



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ NÁVRH POHYBUJÍCÍCH SE OBJEKTŮ
EVOLUTIONARY DESIGN OF MOVING OBJECTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB FAJKUS

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL BIDLO, Ph.D.

BRNO 2018

Abstrakt

Cílem této práce je implementovat systém pro automatizovaný evoluční návrh kontrolérů virtuálních robotů. Pro nalezení vhodného programu, který bude řídit robota tak, že se bude pohybovat po trajektorii, která je definována posloupností bodů, je použita reprezentace založená na lineárním genetickém programování ve spojení s genetickým algoritmem. Pro využití chování robota, kterého křídí kandidátní řešení vygenerované genetickým algoritmem, je použit fyzikální simulátor MuJoCo, který uživateli dovoluje definovat tvar robota. Cílem evoluce je natrénovat kontrolér robota tak, aby následoval definovanou trasu. Trénování kontroléru robota je založeno na optimalizaci vzdálenosti mezi robotem a body definujícími trajektorii. Optimalizace se provádí evolucí kontrolérů po daný počet generací steady-state genetického algoritmu. Je zde prezentováno několik experimentů s využitím jejich výsledků.

Abstract

The aim of this work is to implement a system for automatic evolutionary design of virtual robot controllers. In particular, Linear Genetic Programming representation combined with a steady-state genetic algorithm will be used to find a suitable program that will lead a given virtual robot across a sequence of points denoting a predefined trajectory. The MuJoCo physics engine is applied to allow the user to specify the robot shape and to evaluate its behavior according to candidate programs generated by the genetic algorithm. The goal is to train the robot to follow the given path by optimizing the distance of the robot from the given points during the simulation. The optimization is performed by evolving the programs for a given number of generation of the genetic algorithm. Several sets of experiments will be presented and obtained results will be evaluated.

Klíčová slova

simulátor MuJoCo, lineární genetické programování, genetický algoritmus, virtuální robot, trajektorie

Keywords

MuJoCo physics engine, Linear Genetic Programming, Genetic Algorithm, virtual robot, trajectory

Citace

FAJKUS, Jakub. *Evoluční návrh pohybujících se objektů*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Bidlo, Ph.D.

Evoluční návrh pohybujících se objektů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Bidla, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Fajkus
2018-05-15

Poděkování

Chtěl bych poděkovat Ing. Michalovi Bidlovi, Ph.D. za jeho čas, trpělivost a drahocenné rady.

Obsah

1	Úvod	6
2	Evoluční algoritmy	7
2.1	Koncept evolučních algoritmů	7
2.1.1	Reprezentace	8
2.1.2	Fitness funkce	8
2.1.3	Populace	8
2.1.4	Selekce	8
2.1.5	Křížení	9
2.1.6	Mutace	9
2.1.7	Ukončovací podmínka	10
2.2	Genetické programování	10
2.3	Lineární genetické programování	11
3	Zkoumání pohyblivých objektů	13
3.1	Návrh pohyblivých struktur pomocí L-systémů	13
3.2	Evoluce bipedie s použitím LGP	14
3.3	Vlastní roboti v prostředí simulátoru MuJoCo	14
4	Evoluční návrh pohyblivých objektů	17
4.1	Relizace řízení modelu	17
4.1.1	Interpret	17
4.1.2	Podprogramy	18
4.1.3	Hodnoty vstupních registrů	18
4.1.4	Výstupní registry	19
4.1.5	Simulátor	19
4.2	Instance evolučního algoritmu	20
4.2.1	Reprezentace	20
4.2.2	Fitness funkce	21
4.2.3	Populace a selekce	21
4.2.4	Křížení	22
4.2.5	Mutace	22
4.2.6	Ukončovací podmínka	22
5	Experimenty	23
5.1	Přímka	23
5.1.1	Model trojnožky	23
5.1.2	Model mravence	23

5.2	Spirála	24
5.2.1	Model trojnožky	24
5.2.2	Model mravence	25
6	Výsledky experimentů	26
6.1	Přímka	26
6.1.1	Model trojnožky na přímce	26
6.1.2	Model mravence na přímce	29
6.2	Spirála	31
6.2.1	Model trojnožky na spirále	31
6.2.2	Model mravence na spirále	35
7	Závěr	39
Literatura		40
A	Obsah přiloženého CD	41
B	Instalace a spuštění aplikace	42
B.1	Program compute	42
B.2	Program render	43
B.3	Evoluční framework	43

Seznam obrázků

2.1	Jednobodové křížení	9
2.2	Uniformní křížení	9
2.3	Křížení v GP	10
3.1	Stavební prvky robotů v práci G. Hornbyho	13
3.2	Robot vyvinutý G. Hornbym	14
3.3	Výsledek MJCF souboru 3.1	15
3.4	Model robota zvaný trojnožka.	15
3.5	Model robota zvaný mravenec	16
4.1	Schéma interpretu	18
4.2	Způsob výpočtu hodnot vstupních registrů	19
4.3	Rodzdelení genotypu na části pro podprogramy	20
4.4	Vlastní uniformní křížení	22
5.1	Scéna pro experiment s trojnožkou na přímce	23
5.2	Scéna pro experiment s mravencem na přímce	24
5.3	Scéna pro experiment s trojnožkou na spirále	24
5.4	Scéna pro experiment s mravencem na spirále	25
6.1	Trajektorie nejlepšího řešení trojnožky na přímce	27
6.2	Trajektorie neoptimálního řešení trojnožky na přímce	27
6.3	Vývoj fitness hodnot lepší poloviny populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu trojnožky na přímce	27
6.4	Vývoj fitness hodnot populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu trojnožky na přímce	27
6.5	Průběh evoluce, která vedla k nalezení nejlepšího jedince pro experiment s trojnožkou na přímce	28
6.6	Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 10. generaci	28
6.7	Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 50. generaci	28
6.8	Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 100. generaci	28
6.9	Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 150. generaci	28
6.10	Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 200. generaci	28
6.11	Trajektorie nejlepšího řešení mravence na přímce	29

