



MATEMATICKO-FYZIKÁLNÍ FAKULTA

Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Tomáš Karella

Evoluční algoritmy pro řízení heterogenních robotických swarmů

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Martin Pilát, Ph.D.

Studijní program: Informatika

Studijní obor: Programování a Softwarové Systémy

Praha 2017

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V dne

Podpis autora

Poděkování. Poděkovat Martinovi, Milošovi, Arniimu, Metacentru za počítače

Název práce: Evoluční algoritmy pro řízení heterogenních robotických swarmů

Autor: Tomáš Karella

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Martin Pilát, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Abstrakt.

Klíčová slova: klíčová slova

Title: Evolutionary Algorithms for the Control of Heterogeneous Robotic Swarms

Author: Tomáš Karella

Department: Department of Theoretical Computer Science and Mathematical Logic at Faculty of Mathematics and Physics

Supervisor: Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science Mathematical Logic

Abstract: Abstract.

Keywords: key words

Obsah

Úvod	2
1 Evoluční algoritmy	3
1.1 Historie	3
1.2 Co je evoluční algoritmus?	3
1.3 Části evolučních algoritmů	5
1.4 Diferenciální evoluce	9
1.5 Evoluční strategie	10
2 Robotický swarm	12
2.1 Použití	13
2.2 Řízení robotických swarmů	14
3 Simulátor	19
4 Experimenty	22
4.1 Použité technologie	23
4.2 Wood Scene	25
4.3 Mineral Scene	42
4.4 Competitive Scene	59
4.5 Shrnutí	74
Závěr	75
Seznam použité literatury	76
Seznam obrázků	78
Seznam tabulek	79
Přílohy	80

Úvod

Využití robotických hejn (robotic swarms) patří mezi rentabilní metody pro řešení složitějších úkolů. Existuje řada studií potvrzujících, že velký počet jednoduchých robotů dokáže plně nahradit komplexnější jedince. Dostatečná velikost hejna umožní řešení úloh, které by jednotlivec z hejna provést nesvedl. Navíc robotické hejno přináší několik výhod díky kvantitě jsou odolnější proti poškození i po zničení některých z nich zbytek robotů pokračuje v plnění cílů. Dále výroba jednodušších robotů vychází levněji než komplexní jedinců, což přináší nezanebatelnou výhodu pro práci v nebezpečném prostředí. Hejno také může pokrývat více různých úkolů než specializovaný robot, který bude při plnění úkolů, lišících se od typu úloh zamýšlených při konstrukci, mnohem více nemotorný a nejspíše pomalejší. Hejno pokryje větší plochu při plnění úkolů.

Existuje mnoho aplikací robotických hejn, většinou se používají v úlohách týkajících se průzkumu a mapování prostředí, hledání nejkratších cest, nasazení v nebezpečných místech (Jevtić, 2007). Jako příklad můžeme uvést asistenci záchranným složkám při požáru (Penders a kol., 2011). Mnoho projektů zabývající se řízením robotického hejna se inspiruje přírodou, používá se analogie s chováním mravenců a jiného hmyzu (David a kol., 2001). Objevují se i hardwarové implementace chování hejn, zmiňme projekty Swarm-bots (Dorigo, 2001), Colias (Arvin a kol., n.d.)

Elementárnost senzorů i efektorů jednotlivých robotů vybízí k použití evolučních algoritmů, jelikož prostor řešení je rozlehly a plnění cílů lze vhodně ohodnotit. Vzniklo několik vědeckých prací popisující problematiku tohoto tématu (Gomes a kol., 2013) (Ivan a kol., 2013).

Cíl práce

Všechny zmíněné práce používají pro tvorbu řídicích programů evoluční algoritmy (EA) a pracují pouze s homogenními hejny. Cílem této práce je vyzkoušet využití EA na generování chování hejna heterogenních robotů, tedy skupiny agentů, ve které se objevuje několik druhů jedinců a společně plní daný úkol. V rámci práce byl vytvořen program pro simulaci různých scénářů. Pro otěstování jejich úspěšnosti v rámci EA, byly zvoleny 3 odlišné scénáře, ve kterých se objevují 2-3 druhy robotů.

Struktura práce

Rozdelení práce je následující. První kapitola je věnována obecnému úvodu do tématiky evolučních algoritmů, kde se podrobněji věnuji evolučním strategiím a diferenciální evoluci, protože oba tyto postupy implementuji v programu pro řešení scénářů. Druhá kapitola se zabývá představením robotického hejna, základním principům a několika konkrétnějšími aplikacími. Ve třetí kapitole je nastíněno fungování simulátoru, přiložené dokumentace obsahuje podrobnější informace o implementaci simulátoru. Všechny provedené experimenty se všemi detaily obsahuje čtvrtá kapitola.

1. Evoluční algoritmy

1.1 Historie

Začněme pohledem do historie Evolučních algoritmů na základě knih Mitchell (1998) a Eiben (2015). Darwinova myšlenka evoluce lákala vědce už před průlomem počítačů, Turing vyslovil myšlenku *genetického a evolučního vyhledávání* už v roce 1948. V 50. a 60. letech nezávisle na sobě vznikají 4 hlavní teorie nesoucí podobnou myšlenku. Společným základem všech teorií byla evoluce populace kandidátů na řešení daného problému a jejich následná úprava způsoby hromadně nazývanými jako genetické operátory, například mutace genů, přirozená selekce úspěšnějších řešení, atp.

Rechenberg a Schwefell (1965, 1973) představili *evoluční strategie*, což je metoda optimalizující parametry v reálných číslech, v jejich práci se objevují jako prostředek pro optimalizaci tvaru letadlových křídel. Fogel, Owens, Walsh zveřejnili *evolutionary programming* (evoluční programování), technika využívající k reprezentaci kandidátů konečný automat (s konečným počtem stavů), který je vyvíjen mutací přechodů mezi stavy a následnou selekcí. *Genetické algoritmy* vynalezl Holland v 60. letech a následně se svými studenty a kolegy z Michiganské Univerzity vytvořil první implementaci. U genetických algoritmů, oproti ES a EP, nebylo hlavním cílem formovat algoritmus pro řešení konkrétních problémů, ale přenos obecného mechanismu evoluce jako metody aplikovatelné v informatickém světě. Princip GA spočívá v transformaci populace chromozomů (vektor 1 a 0) v novou populaci pomocí genetických operátorů křížení, mutací a inverze. V 1975 v knize *Adaptation in Natural and Artificial Systems* (Holland, 1976) Holland definoval genetický algoritmus jako abstrakci biologické evoluce spolu s teoretickým základem jejich používání. Někteří vědci ovšem používají pojmenování GA i ve významně hodně vzdálených původní Holandově definici. K sjednocení jednotlivých přístupů přispěl v 90. letech John R. Koza, dále jsou všechny metody zahrnutý pod pojmem *Evoluční algoritmy* jako jejich součásti. Dnes se věnuje tématu EA celá řada konferencí a odborných časopisů. Zmiňme ty nejvýznamnější z nich GECCO, PPSN, CEC a EVOSTAR. V rámci odborných časopisů pr. především o Evolutionary Computation, IEEE Transactions on Evolutionary Computation, Genetic Programming and Evolvable Machines, Swarm and Evolutionary Computation.

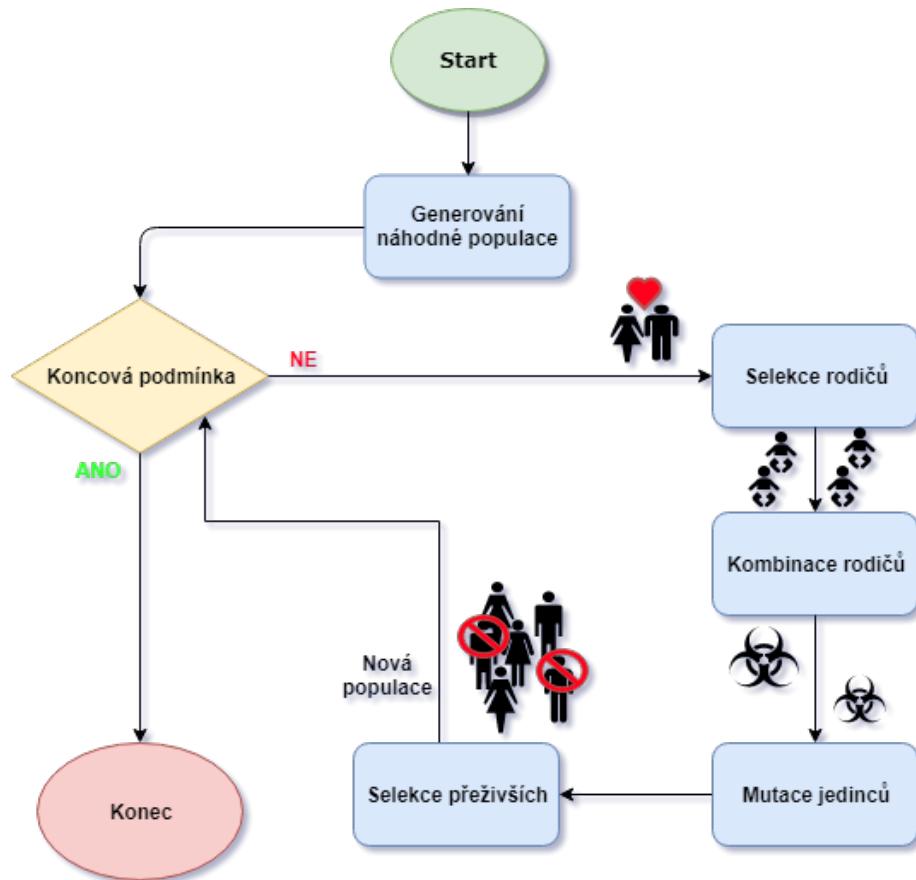
1.2 Co je evoluční algoritmus?

Ač existuje mnoho variant evolučních algoritmů, jak jsme zmínili v krátkém pohledu do historie, spojuje je společná myšlenka populace jedinců uvnitř prostředí s omezenými podmínkami. Jedinci, jinak také nazývání kandidáti, soutěží o zdroje daného prostředí, tím je docíleno přírodní selekce (přežijí jen ti nejlepší). Pokud budeme mít k dispozici kvalitativní funkci, kterou se snažíme maximizovat, pak nebude problém vytvořit náhodné jedince z definičního oboru přesně této funkce. Náhodně vzniklé jedince můžeme ohodnotit, tímto způsobem dáme vzniknout abstrakci pro měření *fitness* (z anglického fit: nejvhodnější, zapadající). Z vzniklých a ohodnocených jedinců lze zvolit ty nejlepší pro tvorbu nové

generace jedinců. Tvorba nové generace probíhá kombinováním zvolených rodičů a mutacemi jedinců. Jako kombinaci uvažujeme operátor, který je aplikován na dva a více zvolených kandidátů (proto se jim také říká rodiče) a tvořící jednoho a více nových jedinců (také nazývány potomci). Mutace je aplikována pouze na jednoho jedince a její výsledkem je také pouze jeden jedinec. Tyto dvě operace aplikované na rodičovskou generaci vedou k vytvoření nových kandidátů (potomků, offspring). I tato nová generace je ohodnocena fitness a dále soutěží se starými jedinci na základě fitness o místo v nové generaci, občas také mimo fitness funkce bereme v potaz stáří kandidáta. Popsaný proces je opakován dokud není nalezen kandidát s dostatečně velkou fitness nebo dosáhneme maximálního počtu iterací (bylo dosaženo požadovaného počtu opakování apod.). Základy evolučního systému pohání dvě základní hnací síly: *variační operátory* a *selektivní operátory*. Variační operátory zajišťují potřebnou různorodost v populaci a tím tvoří nové cesty k úspěšnému kandidátovi. Oproti tomu selektivní operátory zvyšují průměrnou kvalitu řešení v celé populaci. Kombinací těchto operátorů obecně vede ke zlepšování fitness hodnot v následující generaci. Funkčnost evoluce ověříme snadno, stačí nám k tomu pozorovat zda se fitness v populaci blíží více a více k optimálním hodnotám vzhledem k postupu v čase. Mnoho komponent zapříčiňuje, že EA se řadí ke stochastickým metodám, selekce totiž nevybírá nejlepší jedince deterministicky, i jedinci s malou fitness mají šanci být rodiči následující generace. Během kombinování jsou části rodičů, které budou zkombinovány, také zvoleny pomocí dané pravděpodobnosti funkce. Podobně je tomu u mutací. Část která bude změněna, je taktéž určena náhodou, stejně tak nové rysy nahrazující staré. EA se řadí mezi populační stochastické prohledávací algoritmy. Vyhodnocení fitness funkce poskytuje heuristický odhad kvality řešení, prohledávání je řízeno variací a selekcí.

Různé dialekty evolučních algoritmů, zmíněné v historickém okénku, splňují všechny tyto hlavní rysy. Liší se pouze v technických detailech. Kandidáti jsou často reprezentování různě, liší se datové struktury pro jejich uchování i jejich zakódování. Typicky se jedná o řetězce nad konečnou abecedou v případě GA, vektory reálných čísel v ES, konečné automaty v EP a stromy v GP. Důvod těchto rozdílů je hlavně historický. Technicky však lze upřednostnit jednu reprezentaci, pokud lépe odpovídá danému problému, tzn. kóduje kandidáta jednodušeji či přirozeněji formou. Pro ilustraci zvolme řešení splnitelnosti (SAT) s n proměnnými, vcelku přirozeně sáhneme po použití řetězce bitů délky n a obsah ítého bitu označuje, že ítá proměnná má hodnotu (1) - pravda, (0) - nepravda, proto bychom použili jako vhodný EA genetické algoritmy. Oproti tomu evoluce programu, který hraje šachy, by bylo vhodnější použít derivační strom, kde by vrcholy odpovídaly tahům na šachovnici. Přirozenější by tedy bylo použít GP.

Neopomeňme zmínit ještě dva důležité fakty. Kombinační a mutační operátory musí odpovídat dané reprezentaci, např. v GP kombinace pracuje se stromy (kombinují podstromy ...), zatímco v GA na řetězcích (prohazují části řetězců). A dále oproti variacím musí fitness selekce záviset vždy pouze na fitness funkci, takže selekce pracuje nezávisle na reprezentaci.



Obrázek 1.1: Obecný evoluční algoritmus - schéma

1.3 Části evolučních algoritmů

Abychom vytvořili běhu schopný evoluční algoritmus, musíme specifikovat každou zmíněnou část a také inicializační funkci, která připraví první populaci. Pro konečný algoritmus ještě nesmíme opomenout a dodat koncovou podmínsku.

- Reprezentace (definice individuí)
- Ohodnocující funkce (fitness funkce)
- Populace
- Selekce rodičů
- Variační operátory (kombinace, mutace)
- Selekce prostředí

Reprezentace

Při tvorbě EA nejdříve musíme propojit prostor původního problému s prostorem řešení, kde se bude vlastní evoluce odehrávat. K docílení propojení je většinou potřeba zjednodušit či vytvořit abstrakci nad aspekty reálného světa (prostor problému), abychom vytvořili vhodný prostor řešení, kde mohou být jednotlivá řešení

ohodnocena. Neboli chceme docílit, aby možná řešení mohla být specifikována a uložena, tak aby s nimi mohl počítač pracovat. Objekty reprezentující možná řešení v kontextu původního problému jsou nazývány *fenotyp*, zatímco kódování na straně EA prostoru se říká *genotyp*. Reprezentace označuje mapování z fenotypů na genotypy, používá se ve významu funkce z fenotypu na genotyp (encode) i genotypu na fenotyp (decode). Při návrhu se snažíme, aby pro každý genotyp existoval nejvýše jeden fenotyp, ne vždy nám to prostor problému umožní. Například pro optimalizační problém, kde množinou řešení jsou celá čísla. Celá čísla tvoří množinu fenotypu a můžeme se rozhodnout pro reprezentaci v binárních číslech. Tedy 18 by byl fenotyp a 10010 jeho genotyp.

Prostor fenotypů a genotypů se zpravidla velmi liší a celý proces EA se odehrává pouze v genotypovém prostoru, vlastní řešení dostaneme rozkódováním nejlepšího genotypu po ukončení EA. Jelikož nevíme, jak vlastní řešení vypadá, je nanejvýš vhodné umět reprezentovat všechna možná řešení.

- V kontextu původního problému jsou následující výrazy ekvivalentní: fenotyp, kandidát (na řešení), jedinec, individuum (množina všech možných řešení = fenotypový prostor)
- V kontextu EA: genotyp, chromozom, jedinec, individuum (množina, kde probíhá EA prohledávání = genotypový prostor)
- Části individuí jsou nazývány gen, locus, proměnná a dále se dělí na alely, či hodnoty

Populace

Populace je multimnožina genotypů, slouží jako jednotka evoluce. Populace utvářejí adaptaci a změny, zatímco vlastní jedinci se nijak nemění, jen vznikají noví a nahrazují předešlé. Pro danou reprezentaci je definice populace velmi jednoduchá, charakterizuje ji pouze velikost. U některých specifických EA má populace další prostorové struktury, definované vzdáleností jedinců nebo relacemi sousedních jedinců, což by se dalo připodobnit reálným populacím, které se vyvíjejí v různých geografických prostředích. U většiny EA se velikost populace nemění, což vede k soutěživosti mezi jedinci (zůstanou ti nejlepší). Na úrovni populací pracují právě selektivní operátory. Například zvolí nejlepší jedince aktuální populace jako rodiče následující a nahradí nejhorské jedince novými. Rozmanitost populace - vlastnost, které chceme z pravidel docílit, je měřena jako počet různých řešení v multimnožině. Neexistuje však jediné hledisko, podle kterého lze tuto vlastnost měřit, většinou se používá počet rozdílných hodnot fitness, rozdílných fenotypů či genotypů a také entropie (míra neuspořádanosti). Ovšem musíme mít na paměti, že jedna hodnota fitness v populaci neznamená, že populace obsahuje pouze jeden fenotyp, stejně tak jeden fenotyp nemusí odpovídat jednomu genotypu apod.

Selekce rodičů

Rodičovská selekce, někdy také partnerská selekce, vybírá lepší jedince jako rodiče pro příští generaci. Jedinec se stává rodičem, pokud byl zvolen k aplikaci variačních operátorů a tím dal vzniknout novým potomkům. Společně se selekcí

přeživších je rodičovská selekce zodpovědná za zvedání kvality v populacích. Rodičovská selekce je v EA většinou pravděpodobnostní metoda, která dává jedincům s větší kvalitou mnohem větší šanci být vybrán než těm s nízkou. Nicméně i jedincům s nízkou kvalitou je často přidělena malá nenulová pravděpodobnost pro výběr, jinak by se celé prohledávání mohlo vydat slepou cestou a uvíznout v lokálním optimu.

Variační operátory

Hlavní úlohou variačních operátorů (mutace, rekombinace) je vytváření nových individuí ze starých. Z pohledu generate test algoritmů spadají variační operátory do právě do generate části. Obvykle je dělíme na dva typy podle jejich arity, jedná se o unární (Mutace) a nární (rekombinace) operátory.

Mutace

Ve většině případů myslíme unárním variačním operátorem mutace. Tento operátor je aplikován pouze na jeden genotyp a výsledkem je upravený potomek. Mutace řadíme mezi stochastické metody, výstup (potomek) totiž závisí na sérii náhodných rozhodnutí. Mutace však není vhodné pojmenování pro všechny variační operátory, například pokud se jedná o heuristiku závislou na daném problému, která se chová systematicky, hledá slabé místo a následně se jej snaží vylepšit, v tomto případě se nejedná o mutaci v pravém slova smyslu. Obecně by mutace měla způsobovat náhodné a nezaujaté změny. Unární variační operátory hrají odlišnou roli v rozdílných EA opět díky historicky oddělenému vývoji. V GA mají velmi důležitou roli a v EP se jedná dokonce o jediný variační operátor. Díky variačním operátorům aplikovaným v jednotlivých evolučních krocích dostává prohledávací prostor topologickou strukturu. Existují teorie podporující myšlenku, že EA (s dostatečným časem) naleznou globální optimum daného problému opírající se právě o tuto topologickou strukturu a také spoléhají na fakt, že může vzniknout každý genotyp reprezentující možné řešení. Nejjednodušší cesta ke splnění těchto podmínek vede právě přes variační operátory. U mutací tohoto například dosáhneme, když povolíme, aby mohly skočit kamkoliv, tzn. každá alela může být zmutována na jakoukoli další s nenulovou pravděpodobností.

Rekombinace

Rekombinace, také nazývána křížení, je binární variační operátor. Jak název napovídá, spojuje informace ze dvou rodičů (genotypů) do jednoho nebo dvou potomků. Stejně jako mutace patří rekombinace k stochastickým operátorům. Rozhodnutí, jaké části budou zkombinovány a jakým způsobem toho bude docíleno, závisí na náhodě. Role rekombinace se znova liší v rozličných EA, v GP se jedná o klíčové variační operátory, v EP nejsou použity vůbec. Existují rekombinační operátory s vyšší aritou (tzn. používající více než dva rodiče) a jsou také jednoduché na implementaci. Dokonce několik studií potvrdilo, že mají velmi pozitivní vliv na celou evoluci, ale nemají tak hojně zastoupení, nejspíše proto, že neexistuje biologický ekvivalent. Rekombinace funguje na jednoduchém principu, slučuje dvě individua a produkuje jednoho či více potomků, kteří kombinují jejich vlastnosti, tedy potomek by měl být úspěšnější než jeho rodiče. Tento princip

podporuje fakt, že po tisíciletí se aplikací rekombinace na rostliny a zemědělská zvířata mnohokrát podařilo vytvořit nové jedince s vyšším výnosem či jinými výhodnými vlastnostmi (odolnost proti škůdcům atd.). Evoluční algoritmy vytváří množství potomků náhodnou rekombinací a doufáme, že zatímco malá část nové generace bude mít nežádoucí vlastnosti, většina se nezlepší ani si nepohorší a konečně další malá část předčí jejich rodiče. Na naší planetě se sexuálně (kombinací dvou jedinců) rozmnožují pouze vyšší organismy, což vzbuzuje dojem, že rekombinace je nejvyšší forma reprodukce. Neopomeňme, že variační operátory jsou závislé na reprezentaci (genotypu) jedinců. Např. pro genotypy bitových řetězců může použít bitovou inverzi jako vhodný mutační operátor bude-li ovšem genotyp strom musíme zvolit nějaký jiný.

Selekce prostředí

Selekce prostředí, někdy také nazývána selekce přeživších, jak název napovídá má blízko k selekci rodičů, odlišuje jednotlivá individua na základě jejich fitness hodnoty. Oproti rodičovské selekci ji však používáme v jiné části evolučního cyklu. Selekcí prostředí spouštíme hned po vytvoření nových potomků. Jelikož velikost populace kandidátů bývá skoro vždy konstantní, je nutné rozhodnout, které kandidáty zvolíme do další generace. Toto rozhodnutí většinou závisí na hodnotě fitness jednotlivých kandidátů, také se často hledí na dobu vzniku daného kandidáta. Dobou vzniku myslím generaci v které daný jedinec vznikl. Na rozdíl od rodičovské selekce, která bývá stochastická, bývá selekce prostředí deterministickou metodou. Uvedme dvě nejběžnější metody, obě kladou největší důraz na fitness. První vybírá nejlepší segment z množiny nových potomků i z původní generace nezávisle, druhá dělá totéž jen z množiny nových potomků. Selekcí prostředí je uváděna i pod názvem „záměnná“ strategie (replacement strategy). Selekcí prostředí (přeživších) se víceméně používá, pokud je množina nových potomků větší než velikost generace a záměna využíváme když je nových potomků velmi málo.

Inicializace populace

Inicializace populace zastává důležitý úkol a to vytvořit první generaci kandidátů. Její implementace bývá ve většině EA velmi jednoduchá. První generace je vygenerována čistě náhodně. Principiálně by se zde dalo využít nějaké heuristiky k vytvoření vyšší fitness v první generaci, ovšem musela by zohledňovat řešený problém. Jestli tento postup stojí za výpočetní čas, velmi záleží na konkrétní aplikaci EA. Ovšem existují obecná doporučení, která se touto otázkou zabývají.

Ukončovací podmínka

Rozlišme dvě varianty vhodné ukončovací podmínky. Pokud řešený problém má známou optimální hodnotu fitness, potom v ideálním případě ukončovací podmínkou je řešení s touto fitness. Pokud jsme si vědomi, že náš model oproti modelovanému prostředí obsahuje nutná zjednodušení nebo by se v něm mohly vyskytovat nežádoucí šumy, lze se spokojit s řešením, které dosáhlo optima fitness s přesností $\epsilon > 0$. EA díky své stochastičnosti většinou nemohou garantovat dosažení takovéto hodnoty fitness, takže ukončovací podmínka by nemusela být nikdy

splněna a algoritmus by běžel věky. Kdybychom vyžadovali konečnost algoritmu, musíme rozšířit ukončovací podmínsku. Za tímto účelem se nejběžněji používají následující rozšíření:

1. Byl překročen čas vyhrazený pro počítání na CPU
2. Celkový počet evaluací fitness přesáhl svůj předem daný limit
3. Zlepšení fitness v dané časovém bloku (měřenému počtem generací, či evaluací) klesla pod přípustný práh
4. Rozmanitost v populaci nedosahuje předem určených hodnot

Ukončovací podmínka bývá prakticky disjunkce předchozích tvrzení. Může se stát, že optimum známé není. Pak se používá pouze zmíněných podmínek nebo se smíříme s nekonečností algoritmu a ukončíme jeho běh manuálně.

