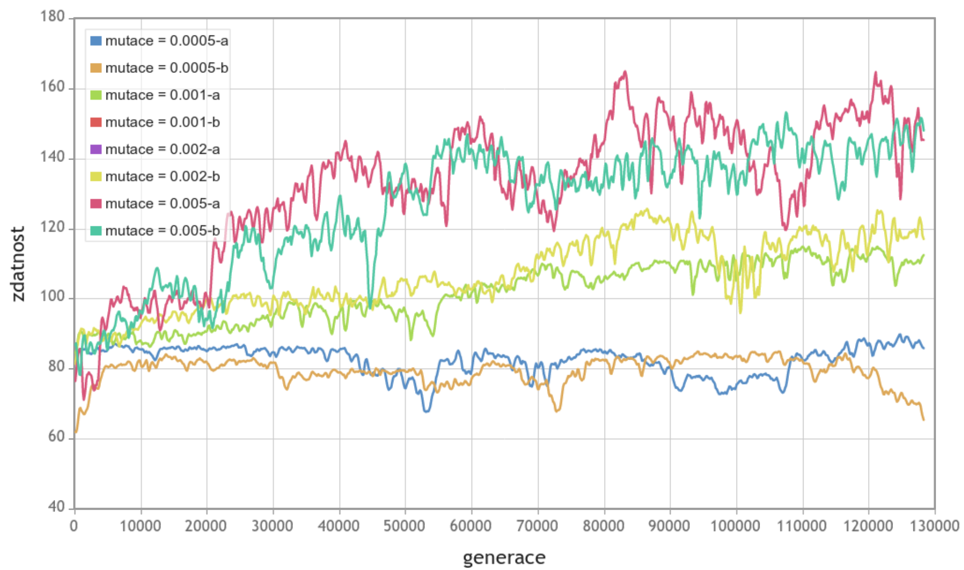
$$fitness = speed * 2 + maxDistance + time * 0.5 + distance * 0.1$$

Pomocí této funkce zdatnosti se agenti už pohybují požadovaným způsobem. V tomto experimentu byl také ve větší míře testován vliv velikosti mutace[1] na rychlost, s jakou evoluce nalezne dobrou strategii.

1. Mutace se aplikuje po křížení, viz strana 14.



Obrázek 6.5: Experiment G-3-htan-fl-1p s velkými hodnotami mutací.



Obrázek 6.6: Experiment G-3-htan-fl-1p s malými hodnotami mutací.

Graf 6.5 měří výkony umělých inteligencí s vyššími mutacemi a graf 6.6 s menšími. Na obrázku 6.5 můžeme vidět, že mutace nad 0.5 již neprodukují dostatečně stabilní DNA a evoluce selhává. Naopak na obrázku 6.6 lze vidět, že hodnoty mutace nižší než 0.001 nedávají dobré výsledky v rozumné době.

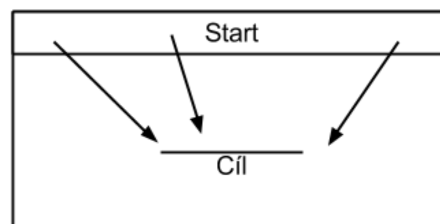
Experiment s malými mutacemi trval $8 * 55$ hodin, v každém virtuálním světě ale uběhlo 89 dní. Zatímco v experimentech s vyššími hodnotami mutací vznikají dobré strategie[2] už kolem generace 15 000, v případě menších mutací se dostáváme na rozumnou strategii až po generaci 80 000.

6.3 Experiment se zadaným cílem

V tomto experimentu je svět prázdný (bez překážek) a agenti jsou při startu rozmístěni podél severní stěny. Je jim zadán cíl dostat se do blízkosti přímky uprostřed mapy. Konfigurace světa je znázorněna na obrázku 6.7. Funkce zdatnosti byla nastavena takto:

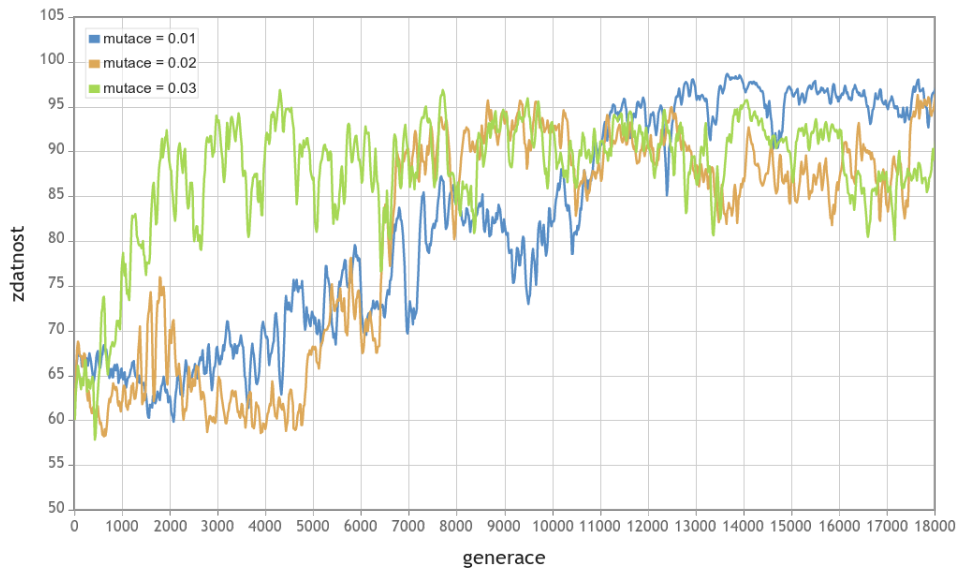
$$fitness = 100 - deathTargetDistance$$

Na výsledném grafu (obrázek 6.8) lze vidět postupné snižování vzdálenosti od cíle. Tato vzdálenost je vypočtena v okamžik zničení agenta[3].

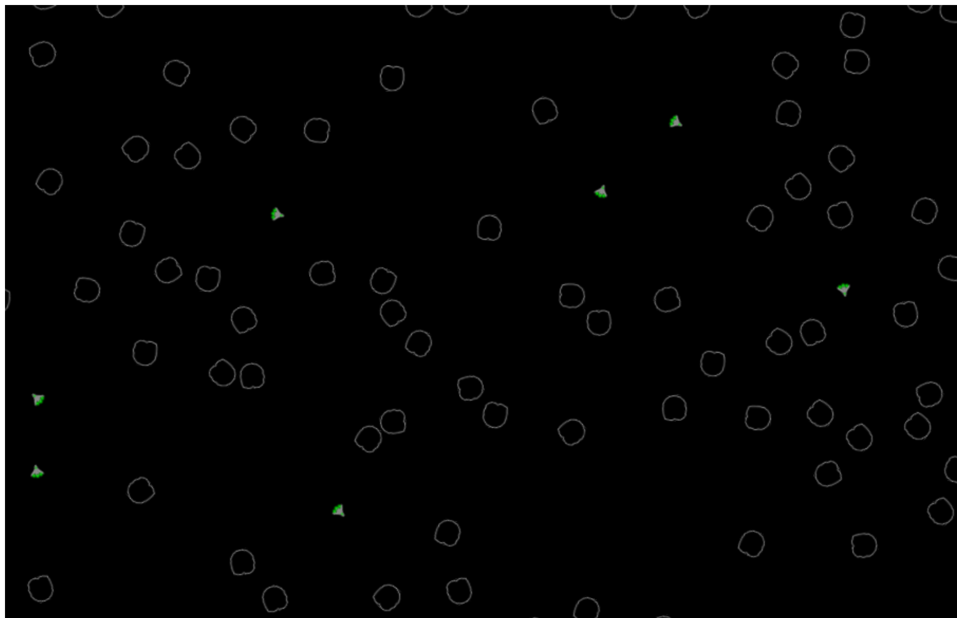


Obrázek 6.7: Konfigurace virtuálního světa.

2. V tomto případě je strategie dobrá, pokud je její hodnota zdatnosti vyšší než 160.
3. Nastává po nárazu nebo vypršení časového limitu pro současnou generaci.



Obrázek 6.8: Experiment E-3-htan-fl-1p – test s cílovou oblastí.



Obrázek 6.9: Ukázka virtuálního prostředí s překážkami.

Kapitola 7

Závěr

Cílem této diplomové práce bylo implementovat obecný systém pro popis charakteristik prostředí v multiagentním systému. Byla požadována možnost konfigurace cílových funkcí agentů (funkce zdatnosti). Zadání prostředí mělo probíhat pomocí konfiguračních souborů.

Návrhem a implementací systému bylo těchto cílů dosaženo. Programová část sestává ze tří částí. Serveru, který se stará o hlavní simulaci. Klienta, který slouží v současné době pro pozorování experimentu. Ten je ale také připraven pro převzetí části zátěže od serveru (výpočet umělé inteligence). Poslední komponentou systému je registr, který udržuje seznam běžících serverů a poskytuje jej klientům, kteří si jej vyžádají.