6.12 Trajektorie neoptimálního řešení mravence na přímce	29
6.13 Vývoj fitness hodnot lepší poloviny populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu mravence na přímce	29
6.14 Vývoj fitness hodnot populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu mravence na přímce	29
6.15 Průběh evoluce, která vedla k nalezení nejlepšího jedince pro experiment s mravencem na přímce	30
6.16 Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 10. generaci	30
6.17 Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 30. generaci	30
6.18 Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 60. generaci	30
6.19 Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 90. generaci	30
6.20 Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 120. generaci	30
6.21 Trajektorie nejlepšího řešení trojnožky na spirále	31
6.22 Trajektorie neoptimálního řešení trojnožky na spirále	31
6.23 Vývoj fitness hodnot lepší poloviny populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu trojnožky na spirále	32
6.24 Vývoj fitness hodnot populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu trojnožky na spirále	32
6.25 Průběh evoluce, která vedla k nalezení nejlepšího jedince pro experiment s trojnožkou na spirále	32
6.26 Histogram celkového počtu jedinců ve všech bězích trojnožky na spirále v 10. generaci	32
6.27 Histogram celkového počtu jedinců ve všech bězích trojnožky na spirále v 50. generaci	33
6.28 Histogram celkového počtu jedinců ve všech bězích trojnožky na spirále v 100. generaci	33
6.29 Histogram celkového počtu jedinců ve všech bězích trojnožky na spirále v 200. generaci	33
6.30 Histogram celkového počtu jedinců ve všech bězích trojnožky na spirále v 300. generaci	33
6.31 Nejznámá testovací trajektorie č.1 pro kontrolér trojnožky – nejlepší řešení .	34
6.32 Nejznámá testovací trajektorie č.1 pro kontrolér trojnožky – druhé nejlepší řešení	34
6.33 Trajektorie řešení s největší fitness mravence na spirále	35
6.34 Trajektorie nejlepšího řešení mravence na spirále	36
6.35 Vývoj fitness hodnot lepší poloviny populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu mravence na spirále	36
6.36 Vývoj fitness hodnot populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu mravence na spirále	36
6.37 Průběh evoluce, která vedla k nalezení nejlepšího jedince pro experiment s mravencem na spirále	37
6.38 Histogram celkového počtu jedinců ve všech bězích mravence na spirále v 10. generaci	37

6.39 Histogram celkového počtu jedinců ve všech bězích mravence na spirále v 50. generaci	37
6.40 Histogram celkového počtu jedinců ve všech bězích mravence na spirále v 100. generaci	37
6.41 Histogram celkového počtu jedinců ve všech bězích mravence na spirále v 200. generaci	38
6.42 Histogram celkového počtu jedinců ve všech bězích mravence na spirále v 300. generaci	38
6.43 Neznámá testovací trajektorie č.1 pro kontrolér mravence	38
6.44 Neznámá testovací trajektorie č.2 pro kontrolér mravence	38
6.45 Neznámá testovací trajektorie č.3 pro kontrolér mravence	38

Kapitola 1

Úvod

Robotika představuje v současné době velmi aktuální oblast aplikace výpočetní techniky, jejímž cílem je řízení nejrůznějších elektromechanických systémů. Uplatnění nacházejí roboti zejména v průmyslu, kde usnadňují a zefektivňují namáhavou a opakující se práci člověku, ale také je populární oblast autonomních robotů používaných např. pro operace v těžko přístupných nebo nebezpečných místech (např. ponorky, podpora záchrannářům apod.). Jedním ze zásadních problémů konstrukce robotů obecně je návrh jejich efektivního řízení ve vztahu ke konkrétnímu typu robota a podmínkám jeho nasazení [1].

Pro tyto potřeby je vhodné mít k dispozici prostředky, které nám umožní rychle a levně prototypovat různé koncepty řízení robotů. Jednou z možností, jak řadiče robotů navrhovat, je použití evolučních algoritmů. Využití evolučních algoritmů osvobozuje konstruktéry od řešení malých detailů při návrhu kontroléru robota a v mnoha případech může evoluce nalézt řešení, které by lidského konstruktéra nenapadlo.

Řízení robota je ovšem možné realizovat mnoha způsoby, např. umělými neuronovými sítěmi [8], konenčními automaty [4] nebo i programy v imperativním jazyce [11]. V závislosti na vybrané metodě řízení robota jsou k dispozici různé algoritmy a automatizované nástroje pro nalezení konfigurace zvoleného způsobu řízení, které bude mít za následek požadované chování robota. Touto konfigurací mohou být např. váhy neuronové sítě, definice konečného automatu nebo sekvence příkazů imperativního jazyka. Právě řízením robota programy napsanými v imperativním jazyce (byť jednoduchém) se zabývá tato práce, ve které jsou použity evoluční algoritmy pro automatizované hledání vhodného řízení robota.

