Mendelova univerzita v Brně
Provozně ekonomická fakulta

Řízení autonomního agenta pomocí neuroevoluce

Diplomová práce

Vedoucí práce:
Ing. Jiří Lýsek, Ph.D.

Bc. Martin Hnátek

Brno, 2018

Rád bych zde poděkoval svému vedoucímu Ing. Jiří Lýskovi, Ph.D. za jeho cenné rady a čas, který mi věnoval při řešení této práce.

Čestné prohlášení

Prohlašuji, že jsem práci: **Řízení autonomního agenta pomocí neuroevoluce** vypracoval samostatně a veškeré použité prameny a informace uvádím v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 29. prosince 2018 .....

Abstract

Autonomous agent control using neuroevolution

Thesis describe theory behind neuroevolution. Then it describes both design and creation of simulated environment for autonomous agent and its training with library Neataptic in environemnts with various difficulity. Thesis also describes process of designing frontend for visualization of results and backend for faster training of agents. At the end it describes resulting agents and proposes enhancements to existing solution.

Key words:
NEAT, AI, MachineLearning, Agent, Javascript

Abstrakt

Řízení autonomního agenta pomocí neuroevoluce

Tato práce představuje teoretický základ neuroevoluce. Zahrnuje také návrh tvorbou simulačního prostředí, pro autonomního agenta a jeho natrénování s pomocí knihovny Neataptic v různě obtížných podmínkách. Popisuje také tvorbu frontendu pro snadnou vizualizaci a serverové části pro rychlé učení. Na závěr se zabývá vyhodnocením natrénovaných agentů a návrhem možných zlepšení.

Klíčová slova:
NEAT, AI, MachineLearning, Agent, Javascript

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod a cíl práce

1.1 Úvod do problematiky

S růstem výpočetního výkonu a rozvojem **GPUGPU** (paralelizace výpočtů na grafické kartě) se neuronové sítě ukázaly jako mocný nástroj pro řešení složitých problémů na které standardní metody umělé inteligence nestačily.

Další oblastí umělé inteligence, které nárust masivní paralelizace prospěl, jsou metaheuristiky, které mohou s nárůstem výpočetního výkonu v rozmném čase pokrýt stále větší stavový prostor a jsou tedy schopné rychle řešit stále složitější problémy.

Neuroevoluce propojuje oba přístupy a využívá je ke generování topologii neuronových sítí a nastavení jejích vah. Výsledkem je neuronová síť, jejiž architektura lépe popisuje daný problém a lze jí aplikovat i na problémy na které by klasické neuronové sítě aplikovat nešly. Jedná se například o problémy, u kterých je těžké získat trénovací data a nelze tedy neuronovou síť natrénovat klasickými metodami, jako je například gradientní sestup.

1.2 Cíl práce

Cílem této práce je aplikovat neuroevoluci pro učení autonomního agenta. Toto zahrnuje návrh a vytvoření vhodného simulačního prostředí. Poté na simulovaném prostředí navrhnout a provést sérii experimentů, které slouží k zjištění limitů učících schopností algoritmu NEAT.

2 Literalní rešerše

Teoretická část práce seznamuje čtenáře s popisem algoritmu NEAT a algoritmů ze kterých vychází.

2.1 Neuronové sítě

Neuronové sítě jsou model strojového učení, který je volně založený na principu zvířecího mozku (PATTERSON, Josh. 2017, s. 41).

Neuron

Neuron je základní výpočetní jednotka neuronových sítí, která je definovaná jako suma všech jejích vstupů a aplikace aktivační funkce.

$$g(\Sigma_{i=0}^{N}\theta_i \cdot x_i + b)$$

Kde:

1. x_i - i-tý vstup do neuronu
2. g - aktivační funkce
3. θ - skrytá váha pro daný vstup
4. b - bias neuronu

Na tomto místě je vhodné podotknout, že výše uvedená notace vychází z přednášek Andrew Y. Ng a je jen jednou z mnoha běžně uváděných.

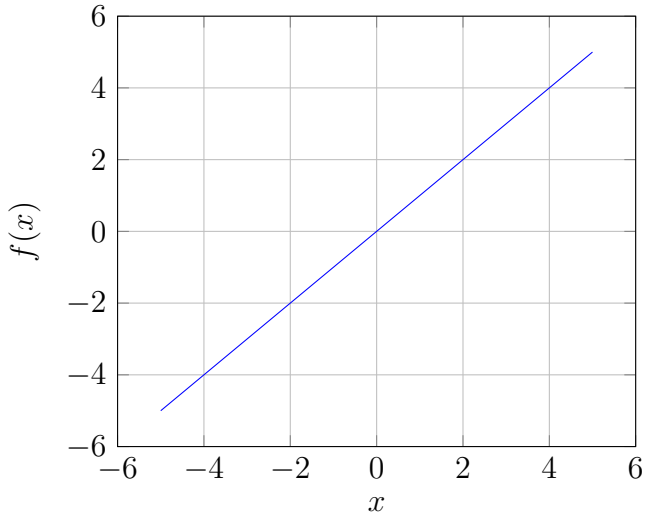
Vrstva

Vrstva je skupina neuronů se stejnou aktivační funkcí.

Aktivační funkce

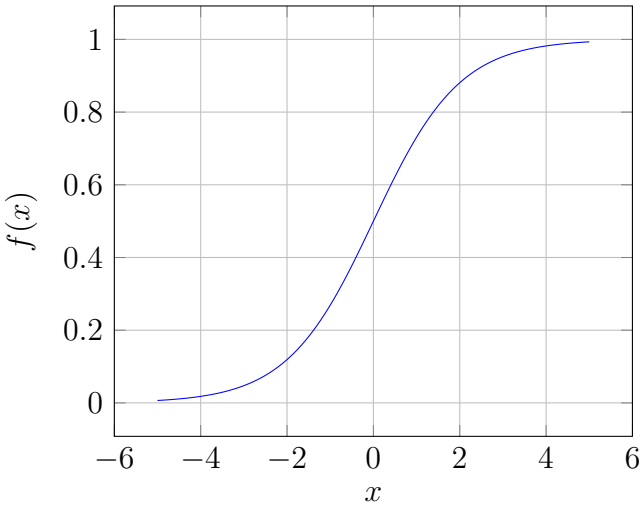
Aktivační funkce se používá pro definování výstupu a zavedení nelinearity. Bez nich by byla neuronová síť schopna aproximovat pouze n-dimenzionální rovinu. (PATTERSON, Josh. 2017, s. 65)
Dalším využitím je omezení výstupních hodnot. Například aktivační funkce sigmoid se s oblibou používá u výstupní vrstvy neuronových sítí určených ke klasifikačním problémům, protože je to relace $\mathbb{R} \rightarrow \{0..1\}$, která se dá jednoduše jako "jistota"neuronu, že se jedná o výstup, který neuron reprezentuje. Podobně se dá uvažovat i o funkcích jako je například softmax a tanh, které také najdou hojné využití u klasifikačních problémů.

Linearní funkce Vrací vstup, tak jak je. Využití najde především u vstupní vrstvy neuronové sítě a u neuronových sítí, které řeší regresní typy úloh.



$$f(x) = x$$

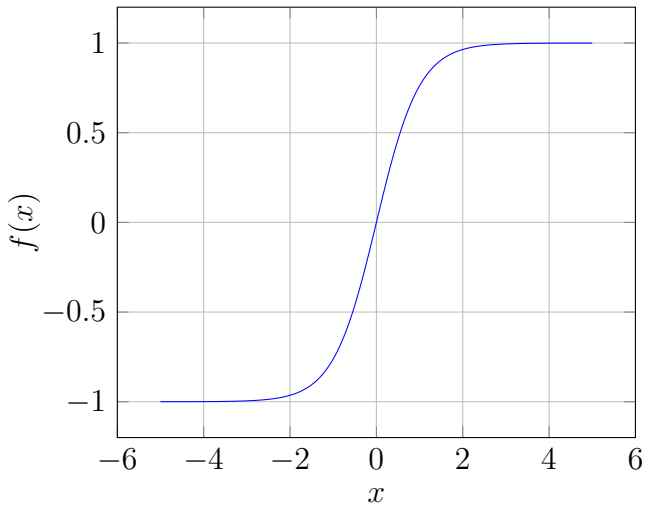
Sigmoid Sigmoid je aktivační funkce, která je schopná potlačit extrémní hodnoty a převést je do rozsahu 0 - 1.