Programová část má téměř dvacet tisíc[1] řádků napsaných v jazyce C++. Programová část je implementována s důrazem na budoucí rozšiřitelnost. Díky častému využívání polymorfismu[Virius(2006)] lze případné nově vytvořené odvozené třídy jednoduše začlenit do programu.

Součástí práce je také příklad netriviální umělé inteligence v podobě konfigurovatelné vícevrstvé dopředné neuronové sítě. Jejím testováním se zabývá kapitola 6.

Dalším cílem bylo implementovat přehledné zobrazení výsledků experimentu. Tohoto cíle bylo dosaženo pomocí generované HTML stránky. Ta umožňuje přehledně porovnávat různé statistiky proběhlých experimentů.

---

1. Převzatý zdrojový kód nebyl do této sumy započten.

7.1 Možnost rozšíření

Díky modulárnímu a objektovému designu se podařilo vytvořit rozšiřitelný program. Nabízí se tedy mnoho způsobů, jak projekt dále rozvíjet.

Asi nejvýznamnější možností je implementace nových typů umělé inteligence. Nabízí se implementace neuronových sítí schopných se učit za svého života podobně jako v Polyworldu (viz strana 3). Nebo cyklických neuronových sítí, jejichž výhodou by mohla být schopnost pamatovat si údaje v průběhu života.

Dobrým námětem na rozšíření je také implementace pokročilých metod mutací, které by měnily sílu mutace například v závislosti na času, hodnotě zdatnosti nebo změně hodnoty zdatnosti (obdoba simulovaného žíhání[Ingber(1993)]).

Také si lze představit například evoluci pseudokódu[Ofria and Wilke(2004)]. Ta by mohla být implementována tak, že třída umělé inteligence by simulovala virtuální počítač, na kterém by se spouštěl evolučně vyvinutý pseudokód. Emulovaný počítač by kromě základních vstupů a výstupů umělé inteligence mohl obsahovat například registry, paměť RAM a ROM.

Dále je možné vytvářet nové komponenty pro plemena a tím rozšířit možnosti řízení evoluce. Bylo by možné i implementovat podporu pro podobný systém evoluce, jaký je použitý v Polyworldu. Také je možné přidat další možnosti ukládání získaných dat.

Velké možnosti mají také komponenty agentů, pomocí nich lze dát agentům různé schopnosti a změnit způsob, jakým se pohybují. Pomocí změny v komponentách starajících se o pohyb agenta ve virtuálním světě, lze docílit agenta využívajícího Ackermannova řízení (simulace automobilu).

Bylo by zajisté zajímavé vytvořit pro agenty složitější svět, kde by se již využily i složité umělé inteligence. Přidat agentům možnost mezi sebou interagovat, předávat energii[2]

Rozšíření je možné i v síťovém kódu. Například implementací kompresního algoritmu by se snížilo využití pásma přenosem dat. Pro velké virtuální světy by bylo vhodné také implemento-

2. V tomto případě by bylo potřeba vytvořit komponentu pro agenta, která by starala o předávaní a uchování energie.

vat odesílání jen části světa klientovi (klient nemusí znát polohu vzdálených objektů), popřípadě snížit frekvenci obnovení informací o velmi vzdálených objektech.

Literatura

[Dawkins(1986)] Richard Dawkins. *The blind watchmaker, Harlow*. Longman, 1986. ISBN 978-0393315707.

[Ingber(1993)] Lester Ingber. Simulated annealing: Practice versus theory. *Mathematical and computer modelling*, 18(11):29–57, 1993.

[Jaroslav(2008)] Flegr Jaroslav. *Zamrzlá evoluce*. Nakladatelství akademia Praha, 2008. ISBN 978-80-200-1526-6.

[Kempter et al.(1999)Kempter, Gerstner, and Van Hemmen] Richard Kempter, Wulfram Gerstner, and J Leo Van Hemmen. Hebbian learning and spiking neurons. *Physical Review E*, 59(4):4498, 1999.

[LeCun et al.(1998)LeCun, Bottou, Orr, and Müller] Yann LeCun, Leon Bottou, Genevieve Orr, and Klaus Müller. *Efficient BackProp*. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1998. ISBN 978-3-540-65311-0.

[Ofria and Wilke(2004)] Charles Ofria and Claus O Wilke. Avida: A software platform for research in computational evolutionary biology. *Artificial life*, 10(2):191–229, 2004.