Cílem této práce je návrh a implementace nástroje pro automatizované hledání vhodného kontroléru robotického modelu pro průchod prostorem po dané trajektorii. Pro návrh vhodného programu je použita technika zjednodušeného Lineárního Genetického Programování [2]. Pro simulaci pohybu robotického modelu byl použit simulátor MuJoCo [10]. Účinnost softwarového řízení je experimentálně doložena s použitím dvou robotických modelů na dvou trajektoriích.

Tato práce začíná úvodem do problematiky evolučních algoritmů a lineárního genetického programování v kapitole 2. Následuje kapitola 3, ve které jsou představeny některé koncepty pohyblivých struktur a poté je popsán vlastní koncept a také dva modely robotů, které jsou zde použity. Kapitola 4 popisuje vlastní přístup, který je použit pro řízení robotického modelu v simulátoru, a způsob, jakým je implementována evoluce programů. Kapitola 5 představuje 4 experimenty, které byly provedeny k ověření funkčnosti vlastních přístupů, prezentovaných v této práci. Výskedy těchto experimentů jsou prezentovány v kapitole 6, po které následuje samotný závěr této práce, ve kterém jsou komentovány dosažené výsledky a možnost pokračování v této práci.

Kapitola 2

Evoluční algoritmy

V této kapitole si nejprve v sekci 2.1 představíme základní koncept evolučních algoritmů. Poté následuje v sekci 2.2 stručný úvod ke Genetickému Programování (GP), na který naváže sekce 2.3 o Lineárním Genetickém Programování (LGP).

2.1 Koncept evolučních algoritmů

Informace v této sekci vycházejí z [3].

Evoluční algoritmy (EA) jsou inspirovány přírodními evolučními procesy a Darwinovou teorií evoluce. EA se využívají ke stochastickému prohledávání stavového prostoru. EA, na rozdíl od jiných metod, pracují s celou populací kandidátních řešení (jedinců), které se vyvíjí paralelně. Každý jedinec v populaci v sobě nese zakódovanou informaci o konkrétním řešení, kterou nazýváme genotyp. Genotyp se poté dekóduje na fenotyp, který už reprezentuje řešení daného problému.

Myšlenku EA můžeme popsat následovně. Máme populaci jedinců, kteří jsou všichni umístěni ve společném prostředí, ve kterém soutěží o zdroje. Každému jedinci je přiřazena kvantitativní míra vyjadřující jeho schopnosti. Během evoluce dochází k přirozenému výběru jedinců k reprodukci, který se projevuje tak, že horší jedinci mají menší pravděpodobnost reprodukce, než-li ti lepsi. Tímto se přirozeně zvyšuje kvalita populace. Reprodukce probíhá dvojím způsobem, a to křížením a mutací. Křížení pracuje se dvěma rodiči a má za následek vytvoření dvou potomků, kteří vznikají kombinací genotypu obou rodičů. Mutace pracuje nad jedním rodičem a produkuje jednoho potomka, který má narozdíl od svého ročice lehce pozměněný genotyp.

Činnost obecného EA můžeme vidět na algoritmu 1:

Algoritmus 1: Obecný evoluční algoritmus

```
inicializuj populaci náhodně vygenerovanými jedinci;  
vyhodnot všechny jedince;  
while (není splněna ukončující podmínka) do  
    vyber rodiče;  
    aplikuj křížení na dvojice rodičů;  
    mutuj potomky;  
    vyhodnot kvalitu potomků;  
    vyber jedince do další generace;  
end
```

2.1.1 Reprezentace

Při návrhu řešení problému je často nutné abstrahovat reálný svět tak, abychom vytvořili prostředí, ve kterém budou existovat kandidátní řešení a ve kterém budou tato řešení využívána. Kandidátní řešení problému nazýváme fenotypy. Zakódované fenotypy nazýváme genotypy. Termín reprezentace se používá ve dvou kontextech. V prvním kontextu specifikuje mapování z fenotypu na genotyp a je synonymem pro kódování. V druhém kontextu označuje spíše strukturu prostoru genotypů.

2.1.2 Fitness funkce

Úlohou fitness funkce je reprezentování požadavků, které by populace měla splňovat. Tvoří základ pro funkci selekčních operátorů. Můžeme mluvit o zobrazení z jedince v prostoru genotypů na reálné nebo celé číslo. Z technického hlediska se jedná o funkci, která měří míru kvality genotypu a přiřazuje mu tzv. fitness. Výpočet fitness funkce zahrnuje dekódování genotypu na fenotyp a následné využití fenotypu, jako kandidátního řešení pro daný problém.

2.1.3 Populace

Úlohou populace je obsahovat množinu kandidátních řešení — jedná se o multimnožinu jedinců. Populace je jednotkou populace, zatímco jedinci jsou statické objekty, které se nemění ani neadaptují. Je to právě populace, která podléhá změnám nebo adaptaci.

Populace je struk

Definice populace se může omezit pouze na její velikost, ale existují i specializované evoluční algoritmy, které pracují i s rozložením jedinců v prostoru.

2.1.4 Selekc

Operátor selekce slouží k výběru rodičů pro reprodukci na základě jejich kvality. Tento operátor je nejčastěji založen na náhodě a to tak, že kvalitnější řešení jsou vybírána s větší pravděpodobností. Změnou parametrů selekce nastavujeme tzv. selekční tlak, který ovlivňuje, do jaké míry jsou upřednostňováni lepsi jedinci.