$$f(x) = \frac{1}{1 + e^{-x}}$$

Tanh

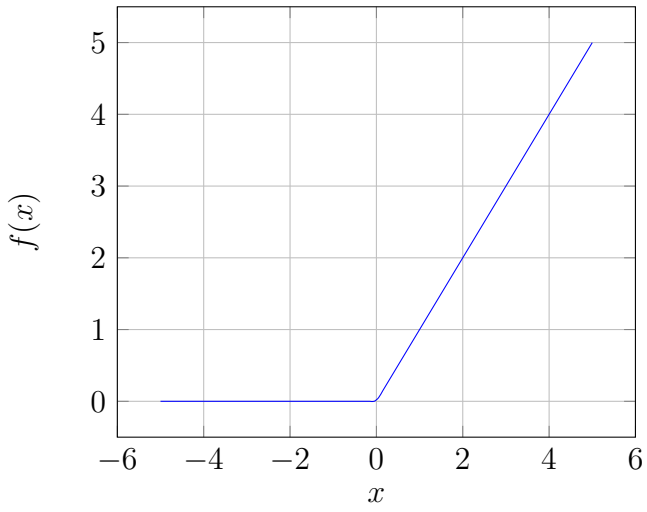
Tanh je funkce obdobná sigmoidu. Hlavní rozdíl mezi ní a sigmoidem je ten, že její obor je v rozmezí -1 a 1 hodí se proto i pro záporná výstupy, které vyžadují záporná čísla. (PATTERSON, Josh. 2017, s. 67)



$$f(x) = tanh(x)$$

RELU

RELU je aktivační funkce, která je podobná lineární aktivační funkci s tím rozdílem, že pokud vstupní hodnota nepřesáhne určitého prahu výstupem je 0. Její hlavní výhodou je to, že zabraňuje problémům s takzvaným explodujícím gradientem (PATTERSON, Josh. 2017, s. 69)



$$f(x) = \begin{cases} x >= 0, & x \\ x < 0, & 0 \end{cases}$$

Rekurentní neurony

Speciální druh neuronů, který si dokáže zapamatovat a reagovat na sekvenci vstupů. Rekurentní neurony značně rozšiřují možnosti neuronových sítí. S nimi je možné řešit

složité úlohy typu strojového překladu nebo sémantické vyhodnocování textu.

V praxi se setkáváme s dvěma druhy neuronů a to LSTM a novější GRU. LSTM a GRU jsou si velmi podobné s jedním podstatným rozdílem. U GRU bylo prokázáno, že je značně rychlejší.

2.2 Genetické algoritmy

Genetické algoritmy slouží k řízenému prohledávání stavového prostoru založené na teorii evoluce. (KOZA, John R., 1992, s. 17)

Princip

Základní myšlenka spočívá ve vygenerování náhodných jedinců (řešení problému) a jejích postupné zlepšování s pomocí operací křížení a mutace. Proces zlepšování genomů probíhá na základě jeho hodnocení (fitness).

Samotný algoritmus pak lze rozdělit na následující kroky (MITCHELL, Melanie., 1996, s. 12)

1. Vygeneruj náhodnou populaci o n chromozomech
2. Pro každý chromozom spočítej jeho fitness
3. Opakuj dokud není splněná podmínka ukončení
   a) Vyber pár chromozomů na základě jejích ohodnocení (selekce)
   b) Proveď spojení chromozomů (křížení)
   c) S určitou pravděpodobností mutuj daného jedince (mutace)
4. Nahraď současnou populaci populací, která vznikla předchozím krokem
5. Jdi na krok 2

Kódování

Někdy se mu též říká genotyp je způsob zápisu řešení problému. Je důležité zároveň uvést pojem fenotyp který označuje vlastní řešení úlohy.

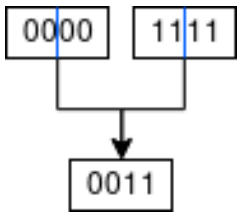
Existuje mnoho různých kódování a každý má své výhody a nevýhody. Většinou se snažíme vybírat kódování, které dobře reprezentuje daný problém. Zde je seznam několika nejpoužívanějších kódování (HYNEK, Josef., 2008, s. 42-43):

1. Binární - řetězec bitů, který může například reprezentovat jednu nebo více numerických hodnot.
2. Reálná čísla - Jedno nebo více reálných čísel
3. Permutační - Sekvence čísel používané například pro řešení problému obchodního cestujícího

Křížení

Křížení je operátor, který kombinuje dva jedince do jednoho. Jedinec, který vzniká přebírá genetickou informaci od obou jedinců v určitém poměru. Jeho implementace je závislá na použitém kódování.

Příkladem může být obrázek 1 ve kterém je ilustrováno tzv. jednobodové křížení (HYNEK, Josef., 2008, s. 50) při kterém dochází k rozdělení obou genomů ve stejném bodě a spojením vzniklých podřetězců do nového genomu.



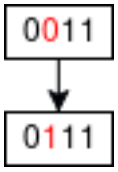
Obrázek 1: Příklad křížení

Mutace

Mutace náhodně modifikuje chromozom a zavádí tak do populace variaci. Díky tomuto se může algoritmus dostat z lokálního extrému.

Existují různé varianty mutace, která se provádí v závislosti na daném kódování.

Například u binárního kódování se může jednat o náhodnou inverzi některého z bitů genomu viz obrázek 2.



Obrázek 2: Příklad mutace

Selekce

Selekce je proces výběru dvou jedinců na něž jsou později aplikovány genetické operátory jako je křížení a mutace.

Využití

Genetické algoritmy naleznou využití v mnoha optimalizačních úlohách. Příkladem může být experiment, který zahrnoval použití gramatické evoluce pro vytvoření regresních modelů pro datasety CASY3 a CASY5 (LÝSEK Jiří, ŠŤASTNÝ Jiří, 2014, s. 2-9).

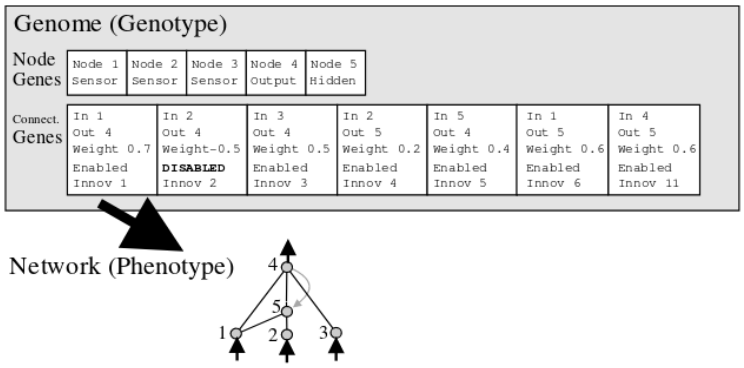
3 NEAT

NEAT (neuroevolution of augmenting topologies) kombinuje neuronové sítě s genetickými algoritmy. Hlavní výhodou této metody je, že generuje jak topologii neuronové sítě, tak její váhy. Výsledkem může být neuronová síť jejíž rozložení lépe popisuje řešený problém.

Algoritmus probíhá stejně jako běžný genetický algoritmus. Rozdílem je použitý fenotyp a operátory mutace a křížení, které se nad ním provádí.

3.1 Genotyp a fenotyp

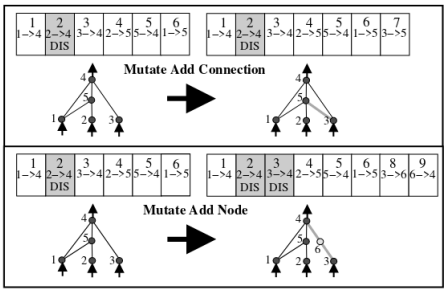
Genotyp a fenotyp je ilustrován na obrázku 3. Genotyp obsahuje jak údaje o jednotlivých neuronech, tak informace o topologii sítě. Metadata, která se týkají topologie neuronové síti jsou údaje o spojení (IN, OUT), váha samotného spojení (weight), informace týkající se toho, zda je spojení použito v fenotypu (ENABLED/DISABLED) a inovační skore. Inovační skore je číslo, které reprezentuje pořadí ve kterém se daný gen objevil v fenotypu (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 9).