[Pedregosa et al.(2011)] Pedregosa et al. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12: 2825–2830, 2011.

[Sims(1994a)] Karl Sims. Evolved virtual creatures. 1994a. `http://www.karlsims.com/evolved-virtual-creatures.html`.

[Sims(1994b)] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 1994b.

[Virius(2006)] Miroslav Virius. *Jazyky C a C++: Kompletní kapesní průvodce programátora*. Grada Publishing, 2006. ISBN 80-247-1494-9.

[Yaeger(1994)] Larry Yaeger. Poly world: Life in a new context. *Proc. Artificial Life*, 3:263–263, 1994. `http://shinyverse.org/larryy/Polyworld.html`.

[Yaeger(2010)] Larry Yaeger. Polyworld. 2010. `http://shinyverse.org/larryy/Polyworld.html`.

Příloha A

Popis konfiguračních souborů

A.1 Konfigurace světa

Na úryvku A.1 je příklad konfigurace virtuálního světa. Při spuštění je serveru předána cesta k takovému konfiguračnímu souboru, server jej pak zpracuje a vytvoří definovaný virtuální svět.

Rozpis částí XML konfigurace:

- Tag server – zde se nachází konfigurace pro vytvoření serveru. Atribut public udává, jestli se server bude hlásit na registr (viz. strana 19). Do atributu address lze zadat jak IP adresa, tak url adresa. Tuto adresu pak nabízí registr klientům pro připojení. Fullspeed udává, jestli server začne se zapnutou maximální rychlostí[1]. Tag description obsahuje popis serveru/experimentu.
- Tag world obsahuje:
  - Rozměry světa a atribut grid udává rozměr akcelerační mřížky (viz strana 41).
  - Tagy typu objekt se používají pro zaplnění virtuálního světa objekty. Jejich atributy udávají počet, polohu a další fyzické vlastnosti objektů. Atribut static udává, že objekt je statický – tj. nemůže být posunut nebo zničen. Tagů typu object může být více než jeden.
- V tagu breeds se nachází popis plemen. Každé plemeno je zapouzdřeno v tagu breed, ten obsahuje název plemene, popis komponent a textový popis plemene.

1. Server není synchronizován s reálným časem. Typicky běží rychleji.

Úryvek A.1: Příklad konfiguračního souboru experimentu

```
<?xml version="1.0" encoding="UTF-8"?>
<server name="Název experimentu" port="20024" public="true"
    verze="0.06" address="adresa.cz" fullspeed="true">
  <description>Popis experimentu.</description>
</server>
<world>
  <size x="140.0" y="140.0" grid="20"></size>
  <object type="asteroid" num="600" size="1.6" random_size="
      0.0" weight="100000.0f" hp="500000.0" static="true"> <
      /object>
</world>
<breeds>
  <breed name="Název plemene">
    <kmp type="onegen" num="30" maxtime="60.0" ai="mlp1"
        color="2"></kmp>
    <kmp type="complex_fitness" equation="speed*2+
        maxDistance+time*0.5+distance*0.1"> </kmp>
    <kmp type="garbage_collector" treshold="0.9"> </kmp>
    <kmp type="save_good_generations" count="3" minGenDist="
        1"> </kmp>
    <kmp type="one_parent_xbest" mutace="0.05" relative="0.3
        " setup="ai_01_nf">
      <mlp activation="2">
        <row count="6"/>
        <row count="7"/>
        <row count="7"/>
      </mlp>
    </kmp>
    <kmp type="simple_end" gen="10000"></kmp>
    <kmp type="simple_HTML_statistics" num="1" path="/cesta/
        k/html/statistice"></kmp>
    <kmp type="log_out"></kmp>
    <kmp type="log_csv"> </kmp>
    <!-- komentář -->
    <kmp type="autoload" save="true" load="true"></kmp>
    <description>Popis plemene.</description>
  </breed>
</breeds>
```

A.2 Konfigurace plemen

Popis některých zajímavých komponent[2] z úryvku A.1 (podrobnější popis funkcí komponent viz strana 36):