Jako představitele selekce si můžeme uvést ruletovou selekci a selekci turnajem.

Při použití ruletové selekce je pravděpodobnost P_r , že jedinec j bude vybrán závisí na poměru fitness tohoto jedince f_j a fitness celé populace: $P_r(j) = \frac{f_j}{\sum_{i=1}^N f_i}$, kde N je velikost populace a f_i je fitness jedince i v populaci.

Selekce turnajem, narozdíl od ruletové selekce nevyžaduje znalost celé populace a přečítávání fitness hodnot. Selekcí turnajem ani nevyžaduje, aby využití fitness funkcí bylo kvantifikované, ale pro jeho funkci stačí, aby mezi jedinci definována operace uspořádání. Díky tomu je tato metoda nenáročná na implementaci a na výpočetní čas. Základní algoritmus selekce turnajem můžeme vidět na algoritmu 2.

Algoritmus 2: Selekcí turnajem

```
// Chceme vytvořit mating pool velikosti λ jedinců;
while (dokud mating pool neobsahuje λ jedinců) do
    Náhodně vyber  $k$  jedinců z populace;
    Porovnej mezi sebou těchto  $k$  jedinců a vyber z nich nejlepšího jedince  $i$ ;
    Vlož jedince  $i$  do mating pool
end
```

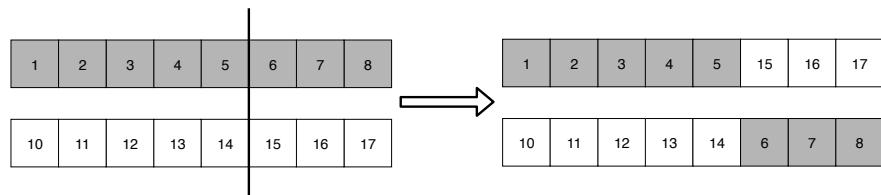
Úpravou parametru k se nastavuje selekční tlak — čím větší je parametr, tím více jedinců je vybráno pro porovnání a to má za následek vyšší selekční tlak.

2.1.5 Křížení

Operátor křížení slouží ke spojení částí genotypů rodičů a tím dochází k vytvoření nových jedinců. Operátor je většinou aplikován pouze s určitou pravděpodobností p_r . Nejčastěji se používá křížení pracující se dvěma rodiči. Výběr, jaké části ze kterého rodiče budou vybrány, je založen na náhodě. Myšlenka za použitím křížení je následující. Zkombinováním rodičů, kteří mají odlišné, ale vhodné vlastnosti, může vzniknout jedinec, který tyto vlastnosti v sobě kombinuje a dosahuje tak vyšší kvality.

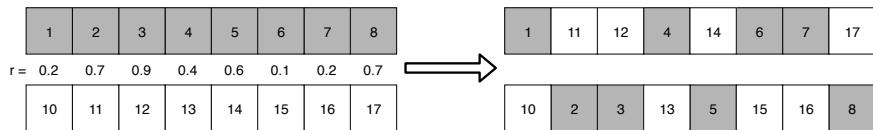
Jako představitele si můžeme uvést jednobodové a uniformní křížení.

Jednobodové křížení bylo původně představeno J. H. Hollandem v [5] a jeho funkce je následující. Vygeneruje se náhodné číslo r z intervalu $\langle 1, L - 1 \rangle$, kde L je délka genomu. Poté se genomy rozdělí v tomto bodě r na počáteční a koncovou část a noví jedinci vznikají záměnou těchto koncových částí, viz obrázek 2.1.



Obrázek 2.1: Jednobodové křížení

Uniformní křížení bylo představeno v [9] a pracuje následovně. Pro každý gen se vygeneruje náhodné číslo r z rovnoměrného rozložení na intervalu $\langle 0, 1 \rangle$. Číslo r se poté porovná s parametrem p , který bývá obvykle roven 0.5, a pokud je $r < p$, tak se použije gen z prvního rodiče, jinak ze druhého. Druhý potomek je vytvořen z genů, které nebyly vybrány do prvního potomka, viz obrázek 2.2.



Obrázek 2.2: Uniformní křížení

2.1.6 Mutace

Operátor mutace se aplikuje na jendoho rodiče a jeho výsledkem je jeden potomek. Cílem mutace je provést malou změnu genotypu. Operátor je většinou aplikován pouze s určitou pravděpodobností p_m a jeho implementace je závislá na použité reprezentaci.

Např. pro reprezentace využívající celá čísla, existují 2 záladní přístupy, které mutují každý gen s pravděpodobností p_m . Prvním přístupem je úplné nahrazení aktuální hodnoty genu novou hodnotou. Tato varianta je vhodná, pokud mezi hodnotami neexistuje relace uspořádání. Druhým způsobem je malá změna aktuální hodnoty, při které se k hodnotě genu přičte nebo odečte náhodně vygenerované malé číslo. Tato varianta je vhodná, jsou-li hodnoty genomu např. parametry systému nebo funkce.

2.1.7 Ukončovací podmínka

Ukončovací podmínka určuje, kdy se zastaví evoluce aktuální populace. K tomuto je možné využít: dosažení předem dané fitness hodnoty, spotřebování určeného strojového času, provedení určeného počtu vyhodnocení fitness funkce, ukončení v případě, že se populace do statečně rychle nezlepšuje nebo pokud dojde ke snížení diverzity populace po daný limit.