Obrázek 3: Genotyp a fenotyp u algoritmu NEAT převzato z (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 9)

3.2 Mutace

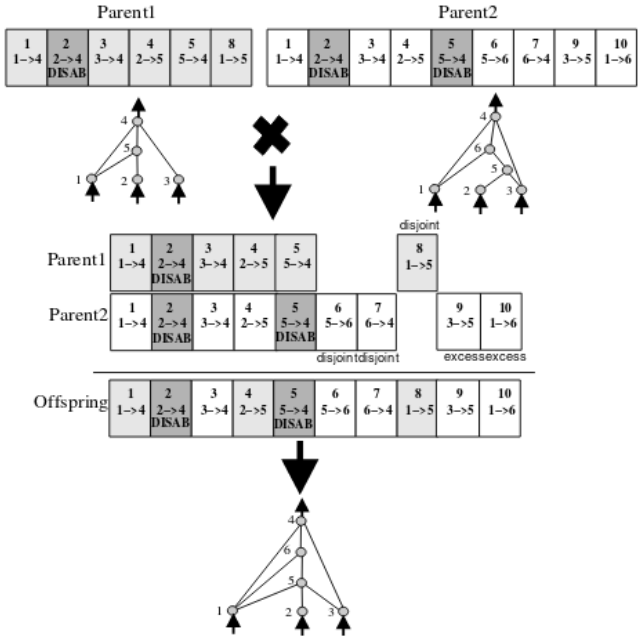
Jak jíž bylo zmíněno nad fenotypem lze provádět různé operace včetně operátoru mutace, který provede náhodnou změnu fenotypu s nadějí, že změna povede k zlepšení řešení. V případě algoritmu NEAT je operátor mutace ilustrován na obrázku 4. Na obrázku je vidět, že mutace může buď přidat další spojení nebo další neuron. V případě, že je přidávan nový neuron je mu přiřazeno náhodné spojení, které je označeno znakem DISABLED (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 10).



Obrázek 4: Mutace v NEAT převzato z (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 10)

3.3 Křížení

Posledním operátorem používaným při algoritmu NEAT je operátor křížení. Operátor křížení ilustruje obrázek 5. Křížení probíhá na základě inovačního čísla. Spojení se stejným inovačním číslem jsou náhodně zděděny z rodičovských genů. Je tu také šance na převzetí genů, které vytváří spojení nenacházející se v jednom z rodičů. Tyto geny jsou převzaty pouze z rodiče, který ma větší fitness. Při křížení je také náhodná šance, která může převzatý genom vypnout (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 12).



Obrázek 5: Křížení v NEAT. Převzato z (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 12)

3.4 Rozšíření algoritmu NEAT

Jelikož je algoritmus NEAT v současnosti předmětem aktivního výzkumu existuje pro něj mnoho rozšíření. Tato sekce se bude zabývat některými z nich.

Instinct

Instinct rozšiřuje původní algoritmus o rekurentní neurony. Rozšíření tak umožňuje případnému agentovi reagovat nejen na okamžitou situaci ale i na předchozí události ve světě. Tento fakt značně rozšiřuje možnosti neuronových sítí, které jsou generované tímto algoritmem ale zároveň je třeba mít na paměti, že se tímto značně zvětšuje prohledávaný stavový prostor a dochází tím k zvýšené časové náročnosti na získání vhodných výsledků.

odNEAT

Varianta algoritmu NEAT aplikovaná na distribuované online učení skupiny autonomních robotů (SILVA, Fernando, Paulo URBANO, Luís CORREIA a Anders Lyhne CHRISTENSEN, 2015, s. 1).

rtNEAT

Rozšíření, které se zaměřuje na neuroevoluci agentů v realném čase.

HyperNEAT

HyperNEAT rozšiřuje neuroevoluci o možnost vytváření velmi velkých neuronových sítí (STANLEY, Kenneth O., David B. D'AMBROSIO a Jason GAUCI., 2009, s. 1).

cgNEAT

Rozšíření zaměřující se na generování náhodného obsahu. Obdobně jako tomu je například u GAN sítí.

3.5 Alternativy k neuroevoluci

Neuroevoluce není jediný přístup, který je dostupný k řešení problému typu trénování autonomního agenta.

Příkladem může být rozsáhlý obor zabývající se zpětnovazebním učením (reinforcement learning). Při zpětnovazebním učení máme agenta, který se nachází v prostředí nad kterým může vykonávat různé akce. Agent může před vykonáním akce pozorovat stav prostředí na základě čehož se rozhoduje, jakou akci vykoná. Po provedení některé se prostředí nějakým způsobem mění a agent dostává zpětnou vazbu

v podobně odměny. Algoritmus zpětnovazebního učení se snaží agenta řídit tak, aby maximalizoval jeho odměnu.

Využití algoritmu NEAT

Algoritmus NEAT a jeho varianty předmětem aktivního výzkumu. Své využítí nalézá hlavně v oblasti robotiky a řešení pro počítač obtížných problémů jako je například hraní počítačových her.

4 Použité technologie

Tato kapitola poskytuje stručný přehled technologii použitých při řešení diplomové práce.

4.1 Grafana

Grafana umožňuje snadnou vizualizaci dat z databáze. Děje se tak definováním vhodných dotazů a volbou grafu, který je reprezentuje. V základní instalaci je na výběr několik druhů zobrazení a v případě potřeby je lze rozšířit s pomocí přídavných pluginů.

4.2 Docker

Docker je nástroj, který umožňuje vytvářet tzv. kontejnery. Kontejner poskytuje izolované softwarové prostředí ve kterém lze spouštět jednu nebo více aplikací. V praxi to umožňuje snadné nasazení libovolné aplikace bez ohledu na aktuální konfiguraci hostitelského systému.

4.3 Vue

Je populární JavaScriptový framework pro snadnou tvorbu uživatelských rozhraní.

4.4 NodeJS

NodeJS je open-source interpret jazyku JavaScript. Využití najde při psaní serverových aplikací v JavaScriptu a jiné případy, kdy je třeba spustit kód napsaný v javascriptu mimo prohlížeč. Lze ho využít například pro spouštění testů při CI (continuous integration) nebo pro tvorbu vysoce serverových aplikací v jazyce JavaScript.

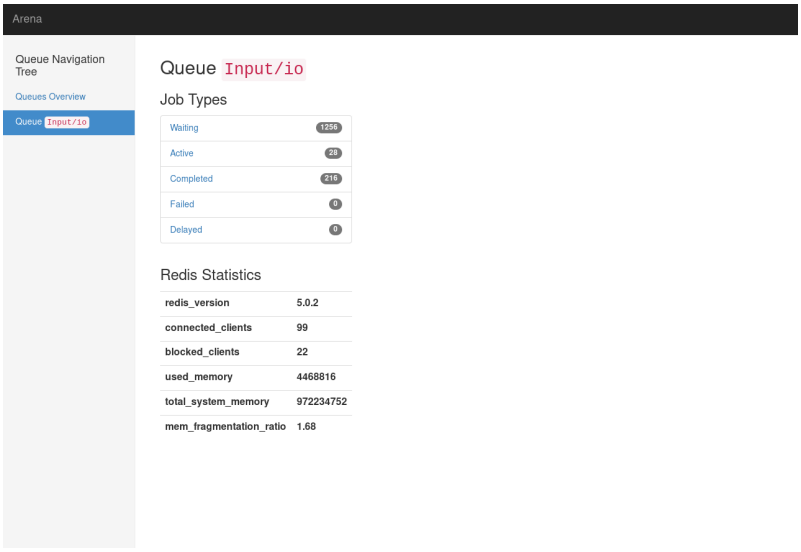
Narozdíl od běžného JavaScriptu obsahuje rozšířenou standardní knihovnu pro snadnou tvorbu serverových aplikací. Tato standardní knihovna umožňuje například javascript kódu prácí se soubory na hostitelském systému, což je něco, co není z bezpečnostních důvodů v klasickém JavaScriptu, který běží v prohlížeči možné.

4.5 Bull

Bull je knihovna pro NodeJS, která poskytuje frontu úkolů založenou na populární in-memory databázi Redis.

4.6 Arena

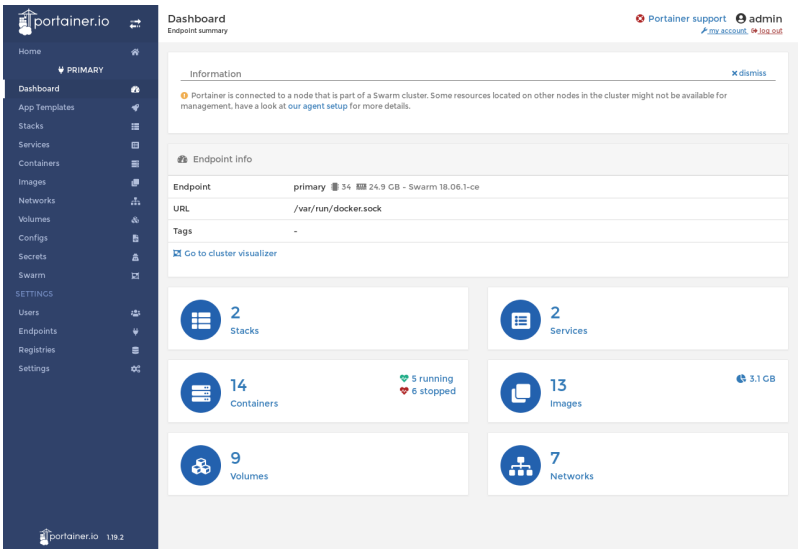
Arena je webová aplikace pro správu a zobrazení aktuálního stavu fronty úkolů poskytované knihovnou bull.