- onegen – komponenta starající se o generace. Zde je nastavena velikost populace 30 jedinců a čas trvání jedné generace je nastaven na 60 sekund.
- complex_fitness – je komponenta pro výpočet funkce zdatnosti, vzorec pro výpočet zdatnosti je zadán v textové formě v atributu equation.
- simple_end – ukončí experiment po 10000 generacích.
- one_parent_xbest – vybírá pro nového agenta jednoho rodiče z x nejlepších (více o způsobech výběru rodičů lze najít na straně 37).
  - Atribut setup určuje název konfigurace agenta (více na straně A.5).
  - Součástí tohoto tagu je také tag mlp. Je to z toho důvodu, že první generace agentů potřebuje mít přístup ke své konfiguraci. Noví agenti dostávají ukazatel do struktury XML právě na tag komponenty starající se o křížení. V něm si naleznou tag, který je pro ně určen (tímto se zamezí načítání konfigurace určené pro jiný typ inteligence).
  - V tagu mlp je definována aktivační funkce neuronové sítě a každý tag row reprezentuje jednu vnitřní (skrytou) vrstvu. První vrstva v konfiguraci odpovídá vrstvě neuronové sítě nejbližší vstupům umělé inteligence. Atributem je počet neuronů ve vrstvě.

U umělé inteligence (v případě typu mlp1 v tagu mlp) lze specifikovat, na které vstupy a výstupy bude reagovat, na výběr jsou v současné implementaci tyto:

---

2. Komponenty jsou definovány XML tagy s názvem „kmp". Konkrétní typ komponenty pak určuje atribut type.

- allowSensors – senzory vzdálenosti.
- allowPain – senzor bolesti.
- allowSpeed – senzor rychlosti, velikost pohybového vektoru.
- allowAngleDifference – rozdíl mezi směrem pohledu a směrem pohybu. Agent je díky této informaci schopen úspěšně přežívat i ve světě se setrvačností.
- allowLife – hodnota života agenta.
- allowTarget – rozdíl úhlu pohledu a úhlu směrem k cíli a normalizovaná vzdálenost.
- allowEngine – povolí umělé inteligenci ovládat pohyb vpřed a vzad.
- allowTurn – povolí umělé inteligenci ovládat otáčení agenta.
- allowStrafe – povolí umělé inteligenci ovládat pohyb do strany u agenta.
- allowButton0-9 – slouží pro aktivaci komponent (například).

A.2.1 Komplexní funkce zdatnosti

Tabulka A.1 popisuje proměnné a funkce, které je možno využít při zadávání funkce zatnosti pomocí komponenty komplexní funkce zdatnosti.

A.3 Konfigurace modelů

V konfiguračním souboru modelů se nachází definice použitelných modelů. V současné době lze použít pouze renderer využívající SDL. Formát souboru s konfigurací modelů počítá s možností rozšíření, proto nastavení modelu zapouzdřeno v tagu SDL.

Při přidání jiného zobrazovacího engine bude možné snadně rozšířit i konfiguraci bez obětování zpětné kompatibility – u každého

Popis	Příkaz	Zkratka
Matematické funkce		
Lineární náhodné číslo	random11	R()
Náhodné číslo 0-x	random(x)	r(x)
Vrátí větší hodnotu	max(a,b)	M(a,b)
Vrátí menší hodnotu	min(a,b)	m(a,b)
Vrátí log(a)	log(a)	l(a)
Vrátí sin(a)	sin(a)	s(a)
Vrátí cos(a)	cos(a)	c(a)
Statistické údaje		
Průměrná rychlost	averageSpeed,speed	a()
Uražená vzdálenost	dist,distance	t()
Doba života	time	T()
Skóre	score	s()
Počet zabití	kills	k()
Minimální vzdálenost k cíli	minTargetDistance	e()
Vzdálenost od cíle při smrti	deathTargetDistance	E()
Počet úmrtí	deaths	K()
Udělené poškození jiným	dmg,damage	g()
Maximální vzdálenost od startu	maxDistance	d()
Vzdálenost od startu při smrti	deathDistance	D()
Počet vystřelených projektilů	shooted	p()

Tabulka A.1: Popis proměnných a funkcí pro komplexní funkce zdatnosti

modelu budou dvě struktury, jedna s tagem SDL, druhá s tagem jiným.

Na úryvku A.2 lze vidět konfigurační soubor modelů s dvěma definovanými modely. Parametr trans určuje použitý způsob při vykreslování objektu.