1.4 Diferenciální evoluce

První verze diferenciální evoluce se objevila jako technický report Storn a Price (1997). Tento evoluční algoritmus popíší podrobněji, protože je jedním z klíčových algoritmů mého řešení.

Charakteristické vlastnosti

Diferenciální evoluce dostala své jméno hlavně díky změnám obvyklých reprodukčních operátorů v EA. Diferenciální mutace, jak jsou zde mutace nazývány, z dané populace kandidátů vektorů v \mathbb{R}^n vzniká nový mutant \bar{x}' přičtením pertubačního vektoru (perturbation vector) k existujícímu kandidátovi.

$$\bar{x}' = \bar{x} + \bar{p}$$

Kde pertubační vektor \bar{p} je vektor rozdílů dvou náhodně zvolených kandidátů z populace \bar{y} a \bar{z} .

$\bar{p} = F \cdot (\bar{y} - \bar{z})$. Faktor škálování $F > 0$, $F \in \mathbb{R}$, který kontroluje míru mutace v populaci. Jako rekombinační operátor v diferenciální evoluci slouží uniformní křížení, pravděpodobnost křížení je dána $C_r \in [0,1]$, definuje šanci jakékoli pozice v rodiči, že odpovídající alela potomka bude brána z prvního rodiče. Diferenciální evoluce také upravuje křížení, neboť jedna náhodná alela je brána vždy z prvního rodiče, aby nedocházelo k duplikaci druhého rodiče.

V hlavních implementacích diferenciální evoluce reprezentují populace spíše listy, odpovídají lépe než množiny, umožňují referenci *itého* jedince podle pozice $i \in 1, \dots, \mu$ v listu. Pořadí kandidátů v této populaci $P = < \bar{x}_1 \dots \bar{x}_i \dots \bar{x}_\mu >$ není závislé na hodnotě fitness. Evoluční cyklus začíná vytvořením populace vektorů mutantů $M = < \bar{v}_1 \dots \bar{v}_\mu >$. Pro každého nového mutantanta \bar{v}_i jsou zvoleny 3 vektory z P , base vektor a dva definující pertubační vektor. Po vytvoření populace mutantů V , pokračujeme tvorbou populace $T = < \bar{u}_1, \dots, \bar{u}_\mu >$, kde \bar{u}_i je výsledek křížení \bar{v}_i a \bar{x}_i (všimněme si, že je zaručené, že nezreplikujeme \bar{x}_i). Jako poslední aplikujeme selekci na každý pár \bar{x}_i a \bar{u}_i a do další generace vybereme \bar{u}_i pokud $f(\bar{u}_i) \leq f(\bar{x}_i)$ jinak \bar{x}_i .

DE algoritmus upravují 3 parametry, škálovací faktor F , velikost populace μ (obvykle značen NP v kontextu diferenciální evoluce) a pravděpodobnost křížení

C_r . Na C_r lze také pohlížet jako na míru mutace, tzn. alela nebude převzata od mutanta.

Vlastnosti diferenciální evoluce:	
Reprezentace:	vektor \mathbb{R}^n
Rekombinace:	uniformní křížení
Mutace:	diferenční mutace
Rodičovská selekce:	uniformní náhodné selekce 3 nezbytných vektorů
Selekce prostředí:	deterministická selekce elity (dítě vs rodič)

Tabulka 1.1: diferenciální evoluce - souhrnné vlastnosti

Varianty diferenciální evoluce:

Během let vzniklo a bylo publikováno mnoho variant diferenciální evoluce. Jedna z modifikací mění výběr base vektoru, když vytváříme populace mutantů M . Base vektor může být náhodně zvolen pro každé v_i , jak bylo řečeno, ale také lze využít jen nejlepšího vektoru a nechat změny na pertubačním vektoru. Další možnost otevírá použití více pertubačních vektorů v mutačním operátoru. Což by vypadalo následovně: $\bar{p} = F(\bar{y} - \bar{z} + \bar{y}' - \bar{z}')$, kde $\bar{y}, \bar{z}, \bar{y}', \bar{z}'$ jsou náhodně vybrány z původní populace.

Abychom odlišili různé varianty, používá se následující notace DE/a/b/c, kde a specifikuje base *vektor(rand, best)*, b specifikuje počet diferečních vektorů při vzniku pertubačního vektoru, c značí schéma křížení (bin=uniformní). Takže podle této notace by základní verze diferenciální evoluce byla zapsána takto: DE/rand/1/bin.

1.5 Evoluční strategie

Evoluční strategie (evolution strategies) byly představeny v německém článku Rechenberg (1981), původně byly určeny pro optimalizaci tvarů křídel. Jedná se o evoluční algoritmus cílící na optimalizaci vektorů reálných čísel. Objevili si dvě schémata (1+1) a (1, 1). Starší (1+1) (one plus one ES) vytvářejí potomka mutací a to přičtením náhodných nezávislých hodnot ke složkám rodičovského vektoru, následně je potomek přijat, pokud získal lepší fitness než jeho rodič. Jako další vznikl (1,1) (one comma one ES), které neimplementovalo elitismus, tzn. měnila vždy rodiče potomkem. Mutační funkce bere náhodná čísla z Normálního rozdělení se střední hodnotou 0 a odchylkou σ . Pro velikost mutačního operátoru (mutation step size) se také používá symbol: σ . Vylepšení původního algoritmu bylo představeno v 70. letech, jedná se o koncept multisložkových ES, které se skládají ze μ jedinců (velikost populace) a λ jedinců vygenerovaných během jednoho cyklu. Opět tento koncept existuje ve dvou verzích (μ, λ) a $(\mu + \lambda)$. Zatímco běžné rekombinační schéma zahrnuje dva rodiče a jednoho potomka, intermediate křížení, které se u ES používá, zprůměruje hodnoty z rodičovských alel. Tímto způsobem můžeme používat i rekombinační operátory s použitím více než dvou rodičů, tyto operátory se nazývají globální rekombinace. Konkrétně se na potomkovi podílí λ rodičů. V praxi se preferuje (μ, λ) před $(\mu + \lambda)$, protože zahazuje všechny rodiče, tím pádem se snadno nezastaví na lokálním optimu, (μ, λ)

se zvládá lépe adaptovat i při hledání pohyblivého optima. Pro typické použití se používá poměr 1:4 a 1:7.

Vlastnosti evolučních strategií:

Reprezentace:	vektor \mathbb{R}^n
Rekombinace:	intermediary křížení
Mutace:	přičítání náhodných hodnot z norm. rozdělení
Rodičovská selekce:	uniformní náhodná
Selekce prostředí:	(sigma,lambda) nebo (sigma + lambda)

Tabulka 1.2: evoluční strategie - souhrnné vlastnosti

2. Robotický swarm

Myšlenka robotického swarmu pochází podobně jako u genetických algoritmů z inspirace matkou Přírodou. Podle souhrnu (Tan a Zheng, 2013) popíší základní myšlenku RS. Pro robotický swarm se také používá výraz rojová robotika či robotický roj, v angličtině je známý pod pojmem swarm robotics.

Základní vlastnosti

Motivací pro použití RS může být chování živočichů na Zemi. Zaměříme se na skupiny živočichů jako jsou mravenci, včely, ryby dokonce i některé savce. Pokud bychom vložili do prostředí jednotlivce z některé ze zmíněných skupin, nebude schopen konkurovat nepřátelskému prostředí a nejspíše příliš dlouho nepřežije. Na druhou stranu, když budeme uvažovat celé společenství, tak se nám ze slabého jedince stane velmi adaptivní, odolný a rychle se vyvýjející roj. Podobnému účinku bychom se chtěli přiblížit v RS. Pro relativně jednoduchého robota, který není schopen plnit obtížný úkol, se pokusíme použít vícero robotů stejného typu, kteří společně zadáný úkol vyřeší. Navíc chceme těžit ze všech výhod hejna.

Jako nejčastější výhody RS oproti jednomu robotovi se nejčastěji uvádějí:

1. Paralelita - Díky malé ceně jedince, si můžeme dovolit velkou populaci jedinců. Malou cenou jedince v ES myslíme, že se jedná o jednoduchého robota s nízkou pořizovací cenou. V kontextu živočichů můžeme uvažovat množství energie, jídla pro tvorbu takového jedince. Velká populace nám umožňuje řešit vícero úkolů naráz, také na velké ploše. Zvláště pro vyhledávací úkoly ušetříme nemalé množství času.
2. Škálovnatelnost - Změna velikosti populace hejna neovlivní chování ostatních jedinců. Samozřejmě plnění úkolu bude rychlejší či pomalejší, ale původní hejno bude stále plnit původní úkol. Tím pádem můžeme celkem snadno upravovat velikost populace bez větší obtíží. V přírodě můžeme pozorovat, že smrt jednotlivých mravenců-dělníků znatelně neovlivní práci celého mraveniště. Nově narození mravenci se mohou vydat do práce, zatímco zbytek mraveniště nemění činnost.
3. Houževnatost - Související se škálovatelností, jen v tomto případě máme na mysli necílenou změnu populace. Jako v předchozím příkladu, u smrti mravenců, část robotů ES může selhat z rozličných důvodů, zbytek hejna však bude pokračovat k cíli, i když ve výsledku jim bude jeho dosáhnutí trvat o něco déle. To se nám může vyplatit v nebezpečných prostředích.
4. Ekonomické výhody - Cena návrhu a konstrukce jednoduchých hejn robotů vyjde většinou levněji než jeden specializovaný robot schopný uspokojit stejné požadavky. V dnešním světě vychází výroba ve velkém množství mnohem levněji než tvorba jednoho drahého konkrétního robota.
5. Úspora energie - Díky menší velikosti a složitosti jednotlivých robotů vyžadují mnohem menší množství energie. To má za důsledek, že si u nich

můžeme dovolit energetickou rezervu na delší čas. Navíc když je pořizovací cena jednoho robota menší než náklady na dobití, můžeme díky škálovatelnosti pouze připojit nové roboty, což u drahého robota jde málokdy.

6. Autonomie a decentralizace - V kontextu RS musí každý jedinec hejna jednat autonomně, jedinci nejsou řízeni žádnou autoritou. Takže umí pracovat i při ztrátě komunikace. Opět se vychází z chování živých organismů. Pokud se chovají jedinci hejna dostatečně kooperativně, mohou pracovat bez centrálního řízení, důsledkem toho se stává celé hejno ještě flexibilnější a odolnější, hlavně v prostředích s omezenou komunikací. Navíc hejno mnohem rychleji reaguje na změny.

Mimo RS existuje i řada jiných přístupů, které se inspirovaly životem hejn v přírodě. Občas jsou zaměňovány za RS, nejčastěji se jedná o multi-agentní systémy a senzorové sítě (sensor networks). V následující tabulce jsou popsány jejich nejklíčovější vlastnosti.

	Robotická hejna	Multi-robotické systémy	Senzorové sítě	Multi-agentní systémy
Velikost populace	Variabilní ve velkém rozsahu	Malá	Fixní	V malém rozsahu
Řízení	decentralizované a autonomní	centralizované	centralizované	centralizované
Odlíšnosti	většinou homogenní	většinou heterogenní	homogenní	homogenní nebo heterogenní
Flexibilita	vysoká	nízká	nízká	střední
Škálovnatelnost	vysoká	nízká	střední	střední
Prostředí	neznámé	známé nebo neznámé	známé	známé
Pohyblivost	ano	ano	ne	vyjímceně

Tabulka 2.1: Porovnání systémů s více agenty

Jednotlivé systémy s více agenty se také liší svojí aplikací. Robotická hejna se nejčastěji používají ve vojenských, nebezpečných úkolech a také pro řešení ekologických katastrof. Multi-robotické systémy potkáváme v transportních, skenovacích úkolech, dále pro řízení robotických fotbalových hráčů. Oproti tomu nejčetnější využití senzorových sítí zasahuje do medicínské oblasti, ochrany životního prostředí. Multi-agentní systémy zase nejvíce zasahují do řízení síťových zdrojů a distribuovaného řízení.

2.1 Použití

Existuje několik vědeckých prací, které studují a navrhují použití RS v reálném nasazení.

Některé jsem zmínil už v úvodu této práce, jako například hasičům asistující roboty (Penders a kol., 2011). Zde si robotické hejno klade za cíl usnadnit a podpořit navigaci lidem v nebezpečném prostředí. Jejich využití je ilustrováno záchrannou misí ve velkém skladišti. Hasiči mají díky kouři velmi omezenou viditelnost. Tím pádem se lokalizace přeživších, epicenter požárů a další důležitých bodů stává obtížným a zdraví ohrožujícím úkolem. Nezřídka se stává, že zasahující hasič zahyne, protože se v hustém dýmu v objektu ztratil. Robotické hejno tedy může prozkoumat celý prostor před vlastním zásahem. Při zásahu ještě asistovat hasičům při orientaci v prostoru.

Robotická hejna se také ukázala jako užitečná u ekologický pohrom. Španělští vědci testovali jejich použití při úniku ropy (Aznar a kol., 2014), či při hledání centra radiace (Bashyal a Venayagamoorthy, 2008).

V prvně zmíněném příkladu autoři mapovali znečištění mořské vody. Dokonce při plavbách bez defektů se do moře uvolňuje palivo, ropa a další nebezpečné látky. Očekává se, že s rostoucí námořní dopravou se rozrostou tyto lokální znečištění ve vážnou hrozbu. Aktuální systémy monitorující znečištění při katastrofách tankerů jsou pro toto využití příliš drahé. Autoři proto navrhují použití hejna dronů, které bude dokumentovat velké vodní plochy a bude schopno odhalovat případné nebezpečí a větší koncentrace cizích látek. Dokonce budou moci na základě získaných informací sledovat znečistovatele.

Po jaderné katastrofě se stává explorace zamořené oblasti v podstatě nemožným úkolem pro lidské průzkumníky. Právě monitorování radiací postiženým územím se stala motivací pro práci (Bashyal a Venayagamoorthy, 2008). Ve zmiňované práci se autoři soustředí na porovnávání autonomních robotických hejn a RS komunikujících s člověkem. V rámci výsledků ukazují, výhody použití robotického hejna pro hledání centra radiace a také že RS interagující s lidmi dosahují lepších výsledků.

Několik prací nezůstalo pouze u simulací a také využívali RS u fyzických robotů. Hlavní motivací pro práci (Duarte a kol., 2016) byl fakt, že většinou se řízení RS vytváří a hlavně testuje pouze v uzavřeném a simulovaném prostředí. Tvůrci se rozhodli pro reálné použití na moři, kde nemohou prostředí jakkoli ovládat či kontrolovat. Snaží se tím ukázat, že i přes šumy a neočekávané situace, RS je stabilní a použitelné pro aplikaci ve skutečném světě. Celé hejno se skládalo z deseti robotů. Jednalo se o malé lodičky s délkou přibližně 60 cm a nízkou pořizovací cenou okolo 300 euro. Každý robot byl vybaven GPS, WIFI, kompasem. Chování bylo připraveno pomocí evolučních algoritmů v simulaci, konkrétně autoři používají neuroevoluci NEAT, jejich simulace obsahovala 4 podúkoly: navádění, shlukování, rozptylování a monitorování prostředí. Poté bylo stvořené řízení ohodnoceno na vodní ploše. Prezentované výsledky vypadají velmi slibně, chování a úspěšnost řízení se velmi blíží mezi simulací a reálným nasazením. Také potvrzují přítomnost výhodných vlastností ze simulaci v reálném světě, jedná se o robustnost, flexibilitu, škálovatelnost. V neposlední řadě také úspěšnost skládání jednoduchých podúkolů do komplexního chování v rámci hejna, které řeší složitý hlavní cíl.

2.2 Řízení robotických swarmů

Chování swarmů se řadí mezi velmi obtížné úkoly pro svět informatiky. Pro reprezentaci chování se využívá *neuronových sítí*, které se optimalizují pomocí nastavování vah jednotlivých perceptronů, neboť se jedná o velký prostor vstupních informací ze senzorů a prostor pro interakci s prostředím je taktéž velmi rozsáhlý. Přímé prohledávání takto obřího prostoru nepřichází v úvahu, proto v poslední době získávají na oblibě evoluční algoritmy. Mezi nejvíce používané patří evoluční strategie, či genetické programování.

Genetické programování a stromy chování

V práci „Evolving behaviour trees for Swarm robotics“ (Jones a kol., 2018) se autoři zaměřují na využití genetického programování pro vytvoření chování robotického hejna. Pro řízení hejna navrhují vcelku zajímavé využití stromů chování (SC)(behaviour tree), které mají uplatnění především v herním průmyslu pro akce charakterů, které nejsou ovládány hráčem. Jako optimalizační algoritmus zvolili genetické programování.

Strom chování je strom, jehož listy interagují s prostředím, vnitřní vrcholy spojují tyto akce dohromady a tvoří rozhodovací a závislostní pravidla. Celý strom je vyhodnocován v pravidelných intervalech, v práci se značí jako *tick*. Opírájí se o článek (Shoulson a kol., 2011), kde bylo ukázáno, že SC může plnohodnotně reprezentovat konečné automaty, dokonce i když budeme používat pravděpodobnosti konečné automaty. Jako jedince z hejna zvolili Kilobot, které byl představen Rubensteinem v (Rubenstein a kol., 2014). Kiloboti se pohybují pomocí dvou vibračních motorů, komunikují přes infračervený kanál, v prostředí se orientují pomocí foto detektoru a signalizují pomocí LED diod s barevným spektrem. Tabulka 2.2 ukazuje, jak byla zjednodušena komunikace s efektory robota a nad nimi optimalizováno SC.

Efektor/Senzor	Read or Write	Popis
motor	W	vypnut, vřed, vlevo, vpravo
přídavná paměť	R&W	libovolná hodnota
vysílač signálu	R&W	vysílá při větší hodnotě než 0.5
přijímač signálu	R	1 pokud přijímá signál
detektor potravy	R	1 pokud světelný snímač vidí potravu
nosič jídla	R	1 pokud nese jídlo
hustota robotů	R	hustota Kilobotů v blízkosti
δh ustota	R	změna v hustotě
$\delta vzdálenost_{potrava}$	R	změna ve vzdálenosti k potravě
$\delta vzdálenost_{hnízdo}$	R	změna ve vzdálenosti k hnizdu

Tabulka 2.2: Parameterizing behavior trees, Motion in Games - podoba stromů

Akce z tabulky 2.2 pak odpovídají listům v SC, vnitřní vrcholy mohou být kompoziční: *seqm*, *selm*, *probm* a tyto mohou mít 2 až 4 syny. Informace procházející mezi vrcholy mohou být následujícího druhu *success*, *failure*, *running*. Zpracovávají informace následujícím způsobem posílají tik do každého syna dokud od nějakého nepřijde hodnota *failure* nebo tik proběhne na všech synech. Pokud proběhne úspěšně tik u všech synů vrací *success*, *failure* jinak. Oproti tomu *selm* vysílá tik, dokud mu nějaký syn nevrátí hodnotu *success* nebo všichni synové provedli tik, pokud se nevrátí jediná hodnota *success*, tak vrací *failure*, v opačném případě *success*. Od obou se liší *probm*, ten s danou pravděpodobností vybere jednoho syna a vrátí jeho odpověď. Vrchol, který má alespoň jednoho syna se statusem *running* vrací stejnou hodnotu.

Vrcholy jen s jedním synem patří do jedné z následujících kategorií: *repeat*, *succeeded*, *failed*. Vrchol *repeat* vrací tik svým synům, s daným počtem pokusů,

dokud nedostane hodnotu *success*. Následující dva vrcholy vrací konstantní odpověď na tik dle svého jména, i přesto pošlou tik svému následníkovi.

Poslední skupinu vrcholu tvoří tzv. akční vrcholy *ml*, *mr*, *mf*, což ve stejném pořadí jsou: zatoč vlevo, zatoč vpravo, jed kupředu. Vrcholy pohybu při prvním tiku vrací *running* při druhém *success* K akčním vrcholům také patří *if*, který slouží k porovnávání jeho dvou synů, pokud porovnání platí vrací *success*. Poslední z akčních vrcholů je *set*, který nastavuje danou hodnotu synům.

V prostředí jsou s konstantní frekvencí prováděny tzv. „update“ cykly. Každý cyklus se skládá ze třech částí po sobě jdoucích částí.

1. Spočítají se hodnoty v synech z vysílaných signálů a prostředí.
2. Na SC je proveden tik, což čte a zapisuje hodnoty do synů.
3. Pohybové motory jsou aktivovány a vysílání je upraveno, oboje dle zapsaných hodnot do synů.

Jako zátěžový test používají obvyklý scénář, který spočívá v hledání potravy a jejím odvážení zpět do hnázda. Fitness se hodnotí podle vzdálenosti doneseného jídla, čím blíže k hnázdu tím lépe. Jako optimalizační metodu autoři vybrali genetické programování a používají DEAP knihovnu (Fortin a kol., 2012). Celá populace velikosti n_{pop} je ohodnocena fitness, každý jedinec se hodnotí podle 10 simulací, každá simulace má jinou startovní konfiguraci. Roboti startují vždy na poli o velikosti 5x5, jejich orientace je vybírána náhodně. Simulaci běží 300 sekund, update cyklus pro vnímání prostředí má frekvenci 8 Hz a u ovladačů s 2Hz. Používané genetické programování implementuje elitismus přenáší n_{elite} nejlepších jedinců do další generace, zatímco zbylá část je zvolena turnajovou selekcí s velikostí t_{size} . Křížící (rekombinační) operátor, který kříží části stromů, je aplikován s pravděpodobností p_{xover} na všechny páry vybrané turnajovou selekcí. Na vzniklé páry se aplikují 3 mutační operátory.

1. S pravděpodobností p_{mutu} je náhodný vrchol stromu vyměněn za nový náhodně vytvořený
2. S pravděpodobností p_{muts} je náhodná větev stromu a je vyměněna za náhodně zvolený terminál (vyskytující se na této větví)
3. S pravděpodobností p_{mutn} je náhodný vrchol vyměněn za nový, ale se stejným počtem argumentů
4. S pravděpodobností p_{mute} je náhodná konstanta vyměněna za jinou náhodnou hodnotu

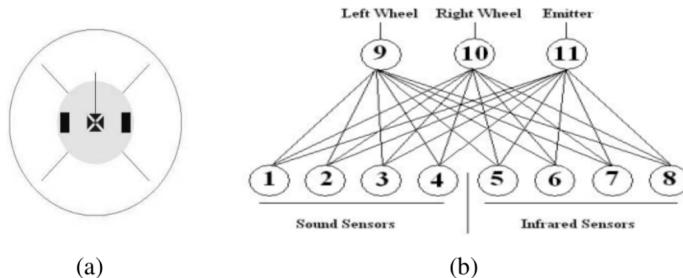
Krom simulace 25 nezávislých běhů evoluce, také byly otestovány algoritmy na reálných strojích. Vytrénované chování bylo otestováno 20 běhy s rozdílnou startovací pozicí a ohodnocení stejnou fitness.

Výsledky simulace byly více než uspokojivé v simulační části si hejno vedlo o trochu lépe 0.075 z maximální hodnoty 0.12 (minimum 0) a co se týče reálného nasazení vygenerovaného chování 0.058. Což opravdu není velký rozdíl, když přihlídneme k tomu, že evoluce probíhala čistě na simulační rovině.

Genetické algoritmy a neuronová síť

Cagri a Yalcin používají ve své práci (Yalcin, 2008) práci neuronových sítí místo SC a pro nastavení vah genetického algoritmu. Shodují se s Winfieldem a jeho kolegy, že evoluce mnoha chování robotického hejna přináší zajímavých strategií, která mohou být mnohem komplexnější než explicitně vytvořené chování. Popisují však také obtížnosti použití evoluce, zvláště volbu evolučního algoritmu a efektivnost celého výpočtu.

Využívají již existujícího simulátoru Cobot2D, pro všechny experimenty bylo použita mapa bez překážek velikosti 400x400. Roboti se pohybují pomocí dvoukolečkových motorů, orientují se 4 infračervenými senzory a 4 zvukovo-směrovými senzory, v jejich středu je umístěn všeobecný zvukový vysílač. Vysílače mají pevný rozsah kruhu vysílání a dynamickou sílu. Zvukové senzory se rozhodují pouze podle signálů, jejichž vysílače spadají do 90° výseče od senzoru a také jejich vzdálenost musí být menší než daná konstanta. Síla signálu se zmenšuje směrem od vysílače a senzory vrací součet sil signálů. Infračervené senzory skenují úsečku dané velikosti a vrací vzdálenost k nejbližšímu průsečíku. V rámci simulace jsou generovány náhodné šumy pro každou interakci s prostředím, aby se simulace přiblížila co nejvíce reálnému nasazení. Pro ovládání robota byla navržena neuronová síť, která má 8 vstupů (4 pro infra-senzory a 4 pro zvukové) a 3 výstupy a není zde žádná skrytá vrstva. Nákres robota a jeho ovládací neuronové sítě můžete vidět na obrázku.



Obrázek 2.1: Cobot2D - nákres robota, zdroj : (Yalcin, 2008)

Fitness funkce: První použitá $fitness_1$ z tohoto experimentu je obrácená hodnota průměrné vzdálenosti do středu robotické skupiny.

$$fitness_1 = 1/(1/n \sum_{r=1}^n d_{rc}),$$

Kde n je počet robotů v robotické skupině, r je robotův index, d_{rc} je euklidovská vzdálenost mezi r a středem robotické skupiny c .