Obrázek 6: Webové aplikace Arena

4.7 Portainer

Portainer je webová aplikace pro zobrazení aktualního stavu a správu instance dockeru a docker swarmu.



Obrázek 7: Rozhraní aplikace portainer

4.8 Databáze

Výsledky experimentů bude nutné někam zapsat. Jako vhodné řešení se jeví relační databáze. S důvodů hw omezení je na ní několik praktický nefunkčních požadavků:

- Musí být schopná běžet na architektuře arm
- Musí být ACID
- Musí být schopná obsloužit více klientů (čtení a zápis)

Z výše uvedených důvodů, z důvodu předchozích zkušeností a s ohledem na to, že se jedná o nekritickou část systému byl vybrán databázový systém PostgreSQL.

PostgreSQL je populární databázový systém, který nabízí robustní open-source alternativu i ke komerčním řešením. Navíc splňuje všechny podmínky stanovené na začátku této sekce.

4.9 PIXI.js

Grafická knihovna pro snadné vykreslování nad HTML značkou canvas, která byla zavedena ve verzi 5. Obaluje jak klasické vykreslovací API, tak modernější WebGL.

4.10 Neataptic

Knihovna implementující samotný algoritmus NEAT ve variantě instinct.

4.11 NPM a YARN

NPM i Yarn jsou kolíčkovací systémy pro javascript. Hlavní rozdíl mezi NPM a Yarn je, že Yarn stahuje balíčky paralelně. Je tedy značně rychlejší oproti NPM.

4.12 CES

Je knihovna, která implementuje tzv. ECS (entity component system). ECS se používá především ve hrách a nabízí určitý způsob, jak se dívat na herní objekty a logiku. Myšlenka je taková, že vše lze rozdělit do níže uvedených podsekcí.

Komponenta

Komponenty bývají jednoduché datové struktury, které vyjadřují vlastnosti entity u níž jsou přiřazeny.

Entita

K entitě se dá přiřadit jedna nebo více komponent (vlastností). Entita tak může představovat libovolný herní objekt.

Příkladem může být vozidlo, které může být reprezentováno složením fyzikální, grafické a ovládací komponenty. Při správné implementaci níže zmiňovaných systému lze pak na jakoukoliv entitu, která má tyto komponenty nahlížet jako na auto.

Je důležité si uvědomit, že na rozdíl od klasického systému dědičnosti je možné entity snadno rozšířit tak, že do nich přidáme další komponenty. Můžeme například k entitě auta přiřadit komponentu životy a udělat ho tak zranitelným.

System

Systém provádí určité akce nad entitami, které mají dané komponent. Chceme-li například propojit grafickou reprezentaci s fyzikálním enginem, můžeme si napsat systém, který po kroku fyzikálního enginu vyhledá všechny entity, které mají grafickou a fyzikální komponentu upraví pozice a rotaci všech grafických objektu tak, aby byla identická s pozicí a rotací fyzikálních objektů, které jsou k ním přiřazeny.

5 Metodika

Na začátku je potřeba provést podrobnou analýzu problému s ohledem na požadavky stanovené konzultacemi s vedoucím. Návrh bude také zahrnovat experimenty, které budou s neuroevolucí provedeny.
Po návrhu bude následovat implementace daného řešení a jejíž popis bude přidán do této práce. V průběhu implementace bude také kód a návrh postupně upravován na základě požadavků, které se mohou objevit až při implementaci navrženého řešení.
Po implementaci bude následovat realizace a vyhodnocení experimentů popsaných v návrhu.
Na závěr proběhne vyhodnocení řešení s návrhem možných zlepšení.

6 Analýza problému

Tato kapitola se zabývá analýzou funkčních a nefunkčních požadavků pro softwarové řešení. Požadavky vznikaly na základě konzultace s vedoucím a vlastní invencí.

6.1 Funkční požadavky

Funkční požadavky jsou rozděleny do několika kategorii.

Simulace

Vyhodnocování agenta bude probíhat jeho nasazením v simulovaném prostředí. Fitness bude pak vyhodnocena na základě jeho akcí v prostředí.

- Testovací prostředí musí být pro všechny agenty stejné
- Fitness funkce musí být deterministická
- Fitness by měla být zjistitelná kdykoliv v průběhu simulace
- Simulace by měla být konfigurovatelná
- Možnost změny obtížnosti simulace pro agenta
- Měla by existovat možnost spuštění více instancí simulace v rámci jednoho programu

Vizualizace

Průběh algoritmu je třeba zobrazit.

- Je třeba provést grafickou vizualizaci fyzikální simulace
- Při zobrazení by mělo být možné vyčíst stav agenta (fitness, senzory, ...)
- Možnost vizualizace průběhu simulace v reálném čase (například pro její demonstraci v předmětu VUI2)

Experimenty

Práce zahrnuje vyhodnocování agenta v různých podmínkách z tohoto vychází následující požadavky:

6.2 Nefunkční požadavky

Spolu s funkčními požadavky jsou na řešení kladeny také požadavky nefunkční.

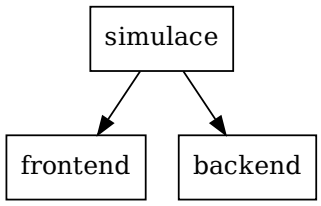
- Škálovatelnost - Možnost spustit a vykreslit libovolné množství simulací

- Rychlost simulace. Simulace by měla být schopná samostatně běžet alespoň rychlostí 30 snímků za vteřinu.
- Robustnost - Simulace by měla být odolná neočekávaným situacím
- Portabilita - bylo třeba zajistit, aby šlo kód rozběhnout v různých platformách v různých konfiguracích.
- Robustnost - Simulace by si měla poradit s neočekávanými vstupy, jako je třeba **NaN**, který vychází z neuronové sítě

7 Návrh řešení

Tato kapitola obsahuje návrh řešení založený na požadavcích identifikovaných v předchozí kapitole. Na základě těchto požadavků byl návrh rozdělen do dvou projektů a to webového frontendu, který slouží jako rozhraní ukazující průběh simulace realném čase a serverové části, která slouží pro rychlý výpočet simulace bez vykreslovaní na platformě Node.js.

Jelikož obě části budou používat stejný simulační kód bylo rozhodnuto, že bude samotná simulace vytvořena jako softwarová knihovna. Tuto závislost ilustruje obrázek 8.



Obrázek 8: Schéma závislostí

7.1 Simulace

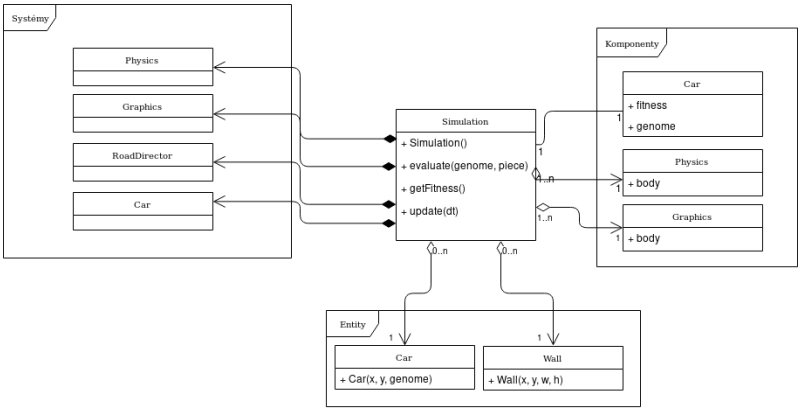
Z obrázku 8 je zřejmé, že simulační kód spojuje obě části dohromady. Je tedy důležité, aby byl navržený tak, aby jej bylo co nejjednodušeji integrovat s oběma řešeními.

Na základě těchto požadavků a podmínek, které jsou stanoveny v předchozí kapitole byl vytvořen následující návrh:

Simulace je navržena v duchu ECS (**Entity component system**) podrobný popis lze nalézt v sekci 4.12. Není tedy žádným překvapením, že se všechny komponenty nalezené v simulaci dají rozložit na systémy, komponenty a entity. Pro lepší představu o implementaci je níže uveden přehled všech systému, entit a komponent použitých v simulaci.

Diagram tříd

Níže popsaný návrh doplňuje diagram tříd, který zobrazuje jejích návaznost a zároveň také do návrhu zavádí třídu **Simulation**, která celou simulaci schovává za jednoduché rozhraní a umožňuje snadnou integraci simulace do různých aplikací.



Obrázek 9: Diagram tříd

Entity

Simulace obsahuje následující entity:

PhysicsGroup Seskupuje fyzikální entity do jedné pro snadnou manipulaci s nimi.