A.4 Konfigurace agentů

Konfigurace agentů obsahuje postup pro složení agenta. Každá konfigurace má svůj tag setup obsahující jméno, krátké jméno (slouží

Úryvek A.2: Příklad konfiguračního souboru modelů

```
<?xml version="1.0" encoding="UTF-8"?>
<models>
  <model name="Earth">
    <SDL>
      <texture file="earth.bmp" trans="2"/>
    </SDL>
  </model>
  <model name="Asteroid">
    <SDL>
      <texture file="asteroid2.bmp" trans="2"/>
    </SDL>
  </model>
</models>
```

Úryvek A.3: Příklad konfigurace agenta

```
<?xml version="1.0" encoding="UTF-8"?>
<shipssetup>
  <setup name="Název␣konfigurace" short="zkracenýNázev" body
     ="zkracený␣název␣fyzickeho␣těla">
    <description>popis</description>
    <kmp num="0" short="TestEngine0"/>
    <kmp num="1" short="Canon1"/>
  </setup>
</shipssetup>
```

pro rychlé vyhledání při tvorbě agenta) a krátký název odkazující na fyzickou schránku agenta (více na straně 69).

Konfigurace dále obsahuje seznam komponent. Na ty jsou použity tagy kmp. Tyto tagy obsahují index na slot a krátký název odkazující na komponentu.

Tento systém se může zdát trochu těžkopádný a složitý, ale splňuje požadavek na konfigurovatelnost a možnou budoucí rozšiřitelnost.

Příkladem takové konfigurace je úryvek A.3.

A.5 Konfigurace fyzické schránky agentů

Konfigurace fyzické schránky agentů se skládá z nastavení fyzikálních vlastností (váha, život, velikost) a seznamu slotů dostupných pro umístění komponent.

U každého slotu je možné nastavit relativní pozici oproti středu objektu a směrový vektor komponenty. Ten se používá při vypouštění projektilů.

Úryvek A.4: Příklad konfiguračního souboru fyzické schránky agenta

```
<?xml version="1.0" encoding="UTF-8"?>
<lodvs>
  <lodv name="Název"
        short="Krátký␣název"
        hp="5.0"
        type="ship"
        model="název␣modelu"
        size="1.0" >
    <description>Defaultni agent.</description>
    <slot positiony="1.0" />
    <slot positiony="2.0" vectorx="0.0" vectory="1.0" button
       ="1"/>
    <slot positiony="2.0" vectorx="0.0" vectory="1.0" button
       ="2"/>
    <slot positiony="0.0" />
  </lodv>
</lodvs>
```

A.6 Konfigurace komponent agentů

Poslední konfigurační soubor se týká komponent. V současné implementaci jsou dostupné dva typy: pohon a zbraň. V úryvku A.5 je ukázka obojího.

Vytváření nové komponenty probíhá tak, že podle atributu **type** se určí typ komponenty. Vytvoří se nová instance a té je poslán odkaz na XML strukturu komponenty. Z té se nová komponenta inicializuje. Díky tomu jsou při vytváření nové komponenty velké možnosti konfigurace.

Tento postup by byl velice zdlouhavý, pokud by se měla XML konfigurace parsovat při každém vytváření nového agenta, všechny komponenty se proto načtou při startu programu a pro agenty se vytvářejí pouze kopie už načtených komponent.

U pohonu je zajímavým atributem **newtonfalse**. Tímto atributem určujeme, jak moc bude agent „postižen“ setrvačností pohybu. Pokud je zde číslo větší, než 0, tak je agentův pohybový vektor upravován tak, aby byl ve větším souladu s jeho vektorem pohledu. Agent s vysokým **newtonfalse** se tedy nemusí starat o problém, že se nemusí nutně pohybovat směrem, kterým se dívá.

Úryvek A.5: Příklad konfiguračního souboru komponent agentů

```
<?xml version="1.0" encoding="UTF-8"?>
<kmps>
  <kmp name="AIEngine1"
       short="AIEngine1"
       type="simple_engine"
       power="2.0"
       backpower="2.0"
       maxspeed="7.0"
       sidepower="1.5"
       rotspeed="1.2"
       rotpower="2.0"
       newtonfalse="3.0"
       >
     <description>Popis</description>
  </kmp>
  <kmp name="Canon1"
       short="Canon1"
       type="simple_weapon"

       speed="15.0"
       ttl="10.0"
       reload_time="2.0"
       accel="0.0"
       time_accel="0.0"
       >

    <projectile size="0.3f"
       weight="0.25f"
       hp="25.0f"
       firmnesss="0.0f"
       firmnessp="0.1f"
```

```
          typ="projectile"
          model="GunFire">
        <description>Popis</description>
    </kmp>
</kmps>
```

Příloha B

Elektronické přílohy

Součástí práce jsou i následující elektronické přílohy:

- Zdrojové soubory.
- Kompletní výsledky provedených testů.