Druhá $fitness_2$ používá metodu *inverse of hierarchical social entropy* (Balch, 2000). Tato metoda počítá kompaktnost skupiny, tím že hledá každou možnou skupinku (cluster) pomocí změn maximální vzdálenosti h mezi jedinci ze stejného clusteru. Přidávají ještě rozšíření od *Shannon's information entropy*, jenž používá pevné h . Toto rozšíření je definováno:

$$H(h) = - \sum_{k=1}^M p_k \log_2(p_k),$$

kde H se nazývá entropie, p_k je proporce jedince z clusteru k , M je počet clusterů pro dané h . Konečně celý předpis daný Balchem vypadá následovně:

$$fitness_2 = \int_0^\infty \frac{1}{H(h)} dh.$$

Použití neuronových sítí a genetického algoritmu se ve výsledku ukázalo jako vhodný prostředek pro učení robotického hejna, neboť se vygenerované chování obstojně shlukuje do úzkých skupin. Autoři dále definují další tzv. cost funkci pro měření úspěchu nalezených chování, aby mohli porovnávat funkce fitness. A $fitness_2$ se ukazuje jako účinější.

Evoluční strategie a neuronová síť

V článku *Self-organised path formation in a swarm of robots* (Sperati a kol., 2011) aplikují pro řízení robotických hejn evoluční strategie. Jako cíl si článek klade problém průzkumu a navigace v neznámém prostředí v kontextu robotických hejn. Experiment, který měl otestovat uvedené vlastnosti robotického hejna, spočíval v co nejrychlejším přesunu celého hejna mezi dvěma prostory v neznámém prostředí.

Pro simulaci bylo využito upravené verze OS Evorobota a jako model jedince z hejna e-puck robot (Mondada a kol., 2009). Tento robot se pohybuje pomocí dvoukoleček, má 8 infračervených senzorů, navíc jeden infračervený senzor na povrch a jeden rozpoznávající barvy vpředu (v tomto případě černobílé prostředí). Navíc mu byla přidělána LED vpředu s modrou barvou a červenou vzadu, která může zapínat a vypínat dle potřeby, a také má snímač barev na vrchu.

Pro ovládání robota zvolili autoři neuronovou síť se 13 vstupy (8 pro infračervené senzory, 1 pro binární podlahový senzor (bílá vs. černá), 4 pro binární vizuální snímače), dále 3 pro skryté neurony a 4 výstupní neurony (2 ovládající kolečka, 2 aktivující přední a zadní led). Formou jsou podobné předchozím modelům robotů.

Fitness funkce je vyhodnocena po nasazení do robotů a provedení simulace, vlastní fitness je pak průměr z 15 běhů. Ve vyznačených místech se roboti nabíjí, což trvá daný čas a roboti s lepší efektivitou přesunou z jednoho místa do druhého stihnout cestu tam a zpět mnohem rychleji.

Výsledky prokazatelně ukazují úspěšné použití evolučních strategií na optimalizaci chování robotického hejna. Pro většinu prostředí dokázali najít efektivní řešení.

3. Simulátor

Součástí bakalářské práce bylo také naprogramování simulátoru mapy, její vizualizace, implementace všech EA, vše v jazyce C#. Veškeré informace o simulátoru a jeho součástech můžete najít v dokumentaci na přiloženém CD. V této kapitole popíší pouze spuštění, základní objekty simulátoru a aplikované optimalizace.

Části simulátoru

- *SwarmSimFramework.exe* - konzolová aplikace, která obsahuje kód pro EA optimalizaci a simulaci mapy
- *SwarmSimVisu.exe* - program pro vizualizaci, implementovaný ve WPF a C#

Spuštění

Pro spuštění optimalizace *SwarmSimFramework.exe* je potřeba soubor s konfigurací experimentu, soubory s konfigurací pro ES mají koncovku *.es* a v případě diferenciální evoluce *.expe*. Jednotlivým EA také odpovídají očekávané argumenty.

- evoluční strategie - *SwarmSimFramework.exe -es konfigurace.es*
- diferenciální evoluce - *SwarmSimFramework.exe -de konfigurace.expe*

V rámci konfiguračních souborů můžeme nastavit charakteristiky mapy, ohodnocení jednotlivých složek fitness, specifikovat druh či počet robotů, název experimentu a název složky s výstupem. Výstupní složka obsahuje serializované nejlepší jedince vždy po 10 generacích, metadata pro graf (číslo generace, hodnoty fitness), serializované veškeré jedince po 100 generacích. V případě ES mají jedinci svou vlastní podsložku z implementačních důvodů (parallelismus). Populace z poslední generace je ještě serializována do složky spuštění. Do konzole, pak program vypisuje základní informace o aktuální generaci.

Pokud si chceme prohlédnout některé z chování vizuálně, spustíme program *SwarmSimVisu.exe*. Kde je připraveno grafické rozhraní, kde pomocí tlačítka navolíme vybraný scénář, nastavení mapy a soubory se serializovaným řízením robota. Poté je možné danou simulaci spuštět a zastavovat, při zastavené simulaci pravým klikem na robota lze zobrazit jeho podrobné informace.

Optimalizace

Simulace mapy patří mezi časově nejvíce náročnou část. Pro její zrychlení jsem použil paralelního programování, jeho přístup se liší podle EA. Za použití profileru jsem nalezl, že nejvíce času simulace zabralo počítání jednotlivých průsečíků pro senzory, efektory a entity samotné. Pro optimalizaci počtu průsečíků jsem implementoval vlastní datovou strukturu *SpatialHash*.

SpatialHash rozděluje mapu na síť malých čtverečků. Každá entita (mimo roboty) je uvedena ve všech čtverečcích do kterých zasahuje. Pro počítání kolizí dostane potenciálně kolidující těleso od SpatialHash množinu obsahující všechny entity zasahující do stejných čtverečků jako potenciálně kolidující těleso. Jedinou výjimku tvoří roboti, neboť se často přesouvají po mapě a ve SpatialHash by neustále měnili pozici. Operace spojené s častým pohybem byly více náročné než počítat průsečíky se všemi roboty, takže roboti jsou uloženi v klasickém listu.

Kvůli dalšímu zrychlení jsem se rozhodl použít paralelního běhu programu. Kde se použití paralelismu liší na základě použitých EA. Zatímco v ES lze paralelně zpracovávat každého jedince zvlášt, protože nepotřebuje v evolučních operátorech celou populaci, tak v diferenciální evoluci je potřeba, aby pro každý evoluční operátor byla k dispozici celá generace rodičů. Tím pádem ES můžeme od sebe oddělit jednotlivé jedince a nemusí sdílet žádnou paměť. U diferenciální evoluce bylo rozdelení poněkud složitější, protože v rámci běhu vybírá náhodně z celé populace. Rozhodl jsem se, že oddělím simulace mapy, která probíhá pro nově vzniklého potomka a na její základě se hodnotí jeho fitness. Takže pro každého člena původní populace vznikne nezávislý proces, který vybere z aktuální populace 3 jedince (pouze čtení) a pak nově vzniklého jedince vloží do thread safe datové struktury (list z knihovny System.Collections.Concurrent). Po dokončení všech procesů se přejde k další generaci. Použití paralelního zpracování znemožňuje přesné zopakování experimentů, zkoušel jsem experimenty pouštět na rozličných strojích a výsledky řádově odpovídají.

Použití

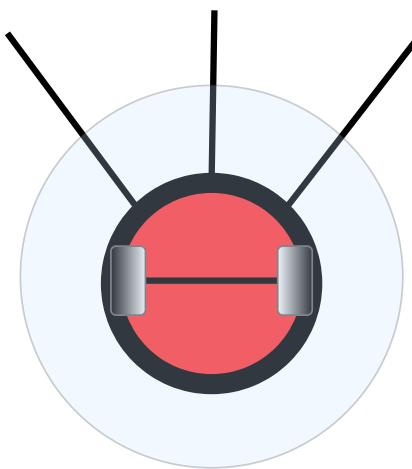
Pro jednotlivé objekty na mapě jsem vytvořil obecný objektový návrh, který umožňuje jednoduché přidání vlastních objektů, případně používat pouze simulaci na mapě pro jiné účely než optimalizaci chování robotů, či implementací nových evolučních algoritmů.

Celá simulace probíhá v rámci instance třídy Map, ve které se nacházejí všechny objekty figurující v simulaci. Jedná se o překážky, materiály určené ke zpracování, roboty atd. Přesný strom dědičnosti je uveden v dokumentaci. Roboti používaní v experimentech v kapitole 4 mají podobnou základní stavbu. Její hlavní části ilustruje obrázek 3.1, červeně je označeno tělo robota, modrý kruh je rádiový přijímač, černé úsečky představují vzdálenostní senzory, šedivě stínované obdélníky zobrazují kolečka (ve vizuálním programu se nezobrazují).

Všichni mnou používaní roboti jsou poháněni dvou kolovým motorem. Pro interakci se svět jím slouží efektory (mění prostředí) a senzory (snímají prostředí), v obou případech mají bud' podobu úsečky nebo kruhu.

Mezi modelové úsečkové senzory patří vzdálenostní senzory, který vrací typ a vzdálenost k objektu kolidující se senzorem, typicky mají roboti 3 tyto senzory vpředu. Jako zástupce kruhovitých senzorů můžeme považovat rádiový přijímač, který vrací robotovi ve vektoru intenzitu signálů, které proťaly jeho kruhovou reprezentaci.

Efektory pracují na podobném principu jen místo čtení hodnot z prostředí očekávají vstupní hodnoty s nastavením. Například efektor zpracovávající stromy na dřevo má podobu úsečkového efektoru, pro zpracování musí kolidovat s entitou stromu a na vstupu musí dostat povel pro aktivaci.



Obrázek 3.1: Ilustrativní obrázek obecného robota

4. Experimenty

Všechny práce zmíněné v úvodní kapitole používají evoluční algoritmy k vytvoření řízení chování homogenního robotického hejna, tzn. s jedním druhem robotů. V následující kapitole podrobně popíší postup hledání optimálního chování pro heterogenního hejna. Optimalizaci jsem navrhl a otestoval na třech rozličných scénářích. V rámci scénářů budu používat DE jako zkratku za diferenciální evoluci a ES pro evoluční strategie

Pracovní názvy scénářů:

1. Wood Scene - zpracování dřeva
2. Mineral Scene - přetvoření minerálů na palivo
3. Competitive Scene - soubojový scénář

Hlavní motivací při tvorbě scénářů bylo vytvořit obtížnější úkoly než se obvykle používají jako například: shlukování, vyhýbání překážkám, atp. Navrhnut je natolik komplexně, aby nebylo možné, že část hejna se nebude podílet na jeho plnění. Také jsem volil scénáře, aby se přiblížily situacím z reálného světa. Každému z nich jsem věnoval samostatnou kapitolu, která zahrnuje popis hlavního úkolu scénáře, seznam robotů i s jejich senzory a efektory, způsob hodnocení fitness, rozdelení do podúkolů s průběhem fitness u ES a DE, vizualizaci a rozbor chování nejlepšího jedince.

Pro řešení problému jsem navrhl řadu postupů, proto v tomto odstavci zmíním ty nejvíce přímočaré a slibné, které se ovšem ukázaly jako neúspěšné. V kapitolách zabývajících se konkrétními scénáři už budu pouze popisovat jen konečné, úspěšné postupy.

Nedostatečný se ukázal pokus provádět evoluci pro fitness hlavního úkolu scénáře. Konkrétně pro Wood Scene počet natěženého a uskladněného dřeva, pro Mineral Scene objem vytvořeného paliva, pro kompetitivní scénář zbylé body zdraví a udělené poškození. Většina hodnocení náhodných chování byla rovna nule, proto EA nedostaly dostatek informací k vhodné exploraci a díky malé pravděpodobnosti vygenerování chování alespoň částečně řešící hlavní úkol nedocházelo ani k exploataci. Což mělo za důsledek neefektivní DE a ES, takže ani jeden z EA nedošel k úspěšnému řešení.

Posun zaznamenala více obecná fitness i když sama o sobě také nedosáhla do kategorie úspěšných postupů. Do fitness jsem zahrnul i menší pozitivní znaky, které byly součástí hlavního úkolu. Například jsem záporně ohodnotil pokusy o pohyb končící kolizí, kladně počet nalezených entit či vhodných objektů v kontejnerech, aktuální stav paliva. Optimalizovaná chování opravdu zaznamenala posun. Ovšem oba EA obtížně hledaly cestu z lokálního optima a ve většině případů optimalizovaly pouze jednoduché části úkolu. I přes přidávání složitějších matematických funkcí do fitness nebyly schopny dosáhnout uspokojivého řešení hlavního úkolu scénáře.

Použitá metoda

Pro finální řešení jsem zvolil metodu, kterou nazývám metodou podúkolů. U každého scénáře podrobně popíší její průběh a nastavení, zde pouze nastíním základní myšlenku. Rozdělil jsem hlavní cíl na několik menších podúkolů (metaúkolů). Každému z nich vytvoříme fitness funkci odpovídající nutné části hlavního cíle. Fitness metaúkolu jsem navrhoval, tak aby necílila na již optimalizované úkony a v každém podúkolu jsem se vždy soustředil pouze na jeden jednoduchý úkon. Díky tomuto principu jsem dosáhl mnohem vyšší odolnosti proti uvíznutí v lokálním minimu. Explorace se tímto procesem také zlepšila, protože hlavní cíl závisí na podúkolech a pokud bylo chování rozmanité a úspěšné, přenesly se tyto vlastnosti i dále. Poté jsem generaci úspěšnou v průzkumu optimalizoval na sbírání materiálů pouze požadované barvy a takto jsem rozděloval až k finálnímu úkolu scénáře.

4.1 Použité technologie

Tuto kapitolu věnuji klíčovým technologiím, jenž jsem použil pro modelování řešeného problému a optimalizaci náhodných chování. Pro ovládání robotů jsem zvolil v poslední době velmi oblíbené *neuronové sítě*, které se často používají v kombinaci s EA. Jednomu jedinci odpovídá jedna neuronová síť ovládající všechny části robota. Tyto neuronové sítě si lze představit jako vektor reálných čísel, což je vhodná reprezentace genotypu pro EA.

Reprezentace Chování - neuronové sítě

Pro reprezentaci jedinců v oblasti robotiky, rozpoznávání obrazů a dalších oblastí umělé inteligence se v poslední době používají nejčastěji neuronové sítě. Neuronová síť se strukturou podobá neuronovým sítím v mozku. Základní sítě se skládají z jednotlivých neuronů, které se v kontextu informatického světa nazývají *perceptrony*. Samostatný perceptron je sám o sobě také neuronovou sítí, ale většinou se propojují do složitějších struktur. Perceptron lze definovat podle (Marsalli, 2006) následovně.

Definice 1 (Perceptron). *Perceptron je funkce $z \mathbb{R}^n \rightarrow \mathbb{R}$, která je dáná následujícím předpisem: $Y = S(\Theta + \sum_{i=0}^n w_i x_i)$, kde pro $i \leq n$ x_i je i-tý prvek vstupního vektoru, w_i se označuje jako váha a většinou w_i se bere z \mathbb{R} . Θ se nazývá práh (bias) a slouží jako váha s konstantním vstupem 1. $S(X)$ je aktivační funkce: $\mathbb{R} \rightarrow \mathbb{R}$ a Y se obvykle označuje jako výstup perceptronu.*

Aktivační nebo také přenosová funkce má za úkol transformovat výstup neuronu, aby výstupní hodnota odpovídala výstupním hodnotám. Mezi nejjednodušší přenosové funkce patří binární funkce, kdy očekáváme od neuronu výstup typu ano, ne. Používají se i více složité funkce.

Jednovrstvou neuronovou sítí pak myslíme n perceptronů, tedy funkci $\mathbb{R}^n \rightarrow \mathbb{R}^n$, kde i-tou složku výstupního vektoru dostaneme aplikací funkce odpovídající i-tému perceptronu na vstupní vektor.

Pokud zapojíme z výstupu jednoho perceptronu na vstup jiného, vznikne *vícevrstvá neuronová sítě*. Což znamená, že podmnožiny výstupů z první vrstvy neuronů neurčují přímo výstup, ale jsou opět zvoleny jako vstupní vektory pro další

jednovrstvou neuronovou sítí. Tímto postupem můžeme vytvářet velmi komplexní struktury.

Skrytá vrstva (hidden layer) je taková jednovrstvá neuronová síť, jejíž výstup(resp. vstup) je pouze vstupem(resp. výstup) jiných perceptronů.

Pro mé účely se jsem testoval řadu různých variant neuronových sítí, ale nejvíce se mi osvědčilo následující nastavení, které poskytovalo uspokojivé výsledky a rozumné časové nároky.

Jako aktivační funkce jednotlivých perceptronů se mi nejvíce osvědčila funkce hyperbolického tangentu často používaná se změněným oborem hodnot pro konkrétní výstup.

V rámci testovaní jsem zvolil jednoduchou architekturu jednovrstvé neuronové sítě, což se ukázalo jako dostatečné pro uspokojivé pro řešení prvních úkolů. Jejich architektura je následující. Pro každé reálné číslo, které očekává robot jako vstup pro efektor, byl připojen perceptron do kterého vstupuje vektor reálných čísel odpovídající vektoru všech hodnot přečtených ze senzorů. Pro dosažení lepších řešení by zde bylo možné nasadit NEAT algoritmus či hledat více specifičtější architektury, případně vyzkoušet vliv vícero vrstev.

Evoluční algoritmy

Neuronovou síť si lze představit jako množinu vektorů, kde jeden perceptron odpovídá vektoru reálných čísel (vah vstupů + práh $v = (x_0, x_1 \dots x_n, \Theta)$). V kontextu evolučních algoritmů se pro optimalizaci vektorů reálných čísel nejčastěji používají ES a DE, I z tohoto důvodu jsem zvolil zmíněné algoritmy jako zástupce pro optimalizaci chování heterogenní skupiny robotů. Oba zmíněné algoritmy důkladně popisují v kapitole 1.4 a 1.5 a má implementace se od popisu v úvodu liší pouze v malý detailech. Do detailu si je lze prohlédnout v přiložené dokumentaci a kódu.

V rámci testovaní jsem vyzkoušel mnoho různých nastavení parametrů EA. V tabulce 4.1 jsou uvedeny nakonec použité parametry, které dosahovali v experimentech největších úspěchů. Jejich vliv popisuji v první kapitole. Jedná se o tradičně používané parametry, osvědčené v řadě optimalizačních problémů.

differenciální evoluce	
F:	0,8
CR:	0,5
evoluční strategie	
alpha	0,05
sigma	0,1

Tabulka 4.1: Nastavení parametrů u EA

4.2 Wood Scene

Vzorem Wood Scene scénáře byla těžba dřeva, představme si dřevorubce s motorovou pilou a silné dělníky nakládající zpracované stromy do transportérů a svázející materiál na společnou hromadu. Roboti odpovídají těmto lidským rolím, samozřejmě je jejich činnost značně zjednodušena. V obou případech je cílem maximalizovat počet zpracovaného dřeva na daném místě, což vyžaduje od obou druhů agentů spolupráci.

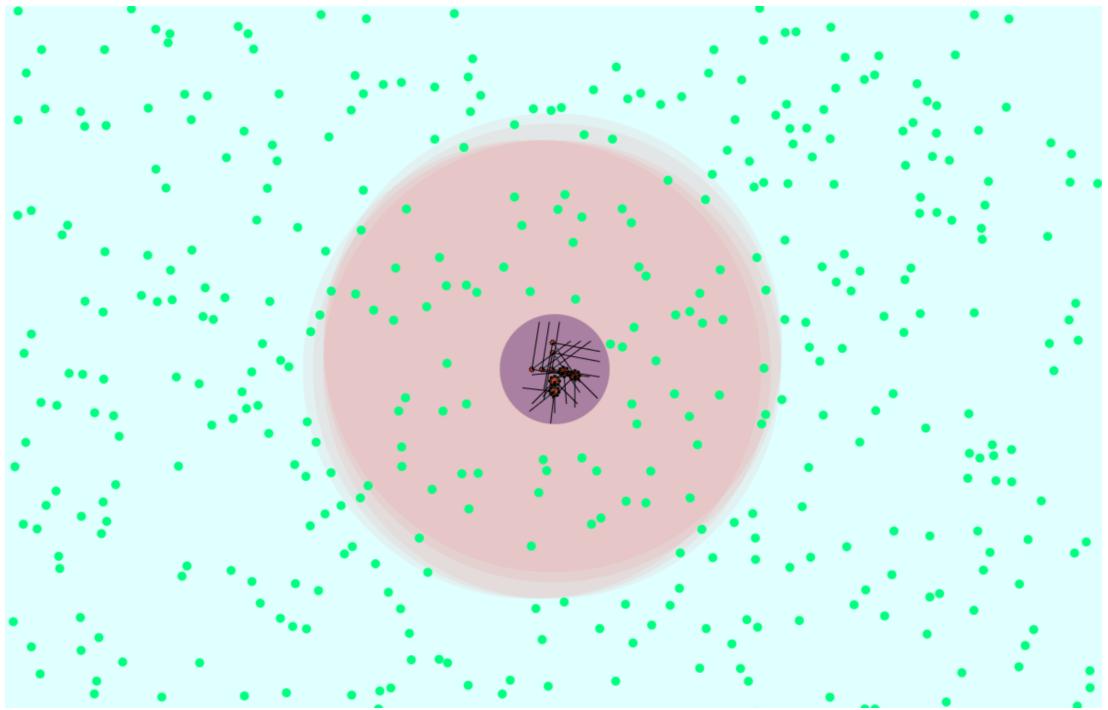
Robotické hejno se ve Wood Scene snaží natěžit a převést co největší množství dřeva na místo označené rádiovým signálem. Rádiové senzory poskytují robotům sílu signálu a vysílaný kód. Pro místo určené na kupení dřeva je určen unikátní kód 2 a je umístěn doprostřed mapy. Žádný jiný rádiový vysílač vysílající signály s kódem 2 se na mapě nenachází.

Celé hejno čítá 9 robotů, jedná se o dva různé druhy, které se liší velikostí, rychlostí, senzory i efektory. Na začátku experimentu jsou náhodně rozmístěny do středu mapy na stejném místě jako se rozprostírá skladovací prostor. Dále jsou na mapě náhodně umístěny stromy. Robot v kontextu scénáře nazývaný Scout odpovídá „dřevorubci“ v teoretickém vzoru, pohybuje se rychle, má menší rozměry, umí nalezený strom zpracovat na dřevo pro komunikaci má přidělený unikátní kód 0. Oproti tomu robot „dělník“ se pohybuje pomaleji, je větší, neumí zpracovávat stromy, ale disponuje nakladačem (vykladačem) a kontejnerem na 5 objektů.

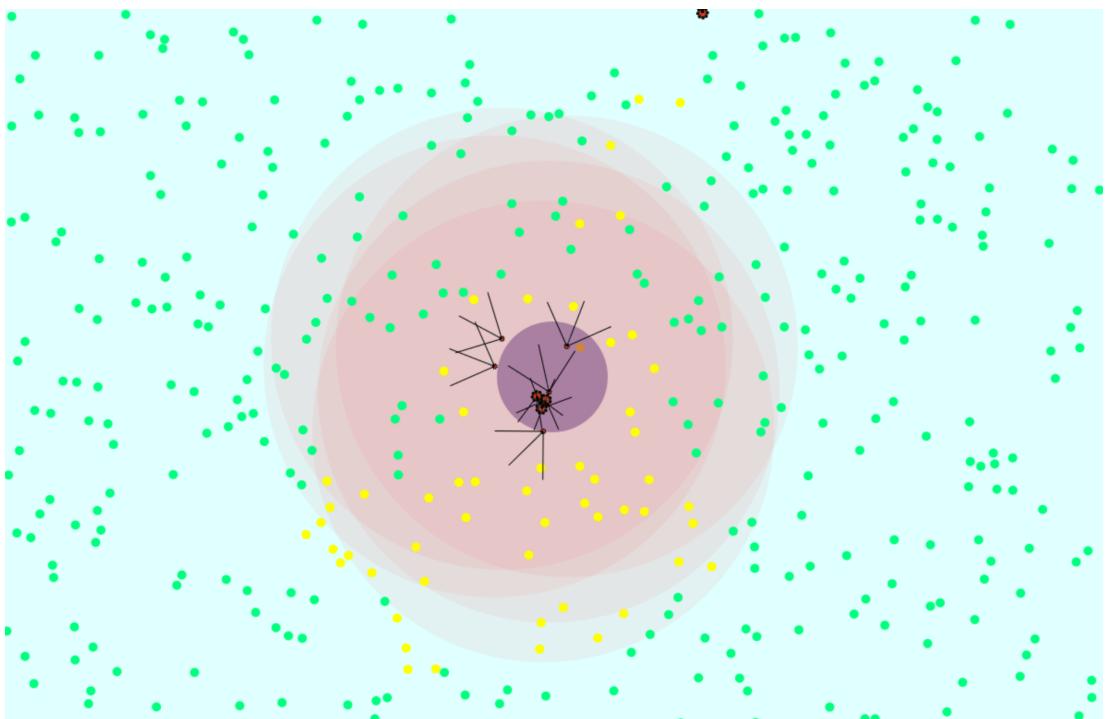
Souhrnně hejno musí strom nalézt, přepracovat na dřevo, poté naložit a odvézt do středu. Celý tento proces zahrnuje typické úkoly pro robotický swarm jako rozprostření, vyhýbání se překážkám, komunikaci mezi jednotlivými agenty, nalezení cesty apod. Z návrhu je zřejmé, že na procesu se musí podílet oba druhy robotů.