RoadPart Entita, která obaluje jednu nebo více překážek tak, aby se s nimi dalo snadno pohybovat používá se pro tvorbu složitějších dílů vozovky.

Car Reprezentuje samotné vozidlo obsahuje jak jeho grafickou reprezentaci, tak kompletní logiku a fyzikální model.

Komponenty

Simulace obsahuje následující komponenty:

- **Car** obsahuje všechny potřebné informace o agentovi. Toto zahrnuje vše od neurnové sítě, která je použitá pro jeho řízení po ovládání jednotlivých kol agenta.
- **Graphics** komponenta, která obsahuje grafické informace pro **Pixi.js**.
- **Physics** komponent, která obsahuje fyzikální entity pro **P2.js**

Systémy

Návrh simulace obsahuje následující systémy:

Car systém, který se stará o ovládání agenta a částečně o vyhodnocování jeho fitness.

Graphics grafický systém, který slouží především k překreslování entit.

Physics krokuje fyzikální engine a synchronizuje grafickou reprezentaci s fyzikální entitou. Tento proces probíhá pro každý snímek a skládá se z přiřazení nové rotace a pozice pro grafickou entitu.

RoadDirector Road director se stará o generování nekonečného prostředí pro agenta. Děje se tak na základě předefinovaných dílu vozovky z nichž každý zaplňuje celou obrazovku simulace. V případě, že agent dorazí až na konec obrazovky je mu určen nový navazující dílek. Agent je pak přehozen na opačnou stranu obrazovky a zároveň je vyměněn díl na kterém se nachází. Hlavní výhodou tohoto přístupu je to, že agent může jezdit po vozovce donekonečna bez starosti o to, že by se dostal na limit fyzikálního enginu (přetečení pozice fyzikálního objektu). Další nesporná výhoda tohoto přístupu je úspora paměťových nároků, kterou by s sebou nesla definice větší mapy a možnost generování náhodných map pro testování agenta.

Fitness funkce

Fitness funkce je důležitou součástí simulace, která zásadně ovlivňuje chování výsledných agentů a je tedy nutné jí volit vhodně. Je nutné, aby funkce agenta motivovala ke správné činnosti.

Po několika pokusech a konzultaci s vedoucím práce byla jako metrika úspěchu agenta zvolena celková vzdálenost, kterou je agent schopný překonat v průběhu jedné generace. Výpočet je realizován s pomocí RoadDirectoru, který si při každém přechodu zaznamená bod, ve kterém se po přesunu agent nachází. Výsledná fitness je pak součet uražených vzdáleností pro každou místnost. Road direktor si pro každou obrazovku uchovává vzdálenost, kterou agent v dané obrazovce překonal. Výsledným fitness je pak součet všech vzdáleností na všech obrazovkách.

7.2 Klientská část

Klientská část je webové rozhraní, které vzniklo z požadavků na vizualizaci a ověření funkčnosti simulační knihovny.

Z požadavků vyšla aplikace, která obsahuje 3 rozdílné obrazovky a to jedne, která slouží pro zobrazení a simulaci vývoje algoritmu v reálném čase. Další slouží pro zpětné přehrávání již vygenerovaných genomů a poslední obrazovka, která umožňuje uživateli vyzkoušet ovládat agenta.

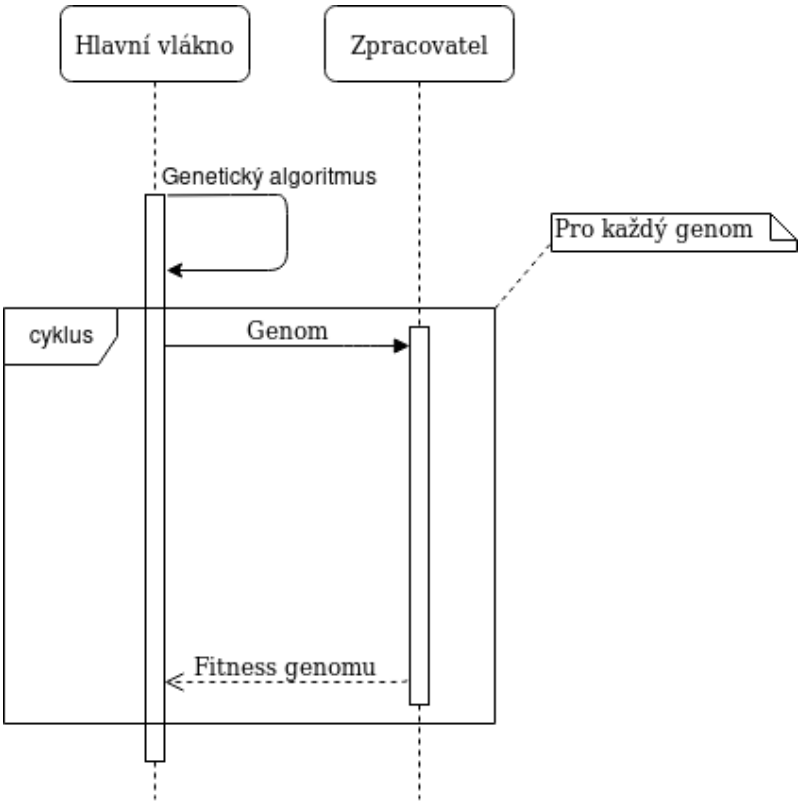
7.3 Serverová část

Serverová část byla nakonec navržena a vytvořena ve dvou na sebe navazujících verzích.

První verze

Je ilustrovaná na obrázku 10. Návrh první verze popisuje jednoduchou aplikaci, která rozkládá vyhodnocování jednotlivých genomů mezi jednoho nebo více zpracovatelů. Každý zpracovatel běží ve vlastním vlákně a zátěž je tedy rozložena mezi dostupná jádra procesoru. Zpracovatel po vyhodnocení genomu vrací hlavnímu vláknu fitness

daného jedince. Ten si ho uloží a po vyhodnocení všech jedinců tímto způsobem provede algoritmus NEAT. Tento proces se opakuje do té doby, než nedojde k naplnění ukončujících podmínek (maximální počet generací) nebo přerušení programu.

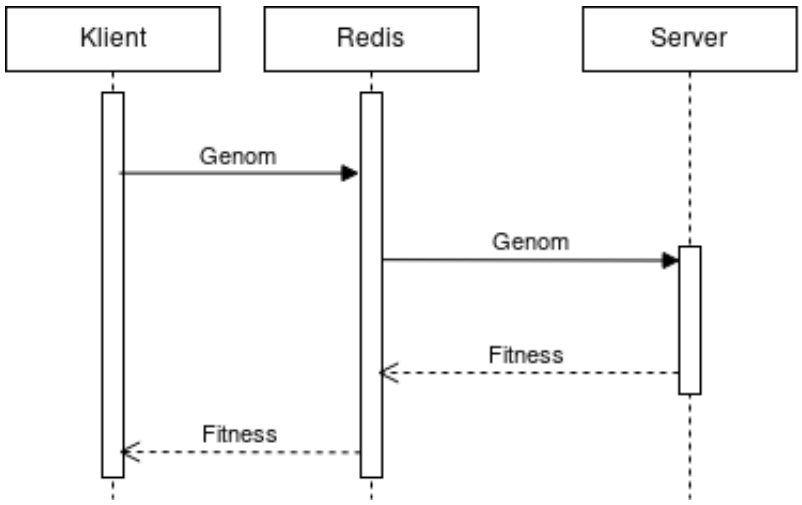


Obrázek 10: První verze serverové části

Druhá verze

Druhá verze rozšiřuje návrh o možnost rozložení výpočtů mezi více počítačů. Byla navržená poté, co bylo zjištěno, že předchozí verze nebyla schopná vyhodnotit dostatečné množství genomů dostatečně rychle.

Pracuje dle diagramu 16, kde je vidět, že klient zadává do fronty úkoly (genom a nastavení simulace). Jednotliví zpracovatelé (počítače v clusteru), kteří si je z ní vyberou, jednotlivé genomy vyhodnotí a hodnotu fitness funkce pošlou zpět na klienta. Toto je ilustrováno v diagramu 11.

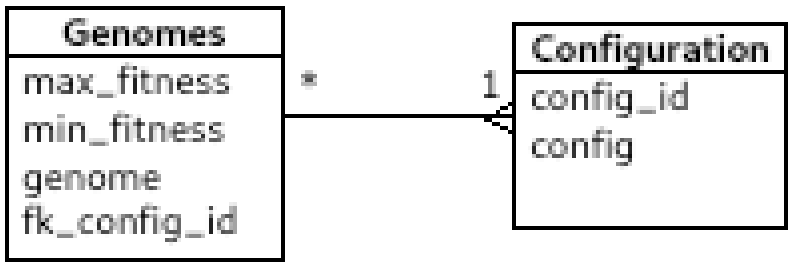


Obrázek 11: Sekvenční diagram komunikace se serverem

Jakmile klient dostane všechny hodnoty zpět provede na populaci algoritmus NEAT (mutace, křížení, …) a poté je nová generace poslána znovu na vyhodnocení.