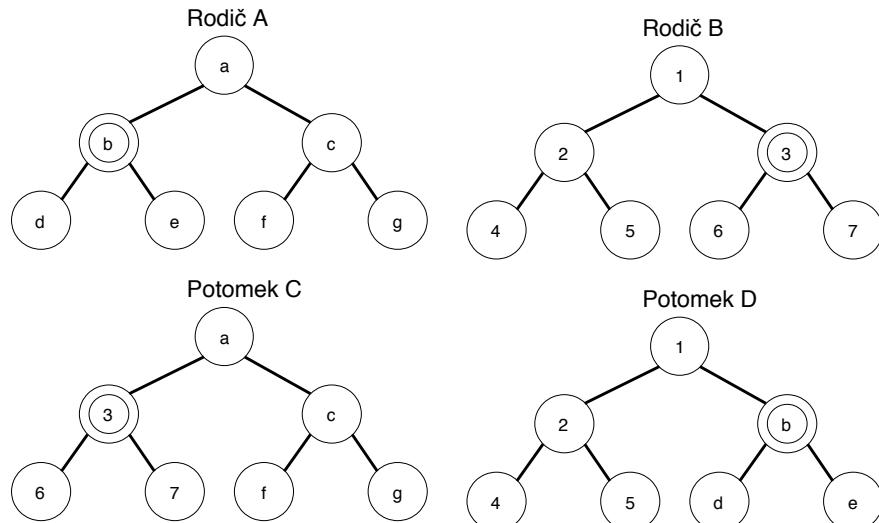
2.2 Genetické programování

Informace v této sekci vycházejí z [7].

Genetické programování (GP) se zabývá evolucí programu, který je reprezentován stromovou strukturou. Programy v GP jsou složeny z funkcí, které ve stromové struktuře odpovídají uzlům, a terminálů, které odpovídají listům.

Vyvíjené programy jsou typicky vyhodnocovány nad sadou vstupů, tzv. fitness cases, u kterých jsou známy požadované výstupy programu. Výsledná fitness programu se pak může počítat jako suma nebo průměr výsledků z každého "fitness case".

V GP je nejdůležitějším operátorem křížení. Křížení pracuje na základě výměny náhodně vybraných podstromů z rodičů. Toto křížení se skládá z několika kroků. Prvním krokem je výběr rodičů. Druhý krok je zvolení náhodného uzlu v každém z rodičů. Tento uzel bude kořenový uzel pro podstromy, které se budou později mezi rodiči vyměňovat. Třetím krokem je vyjmutí podstromu z obou rodičů, jejichž kořenovým uzlem je uzel, který byl vybrán v předchozím kroku. Následuje vytvoření obou potomků. První potomek vznikne tak, že se do stromu prvního rodiče, do místa dříve vybraného uzlu, vloží podstrom z druhého rodiče. Druhý jedinec vzniká obdobným způsobem, viz obrázek 2.3.



Obrázek 2.3: Křížení v GP. Na tomto obrázku je zachycena situace, kdy je v každém z rodičů A a B náhodně vybrán uzel (vyznačen dvojitě). Tyto uzly a jejich podstromy jsou poté zaměněny a vznikají tak potomci C a D.

Velikost programu, definována jako výška stromu, je omezena aby se předešlo velmi velkým programům. Pokud by potomek po křížení přesáhl tuto velikost, nebude vložen do nové generace - místo něj se do nové generace zkopiuje jeden z jeho rodičů.

Mutace v GA hraje menší roli a provádí drobné změny struktur v populaci. Mutace sestává z několika kroků: První krok je zvolení náhodného uzlu ve struktuře. Tento uzel může být vnitřní (funkce) i vnější (terminál). Druhým krokem je odstranění tohoto uzlu i s celým podstromem, který je k němu připojený. Posledním krokem je vygenerování náhodného podstromu, který se poté připojí na místo odstraněného uzlu. Tato operace je řízena parametrem který ovlivňuje výšku nově vygenerovaného podstromu. Speciálním případem mutace je operace, která vloží jeden terminál do náhodně vybraného uzlu ve stromu.

2.3 Lineární genetické programování

V této práci se používá přístup inspirovaný Lineárním genetickým programováním (LGP), které si stručně popíšeme v této sekci, krerá vychází z [2].

LGP je varianta GP, které bylo stručně popsáno v sekci 2.2, ve které jsou programy reprezentovány jako posloupnost instrukcí strojového kódu nebo příkazů vhodného imperativního jazyka. Obecně jsou data, zpracovávaná pomocí programu LGP, uchovávána v registrech, které jsou součástí interpretu LGP.

Interpret má k dispozici sadu registrů, jejichž počet je definován uživatelem. Tyto registry se dělí na vstupní, výstupní a pracovní registry. Vstupní registry obsahují vstupní data, se kterými je program spouštěn. Pracovní registry jsou určeny pro uložení mezinásledků výpočtů a jsou před spuštěním programu inicializovány vhodnou konstantou, např. 1. Jeden nebo více vstupních nebo pracovních registrů může být označen jako výstupní registr, ze kterého se poté po skončení programu čte výsledek.

Programy v LGP mohou být také vykonávány bez interpretu, a to tak, že se převedou do programu v jazyce C. Ukázkový program v jazyce C můžeme vidět na obrázku 2.1.

Instrukční sada definuje programovací jazyk, ve kterém jsou vyvýjené programy napsány. LGP systém je založen na dvou základních typech instrukcí: operace a podmíněné větvení. Příklady operací mohou být: různé aritmetické operace, exponenciální funkce, trigonometrické funkce, nebo booleovské operace. Imperativní instrukce obsahuje operaci, která pracuje nad zdrojovými registry, a přiřazení výsledku do cílového registru.