V rámci bakalářské práce byl připraven program zajišťující vizualizaci chování robotů. Jeho vizuální výstup můžete vidět na obrázku 4.1. Na ní si vysvětlíme jednotlivé entity nacházející se na mapě. Mapa je ohrazena obdélníkovou hranicí a chová se jako zeď. Zelené kroužky znázorňují stromy, které ještě nebyly objeveny. Objevený strom změní barvu na žlutou. Modré označený prostor je určen pro uskladnění zpracovaného dřeva, roboti jej zaznamenají jako rádiový signál. Hnědá kolečka zastupují pokácené dřevo. V některých podíkolech se objevuje dřevo už při inicializaci, proto ještě neobjevené má tmavší barvu a objevené světlejší. Pro roboty je v tomto prostoru vysílán rádiový signál. Roboti jsou vyplněni červenou barvou, jejich senzory a efektory mají černou barvu. Pro každý rádiový signál je určena jedna unikátní barva s alfa kanálem.

Pro potvrzení, že scénář není triviálně řešitelný. Bylo vygenerováno tisíc náhodných chování. Hodnoceny byly dle fitness funkce podíkolem kooperace popsaných níže. Nejlepší z nich můžete vidět těsně po inicializaci mapy 4.1. Výsledek krátce před 10 000. iterací je zachycen v obrázku 4.2. Největší světle zelená plocha je volný prostor, kde se mohou roboti pohybovat. Světle růžový kruh je právě rádiové vysílání, protože všichni roboti vysílají současně je barva celkem sytá. Tmavě fialový kruh označuje místo pro uskladnění, obvykle má modrý odstín, ale protože jej překrývají signály robotů zbarvil se do fialové. Na obrázku 4.2 vidíme, že se robotům povedlo jeden strom pokáct a uložit. Většina velkých robotů se dostala do kolize a menší roboti nedokázali objevit ani pětinu stromů.



Obrázek 4.1: Příklad WoodScene mapy: start náhodného chování



Obrázek 4.2: Příklad WoodScene mapy: po 9000 iteracích náhodného chování

Roboti

Devítičlenné hejno obsahuje 5 Scout robotů a 4 Worker roboty. U každého z robotů popíši jejich efektory a senzory. Pro každý druh robota je připravena jedna shodná neuronová síť. Jedinec odpovídá vektoru vah neuronové sítě. Pokud v rámci experimentu optimalizuji chování více druhů robotů, jedinec jsou dva vektory vah pro každý druh robota jedna neuronová síť. Proto fitness funkce hodnotí jejich výsledné snažení dohromady a evoluční operátory pracují nad celou dvojicí. Podívejme se na jednotlivé druhy podrobně.

Scout robot

Scout robot je robot, který má na starosti průzkum mapy a kácení nalezených stromů. Na zpracování dřeva používá efektor, nazývám jej refaktor, který má podobu úsečky a vyčnívá z čela robota. Pro zpracování musí refaktor kolidovat se stromem v mapě, poté je strom prohozen za entitu dřeva. Aby mohl komunikovat má přidělený rádiový signál s kódem 0, při jeho vysílání přidá na mapu signál jako kruh se středem odpovídajícím pozici robota. Jedná se o menšího robota, oproti Worker robotovi je rychlejší a jeho senzory mají větší dosah. Tabulka 4.2 obsahuje základní charakteristiky, počty a dosahy jednotlivých senzorů a efektorů.

Scout Robot	
Tvar:	<i>Kruh</i>
Poloměr:	2,5
Název:	<i>WoodCutterM</i>
Velikost kontejneru:	0
Efektory	
Motor:	<i>Dvoukolečkový</i>
Maximální rychlosť:	3
Kód rádiového signálu:	0
Poloměr signálu:	200
Refaktor:	<i>Strom ⇒ Dřevo</i>
Dosah refaktoru:	10
Počet paměťových slotů:	10
Obsah slotu:	<i>float</i>
Senzory	
Počet line senzorů:	3
Délka line senzorů:	50
Orientace line senzorů:	$0^\circ, \pm 45^\circ$
Poloměr type senzoru:	50
Poloměr rádiového přijímače:	100
Počet touch senzorů:	8
Lokátor senzor	

Tabulka 4.2: Wood Scene - Scout robot specifikace

Worker robot

Worker robot se stará o manipulaci a následný transport objektů na mapě. Pohybuje se pomaleji než Scout a také je o něco rozměrnější. Picker, úsečkový efektor sloužící pro nakládání a vykládání objektů, funguje na podobném principu jako refaktor pro naložení musí kolidovat s objektem a pro vyložení s ním nesmí nic kolidovat. Ke komunikaci mu byl vyhrazen rádiový signál s kódem 1. Sebrané objekty ukládá do kontejneru, kam se vejde celkem 5 entit a vykládat umí pouze entitu na vrchu. Tabulka 4.3 popisuje další podrobnosti.

Worker Robot	
Tvar:	<i>Kruh</i>
Poloměr:	5
Název:	<i>WoodWorkerM</i>
Velikost kontejneru:	5
Efektor	
Motor:	<i>Dvoukolečkový</i>
Maximální rychlosť:	2
Kód rádiového signálu:	0
Poloměr signálu:	200
Dosah pickeru:	10
Počet paměťových slotů:	10
Obsah slotu:	<i>float</i>
Senzory	
Počet line senzorů:	3
Délka line senzorů:	30
Orientace line senzorů:	$0^\circ, \pm 45^\circ$
Poloměr rádiového přijímače:	100
Počet touch senzorů:	8
Lokátorový senzor	

Tabulka 4.3: Wood Scene - Worker robot specifikace

Vyhodnocování Fitness

Fitness funkce pro ohodnocení WoodScene scénáře probíhá vždy až na konci simulace a má podobu váženého součtu charakteristik mapy. I když se úspěšnost v podílkolech vždy posuzuje jinak, celou fitness funkci lze shrnout do následujícího cílů. Roboti jsou odměňováni za:

1. *nalezené stromy* - stromy o které zavadil line senzor
2. *pokácené stromy* - stromy, které refaktor změnil
3. *sebrané dřevo* - zpracované dřevo, které mají roboti uvnitř kontejnerů
4. *uskladněné dřevo* - dřevo, které dovezli na vyznačené místo

Trestání za:

1. *kolize* - počet pokusů o pohyb při kterém by došlo ke kolizi
2. *sebrané entity mimo dřevo* - počet entit v kontejnerech, které nejsou zpracované dřevo

Podúkoly

Rozdělil jsem hlavní cíl na následující podúkoly. Jejich obtížnost postupně roste a finální metaúkol už odpovídá řešenému problému. Pro ES a DE jsem použil stejné, aby bylo možné porovnat jejich fungování.

1. vygenerování robotů - Na začátku je vygenerováno chování robotů naprostě náhodně. Pro každého robota, je vygenerována náhodná jednovrstvá neuronová síť.
2. učení chůze - Pro oba roboty je velmi důležité, aby se pohybovali bez kolizí po celé mapě a objevovali co největší prostor. Roboti jsou vyvíjeni odděleně a fitness se soustředí na počet kolizí (záporným ohodnocením) a na nalezené stromy (kladným ohodnocením).
3. těžba stromů - Scout roboty, kteří se už obstojně po mapě pohybují, je třeba naučit káct stromy. Proto dalším cílem ve fitness funkci je počet pokácených stromů. Nicméně stále také na počet stromů nalezených.
4. převoz dřeva - Správně pohybující chceme naučit sbírat vytěžené dřevo. Fitness hodnotí počet sebraného dřeva, případně i uskladněné dřevo. Na těchto mapách jsou už na začátku připraveny pouze entity zpracovaného dřeva.
5. kooperace - V posledním experimentu, se hodnotí pouze sebrané a uskladněné dřevo. A evolvují se oba druhy robotů současně.

U každého experimentu uvedu myšlenku, tabulkou s přesným nastavením a poté graf s průběhem fitness jednotlivých EA plus jejich vzájemné porovnání. ES v rámci mutačních operátorů vytváří několik zmutovaných jedinců a na základě jejich fitness utváří potomka. Vyhodnocování fitness je časově nejnáročnější výpočet, protože se musí probíhat na mapě celá simulace. Aby časy běhu DE a ES

byly srovnatelné, odpovídá velikost populace u DE, velikosti populace krát počet mutovaných jedinců u ES. Při porovnání EA zobrazují průměrné hodnoty fitness jedinců v rámci daného podúkolu.

Scout chůze - nastavení experimentu

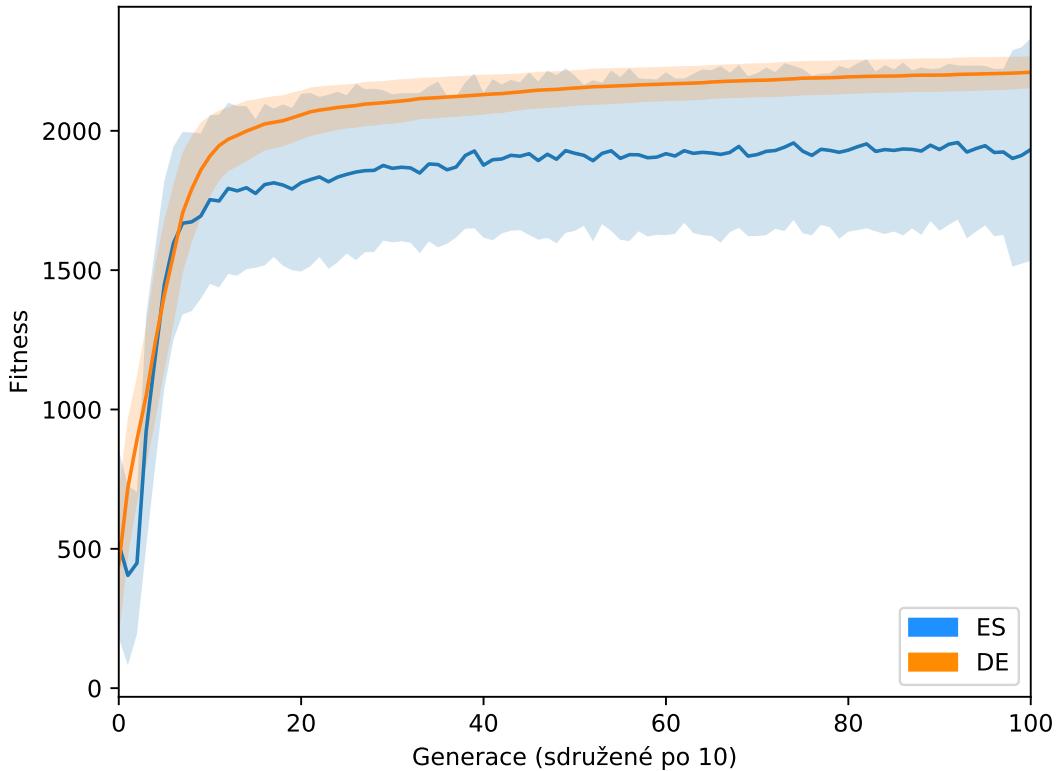
Nejdříve jsem se zaměřil na schopnost pohybu jednotlivých robotů po mapě. Oddělil jsem oba druhy od sebe, protože díky rychlejšímu pohybu Scout robotů EA optimalizovalo pouze jejich pohyb. Roboti byli oceněni za nalezené stromy, tato fitness je nutila rozprostřít se po mapě. Následuje tabulka s nastavením fitness a EA.

Nastavení mapy a EA	
Roboti:	<i>Scout</i> – 5
Počet generací:	1000
Počet iterací map	1000
Velikost generace(DE)	200
Počet jedinců(ES)	10
Počet mutovaný potomků(ES)	20
Elitismus(ES)	<i>Ano</i>
Elitismus(DE)	<i>Ne</i>

Fitness funkce	
Hodnota nalezeného stromu	10
Ostatní hodnoty:	0
Počet stromů:	300
Počet už pokácených stromů	100

Tabulka 4.4: Wood Scout chůze - nastavení experimentu

Výsledky experimentu ilustrují grafy na další stránce. Jednotlivé průběhy fitness na obrázku 4.3 ukazují střední hodnotu fitness a její rozptyl v závislosti na generaci, jedinci jsou kvůli přehlednosti sloučeni po 10 generacích. V obou případech docházelo k největšímu růstu do 200 generace (v grafech 20). Oba EA lze označit jako úspěšné, protože vygenerovaná chování objevila více než 50% stromů na mapě, v případě DE dokonce více dvě třetiny. U ES jsem nepoužíval elitismus, proto křivka více osciluje než je tomu u DE.



Obrázek 4.3: Wood Scout chůze - porovnání průměrné fitness ES a DE

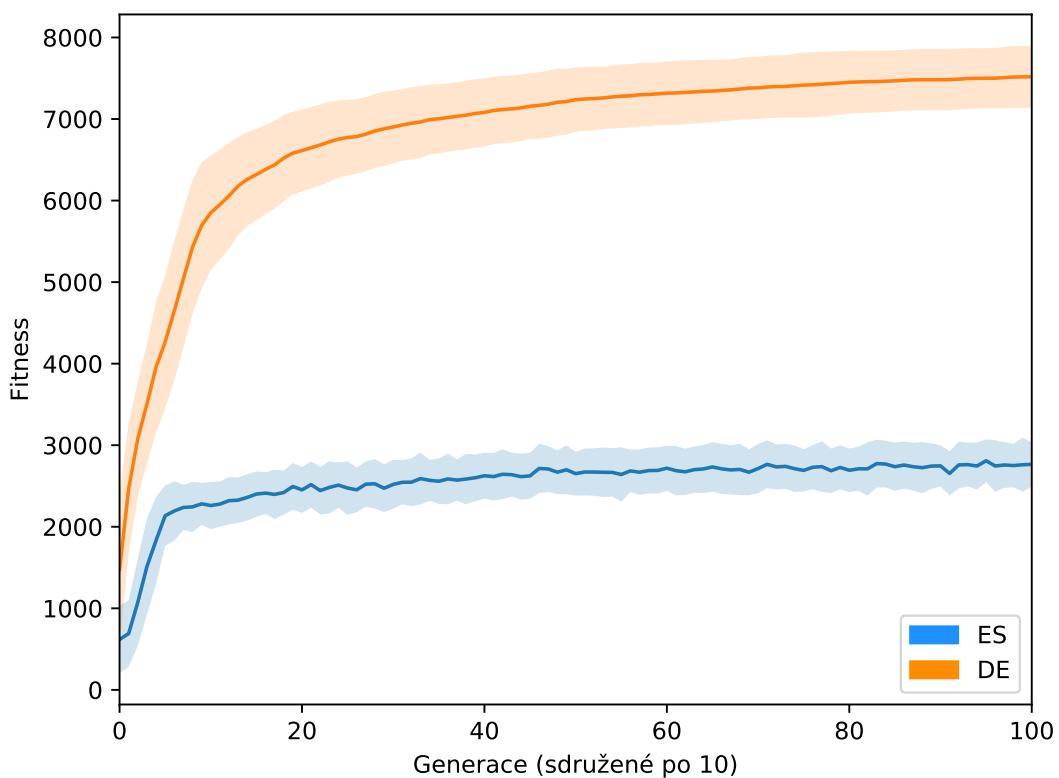
Worker chůze - nastavení experimentu

Worker chůze aplikuje podobný postup jako v předchozím experimentu na Worker roboty. Jen jsem nehodnotil počet nalezených stromů, ale do každé mapy jsem umístil už zpracované stromy. Roboti tedy byli oceněni za nalezení právě tohoto dřeva. Opět fitness funkce nutí roboty, co nejvíce se rozprostřít po mapě a navíc ještě vyhýbat se nepokáceným stromům.

DE dokázal už ve 200. generaci najít většinu zpracovaného dřeva na mapě, jak můžeme vidět na grafu DE v obrázku 4.4. ES neoptimalizovalo v tomto případě příliš rychle a uvízlo v lokálním optimu, jak ukazuje křivka ES od 200. generace. Nejlepší jedinec optimalizovaný pomocí DE byl schopen nalézt až 3x více zpracovaného dřeva než ten pomocí ES. Nutno dodat, že náhodná pozice entit na mapě se liší pro ES a DE. Zkoušel jsme, proto měnit seed u generátoru náhodných čísel a výsledky odpovídaly grafům v 4.4..

Nastavení mapy a EA	
Roboti:	<i>Worker - 4</i>
Počet generací:	1000
Počet iterací map	1000
Velikost generace(DE)	200
Počet jedinců(ES)	10
Počet mutovaný potomků(ES)	20
Elitismus(ES)	<i>Ano</i>
Elitismus(DE)	<i>Ne</i>
Fitness funkce	
Hodnota nalezeného pokáceného stromu	20
Ostatní hodnoty:	0
Počet stromů:	0
Počet už pokácených stromů	400

Tabulka 4.5: Wood Worker chůze - nastavení experimentu



Obrázek 4.4: Wood Worker chůze - porovnání průměrné fitness ES a DE

Scout kácení - nastavení experimentu

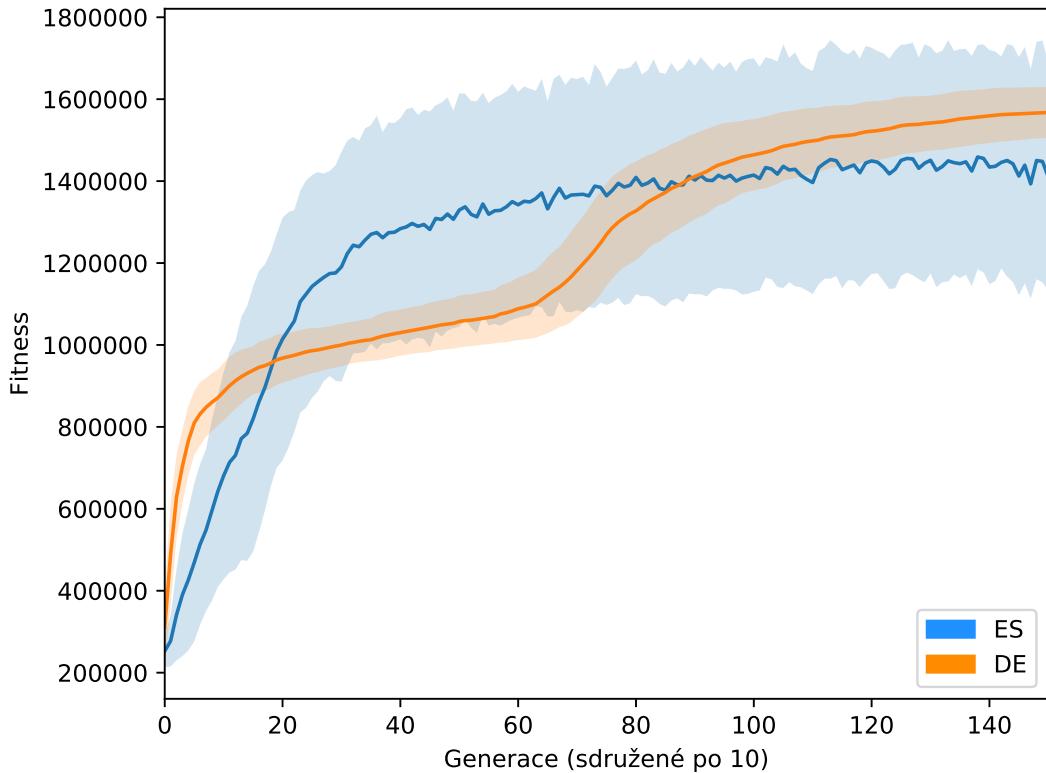
Dalším úkolem pro Scout robota bylo nalezené stromy pokácer. Použil jsem tedy optimalizované neuronové sítě z experimentu Scout chůze a tentokrát přidal do fitness pozitivní body za pokácené stromy. Refaktor je o mnoho kratší než line sensory, proto pro kácení musí robot ke stromu přijet blíže. Abych ještě více vylepšil pohyb po mapě, tak každá kolize byla potrestána negativním bodem do fitness. Přesné nastavení obsahuje následující tabulka.

Nastavení mapy a EA	
Roboti:	<i>Scout</i> – 5
Počet generací:	1500
Počet iterací map	1000
Velikost generace(DE)	200
Počet jedinců(ES)	10
Počet mutovaný potomků(ES)	20
Elitismus(ES)	<i>Ano</i>
Elitismus(DE)	<i>Ne</i>

Fitness funkce	
Hodnota nalezeného stromu	1000
Hodnota pokáceného stromu	10000
Hodnota kolize	-1
Ostatní hodnoty:	0
Počet stromů:	400
Počet už pokácených stromů	0

Tabulka 4.6: Wood Scout kácení - nastavení experimentu

Dále můžete vidět grafy popisující průběh fitness u DE, ES a porovnání jejich průměrné fitness. Z 4.5 můžeme vyčíst, že použité DE více cílí na exploataci a ES na exploraci. DE jsou díky tomu mnohem náchylnější k uvíznutí v lokálním optimu. Fitness se v tomto případě skládá ze dvou složek, pro jedince bylo mnohem jednodušší stromy objevovat a náhodou nějaké pokácer. V prvních 650 generacích DE se tedy drží tento trend a pak se objeví jedinci, kteří cílí na kácení. Toto chování se rychle rozšířilo a fitness celé populace okolo 700. generace prudce vzrostla. Zatímco fitness v ES rostla postupně, ale nedosáhla tak vysoké úrovně jako DE. Nejlepší jedinci pokácí více než 60% stromů.



Obrázek 4.5: Wood Scout kácení - porovnání průměrné fitness ES a DE

Worker sbírání - nastavení experimentu

Worker v rámci zadání musí řešit více složitý úkol než Scout robot. Celý proces uložení se skládá nejdříve z nalezení dřeva, naložení, poté přesunu do místa pro skladování a nakonec vyložení. Optimalizace fungovala lépe po rozdělení na část nakládání, kterou se zabývá tento experiment a na část vykládání, na kterou se podíváme později. Ve fitness jsem se soustředil na správné entity v kontejneru a na jejich počet. Abych nezahodil dobré chování, které dokáže dřevo i ukládat, za uložení dřeva dostali roboti také pozitivní body. Níže můžete vidět tabulku s konkrétním nastavením.

V grafech na 4.6 je vidět, že roboti dokázali maximálně naplnit kontejnery zpracovaným dřevem, v případě DE už po 250. generaci zvládala tento úkol celá populace. Chování optimalizované ES mají mnohem větší rozptyl díky vysoké exploraci, ale i její nejlepší jedinci zvládli maximální naplnění již kolem 250. generace.