Databáze

Všechny verze serverové části zaznamenávají průběh algoritmu NEAT do databáze jejíž schéma lze vidět v obrázku 12. Databáze obsahuje tabulku **Genomes**, která uchová fitness nejlepšího a nejhoršího jedince v generaci, zároveň také obsahuje nejlepší genom z dané generace. Každý záznam v tabulace **Genomes** má přidělenou konfiguraci se kterou byl spuštěn. Toto je důležité při vyhodnocování více konfigurací zároveň, které je popsáno níže.



Obrázek 12: Schéma databáze

7.4 Fitness funkce

Agenta je ovšem kromě motivace třeba také penalizovat za akce, které jsou nepřípustné. V případě simulace se jedná především o kolizi s překážkou, za což je agent penalizován předčasným ukončením simulace a nemožností tedy zvýšit svoji fitness.

7.5 Agent

Definice agenta zásadně ovlivňuje výsledek simulace, protože stanovuje vstupy a výstupy do a z neuronové sítě. V této práci je testováno několik konfigurací, které jsou popsány níže.

V případě této práce je agentem auto, které je vybaveno vzdálenostními senzory. Měření těchto senzorů je normalizováno (maximální vzdálenost měřícího paprsku je 800 m) a předáno jako vstup do neuronové sítě.

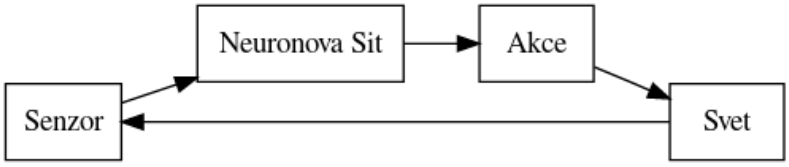
Agent bude testován v různých konfiguracích:

Řízení agenta

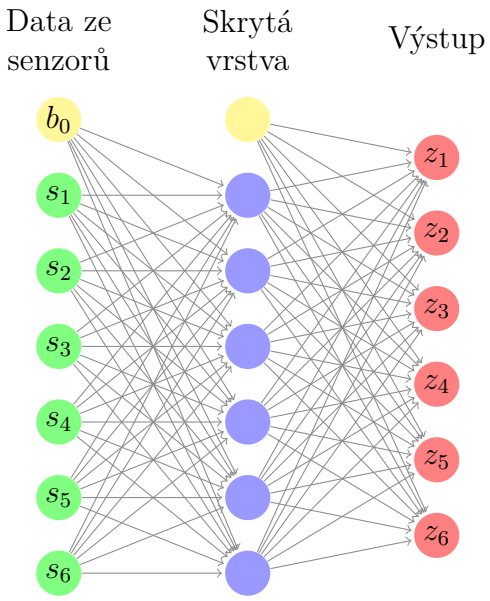
Řízení agenta probíhá tak, že se každý snímek s pomocí neuronové sítě na obrázku 14 rozhoduje, jakou akci podnikne. Má následující možnosti:

1. Ovládání volantu
   a) z_1 Otočení volantem o určitý počet stupňů doleva
   b) z_2 Otočení volantem o určitý počet stupňů doprava
2. Rychlostní stupně
   a) z_3 - zpátečka
   b) $z_4 - z_6$ - rychlosti dopředu

Ovládání volantu i volba rychlostního stupňů probíhá zároveň a to tak, že se vždy z dané skupiny neuronů vybere ten, který má největší hodnotu. Tento přístup je identický tomu, který se používá například u neuronových sítí pro klasifikaci.



Obrázek 13: Řízení agenta



Obrázek 14: Neuronová síť agenta

Možné konfigurace agenta

Výše uvedenou konfiguraci lze rozšířit o níže uvedené vstupy/výstupy.

Modifikace vstupů umožní agentovi vnímat více než jen vzdálenosti z jednotlivých senzorů. Bude zajímavé pozorovat, jak se agent s jednotlivými vjemy poradí. V práci se vyzkouší následující přidané vstupy:

- aktuální rychlost
- náklon volantu

Modifikovat lze také výstupy neuronové sítě toto rozšíří nebo omezí možnosti agenta:

- Přidáním/odebráním možnosti brzdění
- Přidáním/odebráním možnosti udržení volantu ve stejné pozici
- Modifikací možností rychlostí a to:
  - Foo

8 Implementace

Vlastní práce se skládá ze dvou částí klientská část, která slouží k vizualizaci algoritmu a zobrazení výsledků ze serverové části. Serverová část pro maximální urychlení simulace.

8.1 Simulace

Simulace je realizovaná jako knihovna pro Node.js, lze jí tedy použít jak u klientské částí, tak u serverové části. Poskytuje kompletní fyzikální simulaci agenta, prostředí ve kterém se pohybuje, jeho ovládání a výpočet fitness funkce. Součástí simulačního prostředí je také kód pro její vizualizaci.

Rychlost byla zajištěna implementací profilovacího programu (**benchmark.js** ve složce simulation), který spouští simulaci na předem připravené populaci jedinců. Výstupem je pak doba, za jakou jí vyhodnotil na jednom jádře procesoru. Tento údaj byl pak používán při implementaci simulace pro orientační představu, jak moc případné změny v kódu ovlivňují rychlost samotné simulace. Dále byla simulace podrobena občasnému profilování v klientské částí s pomocí vývojářských nástrojů prohlížeče chrome, na kterém simulace jede nejlépe.

Nenáročnost, která souvisí s rychlostí pak byla zajištěna tím, že bylo v průběhu psaní kódu dbáno na to, aby v průběhu simulace nedocházelo k přebytečným alokacím, které by nejen že mohli způsobit přebytečný nárůst požadované paměti ale způsobovali by také nepředvídatelné zpomalení, které s sebou přináší jazyk využívající garbage kolektor.

Robustnost je podrobněji vysvětlená v sekci 7.4 a popis toho, jak bylo dosaženo stejných podmínek pro všechny agenty lze nalézt v návrhu 7.1 především v popisu RoadManageru.

8.2 Serverová část

Serverová část vyhodnocuje jednotlivé jedince distribuovaně s pomocí fronty úkolů. Frontu poskytuje knihovna **Bull**, která používá **Redis** pro správu údajů o jednotlivých úkolech.

Cílem byl návrh robustního systému, který v ideálním případě rozloží výpočetní zátěž mezi jednotlivé uzly rovnoměrně. Dalším požadavkem byla možnost odpojení kdykoliv kteréhokoliv z počítačů, jelikož ne všechny bylo možné nechat běžet přes noc.

8.3 Výpočetní cluster

Ukázalo se, že vyhodnocování simulace zabírá neúměrné množství času a to i na nejvýkonnějším dostupném počítači. Například vyhodnocení jedné generace populace o 1024 jedincích zabralo 290 s na nejsilnějším dostupném pc. Z tohoto důvodu bylo

rozhodnuto o distribuce výpočetní zátěže mezi více počítačů. Byl vytvořen výpočetní cluster se specifikací popsanou v tabulce 1.

Procesor	RAM	Počet	Architektura
S5P6818 Octa core	1 GB	2	arm64
Broadcom BCM2837B0 quad-core	1 GB	1	arm32
Phenom X4 965	8 GB	1	x64
Intel Core i5-2300	4 GB	1	x64
Intel atom x5-Z8350	2 GB	1	x64
Cortex-A5	1 GB	1	armv7l

Tabulka 1: Použitý hardware

Ověření rychlosti cluste

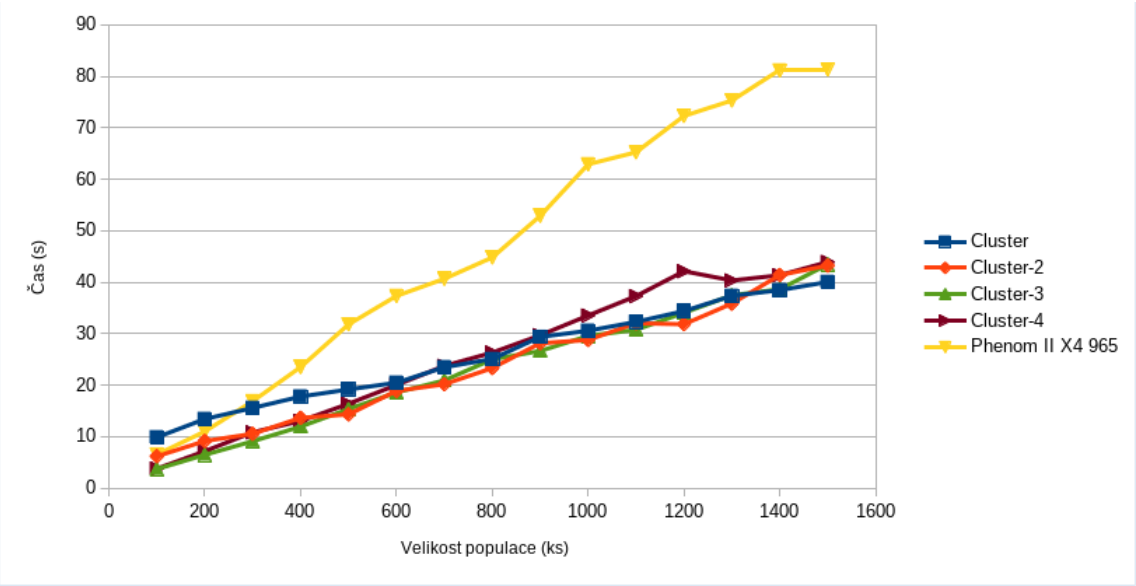
Lze i namítnout, že se zde projevuje určitá režie při síťové komunikaci se serverem, což může být zdrojem určitého zpomalení.