V LGP je nutné zajistit, aby byly vytvářeny pouze programy, které jsou validní. Gentické operátory mutace a křížení proto musí zachovat syntaktickou správnost nově vytvořených programů. To lze zajistit například tak, že bod křížení nemůže být určen urvnitř instrukce, nebo při aplikování operátoru mutace zakážeme záměnu operátoru instrukce za registr. Pro zajištění sémantické správnosti je nutné ošetřit všechny operátory a funkce, které nejsou definovány pro všechny možné hodnoty tak, aby pro nedefinovaný výstup vraceły konstantu, např. velké celé číslo.

Schopnost genetického programování nalézt řešení velmi závisí na zvolené instrukční sadě. Kompletní instrukční sada je taková, která obsahuje všechny prvky, které jsou potřebné k vytvoření optimálního řešení za předpokladu, že je počet registrů a rozsah použitých konstant dostatečný. Na druhou stranu, rozměr stavového prostoru, který obsahuje všechny programy, které je možné sestavit z dostupných instrukcí, roste exponenciálně s počtem instrukcí a registrů.

Na programu 2.1 v jazyce C si ukážeme průběh vykonávání programu pro výpočet průměru 2 čísel. Tento program má k dispozici výstupní registr $r[0]$, ve kterém bude uložen výsledek, dva vstupní registry $r[1]$ a $r[2]$, které obsahují vstupy programu a jeden pracovní registr $r[3]$. Před spuštěním programu jsou do vstupních registrů nakopírovány hodnoty a pracovní registr je inicializován na hodnotu 1. První instrukcí programu je sečtení dvou

vstupních registrů a uložení výsledku do pracovního registru. Tento pracovní registr je poté využit jako operand pro instrukci dělení, která uloží výsledek do výstupního registru r[0].

```
double r[4];  
  
void lgp(r)  
{  
    r[3] = r[1] + r[2];  
    r[0] = r[3] / 2;  
}
```

Program 2.1: LGP program v C pro sečtení dvou čísel

Kapitola 3

Zkoumání pohyblivých objektů

Tato práce se zabývá evolučním návrhem řízení vybraných typů pohyblivých struktur (robotů) pomocí evolučních algoritmů. V této kapitole shrneme myšlenky vycházející z relevantní literatury a představíme vlastní koncept evolučního návrhu řízení robotů.

V sekci 3.1 si nejprve stručně shrneme použití L-systémů pro development robotů, poté krátce představíme použití LGP pro vývoj kontrolérů modelu v sekci 3.2 a nakonec je zde představen vlastní konceptu modelů v sekci 3.3.

3.1 Návrh pohyblivých struktur pomocí L-systémů

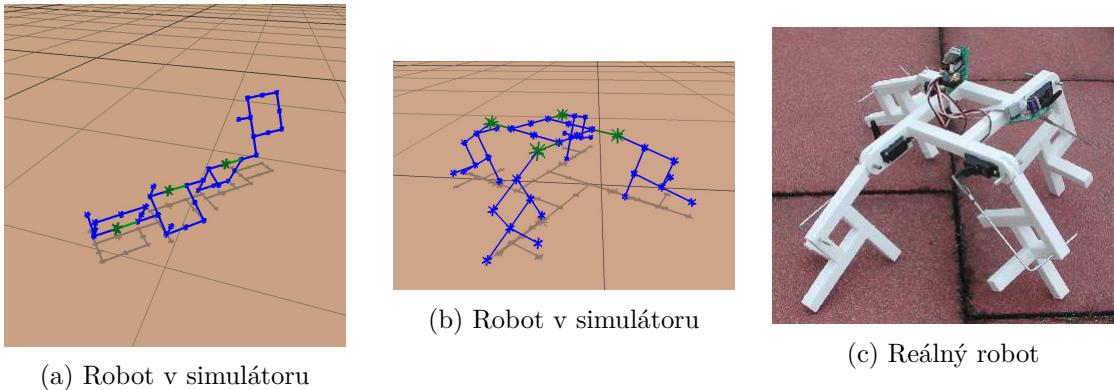
G. Hornby ve své disertaci [6] pomocí generativních reprezentací (L-systémů) mimo jiné vyvíjel fyzickou strukturu i řízení robotických modelů. Pro vývoj řízení byly použity 2 přístupy. První přístup byl založen na oscilujících poháněných kloubech, které měly 2 parametry – rychlosť oscilace a fázový posun. Druhý přístup byl založen na neuronových sítích, které se budovaly použitím instrukcí a které svým výstupem ovládaly jednotlivé poháněné klouby.

V této práci byli roboti složeni ze 3 prvků – tyče fixní délky, bloky pro spojení těchto tyčí a poháněné klouby, viz obrázek 3.1.



Obrázek 3.1: Stavební prvky robotů v práci G. Hornbyho: tyče fixní délky, bloky pro spojení těchto tyčí a poháněné klouby. Zdroj: [6]

V programu, který konstruoval robota, se vyskytovaly instrukce pro budování fyzické struktury a také instukce pro konfiguraci kloubů nebo neuronové sítě. Tímto se struktura robota vytváří současně s vytvářením jeho řízení.