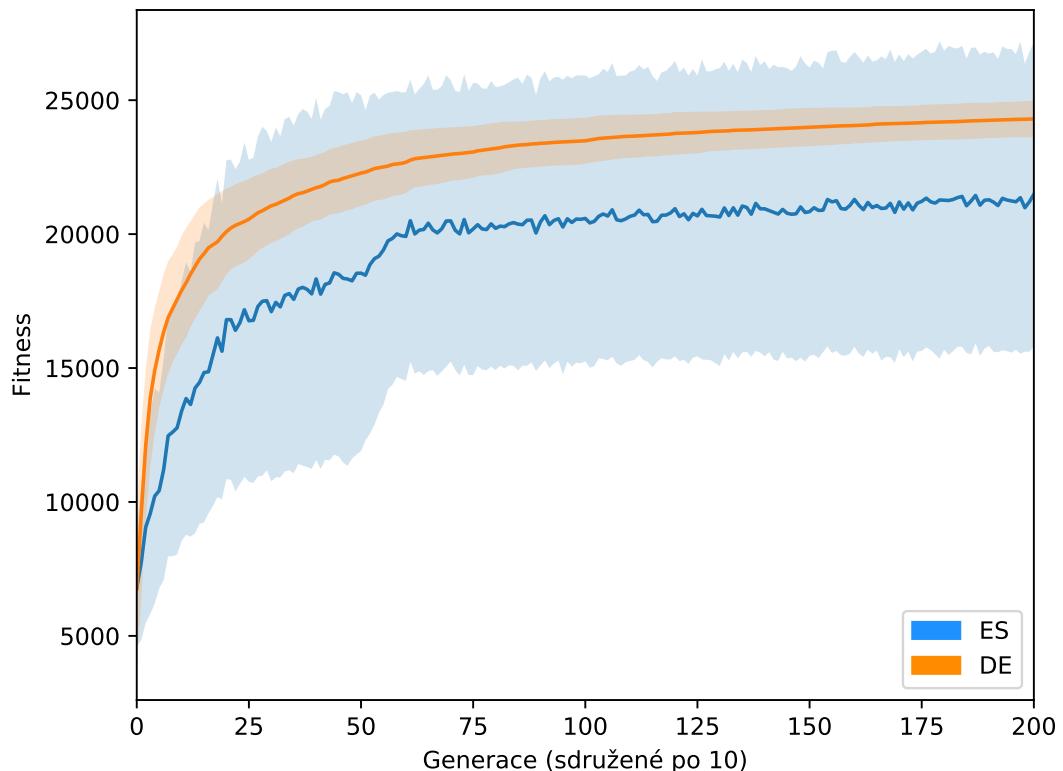
Nastavení mapy a EA

Roboti:	<i>Worker - 4</i>
Počet generací:	2000
Počet iterací map	1000
Velikost generace(DE)	200
Počet jedinců(ES)	10
Počet mutovaný potomků(ES)	20
Elitismus(ES)	<i>Ano</i>
Elitismus(DE)	<i>Ne</i>

Fitness funkce

Hodnota nalezeného pokáceného stromu	100
Hodnota uloženého dřeva	1010
Hodnota dřeva v kontejneru	1000
Hodnota jiné entity v kontejneru	-100
Hodnota kolize	-1
Ostatní hodnoty:	0
Počet stromů:	200
Počet už pokácených stromů	200

Tabulka 4.7: Wood Worker sbíráni - nastavení experimentu



Obrázek 4.6: Wood Worker sbíráni - porovnání průměrné fitness ES a DE

Worker ukládání doprostřed - nastavení experimentu

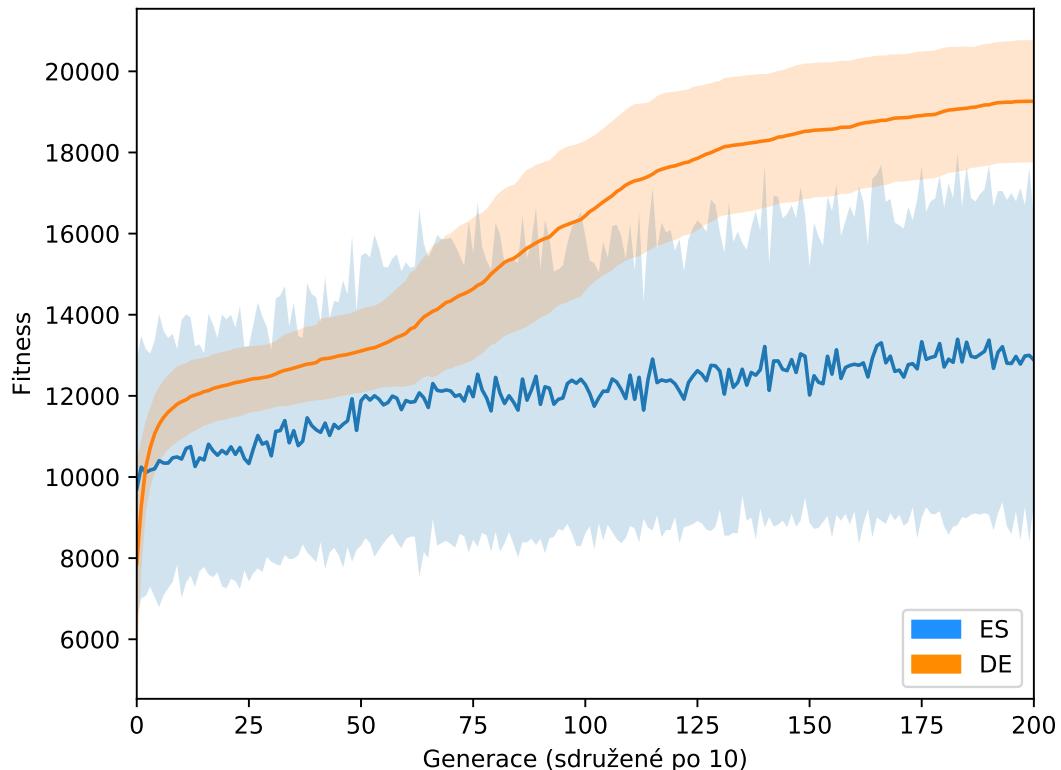
Ukládání zpracovaného dřeva byl poslední experiment, který plnili Worker roboti samostatně. Zde jsem se soustředil na celý úděl Worker robota a využil už optimalizovaných neuronových sítí na sbírání zpracovaného dřeva z předchozího experimentu. Nejvyšší odměnu roboti získávali za uskladnění dřeva a minoritní odměny za vhodné chování dle předchozích fitness. Vlastní nastavení odpovídá tabulce 4.8.

Nastavení mapy a EA	
Roboti:	<i>Worker</i> – 4
Počet generací:	2000
Počet iterací map	2000
Velikost generace(DE)	200
Počet jedinců(ES)	10
Počet mutovaný potomků(ES)	20
Elitismus(ES)	<i>Ano</i>
Elitismus(DE)	<i>Ne</i>

Fitness funkce	
Hodnota nalezeného pokáceného stromu	100
Hodnota uloženého dřeva	1000
Hodnota dřeva v kontejneru	100
Hodnota jiné entity v kontejneru	-100
Hodnota kolize	-1
Ostatní hodnoty:	0
Počet stromů:	200
Počet už pokácených stromů	200

Tabulka 4.8: Wood Worker ukládání doprostřed - nastavení experimentu

Lokace skladovacího místa působila robotům velké obtíže, proto růst fitness byl mnohem mírnější než u předchozích experimentů. DE dokázala už optimalizované chování rychleji vylepšovat narozdíl od ES, která doplácí na velkou míru explorace viz. 4.7. Nejlepší jedinci jsou ovšem už schopni plnit obstoně hlavní úkol Worker robotů. Objevila se obtíž s efektivním urovnáním zpracovaného dřeva na skladovací místo. Pokoušel jsem se tento problém řešit přidáním kladných bodů za menší vzdálenost mezi středem skladovacího místa a uloženým dřevem. Nedosáhl jsem však lepších výsledků než v tomto experimentu.



Obrázek 4.7: Wood Worker ukládání doprostřed - porovnání průměrné fitness ES a DE

Kooperace hlavní úkol - nastavení experimentu

V rámci posledního úkolu jsem složil už optimalizovaná chování dohromady, abych vytvořil heterogenní hejno řešící problém Wood Scene scénáře. Spojoval jsem náhodné chování pro Scout robota s náhodným u Worker robota, aby se našla nejlepší jejich kombinace, tak bylo nutné navýšit počet generací na 4000. Fitness funkce(4.9) byla v tomto případě průnik posledních experimentů Worker ukládání doprostřed a Scout kácení.

Nastavení mapy a EA

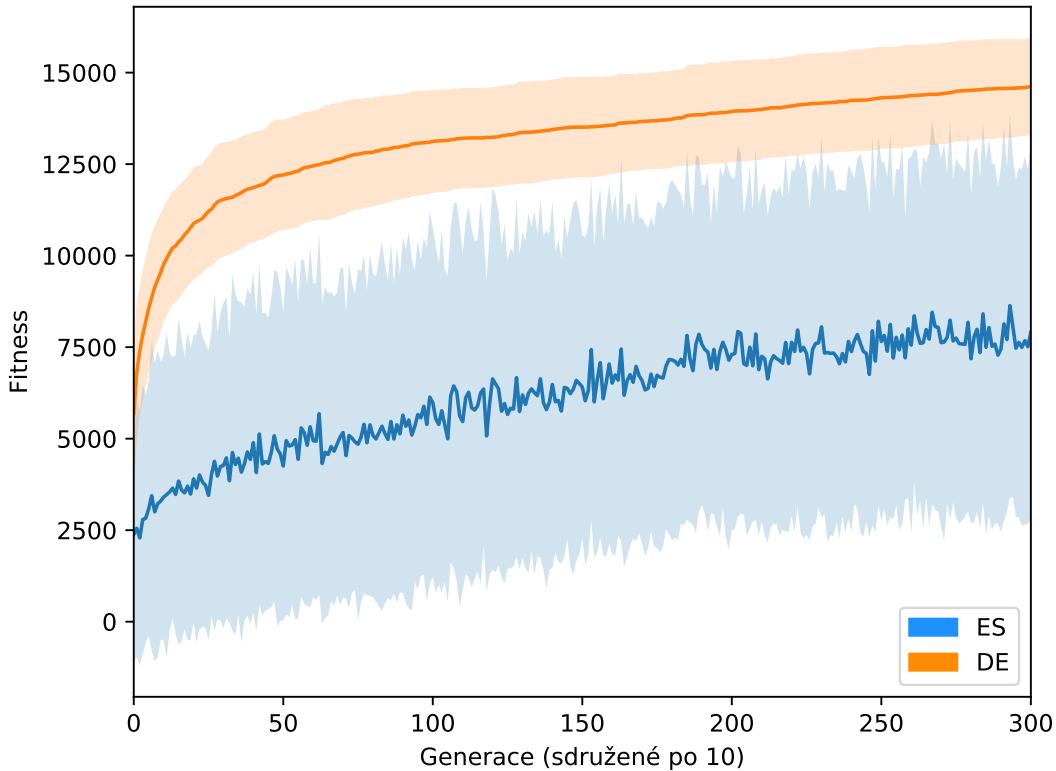
Roboti:	<i>Scout – 5, Worker – 4</i>
Počet generací:	4000
Počet iterací map	2000
Velikost generace(DE)	200
Počet jedinců(ES)	10
Počet mutovaný potomků(ES)	20
Elitismus(ES)	<i>Ano</i>
Elitismus(DE)	<i>Ne</i>

Fitness funkce

Hodnota nalezeného pokáceného stromu	100
Hodnota uloženého dřeva	1000
Hodnota dřeva v kontejneru	100
Hodnota jiné entity v kontejneru	–100
Hodnota kolize	–1
Ostatní hodnoty:	0
Počet stromů:	400
Počet už pokácených stromů	0

Tabulka 4.9: Wood hlavní úkol - nastavení experimentu

Na 4.8 lze pozorovat, že pro ES bylo velmi obtížné skládat už úspěšné chování do lepších celků a velmi osciluje. ES uškodilo agresivní prohledávání prostoru a DE díky své architektuře vytěžilo z optimalizovaných chování maximum. Nejlepšímu jedinci se budeme věnovat podrobněji v další kapitole.



Obrázek 4.8: Wood hlavní úkol - porovnání průměrné fitness ES a DE

Výsledky Experimentu

Výsledkem posloupnosti všech podúkolů vzniklo poměrně velmi komplexní chování. Finální neuronové sítě ať u Worker robotů, či Scout robotů se zvládají vyhýbat překážkám. Scout roboti kácí stromy, které naleznou. Worker roboti nakládají zpracované dřevo, pokud na něj narazí, když zachytí signál úložiště, tak vyloží aktuální náklad. Některá chování byla také schopna předejít zaseknutí o nějaký shluk objektů, pokud byl jejich pohyb vpřed neúspěšný, tak po několika pokusech roboti vycouvali a vydali se cestou okolo kritického místa. U většiny se také objevilo použití rádiových signálů jako prostředku pro největší možné rozptýlení po mapě, jakmile zachytí cizí signál vydají se opačným směrem. Průběh fitness jednotlivých podúkolů je zachycena na předchozích grafech 4.3 až 4.7. V tabulce 4.11 můžete vyčíst průměrné počty nalezených stromů, uskladněného materiálu, apod ze 100 simulací mapy.

Ač se jedná o nejlepší dosažené chování objevují se nějaké nedostatky. Worker robot se občas dostane do pozice, ze které není schopen vyjet, jedná se především o kolize s vícero entitami. Tento problém by mohlo vyřešit použití vícevrstvých sítí či evolučního algoritmu evolvujícího i architekturu sítě. Skládání zpracovaného materiálu po obvodu skladiště není efektivní způsob, jak do něj naskládat maximální množství dřeva. V tomto případě jsem se snažil vylepšit tento nedostatek promítnutím vzdálenosti dřeva od středu skladiště do celkové fitness, ovšem bez znatelného zlepšení v chování. Nejspíše by bylo třeba použít rádiový senzor poskytující více informací o směru k zachycenému signálu.

Inicializační nastavení:

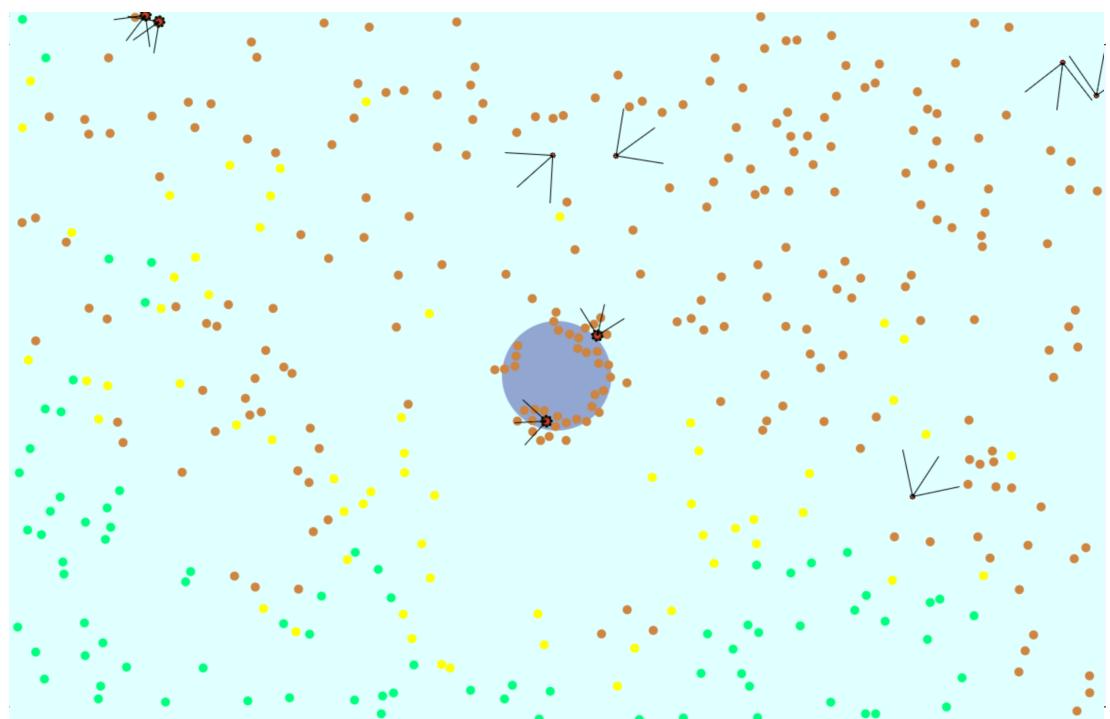
Výška	800
Šířka	1200
Počet iterací	10000
Počet stromů	400
Počet Scout robotů	5
Počet Worker robotů	4

Tabulka 4.10: Wood Scene - nastavení mapy pro testovací experiment

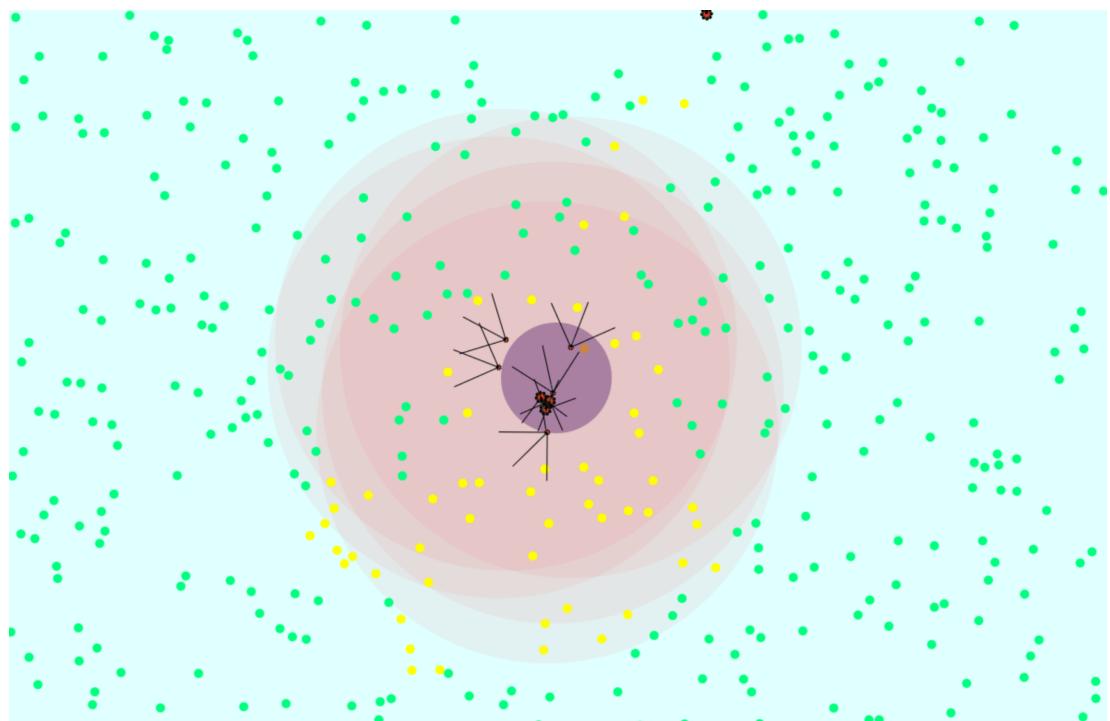
Výsledky

Zpracované dřevo zanechané v mapě	225,22
Stromy v mapě	156,22
Z toho nalezené	52,84
Dřevo v kontejnerech	18,48
Uskladněné dřevo	17,56

Tabulka 4.11: Wood Scene - výsledky simulace nejlepšího jedince, průměr ze 100 simulací testovacího experimentu



Obrázek 4.9: Wood nejlepší jedinec - 10000 iterací simulace



Obrázek 4.10: Wood náhodný jedinec - 10000 iterací simulace

4.3 Mineral Scene

Tento scénář si bere jako inspiraci strategické hry např. Starcraft (Blizzard-Entertainment, 1998) a hypotetické přežití robotů na cizí planetě, kde si budou muset obstarat vlastní nerostné suroviny pro běh. Opět je klíčová spolupráce mezi roboty, kdy roboti mají různé cíle. Jejich společným cílem je maximalizovat množství vyrobeného paliva.

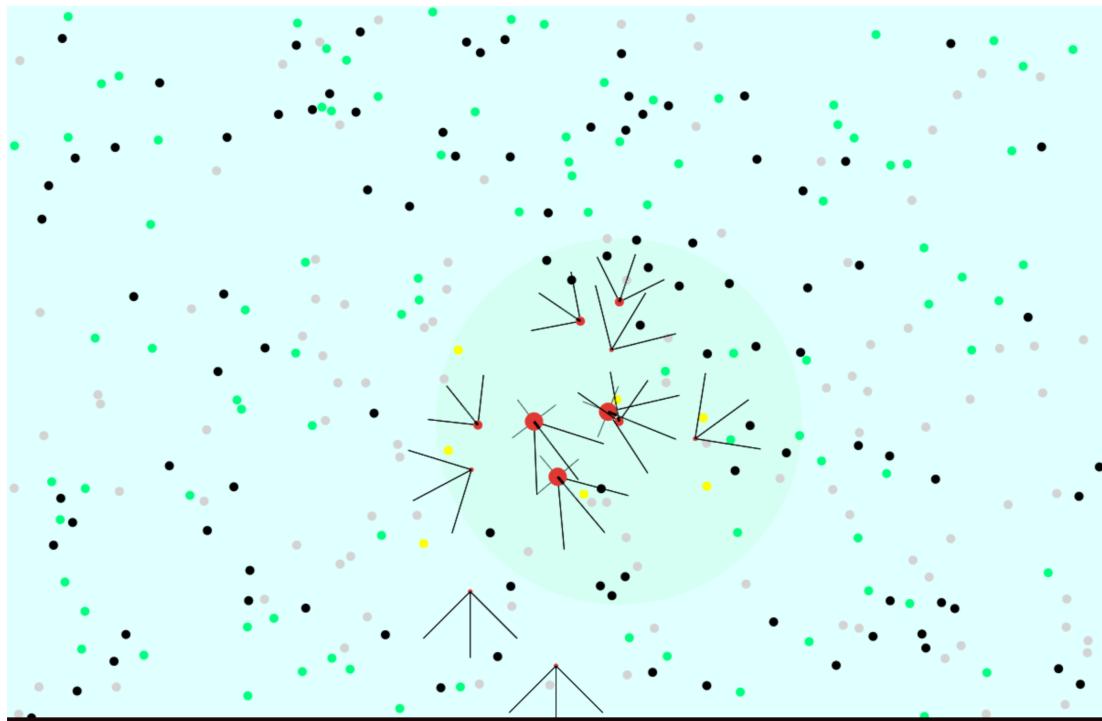
V Mineral Scene scénáři se palivo vyrábí z minerálů, které jsou náhodně rozmištěny po mapě. Transformaci minerálu na palivo dokáže pouze největší robot *Refactor*. V mapě se dále nacházejí překážky a volné palivo, které mohou roboti ihned po sebrání použít. V rámci hlavního cíle roboti musí nejdříve nalézt minerál, Refactor jej přeměnit a pak případně předat ostatním robotům.

V Mineral Scene figurují dohromady tři rozliční roboti, všichni potřebují pro pohyb odlišné množství paliva. Nejmenší robot *Mineral Scout* disponuje pouze senzory k exploraci prostředí a rádiovým vysílačem pro komunikaci se skupinou, opět má unikátní kód jako v předchozím scénáři. Robot střední velikosti *Mineral Worker* se pohybuje o něco pomaleji než Mineral Scout, ale umí přesouvat objekty i více najednou. Robot pro přeměnu minerálu (suroviny na výrobu paliva), označen v simulaci jako Mineral Refactor, se přemisťuje nejpomaleji a má možnost přeměnit minerál na palivo.

Jedná se o složitější cíl než v předchozím scénáři. V mapě se vyskytuje navíc překážky a celkový počet entit na mapě je vyšší. Zatímco ve Wood Scene je místo přesunu surovin pevně dané, zde musí transportní roboti hledat Refaktor roboty.

Více detailů u obrázků 4.11.

Na obrázku 4.11 je vyobrazena vizualizace daného scénáře. Roboti jsou vybarveni červenou barvou a jednotlivé druhy od sebe lze snadno rozoznать podle velikosti (nejmenší Scout robot, poté Worker robot a největší Refaktor robot), Zelené kroužky představují minerály pokud je minerál objevený hejnem obarví se žlutě. Šedivou barvou jsou vyvedeny překážky a černou palivo.



Obrázek 4.11: Příklad Mineral Scene mapy: konfigurace po startu s náhodným chováním

Roboti

V Mineral Scene scénáři se vyskytuje 12 robotů. V hejnu se vyskytují 3 různé druhy robotů: Scout robot, Worker robot, Refaktor robot. V implementaci jsou odlišeni od předchozího experimentu prefixem Mineral. Jejich ovládání stejně jako v předchozím experimentu probíhá pomocí neuronových sítí. Jedinec má opět podobu vektoru vah jednotlivých neuronových sítí a hodnoceno je celé hejno. Každý robot má nádrž na palivo a každá iterace mapy ho stojí jednu jednotku paliva. Nyní popíšeme jednotlivé roboty.

Scout robot

Jak název napovídá Scout robot prozkoumává mapu. Oproti Wood Scene nemá žádnou další funkci. Ze všech robotů se pohybuje nejrychleji a díky malé velikosti dokáže projíždět i užšími prostory. Informace o prostředí získává pomocí úsečkových senzorů a velkého kruhového senzoru, který poskytuje počet jednotlivých druhů entit v celém okruhu. Pro komunikaci se zbytkem používá rádiový signál s kódem 0.

Scout Robot	
Tvar:	<i>Kruh</i>
Poloměr:	2,5
Namespace:	<i>MineralRobots</i>
Název:	<i>ScoutRobotMem</i>
Velikost kontejneru:	0
Efektory	
Motor:	<i>Dvoukolečkový</i>
Maximální rychlosť:	3
Kód rádiového signálu:	0
Poloměr signálu:	200
Počet paměťových slotů:	10
Obsah slotu:	<i>float</i>
Senzory	
Počet line senzorů:	3
Délka line senzorů:	70
Orientace l. senzorů:	0°, ± 45°
Počet fuel senzorů:	3
Délka fuel senzorů:	70
Orientace f. senzorů:	0°, ± 45°
Poloměr rádiového přijímače:	100
Počet touch senzorů:	3
Lokátorový senzor	

Tabulka 4.12: Mineral Scene - Scout robot specifikace

Worker Robot

Mineral Worker roboti zastupují úlohu transportu minerálů. Pohybují se rychleji než Refaktor roboti, ale zase pomaleji než Scout roboti. Uloží do svého kontejneru až 5 minerálů. Pro komunikaci využívají rádiového signálu s kódem 1. Má k dispozici Picker pro nákládání a vykládání entit z kontejneru.

Worker Robot	
Tvar:	<i>Kruh</i>
Poloměr:	5
Namespace:	<i>MineralRobots</i>
Název:	<i>WorkerRobotMem</i>
Velikost kontejneru:	5
Efektor	
Motor:	<i>Dvoukolečkový</i>
Maximální rychlosť:	1,5
Kód rádiového signálu:	1
Poloměr signálu:	200
Dosah pickeru:	10
Počet paměťových slotů:	10
Obsah slotu:	<i>float</i>
Senzory	
Počet line senzorů:	3
Délka line senzorů:	50
Orientace l. senzorů:	0°, ±45°
Počet fuel senzorů:	3
Délka fuel senzorů:	50
Orientace f. senzorů:	0°, ±45°
Poloměr rádiového přijímače:	100
Počet touch senzorů:	3
Lokátorový senzor	

Tabulka 4.13: Mineral Scene - Worker robot specifikace

Refaktor robot

Nenahraditelnou roli zastává Mineral Refaktor, dokáže totiž měnit minerál na jednotku paliva. Pro přeměnu musí být minerál připraven na vrcholu kontejneru a po procesu přeměny se místo minerálu objeví palivo. Refaktor robot je však oproti ostatním robotům značně neohrabaný, zvláště kvůli jeho velikosti. Poloměr Refaktor robota odpovídá dvěma Worker robotům (resp. čtyřem Scout robotům). Pohybuje se z nich také nejpomaleji. Jeho rádiové signály nesou kód 2. Nakladače (vykladače) má do všech čtyřech světových směrů.