Pro ověření rychlosti bylo provedeno měření výkonu clusteru a jeho porovnání s nejvýkonnější dostupnou sestavou. Měření bylo provedeno nad náhodně vygenerovanými populacemi. Jelikož se simulace může ukončit předčasně (například při kolizi s překážkou) byla simulace provedena pro každou velikost 10× a výsledek byl zprůměrován. Naměřená data lze nalézt v tabulce **??** ze které vychází obrázek 15 na kterém lze vidět výsledky tohoto srovnání.

Porovnání rychlosti clusteru s nejvýkonnějším dostupným počítačem ukazuje, že cluster je ve většině případů skoro stejně nebo výrazně rychlejší než samostatný výpočet. Jediné dvě naměřené instance, kde to neplatí je u populací o velikosti 100 a 200, kde si cluster vede mírně hůře, než nejvýkonnější dostupná sestava. Toto lze vysvětlit, jak přítomností méně výkonného hardwaru v clusteru (především se jedná o S5P6818) na které se musí u menších populací čekat. Pro ověření této teorie byl cluster spuštěn v dalších konfiguracích, kde byly postupně odebírány jednotlivé počítače a měření bylo opakováno.

- Cluster - Celý cluster
- Cluster-2 - Odebrán S5P6818 (obě jednotky)
- Cluster-3 - Odebrán Intel atom x5-Z8350
- Cluster-4 - Odebrán AMD A4-4300M

Je nutné však podotknout, že proměnlivá doba u vyhodnocování jedince znamená, že měření není zcela přesné. Nicméně lze na základě dat usoudit, že u větších populací dochází k přibližně 2× zrychlení.



Obrázek 15: Porovnání rychlosti clusteru s jedním PC

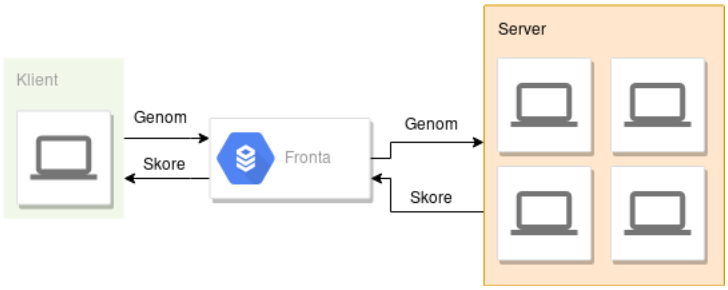
Docker swarm

Pro snadnou distribuci a správu byly všechny počítače zorganizovány do docker swarmu. Docker swarm obsahoval jednoho manažera (Broadcom BCM2837B0 quad-core), který zároveň spouštěl klientskou aplikaci a další služby:

1. **Portainer** pro správu clusteru
2. **Arena** webové UI pro správu knihovny **Bull**
3. **Redis** používaný knihovnou **Bull**

Na manažerovy nebyl spuštěn zpracovatel, aby se zabránilo jeho přetížení (manažer swarmu by měl být vždy dostupný).

Použití docker swarmu umožňuje především snadné nasazení a správu zpracovávajících procesů. Zároveň zajišťuje, že všechny instance zpracovatelů mají unifikovanou konfiguraci, což je zvláště důležité pro dosažení konzistentních výsledků.



Obrázek 16: Schéma distribuovaných výpočtů

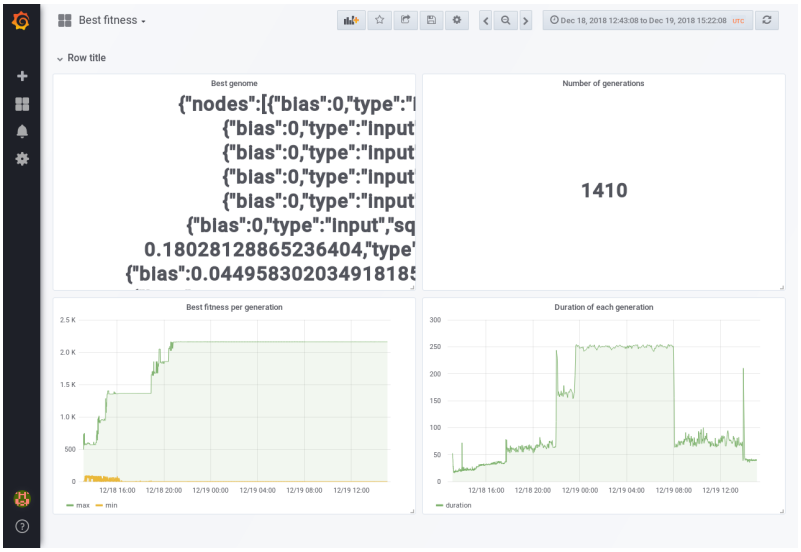
Tento přístup má několik výhod a to:

1. Robustnost - Pokud jeden nebo více zpracovatelů selže (je například odpojen ze sítě) je možné pokračovat ve vyhodnocování (neúspěšný úkol lze vrátit zpátky do fronty). Toto v kombinaci s výše zmíněným docker swarmem znamená, že jakýkoliv výpočetní uzel lze kdykoliv vypnout a po znovu zapojení do sítě si načte nejnovější konfiguraci a začne znovu vyhodnocovat bez potřeby jakékoliv manipulace s jakoukoliv částí swarmu.
2. Dobré rozložení zátěže - Jelikož si zpracovatel vytahuje úkoly z fronty, je vždy optimálně zatížen, a není třeba řešit rozložení mezi různě výkonnými a zatíženými počítači.
3. Škálovatelnost - problém lze škálovat až do doby, kdy počet procesorů nepřesáhne počet potřebných simulací. Chceme-li tedy vypočítat generaci o tisíci jedincích můžeme na ně nasadit až tisíc procesorů.

Monitorování stavu

Pro monitorování stavu algoritmu byla použita webová aplikace Grafana. Jak jíž bylo řečeno v sekci 4.1 jedná se o nástroj pro snadnou vizualizaci dat v databázi. V případě této práce posloužila k vytvoření jednoduchého kontrolního panelu, který obsahoval následující informace:

- Graf zobrazující fitness nejlepšího/nejhoršího jedince dle generací
- Textové pole, které obsahuje nejlepší genom
- Numerické pole, které obsahuje počet generací
- Graf zobrazující dobu výpočtu dle generací



Obrázek 17: Kontrolní panel v aplikaci grafana

9 Klientská část

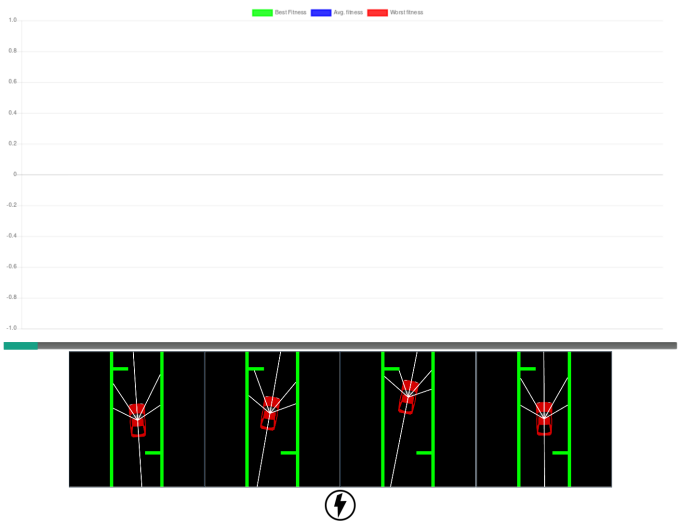
Klientská část byla navržena tak, aby byla schopná vizualizovat průběh algoritmu NEAT a zároveň měla možnost znovu vyhodnocení existujících genomů vygenerovaných serverovou částí. První požadavek vznikl na základě konzultace s vedoucím, který chtěl algoritmus NEAT demonstrovat v hodinách předmětu VUI2. Druhý požadavek vznikl z důvodu potřeby vizualizace řešení, které generoval sever.