Obrázek 3.2: Robot vyvinutý G. Hornbym. Zdroj: [6]

G. Hornby touto metodou vyvinul množství robotů, z nichž některé i skutečně vyrobil, viz obrázek 3.2.

3.2 Evoluce bipedie s použitím LGP

Wolff a Wahde ve své práci [11] využili koncept LGP pro vývoj kontroléru pro humanoidního robota, který měl za úkol chodit v simulovaném prostředí. Použili zde 7 typů instrukcí: sčítání, odčítání, násobení, dělení, funkce sinus, a dvě varianty větvení. Programy měly k dispozici řadu hodnot ze senzorů, jako např. natočení jednotlivých kloubů modelu, aktuální zrychlení jednotlivých částí robota nebo relativní natočení částí těla k některé z os.

Jejich řízení bylo navrženo tak, že se v definovaných časových okamžicích simulace spustil celý program, který na základě vstupních hodnot vypočetl výstupní hodnoty, které se použily pro řízení kloubů modelu. Pro evoluci programů zde použili steady state algoritmus, selekci turnajem velikosti 4, klasické 2 bodové křížení a mutaci, která náhodně vybranou instrukci nahradí novou, náhodně vygenerovanou.

V této práci se podařilo vyvinout kontroléry, které byly schopny řídit robota tak, aby se pohyboval. Tento pohyb byl ovšem velmi pomalý (0.054 m/s), ačkoli trvalý (až 20 minut).

3.3 Vlastní roboti v prostředí simulátoru MuJoCo

Podobně jako v práci G. Hornbyho [6] jsou roboti v této práci složeni z pevných tyčí, které mají různou délku, bloků pro spojení těchto tyčí a poháněných kloubů. Pro účely této práce byly vytvořeny dva počítačové modely robotů, pro které bylo cílem evolučně navrhnut řídicí program tak, aby robot vykazoval požadované chování. Pro tyto potřeby byl použit zjednodušený koncept LGP, který bude blíže představen v sekci 4.1, a simulátor MuJoCo¹.

MuJoCo je fyzikální simulátor určený k vývoji v oblasti robotiky, biomechaniky, grafiky a simulace, strojového učení a všech dalších oblastí, ve kterých je potřeba provádět rychlé a přesné simulace komplexních dynamických systémů. Tento simulátor byl od začátku postaven tak, aby byl co nejvýkonnější a bylo jej tedy možné využít v situacích, kdy je potřeba provádět simulace mnohem rychleji, než-li v reálném čase. Simulátor je napsán

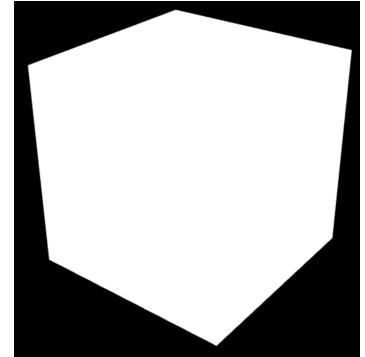
¹<http://www.mujoco.org/index.html>

jako dynamická knihovna s rozhraním v jazyce C a má zabudovanou interaktivní vizualizaci, založenou na OpenGL.

Modely pro tento simulátor se specifikují v XML souboru ve formátu MJCF, který je lidsky čitelný a je relativně jednoduché s ním pracovat. Tento soubor také obsahuje nastavení všech parametrů simulace. Všechny parametry mají své výchozí hodnoty, takže je není nutné pro jednoduché pokusy nastavovat. V programu 3.1 můžeme vidět ukázku minimalistického MJCF souboru definující objekt, který je vidět na obrázku 3.3. Tento MJCF soubor je při spuštění simulace přeložen na nízkourovňové struktury, se kterými se poté pracuje v samotné simulaci.

```
<mujoco>
  <worldbody>
    <geom type="box" pos="1 1 1" size="1 1 1"
          rgba="255 255 255 1"/>
  </worldbody>
</mujoco>
```

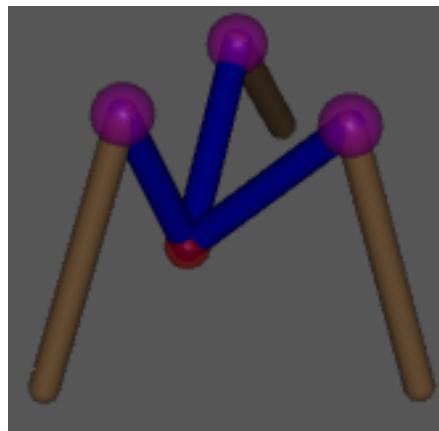
Program 3.1: Minimalistický MJCF soubor. Tento soubor definuje bílou krychli (viz obrázek 3.3) jednotkové velikosti, která je umístěna v prostoru na souřadnicích 1,1,1.



Obrázek 3.3: Výsledek MJCF souboru 3.1

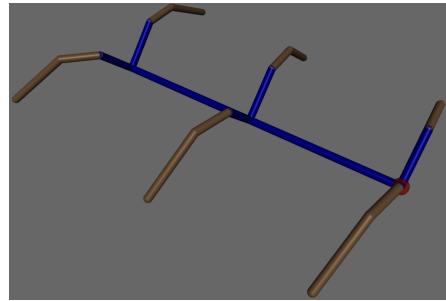
Níže si popíšeme dva modely robotů, se kterými buou v této práci představeny různé experimenty. Tyto experimenty zahrnovaly pohyb po přímce a spirále.