Refaktor Robot	
Tvar:	<i>Kruh</i>
Poloměr:	10
Namespace:	<i>MineralRobots</i>
Název:	<i>RefactorRobotMem</i>
Velikost kontejneru:	5
Efektor	
Motor:	<i>Dvoukolečkový</i>
Maximální rychlosť:	1,5
Kód rádiového signálu:	2
Poloměr signálu:	200
Počet pickerů	4
Orientace pickerů	0°, 90°, 180°, 270°
Dosah pickerů:	20
Počet paměťových slotů:	10
Obsah slotu:	<i>float</i>
Refaktor:	<i>Minerál ⇒ Palivo</i>
Dosah refaktoru:	<i>kontejner</i>
Kapacita refaktoru:	1
Senzory	
Počet line senzorů:	3
Délka line senzorů:	70
Orientace l. senzorů:	0°, ±35°
Počet fuel senzorů:	3
Délka fuel senzorů:	70
Orientace f. senzorů:	0°, ±35°
Poloměr rádiového přijímače:	100
Počet touch senzorů:	3
Lokátorový senzor	

Tabulka 4.14: Mineral Scene - Refaktor robot specifikace

Vyhodnocování Fitness

Jako fitness funkci pro Mineral Scene scénář jsem, po dobrých zkušenostech z předchozího experimentu, použil vážený součet následujících charakteristik mapy po konci simulace. Tento součet je specifický pro každý podúkol zvlášť. Kvůli komplexnosti tohoto úkolu a velikosti hejna v jednotlivých podúkolech nevystupují roboti vždy v plném počtu, ale nejdříve se optimalizuje chování pro pár jedinců od každého druhu. I tak bylo potřeba zmenšit velikost populace pro rozumnou dobu času běhu. Pozitivní hodnocení roboti získávali za:

- *objevené minerály* - minerály o které zavadil line senzor
- *uložené minerály* - minerály nacházející se v kontejnerech robotů
- *přeměněné palivo* - palivo nacházející se na mapě či v kontejnerech robotů
- *palivo v nádržích* - palivo uvnitř nádrží jednotlivých robotů
- *odložené minerály* - minerály na pomocném prostoru označeným rádiovým signálem

A negativní pouze za:

- *kolize* - počet pokusů o pohyb který by vedl ke kolizi

Podúkoly

Podle výsledků předchozího experimentu jsem se tentokrát výhradně soustředil na DE, které vycházelo lépe.

Opět jsem vygeneroval první neuronové sítě náhodně i další kroky jsou podobné jako v předchozím případě, nejdříve jsem navrhl fitness funkce pro učení chůze, vyhýbání, sbírání objektů a jejich skládání. Dále však nebyla zřejmá posloupnost jednotlivých úkonů, proto fitness funkce odpovídá už výslednému cíli, množství vytvořeného paliva a sebraným minerálů.

Od následující stránky se budu soustředit na jednotlivé podúkoly. U každého podúkolu bude uveden krátký popis cíle jednotlivého experimentu, obrázek s grafy průběhu fitness, stručné shrnutí chování a grafů s průběhy. Grafy budou vždy stejného formátu, osa y zobrazuje hodnotu fitness, osa x odpovídá číslu generace, pro lepší čitelnost jsou generace sdruženy po 10.

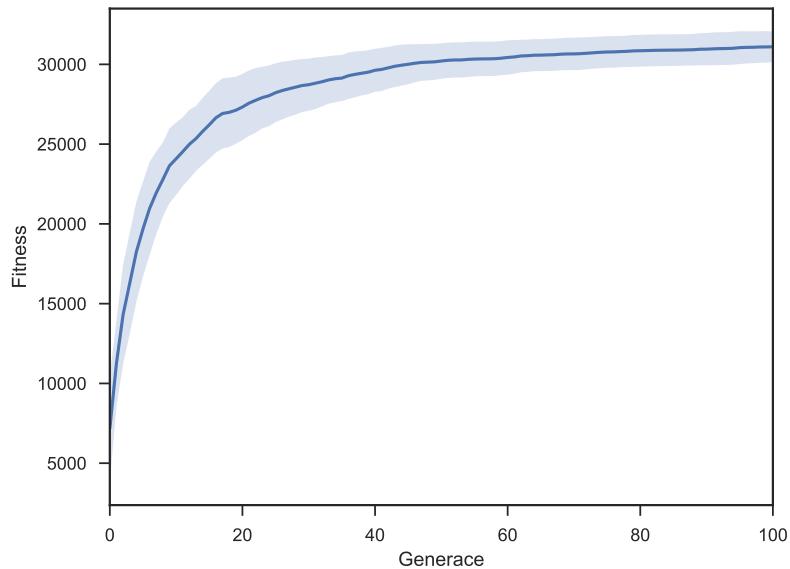
Scout chůze - nastavení experimentu

Stejně jako u předchozího scénáře, jsem roboty nejdříve učil pohybu. Opět jsem roboty oddělil, aby optimalizace neupřednostňovala ty rychlejší. Nejprve byly vygenerované náhodné neuronové sítě pro řízení Scout robotů. Poté byly optimalizovány pomocí fitness, která oceňuje chování podle počtu objevených minerálů.

Z grafu 4.12 lze vyčíst, že ani větší množství entit v mapě nevadilo pro optimální slušného chování pro vyhýbání se překážkám a prohledávání mapy. Nejlepší chování dokáže odhalit okolo 75% minerálů na mapě.

Nastavení mapy a EA	
Roboti a jejich počet:	<i>Scout</i> – 5
Počet generací:	1000
Počet iterací map	1000
Velikost generace(DE)	100
Fitness funkce a objekty na mapě	
Hodnota nalezeného minerálu	100
Ostatní hodnoty:	0
Počet minerálů:	400
Počet překážek	100
Počet paliva	100

Tabulka 4.15: Mineral Scout chůze - nastavení experimentu



Obrázek 4.12: Mineral Scout chůze - průběh fitness DE

Worker chůze - nastavení experimentu

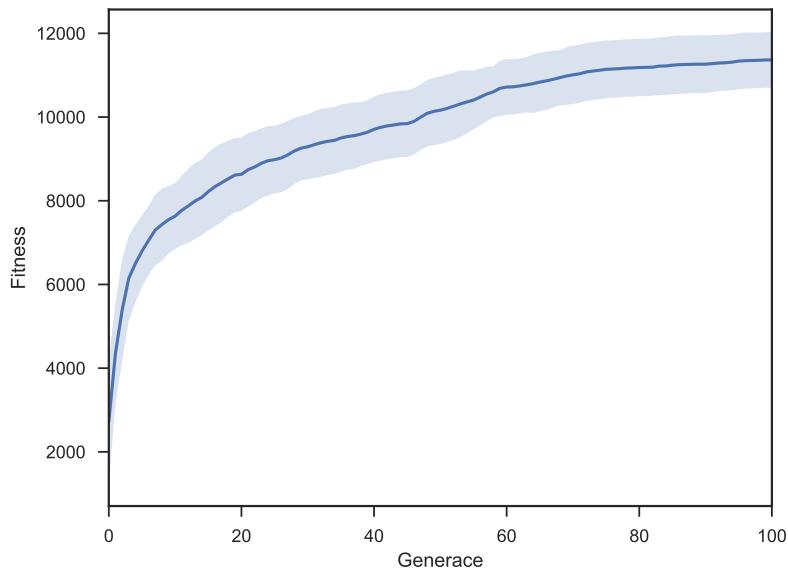
I pro Worker robota bylo potřeba optimalizovat chůzi v mapě. Stejně jako u Scout robotů byli Worker roboti odměňováni za nalezené minerály.

Nejlepší chování u Worker robota dokáže objevit více než čtvrtinu minerálů, jak ukazuje obrázek 4.13 Což je také uspokojivý výsledek, přihlédneme-li k tomu, že Scout robot má poloviční rychlosť a velikost než Worker robota. Navíc díky delším senzorům objevuje Scout robot objekty na větší vzdálenosti.

Nastavení mapy a EA	
Roboti a jejich počet:	<i>Worker</i> – 4
Počet generací:	1000
Počet iterací map	1000
Velikost generace(DE)	100

Fitness funkce a objekty na mapě	
Hodnota nalezeného minerálu	100
Ostatní hodnoty:	0
Počet minerálů:	300
Počet překážek	100
Počet paliva	100

Tabulka 4.16: Mineral Worker chůze - nastavení experimentu



Obrázek 4.13: Mineral Worker chůze - průběh fitness DE

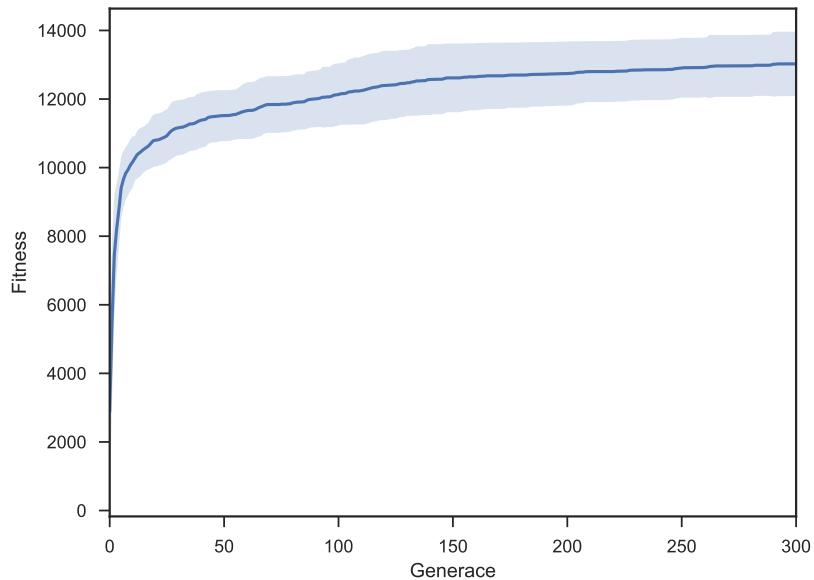
Worker sbírání - nastavení experimentu

Na rozdíl od Wood Scene zde není zcela jasné, kam mají Worker roboti ukládat sebrané minerály. Aby je v budoucích podíkolech umisťovali roboti, co nejbližše k Refaktor robotům, přidal jsem prozatím do středu mapy pomocný rádiový signál se stejným kódem jako mají Refaktor roboti.

Roboti v prvních 500 generacích zaplní své kontejnery minerály, což může pozorovat na grafu 4.14. Dále dokonce dováželi roboti do středu několik minerálů.

Nastavení mapy a EA	
Roboti a jejich počet:	<i>Worker</i> – 2
Počet generací:	3000
Počet iterací map	1000
Velikost generace(DE)	100
Fitness funkce a objekty na mapě	
Pomocný rádiový signál:	<i>Ano</i>
Hodnota nalezeného minerálu	1
Hodnota minerálu v pomoc. signálu	1010
Hodnota uložených minerálů	1000
Ostatní hodnoty:	0
Počet minerálů:	400
Počet překážek	100
Počet paliva	100

Tabulka 4.17: Mineral Worker sbírání - nastavení experimentu



Obrázek 4.14: Mineral Worker sbírání - průběh fitness

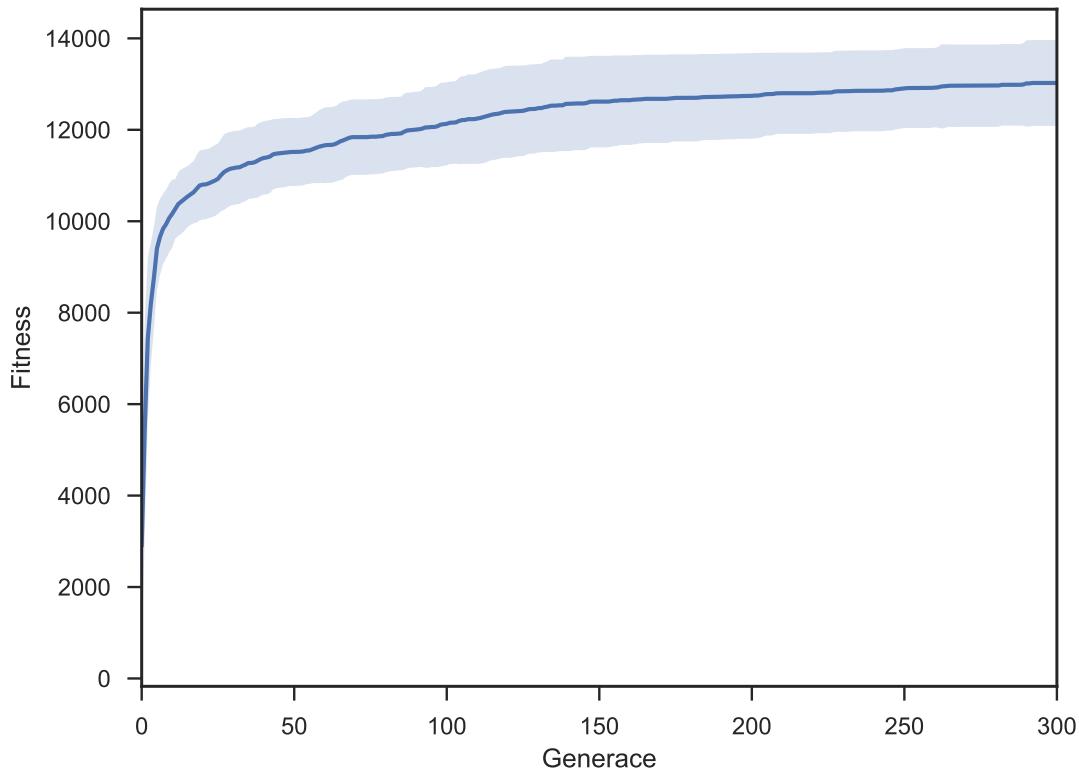
Worker skládání - nastavení experimentu

V dalším metaúkolu jsem se soustředil výhradně na skládání minerálů do středu na místo označené kódem 2. Konkrétní nastavení experimentu popisuje tabulka 4.18. Oproti předchozímu scénáři optimalizuji Workery v plném počtu, aby se optimalizovala komunikace a rozptylování Worker robotů.

Výsledek tohoto podúkolu odpovídá Wood Scene - sbírání, roboti dokázali do středu převést přibližně 10 minerálů, opět měli obtíže s uspořádáním. Vizuální výsledek potvrzuje graf 4.15.

Nastavení mapy a EA	
Roboti a jejich počet:	<i>Worker</i> – 4
Počet generací:	6000
Počet iterací map	1000
Velikost generace(DE)	100
Fitness funkce a objekty na mapě	
Pomocný rádiový signál:	<i>Ano</i>
Hodnota nalezeného minerálu	1
Hodnota minerálu v pomoc. signálu	1000
Hodnota uložených minerálů	10
Ostatní hodnoty:	0
Objekty na mapě	
Počet minerálů:	400
Počet překážek	100
Počet paliva	100

Tabulka 4.18: Mineral Worker skládání - nastavení experimentu



Obrázek 4.15: Mineral Worker skládání - průběh fitness DE

Refaktor chůze - nastavení experimentu

Poslední metaúkol zabývající chůzí byl pro Refaktor robota i v jeho případě byl hodnocen podle počtu nalezených minerálů jako u ostatních robotů. Tabulka 4.16 ukazuje přesné nastavení metaúkolu.

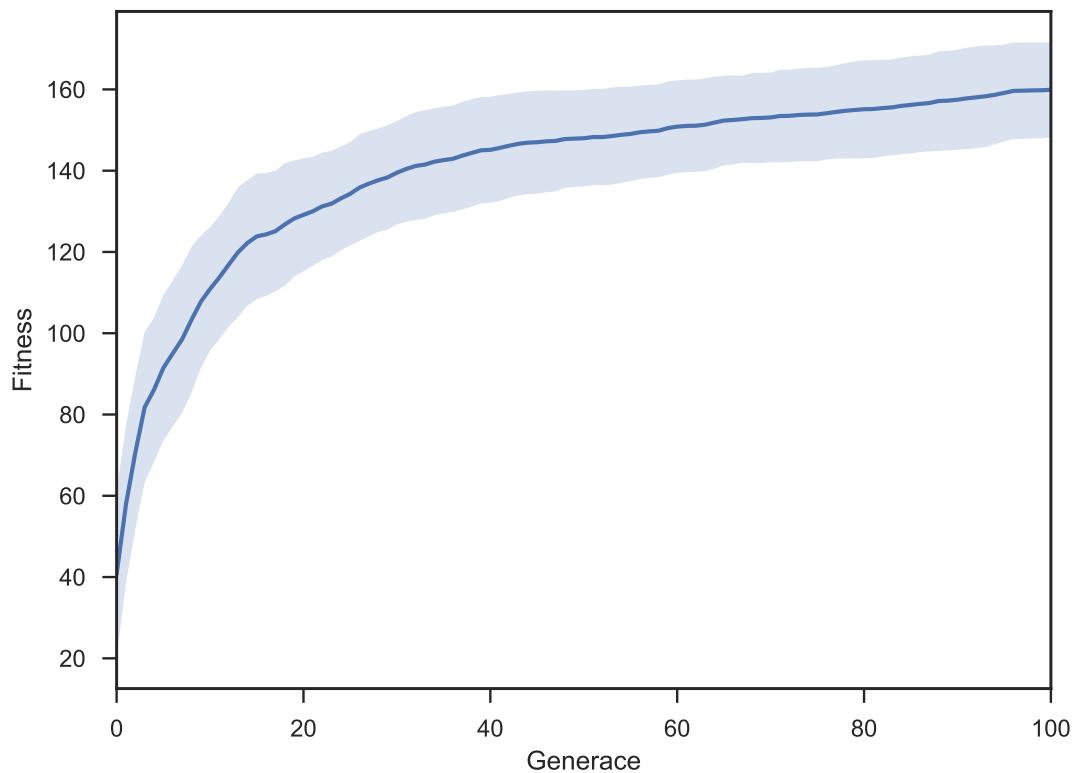
Už v 200 generaci dokážou Refaktor robot odhalí jednu pětinu minerálů, jak je vidět v průběhu grafu na 4.16. Ač podle fitness se zdá, že roboti prohledávají mapu a vyhýbají se překážkám. Po krátkém zkoumání jejich chování jsem zjistil, že používají nakladače a přehazují překážky za sebe.

Nastavení mapy a EA	
Roboti a jejich počet:	<i>Refaktor</i> – 3
Počet generací:	1000
Počet iterací map	1500
Velikost generace(DE)	100

Fitness funkce	
Hodnota nalezeného minerálu	1
Ostatní hodnoty:	0

Objekty na mapě	
Počet minerálů:	500
Počet překážek	100
Počet paliva	100

Tabulka 4.19: Mineral Refaktor chůze - nastavení experimentu



Obrázek 4.16: Mineral Refaktor chůze - průběh fitness u DE

Refaktor Worker kooperace - nastavení experimentu

Jako první kooperativní metaúkol jsem zvolil spolupráci Refaktor robota s Workera robota. Zamýšlel jsem, že Worker roboti seberou minerály a budou je přiblížovat k Refaktor robotovi, který je bude přetvářet na palivo. Tomuto účelu jsem přizpůsobil ohodnocení fitness, jedinci jsou nejvíce oceněni za přeměněné palivo vložené do mapy nebo už v nádrži. Další podrobnosti v tabulce 4.17

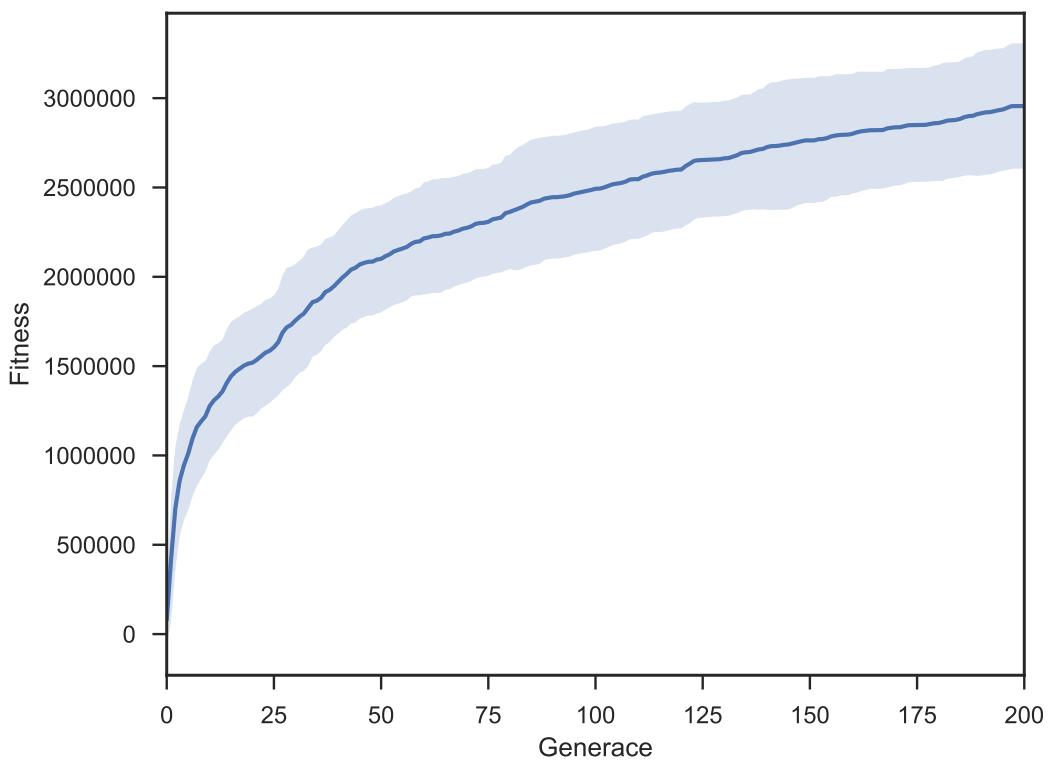
Ač se podle grafu 4.17, že optimalizaci obou robotů proběhla úspěšně. Ale po vizuální kontrole jsem zjistil, že se zlepšoval pouze Refaktor robot. EA však v rámci nejlepších chování nezapojila Worker robota.

Nastavení mapy a EA	
Roboti:	<i>Refaktor, Worker</i>
Počty robotů:	$R - 1, W - 2$
Počet generací:	2000
Počet iterací map	1500
Velikost generace(DE)	100

Fitness funkce	
Hodnota uložených minerálů	1
Hodnota přeměněho paliva	1000
Hodnota paliva v nádržích	1000
Ostatní hodnoty:	0

Objekty na mapě	
Počet minerálů:	400
Počet překážek	100
Počet paliva	0

Tabulka 4.20: Mineral Refaktor Worker kooperace - nastavení experimentu



Obrázek 4.17: Mineral Refaktor Worker kooperace - průběh fitness DE

Hlavní úkol kooperace - nastavení experimentu

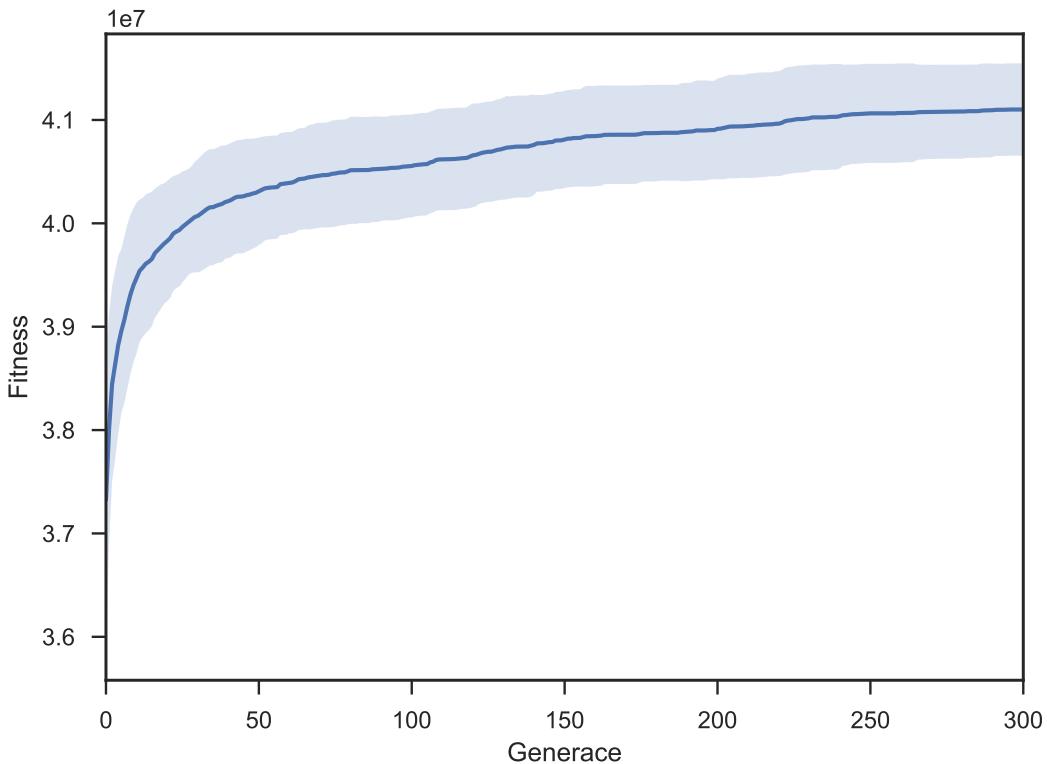
V rámci finálního podúkolu byla fitness nastavena, aby odpovídala hlavnímu cíli scénáře. Oceňuje roboty pouze za přetvořené palivo a palivo v nádržích. V tomto podúkolu už figurují všechny druhy robotů. Další podrobnosti jsou zaneseny v tabulce 4.21.