9.1 Vizualizace

Vizualizace se skládá z jednoduchého rozhraní, které lze vidět na obrázku 18. V horní části je graf, zobrazující průběh genetického algoritmu. Lze v něm nalézt fitness nejlepšího, nejhoršího a průměrného jedince v populaci.

Další část se skládá z konfigurovatelného množství simulačních prostředí. Jednotliví jedinci v generaci jsou pak rovnoměrně rozloženi mezi všechna simulační prostředí a uživatel může pozorovat vývoj jedinců v reálném čase.

Poslední tlačítko slouží k urychlení simulace. Způsobí to, že se simulace začne obnovovat bez vykreslování. Toto jí značně zrychlí.



Obrázek 18: Uživatelské rozhraní klientské části

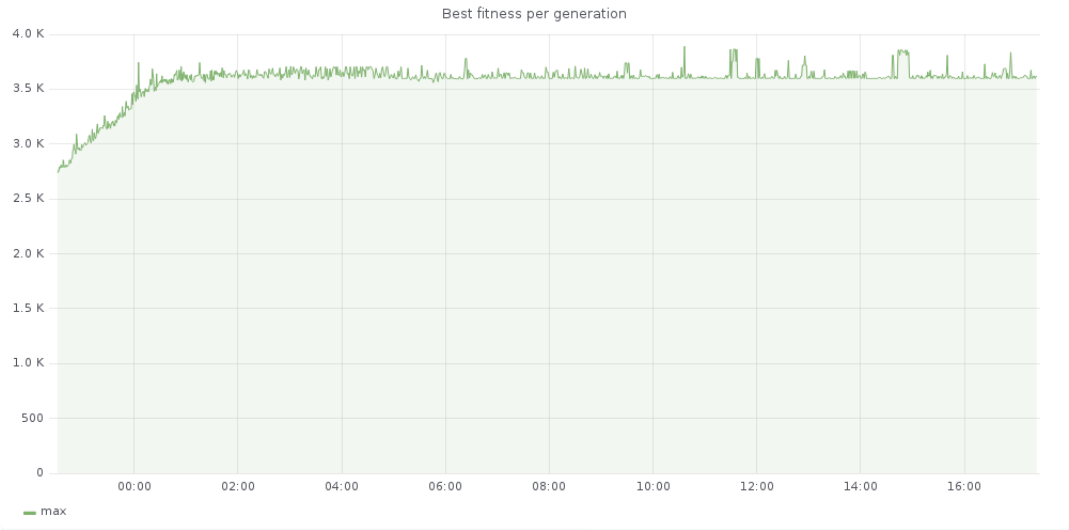
10 Experimenty

Po návrhu simulačního prostředí byl agent vyzkoušen v několika situacích se stupňující se obtížností. Každá simulace probíhala s 1000 jedinci po 2000 generací. Ačkoliv je pravděpodobné, že by delší doba evaluace by pravděpodobně vyústila v lepší výsledky její výpočet v různých konfiguracích se ukázal jako příliš časově náročný navíc empirické pozorování ukázalo, že tato konfigurace poskytuje dostatečně dobré výsledky za snesitelný čas.

S ohledem na časovou náročnost výpočtů (výpočet tisíce generací trvá na výpočetním clusteru přibližně 5 hodin) byly zkoumany jen tyto konfigurace:

10.1 Nekonečná silnice ve tvaru I

Agent byl umístěn do nekonečné rovné silnice ve tvaru I. Cílem bylo pozorovat, zda se agent bude schopný naučit řídit rovně. Agent po tisící generací dosáhl fitness 3 500 a naučil úspěšně kývavým pohybem udržet uprostřed vozovky.



Obrázek 19: Fitness agenta v průběhu času

11 Možná vylepšení

Tato kapitola se bude zabývat možnými vylepšení současného řešení.

12 Závěr

13 Reference

[BUDUMA, Nikhil 2017] BUDUMA, Nikhil. *Fundamentals of deep learning: designing next-generation machine intelligence algorithms.* Sebastopol: O'Reilly, 2017. ISBN 978-149-1925-614..

[PATTERSON, Josh. 2017] PATTERSON, Josh. *Deep learning : a practitioner's approach Deep learning : a practitioner's approach. 1.* Beijing ; Boston ; Farnham ; Sebastopol ; Tokyo: O'Reilly, 2017. ISBN 978-1-491-91425-0..

[MITCHELL, Melanie., 1996] MITCHELL, Melanie. *An introduction to genetic algorithms.* Cambridge: Bradford Book, c1996. ISBN 0-262-13316-4..

[HYNEK, Josef., 2008] HYNEK, Josef. *Genetické algoritmy a genetické programování.* Praha: Grada, 2008. Průvodce (Grada). ISBN 978-80-247-2695-3..

[LÝSEK Jiří, ŠŤASTNÝ Jiří, 2014] LÝSEK, Jiří a ŠŤASTNÝ, Jiri. (2014). *Automatic discovery of the regression model by the means of grammatical and differential evolution.* Agricultural Economics (AGRICECON). 60. 546-552. 10.17221/160/2014-AGRICECON. .

[STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002] STANLEY, Kenneth O. a Risto MIIKKULAINEN. *Evolving Neural Networks through Augmenting Topologies.* In: Evolutionary Computation [online]. 2002, 10(2), s. 99-127 [cit. 2018-12-08]. DOI: 10.1162/106365602320169811. ISSN 1063-6560. Dostupné z: http://www.mitpressjournals.org/doi/10.1162/106365602320169811.

[SILVA, Fernando, Paulo URBANO, Luís CORREIA a Anders Lyhne CHRISTENSEN, 20 SILVA, Fernando, Paulo URBANO, Luís CORREIA a Anders Lyhne CHRISTENSEN. *OdNEAT: An Algorithm for Decentralised Online Evolution of Robotic Controllers.* Evolutionary Computation. 2015, 23(3), 421-449. DOI: 10.1162/EVCO_a_00141. ISSN 1063-6560. Dostupné také z: http://www.mitpressjournals.org/doi/10.1162EVCO_a_00141.

[STANLEY, Kenneth O., David B. D'AMBROSIO a Jason GAUCI., 2009] STANLEY, Kenneth O., David B. D'AMBROSIO a Jason GAUCI. *A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. Artificial Life [online].* 2009, 15(2), 185-212 [cit. 2018-12-15]. DOI: 10.1162/artl.2009.15.2.15202. ISSN 1064-5462. Dostupné z: http://www.mitpressjournals.org/doi/10.1162/artl.2009.15.2.15202.

[RUSSELL, Stuart J., Peter NORVIG a Ernest DAVIS, 2010] RUSSELL, Stuart J., Peter NORVIG a Ernest DAVIS. *Artificial intelligence: a modern approach. 3rd ed.* Boston: Pearson, c2010. Prentice Hall series in artificial intelligence. ISBN 978-0-13-207148-2..

[KOZA, John R., 1992] KOZA, John R. *Genetic programming: on the programming of computers by means of natural selection.* Cambridge, Mass.: MIT Press, c1992. ISBN 0-262-11170-5..

Přílohy

A CD se zdrojovým kódem

B Tabulka s naměřenými rychlostmi clusteru

Jedinců	Cluster	Cluster-2	Cluster-3	Cluster-4	Phenom II X4 965
100	9.853	6.1684	3.6887	3.7583	6.5186
200	13.3993	9.126	6.451	7.1599	10.9839
300	15.5628	10.5351	9.08	10.8206	16.8723
400	17.7699	13.6263	11.9285	13.0231	23.5019
500	19.1542	14.303	15.349	16.3707	31.7831
600	20.4675	18.8677	18.571	20.0113	37.3242
700	23.4671	20.1617	20.9039	23.7305	40.66
800	25.07	23.3143	24.9978	26.3178	44.8019
900	29.3611	28.1234	26.6288	29.6816	52.8829
1000	30.5498	28.7635	29.5586	33.5195	62.8967
1100	32.2825	32.0137	30.7372	37.2753	65.2363
1200	34.3818	31.8152	34.0371	42.1325	72.2926
1300	37.3422	35.8219	37.4416	40.3347	75.2937
1400	38.4452	41.3896	38.6274	41.3493	81.1193
1500	40.0487	43.225	43.4606	43.9233	81.3101