Křivka opět ukazuje vzrůst fitness, hodnota se ustaluje kolem $4.1 \cdot 10^7$. Při přeměně minerálu vznikne 100 jednotek paliva. Roboti dostali na začátku simulace více paliva než potřebovali na běh, tohoto paliva jim zbylo 8 500 jednotek (ve fitness $0.85 \cdot 10^7$). Také výsledná fitness tohoto experimentu odpovídá přibližně 300 zpracovaným minerálům. Nejlepšímu chování se budeme věnovat v rámci výsledků experimentů 4.3.

Nastavení mapy a EA	
Roboti:	<i>Refaktor, Worker, Scout</i>
Počty robotů:	$R = 2, W = 3, S = 4$
Počet generací:	2000
Počet iterací map	1500
Velikost generace(DE)	100

Fitness funkce a objekty na mapě	
Hodnota přeměného paliva	1000
Hodnota paliva v nádržích	1000
Ostatní hodnoty:	0
Počet minerálů:	400
Počet překážek	100
Počet paliva	0

Tabulka 4.21: Mineral Refaktor Worker kooperace - nastavení experimentu



Obrázek 4.18: Mineral Refaktor Worker kooperace - průběh fitness DE

Výsledky Experimentu

Ač předchozí graf fitness 4.18 vypadal velmi optimisticky, výsledky 4.23 z vícero běhů na různých mapách jim neodpovídají.

Po dlouhém pozorování optimalizovaných jedinců jsem dospěl k závěru, že optimalizace se u takto složitých úkolů soustředila pouze na Refaktor robota. A to z jednoduchého důvodu, protože sám o sobě dokázal nejvíce ovlivňovat fitness funkci pro optimalizaci byl tutíž nejvhodnější volbou. Dále jelikož úkony spojené s hlavním cílem scénáře byly velmi složité, tak se chování optimalizovalo v závislosti na aktuálně vygenerované mapě. Každému experimentu odpovídá jedna náhodně generovaná mapa kvůli jednoznačnosti ohodnocení fitness. Nicméně v jednodušších podílkolech dokázala DE vyevolvovat univerzální a úspěšné jedince. V další optimalizaci je na obtíž, že Refaktor robot zvládá všechny úkony sám a díky tomu by bylo potřeba navrhnout fitness reflektující tento hlavní problém, tak aby nutila ostatní roboty k činnosti. Například v rámci vah velkou hodnotou penalizovat malý pohyb menších robotů, případně hodnotit jen minerály objevené menšími roboty.

Pro testovací běhy jsem použil mapu s nastavením podle tabulky 4.22. Jedná se opět o 100 různých běhů a v tabulce je možné vidět průměrné hodnoty charakteristik výsledku.

Inicializační nastavení:

Výška:	800
Šířka:	1200
Počet iterací:	1500
Počet minerálů:	400
Počet překážek:	50
Počet Scout robotů:	4
Počet Worker robotů:	3
Počet Refaktor robotů:	2
Inicializační palivo:	1500
Spotřeba paliva robotů:	1/kolo

Tabulka 4.22: Mineral Scene - nastavení mapy pro testovací experiment

Výsledky testovacích běhů jsou zaneseny v tabulce 4.23.

Výsledky

Překážky v mapě	49,98
Objevené překážky	2,51
Minerálů v mapě	393,7
Z toho nalezených	17,25
Palivo v kontejnerech	0,02
Zbylé palivo v nádržích	59
Kolize	9

Tabulka 4.23: Mineral Scene - výsledky simulace nejlepšího jedince, průměr ze 100 simulací testovacího experimentu

V tomto případě ani nepřikládám obrázek s koncem běhu tohoto chování, neboť je velmi podobný tomu z náhodného běhu. Všechna chování nejlepší generace si lze prohlédnout v programu na přiloženém CD.

4.4 Competitive Scene

Poslední ze scénářů se týká soutěžení dvou týmů (hejn), kteří se snaží ty druhé zničit. Úspěšnost týmu je dána zachovanými jednotkami zdraví robotů a uděleným poškozením do nepřátelské skupiny robotů.

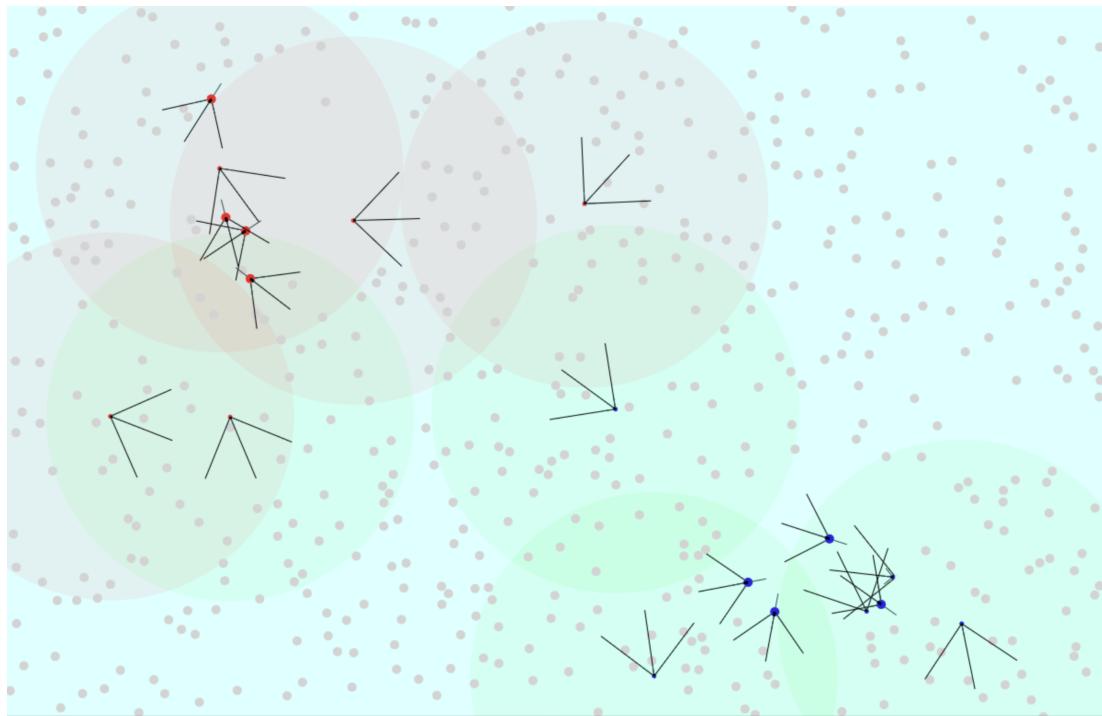
V mé Competitive Scene scénáři proti sobě stojí dvě hejna, která čítají každá 9 robotů. Mimo roboty jsou v mapě také náhodně rozmístěny překážky, kterým se musí roboti vyhýbat. Týmy začínají v první a poslední čtvrtině mapy, kde startují na náhodné pozici.

Hejno se skládá ze dvou druhů robotů. Roli průzkumníka zastává *Scout robot*, který je malý a rychlý. I Scout robot může způsobovat poškození, ovšem pouze jednu pětinu oproti druhému robotovi, pro kterého používám v rámci tohoto scénáře název *Fighter robot*. Fighter robot se pohybuje méně obratně, jelikož je dvakrát větší a pomalejší.

V tomto scénáři mají všichni roboti k dispozici rádiové signály s kódy nula až 4, tudíž je čistě na optimalizaci, jak je využije. Competitive Scene obsahuje základní rojové scénáře jako je komunikace, vyhybání překážkám, hledání v mapě, apod. Tentokrát budou mít roboti obtížnější hledání cílů, protože budou pohyblivé a také jim mohou způsobit poškození.

Popis jednotlivých entit v mapě se nachází u obrázku 4.19

Na obrázku se nachází dvě devítičlenná hejna. Jednotlivé týmy jsou od sebe odděleny barvou, červená barva odpovídá týmu jedna a modrá týmu dva. Opět na mapě můžeme vidět rádiové signály, jedná se o kruhy s průhlednou barvou, jednotlivé kódy jsou barevně odlišeny. Šedivá kolečka pak znázorňují stejně jako v předchozím scénáři překážky.



Obrázek 4.19: Příklad Competitive Scene mapy: start náhodného chování

Roboti

Jak už jsem zmínil, ve Competitive Scene scénáři se objevují dva druhy robotů. V každém týmu se objevují 4 Scout roboti a 5 robotů typu Fighter. Pro účely správného vyhodnocování fitness zůstává po celou dobu běhu experimentu nepřátelský tým řízený stejným chováním. Typicky je na začátku pro nepřátelský tým vygenerováno náhodné chování. Podoba jedinců opět odpovídá předchozím experimentům i v kontextu nepřátelských robotů. V implementaci má aktuálně optimalizovaný tým označení tým 1 a protivník tým 2. Ted se podíváme na jednotlivé roboty.

Fighter Scout robot

Roli lehkého útočníka ve scénáři zastává Scout robot. Jedná se o malého a obratného robota. Dokáže způsobit poškození za 100 bodů zdraví. K poškození slouží úsečkový efektor (v rámci implementace pojmenovaný *Weapon*). Efektory *Weapon* fungují na stejném principu jako ostatní úsečkové efektory, pro udělení poškození musí kolidovat s jiným robotem. V rámci tohoto experimentu je možné udílet poškození i robotům z vlastního týmu. Jeho podrobnou specifikaci poskytuje tabulka 4.24

Fighter robot

Fighter robot je navržen jako těžký bitevník. Oproti Scout robotovi je větší a pomalejší. Nejmarkantnější rozdíl tvoří body zdraví a síla útoku. Fighter robot způsobuje pětkrát vyšší poškození a disponuje trojnásobkem bodů zdraví. Navíc může útočit najednou až čtyřmi zbraněmi, zatímco Scout robot pouze třemi. I on může útočit do vlastních řad. Jeho konkrétní specifikaci lze nalézt v tabulce 4.25.

Fighter Scout Robot	
Tvar:	<i>Kruh</i>
Poloměr:	2,5
Body zdraví:	500
Namespace:	<i>CompetitiveRobots</i>
Název:	<i>FighterScoutRobotMem</i>
Efektor	
Motor:	<i>Dvoukolečkový</i>
Maximální rychlosť:	3
Kód rádiového signálu:	0,1,2
Poloměr signálu:	200
Počet paměťových slotů:	10
Obsah slotu:	<i>float</i>
Počet zbraní:	3
Dosah zbraní:	10
Orientace zbraní:	0°, ±45°
Útok zbraní:	100
Senzory	
Počet line senzorů:	3
Délka line senzorů:	70
Orientace l. senzorů:	0°, ±45°
Poloměr rádiového přijímače:	100
Poloměr type senzoru:	50
Počet touch senzorů:	3
Lokátorový senzor	

Tabulka 4.24: Competitive Scene - Fighter Scout robot specifikace

Fighter Scout Robot	
Tvar:	<i>Kruh</i>
Poloměr:	5
Namespace:	<i>CompetitiveRobots</i>
Název:	<i>FighterRobotMem</i>
Body zdraví:	1500
Efektor	
Motor:	<i>Dvoukolečkový</i>
Maximální rychlosť:	1,5
Kód rádiového signálu:	0,1,2
Poloměr signálu:	200
Počet paměťových slotů:	10
Obsah slotu:	<i>float</i>
Počet zbraní:	4
Orientace zbraní:	0°, ±45°, 180°
Útok zbraní:	500
Senzory	
Počet line senzorů:	3
Délka line senzorů:	70
Orientace l. senzorů:	0°, ± 45°
Poloměr rádiového přijímače:	100
Poloměr type senzoru:	50
Počet touch senzorů:	3
Lokátorový senzor	

Tabulka 4.25: Competitive Scene - Fighter robot specifikace

Vyhodnocování Fitness

Hlavní cíl Competitive Scene scénáře se skládá z mnoha úkonů. Proto jsem ho stejně jako v předchozích scénářích rozdělil na menší metaúkoly. Tradičně jsem začal s učením pohybu a vyhybáním se překážkám. Poté jsem pokračoval přes uchování co nejvíce životních bodů, až po způsobení maximálního poškození protivníka. Fitness funkce má podobu váženého součtu stejně jako u Wood Scene a Mineral Scene. A používal jsem k optimalizaci pouze DE, která se ukázala v prvním scénáři jako účinnější metoda optimalizace chování.

Na konci simulace mapy jsou roboti oceněni za:

1. *nalezené překážky* - překážky o které zavadil line sensor
2. *zabité roboty* - mrtvé roboty nepřátelského týmu
3. *udělený útok* - útok udělený nepřátelským robotům
4. *zbývající životy* - životy zbývající aktuálním robotům

Trestáni za:

1. *kolize* - počet pokusů o pohyb při kterém by došlo ke kolizi

Podúkoly

V prvních podúkolech opět tradičně figurují roboti odděleně. Jejich princip je shodný, protože roboti mají velmi podobné funkce.

Po vzoru Wood Scene se jedná o :

1. vygenerování náhodného chování - Pro každého robota je vygenerována jednovrstvá neuronová síť s náhodnými vahami.
2. učení chůze - Roboti se jsou odděleně optimalizováni, aby dokázali objevit, co nejvíce překážek a nedocházelo ke kolizím.
3. šetření životů - Roboti se snaží udržet co největší počet zdravotních bodů. I tento scénář probíhá odděleně.
4. agresivní chování - Každý druh zvláště je optimalizován, aby působil maximální poškození. Roboti jsou oceňováni za způsobený útok a zabité roboty.
5. agresivní spolupráce - V posledním podúkolu figuruje celé hejno. Fitness je určena tak, aby roboty vedla ke zničení nepřátelského týmu.

Na další stránce budou následovat jednotlivá přesná nastavení podúkolů. Po psány budou obvyklou formou, u každého metaúkolu bude uveden stručný obsah podúkolu, tabulka s přesným nastavením a graf s průběhem fitness. Gafy stále dodržují předchozí vzor, osa y znázorňuje hodnotu fitness, osa x odpovídá číslu generace. Generace jsou sdruženy po 10. Křivkou je vyvedena průměrná fitness a modrá plocha odpovídá průměru fitness \pm směrodatná odchylka.

Scout chůze - nastavení experimentu

Podúkoly spojený s chůzí je podobný jako u předchozích scénářů, zvolil jsem pro fitness funkci počet překážek v mapě, jiné objekty ani na mapě rozmístěny nejsou.

Nastavení mapy a EA	
Roboti a jejich počet:	<i>Scout</i> – 5
Počet generací:	1000
Počet iterací map:	1000
Velikost generace(DE):	100

Fitness funkce	
Hodnota nalezené překážky:	100
Ostatní hodnoty:	0

Objekty na mapě	
Počet překážek:	500

Tabulka 4.26: Competitive Scout chůze - nastavení experimentu

Fighter chůze - nastavení experimentu

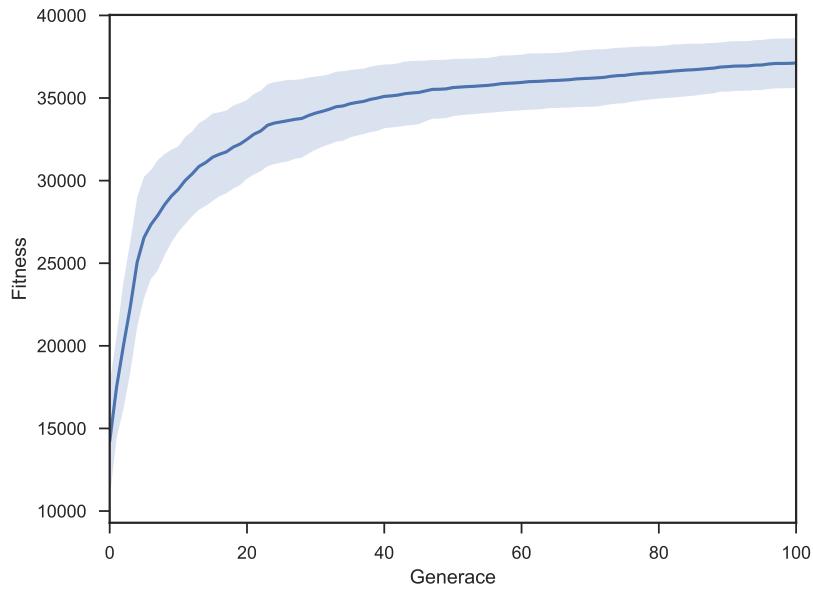
Tento podúkol odpovídá předchozí variantě pro Scout robota.

Nastavení mapy a EA	
Roboti a jejich počet:	<i>Fighter</i> – 4
Počet generací:	2000
Počet iterací map:	1000
Velikost generace(DE):	100

Fitness funkce	
Hodnota nalezené překážky:	100
Ostatní hodnoty:	0

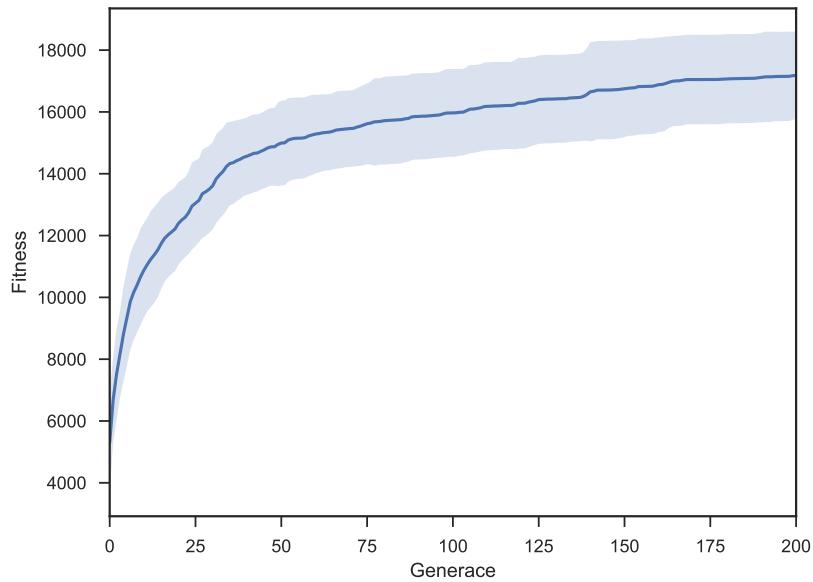
Objekty na mapě	
Počet překážek:	500

Tabulka 4.27: Competitive Fighter chůze - nastavení experimentu



Obrázek 4.20: Competitive Scout chůze - průběh fitness DE

V grafu 4.20 můžeme vidět, že Scout robot dokáže nalézt přes 350 překážek na mapě. Fighter objeví podle 4.21 přes 150 překážek mapy, což k přihlédnutí k délce senzorů a jeho velikost je také velmi dobrý výsledek.



Obrázek 4.21: Competitive Fighter chůze - průběh fitness DE

Scout životy - nastavení experimentu

V následujícím podúkolu byly roboti optimalizováni, aby nepůsobili poškození mezi sebou týmu. Nicméně bylo stále žádoucí, aby objevovali celou mapu, proto ve fitness zůstávají kladné body i za objevení překážek.

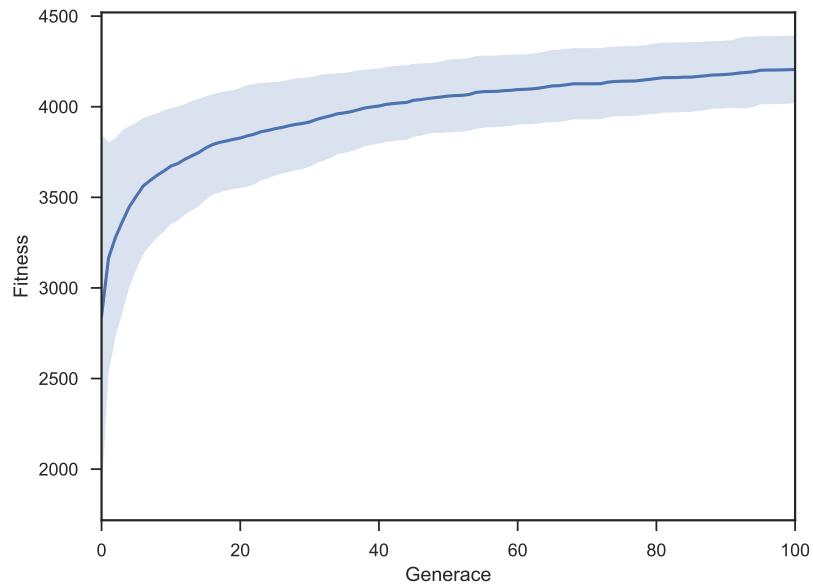
Celkový součet bodů zdraví u Scout robotů je 2500 (bodů fitness 25000). Z průběhu průměrné hodnoty fitness na grafu 4.22 můžeme vidět, že naprostá většina robotů v populaci, buď udržela své body zdraví či vyvážila jejich ztrátu nalezením dostatečného množství překážek.

Nastavení mapy a EA	
Roboti a jejich počet:	<i>Scout</i> – 5
Počet generací:	1000
Počet iterací map:	1000
Velikost generace(DE):	100

Fitness funkce	
Hodnota nalezené překážky:	10
Hodnota bodu zdraví:	10
Ostatní hodnoty:	0

Objekty na mapě	
Počet překážek:	500

Tabulka 4.28: Competitive Scout životy - nastavení experimentu



Obrázek 4.22: Competitive Scout životy - průběh fitness DE

Scout agresivní - nastavení experimentu

V posledním podúkolu, kde vystupují Scout roboti osamoceně. Fitness funkce cílí pouze na udělování poškození nepřátelským robotům a jejich zničení.

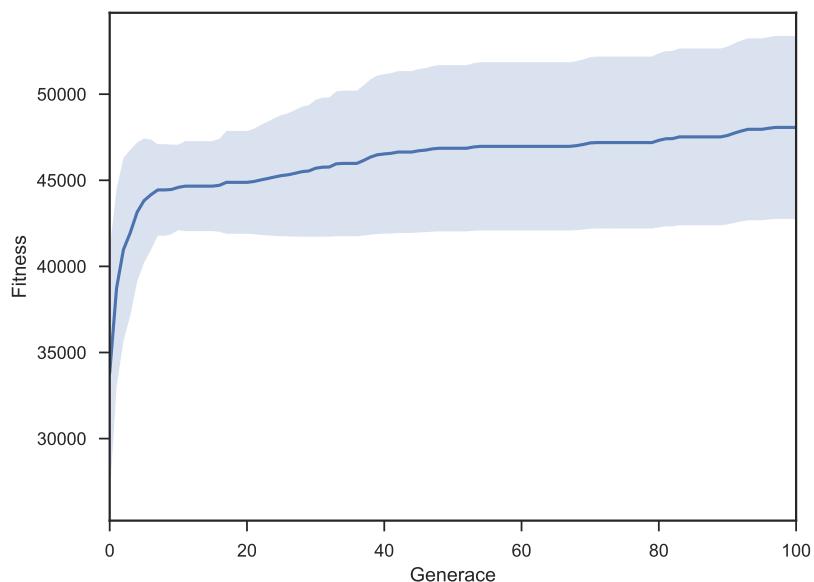
DE je schopno už během první 200 generací aktivně útočit na nepřátelské Scout roboty. Mapa je však vzhledem k velikosti robota velká a tudíž je obtížné někoho z nepřátelského týmu najít a udržet se za ním. Díky této obtížnosti roste směrodatná odchylka s generacemi, jak můžeme sledovat na grafu 4.23. Nejlepší jedinci, pak jsou schopni udělit až 500 bodů poškození, což je velmi dobrý výsledek vzhledem k obtížnosti úkolu.

Nastavení mapy a EA	
Roboti a jejich počet:	<i>Scout</i> – 5
Počet generací:	1000
Počet iterací map:	1500
Velikost generace(DE):	100

Fitness funkce	
Mrtvý nepřátelský robot:	1000
Udělený bod poškození:	100
Ostatní hodnoty:	0

Objekty na mapě	
Počet překážek:	500

Tabulka 4.29: Competitive Scout agresivní - nastavení experimentu



Obrázek 4.23: Competitive Scout agresivní - průběh fitness DE

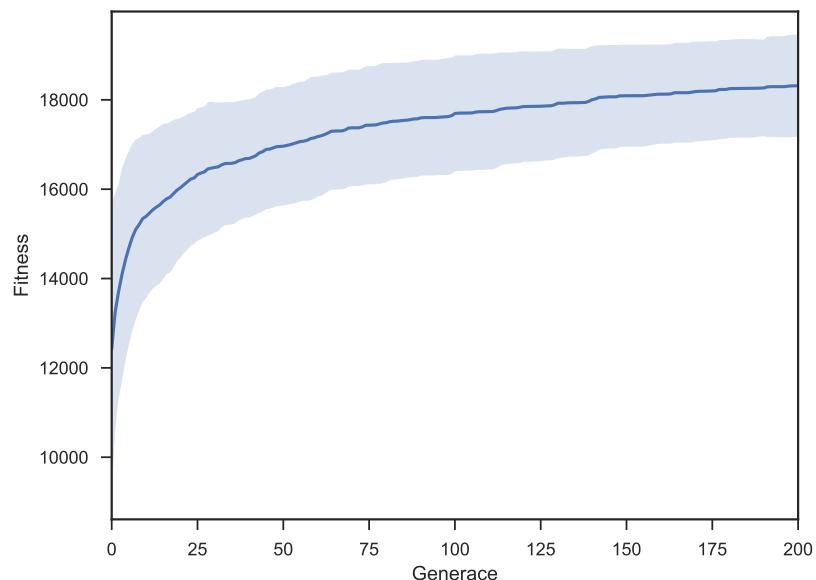
Fighter životy - nastavení experimentu

V tomto podúkolu chceme stejně jako ve variantě pro Scout robota, aby jedinci neničili roboty ze svého týmu. Ovšem není žádoucí, aby se hejno stalo neaktivní, proto fitness stále kladně hodnotí objevené překážky.

Čtyři Fighter roboti mají dohromady 6 000 bodů zdraví. Což odpovídá třetině hodnoty fitness nejlepšího jedince, jak je vidět na grafu 4.24. Takže jedinci z finální populace neútočí na sebe a ještě jsou schopni objevit více než pětinu překážek na mapě.

Nastavení mapy a EA	
Roboti a jejich počet:	<i>Fighter</i> – 4
Počet generací:	2000
Počet iterací map:	1000
Velikost generace(DE):	100
Fitness funkce	
Hodnota nalezené překážky:	100
Hodnota bodu zdraví:	1
Ostatní hodnoty:	0
Objekty na mapě	
Počet překážek:	500
Nepřátele roboti:	<i>Fighter</i>
Jejich počet:	$F - 4$

Tabulka 4.30: Competitive Fighter životy - nastavení experimentu



Obrázek 4.24: Competitive Fighter životy - průběh fitness DE

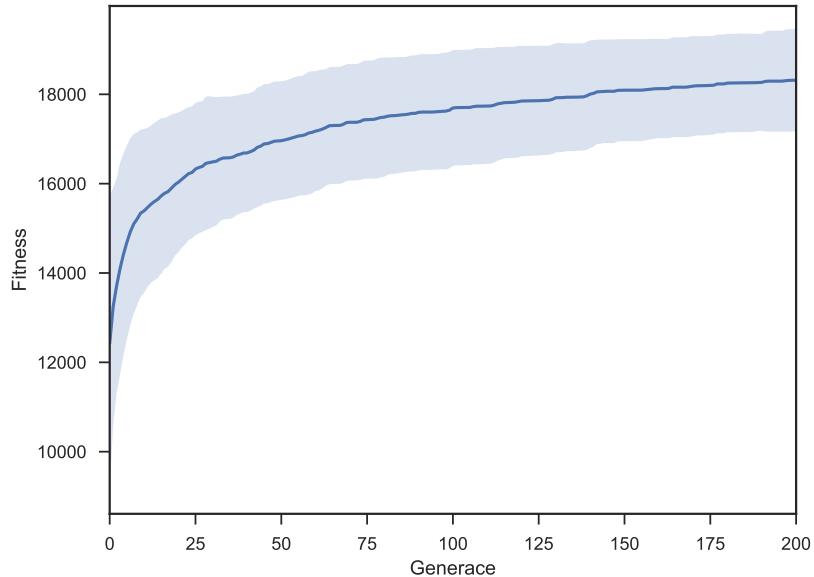
Fighter agresivní - nastavení experimentu

Agresivní podúkol pro Fighter robota koresponduje s nastavení pro Scout robota. Konkrétní nastavení lze vyčíst z tabulky 4.31. Kvůli velikosti mapy jsem přidal o jednoho Fightera robota více, než bylo v předchozím metaúkolu.

Křivka v grafu 4.25 ukazuje, že Fighter robot má ještě větší potíže najít nepřátele na mapě a udržet s nimi krok. Nejlepší jedinci jsou schopni zasáhnout jen 3x do nepřátel.

Nastavení mapy a EA	
Roboti a jejich počet:	<i>Fighter</i> – 5
Počet generací:	1000
Počet iterací map:	1500
Velikost generace(DE):	100
Fitness funkce	
Mrtvý nepřátelský robot:	100
Udělený bod poškození:	10
Ostatní hodnoty:	0
Objekty na mapě	
Počet překážek:	500
Nepřátelští roboti:	<i>Fighter</i>
Jejich počet:	<i>F</i> – 5

Tabulka 4.31: Competitive Fighter agresivní - nastavení experimentu



Obrázek 4.25: Competitive Fighter agresivní - průběh fitness DE

Competitive Scene kooperace - nastavení experimentu

Finální podúkolem Competitive scénáře je kooperace, kde vystupuje už celé hejno. Ohodnocení fitness odpovídá hlavnímu úkolu celého scénáře. Cílem je tedy maximalizovat udělené body poškození a počet zničených robotů soupeře.

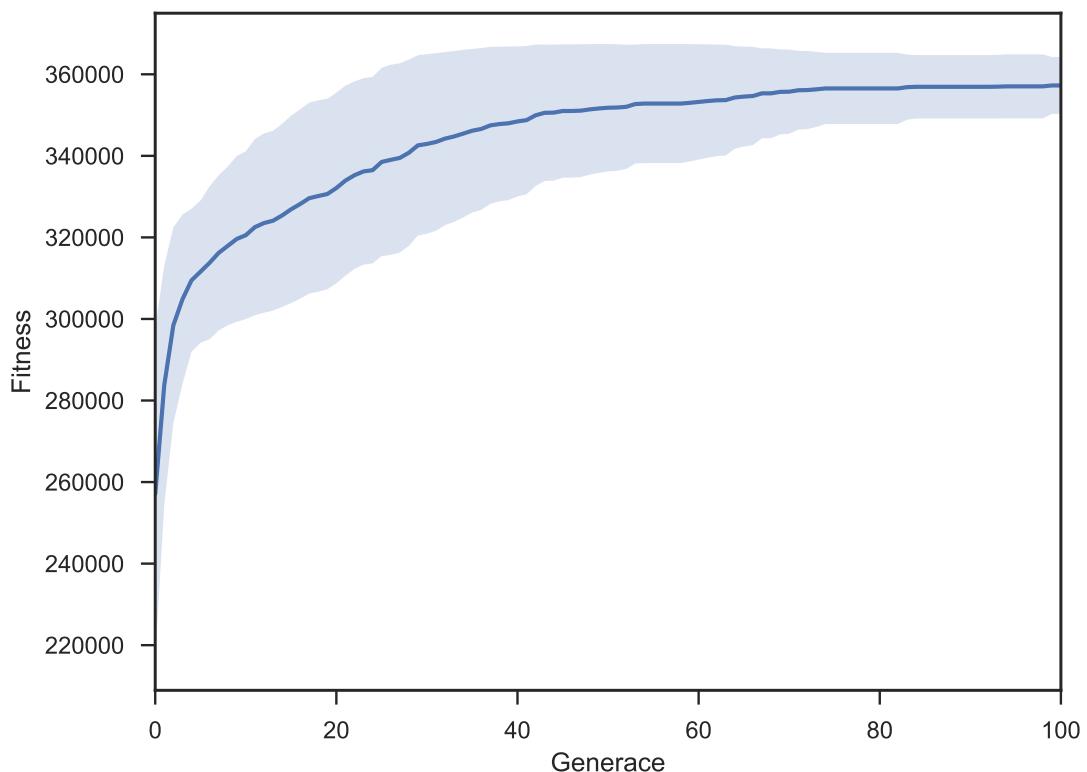
Z grafu 4.26 můžeme vyčíst, že nejlepší jedinci jsou schopni udělit až 3600 bodů poškození, což lze považovat za velmi dobrý výsledek. Neboť to znamená, že během 1500 iterací bylo hejno schopno zničit až dva Fighter roboty a jednoho Scouta. Nejlepšímu z jedinců se budeme věnovat více v rámci výsledků experimentu.

Nastavení mapy a EA	
Roboti:	<i>Scout, Fighter</i>
Počty robotů:	$S - 5, F - 4$
Počet generací:	1000
Počet iterací map:	1500
Velikost generace(DE):	100

Fitness funkce	
Mrtvý nepřátelský robot:	1000
Udělený bod poškození:	100
Ostatní hodnoty:	0

Objekty na mapě	
Počet překážek:	500
Nepřátelští roboti:	<i>Scout, Fighter</i>
Jejich počet:	$S - 5, F - 4$

Tabulka 4.32: Competitive kooperace - nastavení experimentu



Obrázek 4.26: Competitive kooperace - průběh fitness DE

Výsledky Experimentu

Nejlepší chování z Competitive Scene mělo řadu dobrých vlastností, umělo se efektivně pohybovat po mapě, v rámci komunikace se rozptylilo po větším území.

Poslední podúkol optimalizoval maximální poškození robotů, což mělo za důsledek, že celé hejno je agresivní na každého robota, kterého zachytí na svých senzorech, tzn. i na své vlastní kolegy. Opět se jedná o velmi komplexní scénář a chování v jednotlivých podúkolech je více než uspokojivé.

Tento problém by měl být vyřešen úpravou fitness, kdy i v rámci posledního experimentu bude fitness funkce zahrnovat i pozitivní ohodnocení zbylých bodů zdraví. Případně použít složitější architektury neuronové sítě. Ovšem z časových důvodů nebyly tyto alternativy vyzkoušeny.

Inicializační nastavení:

Výška:	800
Šířka:	1200
Počet iterací:	1500
Počet překážek:	400
Počet Scout robotů	5
Počet Fighter robotů	4

Tabulka 4.33: Competitive Scene - nastavení mapy pro testovací experiment

Výsledky

Počet nalezených minerálů	309,35
Součet zdraví	-266
Součet zdraví protivníka	-287

Tabulka 4.34: Competitive Scene - výsledky simulace nejlepšího jedince, průměr ze 100 simulací testovacího experimentu

4.5 Shrnutí

DE společně s metodou podíkolů dokázala efektivně optimalizovat chování heterogenního hejna při řešení typických úloh pro robotický swarm.

Jedná se především o pohyb v mapě, kde se roboti vyhýbají ostatním objektů i dalším členům hejna. Dále byli roboti schopni se rozptýlit po mapě. Hledání a sbírání objektů pro ně také nepředstavovalo problém. Pokud bylo hejno vedeno ke konkrétním úkonům jako tomu bylo v případě Wood Scene, byla evolvovaná chování schopna vykonávat i mnohem komplexnější úkoly zahrnující vzájemnou komunikaci, jako je například kácení stromů a jejich následné uskladnění.

Konkrétně ve Wood Scene scénáři, kde figurují dva druhy robotů z nichž jeden může provádět pouze transport a druhý pouze zpracování objektů, dokázali nejlepší jedinci najít požadované objekty na mapě, provést jejich zpracování, následně v rámci komunikace předat informaci o jejich nalezení, po přijetí informace o objektech připravených k transportu je naložit a odvést na místo určení.

DE se pro tyto účely ukázali jako vhodnější optimalizační nástroj, v rámci prvního experimentu proběhlo několik testovacích běhů potvrzující tuto domněnku.

V dalších dvou experimentech byli představeny limity zmíněného řešení. V těchto experimentech hejno mělo řešit obtížnější úlohy a v rámci podíkolů se optimalizovali obecnější cíle. Pro jejich jednoduché části jako například pohyb opět fungovala DE velmi dobře, ovšem poté se objevili problém u obecně zadaných podíkolů.

V Mineral Scene se nezdařilo zabránit evoluci, aby optimalizovala pouze jednoho ze členů, který dokáže plnit veškeré úkony hlavního cíle scénáře. U Competitive Scene scénáře se bohužel nepodařilo v posledním podíkolu eliminovat nepřátelské chování k členům vlastního hejna. Pro evoluční algoritmus bylo jednodušší optimalizovat obecně nepřátelské chování i přesto, že ve fitness funkci bylo hejno odměňováno pouze za útok do nepřátel.

V případech obou méně úspěšných scénářů by pravděpodobně bylo možné zabránit nežádoucímu chování, kdybychom jej jako negativní složku zahrnuli do fitness funkce a použili složitější architekturu neuronové sítě. Kvůli časové náročnosti simulací mapy nebyly tyto pokusy provedeny.

V konečném důsledku se DE v kombinaci s konkrétními metaúkoly osvědčila jako vhodný optimalizační metoda chování heterogenního hejna. Nejlépe však funguje pokud ji programátor vede přímým směrem.

Závěr

V rámci této práce byl implementován kompletní 2D simulátor umožňující simuloval chování robotických hejn, simulátor umožňuje hejnu, abys se skládalo z rozličných jedinců. Tento simulátor také obsahuje metody EA, konkrétně DE a ES jako prostředky pro optimalizaci řízení robotických hejn. Pro vizualizaci těchto chování vznikl program umožňující prohlížení už optimalizovaných chování. Simulace byla také optimalizována a bylo použito paralelního zpracování.

Za pomocí těchto programů byla otestována evolucí generované ovládání heterogenních hejn v rámci tří rozličných scénářů. Úkoly v těchto scénářích byly tvořeny z tradičních problémů robotických hejn jako je shlukování, pohyb v prostředí, rozptylování, apod. Dohromady tvořily scénáře komplexní problémy s ne-triviálním řešením.

Během prvního experimentu byly porovnány dva evoluční algoritmy pro optimalizaci neuronových sítí pro řízení heterogenních swarmů, konkrétně se jednalo o DE a ES. V rámci toho experimentu bylo provedeno několik testovacích běhů pro zmíněné srovnání. DE se ukázala jako lepší varianta pro optimalizaci heterogenního robotického hejna.

V dalších dvou experimentech se bylo potvrzeno, že tuto metodu lze použít jako vhodný nástroj pro optimalizaci jednoduchých chování hejna. Ovšem objevily se i limity spojené s obtížnými úkony, kde DE nepracovala uspokojivě. Bylo navrženo řešení těchto nedostatků, ovšem z časových důvodů nebylo vyzkoušeno.

Možná rozšíření

Tato práce poskytuje řadu zajímavých možností k rozšíření ať už se jedná o řešení nevyzkoušených metod na neúspěšných scénářích či porovnání s jinými metodami.

V rámci nedostatků scénářů Mineral Scene a Competitive Scene by bylo žádoucí vyzkoušet evoluční algoritmy, které umí generovat složitější neuronové sítě. Jedná se především o algoritmy odvozené z NEAT rodiny. Otestovat je na těchto obtížných scénářích.

Pro porovnání úspěšnosti evolučního algoritmu s jinýmu učícími algoritmy by bylo vhodné otestovat DE a ES s tradičním *backpropagation* algoritmem pro neuronové sítě.

V neposlední řadě by zajímavou cestu tvořilo přenesení vyvinutých chování na fyzické roboty. Což by ale vyžadovalo velké úpravy na simulátoru, protože simulátor značně zjednodušuje akce v prostředí. Navíc by bylo třeba přidat šumy prostředí neboť senzory jsou jimi v reálném světě značně zkresleny.

Seznam použité literatury

- ARVIN, F., MURRAY, J., ZHANG, C. a YUE, S. (n.d.). Colias: An autonomous micro robot for swarm robotic applications. *International Journal of Advanced Robotic Systems*, **11**. ISSN 17298806.
- AZNAR, F., SEMPERE, M., PUJOL, M., RIZO, R. a PUJOL, M. J. (2014). Modelling oil-spill detection with swarm drones. *Abstract & Applied Analysis*, pages 1 – 14. ISSN 10853375.
- BALCH, T. (2000). Hierarchic social entropy: An information theoretic measure of robot group diversity. *Autonomous robots*, **8**(3), 209–238.
- BASHYAL, S. a VENAYAGAMOORTHY, G. K. (2008). Human swarm interaction for radiation source search and localization. In *2008 IEEE Swarm Intelligence Symposium*, pages 1–8. doi: 10.1109/SIS.2008.4668287.
- BLIZZARD-ENTERTAINMENT (1998). Starcraft. URL <https://starcraft.com/en-us/>.
- DAVID, P., MIKE, D., REGINA, E., MIKE, H. a CRAIG, L. (2001). Pheromone robotics. *Autonomous Robots*, **3**(3), 319. ISSN 0929-5593.
- DORIGO, M. (2001). Swarm-bots. URL <http://www.swarm-bots.org>.
- DUARTE, M., COSTA, V., GOMES, J., RODRIGUES, T., SILVA, F., OLIVEIRA, S. M. a CHRISTENSEN, A. L. (2016). Evolution of collective behaviors for a real swarm of aquatic surface robots. *PLoS ONE*, **11**(3), 1 – 25. ISSN 19326203.
- EIBEN, A. (2015). *Introduction to evolutionary computing*. Springer, Berlin. ISBN 978-3662448731.
- FORTIN, F.-A., RAINVILLE, F.-M. D., GARDNER, M.-A., PARIZEAU, M. a GAGNÉ, C. (2012). Deap: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, **13**(Jul), 2171–2175.
- GOMES, J. C., URBANO, P. a CHRISTENSEN, A. L. (2013). Evolution of swarm robotics systems with novelty search. *CoRR*, **abs/1304.3362**. URL <http://arxiv.org/abs/1304.3362>.
- HOLLAND, J. H. (1976). *Adaptation in Natural and Artificial Systems*, volume 18. MIT Press; New Ed edition (1 July 1992).
- IVAN, T., TÜZE, K. a KATSUNORI, S. (2013). Hormone-inspired behaviour switching for the control of collective robotic organisms. *Robotics, Vol 2, Iss 3, Pp 165-184 (2013)*, **3**(3), 165. ISSN 2218-6581.
- JEVTIĆ, ALEKSANDAR, A. D. L. F. D. (2007). Swarm intelligence and its applications in swarm robotics.

- JONES, S., STUDLEY, M., HAUERT, S. a WINFIELD, A. (2018). *Evolving Behaviour Trees for Swarm Robotics*, pages 487–501. Springer International Publishing, Cham. ISBN 978-3-319-73008-0. doi: 10.1007/978-3-319-73008-0_34. URL https://doi.org/10.1007/978-3-319-73008-0_34.
- MARSALLI, M. (2006). McCulloch-pitts neurons. URL <http://www.mind.ilstu.edu/curriculum/modOverview.php?modGUI=212>.
- MITCHELL, M. (1998). *An introduction to genetic algorithms*. Complex adaptive systems. Cambridge : MIT Press, 1998. ISBN 0-262-13316-4.
- MONDADA, F., BONANI, M., RAEMY, X., PUGH, J., CIANCI, C., Klaptocz, A., MAGNENAT, S., ZUFFEREY, J.-C., FLOREANO, D. a MARTINOLI, A. (2009). The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, volume 1, pages 59–65. IPCB: Instituto Politécnico de Castelo Branco.
- PENDERS, J., ALBOUL, L., WITKOWSKI, U., NAGHSH, A., SAEZ-PONS, J., HERBRECHTSMEIER, S. a EL-HABBAL, M. (2011). A robot swarm assisting a human fire-fighter. *Advanced Robotics*, **25**(1/2), 93 – 117. ISSN 01691864.
- RECHENBERG, I. (1981). "evolutionsstrategie-optimierung technischer systems nach prinzipien der biologischen evolution, stuttgart: Frommannholzboog, 1973. New York: John Wiley.
- RUBENSTEIN, M., AHLER, C., HOFF, N., CABRERA, A. a NAGPAL, R. (2014). Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, **62**(Reconfigurable Modular Robotics), 966 – 975. ISSN 0921-8890.
- SHOULSON, A., GARCIA, F., JONES, M., MEAD, R. a BADLER, N. (2011). Parameterizing behavior trees. *Motion in Games*, pages 144–155.
- SUPERATI, V., TRIANNI, V. a NOLFI, S. (2011). Self-organised path formation in a swarm of robots. *Swarm Intelligence*, **5**(2), 97–119.
- STORN, R. a PRICE, K. (1997). Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, **11**(4), 341–359.
- TAN, Y. a ZHENG, Z.-Y. (2013). Research advance in swarm robotics. *Defence Technology*, **9**, 18 – 39. ISSN 2214-9147.
- YALCIN, C. (2008). Evolving aggregation behavior for robot swarms: A cost analysis for distinct fitness functions. pages 1–4.

Seznam obrázků

1.1	Obecný evoluční algoritmus - schéma	5
2.1	Cobot2D - nákres robota, zdroj : (Yalcin, 2008)	17
3.1	Ilustrativní obrázek obecného robota	21
4.1	Příklad WoodScene mapy: start náhodného chování	26
4.2	Příklad WoodScene mapy: po 9000 iteracích náhodného chování	26
4.3	Wood Scout chůze - porovnání průměrné fitness ES a DE	31
4.4	Wood Worker chůze - porovnání průměrné fitness ES a DE	32
4.5	Wood Scout kácení - porovnání průměrné fitness ES a DE	34
4.6	Wood Worker sbírání - porovnání průměrné fitness ES a DE	35
4.7	Wood Worker ukládání doprostřed - porovnání průměrné fitness ES a DE	37
4.8	Wood hlavní úkol - porovnání průměrné fitness ES a DE	39
4.9	Wood nejlepší jedinec - 10000 iterací simulace	41
4.10	Wood náhodný jedinec - 10000 iterací simulace	41
4.11	Příklad Mineral Scene mapy: konfigurace po startu s náhodným chováním	43
4.12	Mineral Scout chůze - průběh fitness DE	48
4.13	Mineral Worker chůze - průběh fitness DE	49
4.14	Mineral Worker sbírání - průběh fitness	50
4.15	Mineral Worker skládání - průběh fitness DE	52
4.16	Mineral Refaktor chůze - průběh fitness u DE	53
4.17	Mineral Refaktor Worker kooperace - průběh fitness DE	55
4.18	Mineral Refaktor Worker kooperace - průběh fitness DE	57
4.19	Příklad Competitive Scene mapy: start náhodného chování	60
4.20	Competitive Scout chůze - průběh fitness DE	66
4.21	Competitive Fighter chůze - průběh fitness DE	66
4.22	Competitive Scout životy - průběh fitness DE	67
4.23	Competitive Scout agresivní - průběh fitness DE	68
4.24	Competitive Fighter životy - průběh fitness DE	69
4.25	Competitive Fighter agresivní - průběh fitness DE	70
4.26	Competitive kooperace - průběh fitness DE	72

Seznam tabulek

1.1	diferenciální evoluce - souhrnné vlastnosti	10
1.2	evoluční strategie - souhrnné vlastnosti	11
2.1	Porovnání systémů s více agenty	13
2.2	Parameterizing behavior trees, Motion in Games - podoba stromů	15
4.1	Nastavení parametrů u EA	24
4.2	Wood Scene - Scout robot specifikace	27
4.3	Wood Scene - Worker robot specifikace	28
4.4	Wood Scout chůze - nastavení experimentu	30
4.5	Wood Worker chůze - nastavení experimentu	32
4.6	Wood Scout kácení - nastavení experimentu	33
4.7	Wood Worker sbírání - nastavení experimentu	35
4.8	Wood Worker ukládání doprostřed - nastavení experimentu	36
4.9	Wood hlavní úkol - nastavení experimentu	38
4.10	Wood Scene - nastavení mapy pro testovací experiment	40
4.11	Wood Scene - výsledky simulace nejlepšího jedince, průměr ze 100 simulací testovacího experimentu	40
4.12	Mineral Scene - Scout robot specifikace	44
4.13	Mineral Scene - Worker robot specifikace	45
4.14	Mineral Scene - Refaktor robot specifikace	46
4.15	Mineral Scout chůze - nastavení experimentu	48
4.16	Mineral Worker chůze - nastavení experimentu	49
4.17	Mineral Worker sbírání - nastavení experimentu	50
4.18	Mineral Worker skládání - nastavení experimentu	51
4.19	Mineral Refaktor chůze - nastavení experimentu	53
4.20	Mineral Refaktor Worker kooperace - nastavení experimentu	54
4.21	Mineral Refaktor Worker kooperace - nastavení experimentu	56
4.22	Mineral Scene - nastavení mapy pro testovací experiment	58
4.23	Mineral Scene - výsledky simulace nejlepšího jedince, průměr ze 100 simulací testovacího experimentu	58
4.24	Competitive Scene - Fighter Scout robot specifikace	62
4.25	Competitive Scene - Fighter robot specifikace	63
4.26	Competitive Scout chůze - nastavení experimentu	65
4.27	Competitive Fighter chůze - nastavení experimentu	65
4.28	Competitive Scout životy - nastavení experimentu	67
4.29	Competitive Scout agresivní - nastavení experimentu	68
4.30	Competitive Fighter životy - nastavení experimentu	69
4.31	Competitive Fighter agresivní - nastavení experimentu	70
4.32	Competitive kooperace - nastavení experimentu	71
4.33	Competitive Scene - nastavení mapy pro testovací experiment	73
4.34	Competitive Scene - výsledky simulace nejlepšího jedince, průměr ze 100 simulací testovacího experimentu	73

Přílohy