První model, zvaný trojnožka, je vidět na obrázku 3.4. Robot má 3 nohy, každá z nich je spojena s jádrem robota kloubem, který se otáčí v jedné ose. Rozsah pohybu těchto kloubů je omezen na 50 stupňů. Model robota má v horní části v místě kloubů umístěny kontaktní body. Tyto body slouží k detekci převrácení robota (kolizi ze zemí).



Obrázek 3.4: Model robota zvaný trojnožka. Tento model je složen z jádra robota (modré) a 3 nohou (hnědě). Každá z nohou je s jádrem spojena kloubem, který se otáčí pouze v jedné ose a je umístěn pod fialovou koulí. Tyto fialové koule slouží k detekci převrácení robota. Ve středu robota je bod zvaný hlava (červeně).

Druhý robot, zvaný mravenec, je vidět na obrázku 3.5. Robot má 3 páry nohou, které jsou všechny připojeny k tělu robota. Rozsah pohybu kloubů, které rotují kolem vertikální osy a spojují tělo robota s jeho nohou, je omezen na 100 stupňů. Rozsah kloubů, které rotují kolem horizontální osy a spojují dvě části nohy, je omezen na 65 stupňů.



Obrázek 3.5: Model robota zvaný mravenec. Tento model je složen z těla robota (modře) a 6-ti nohou (hnědě). V přední části robota je bod zvaný hlava (červeně). Každá noha je složena ze dvou pevných částí a jednoho kloubu. Tento kloub rotuje kolem horizontální osy, tj. může zvedat a snižovat robota. Celá noha je připojena k tělu robota kloubem, který rotuje kolem vertikální osy, tj. slouží k odrážení.

Kapitola 4

Evoluční návrh pohyblivých objektů

V této kapitole si nejdříve v sekci 4.1 popíšeme způsob, jakým je realizováno řízení robota a poté si v sekci 4.2 popíšeme instanci EA, která je v této práci použita.

4.1 Relizace řízení modelu

V simulátoru MuJoCo je vytvořena scéna, která obsahuje model a množinu referenčních bodů, které svým pořadím a umístěním ve scéně definují trajektorii, kterou má model následovat. Tato scéna je poté využita v simulaci, ve které probíhá vyhodnocování daného robotického modelu s vyvýjeným řízením.

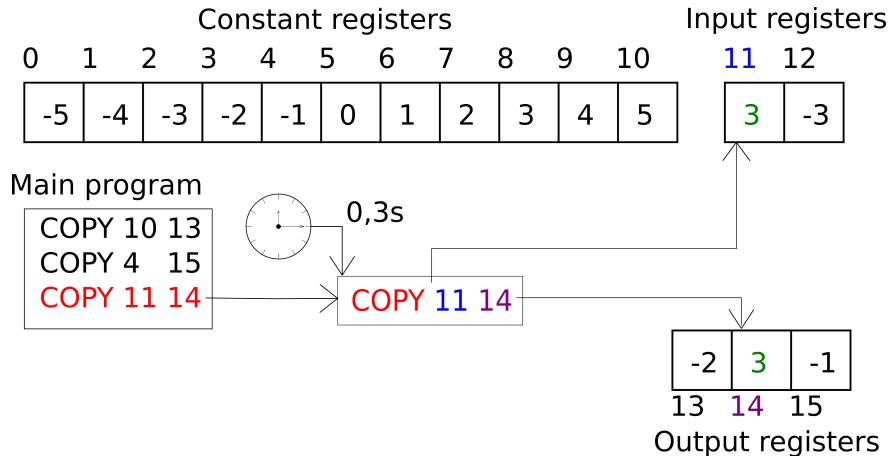
4.1.1 Interpret

Program, který řídí model robota, je vykonáván v interpretu. Interpret obsahuje množinu registrů, kde každý registr je identifikovaný unikátním číslem, které se navýzvá index. Tyto registry mohou obsahovat celočíselné hodnoty v rozsahu od -5 do 5.

Pracuje se zde se třemi typy registrů. Prvním typem jsou vstupní registry, které obsahují vstupní data programu, jsou chráněné proti přepisu a v interpretu jsou 2. Dalším typem jsou konstantní registry, kterých je v interpretu 11. Tyto registry obsahují všechna celá čísla (v rozsahu od -5 do 5), které mohou být do registrů uložena, a tyto registry jsou také chráněny proti přepisu. Posledním typem jsou výstupní registry, které řídí jednotlivé klouby modelu a jejich obsah je modifikován instrukcemi. Počet těchto registrů závisí na počtu poháněných klubů modelu, pro který je program určen – pro modely, které se v této práci používají, jsou počty výstupních registrů 3 a 12.

Interpret vykonává program, který je složen z instrukcí. Je zde použit jen jeden druh instrukce, a to instrukce s názvem COPY. Tato instrukce má 2 parametry: zdrojový registr a cílový registr. Výsledkem této instrukce je zkopirování hodnoty ze vstupního, nebo konstantního, registru do výstupního registru. Interpret je schématicky znázorněn na obrázku 4.1.

V průběhu simulace je vykonáván program, který čte hodnoty ze vstupních registrů, nebo konstantních registrů a zapisuje hodnoty do výstupních registrů, které se převádí na sílu, která je aplikována v jednotlivých kloubech.



Obrázek 4.1: Schéma interpretu. Je zde znázorněna situace, ve které interpret vykonává instrukci z podprogramu main. Tato instrukce, s parametry 11 a 14 způsobí zkopírování hodnoty 3 ze vstupního registru s indexem 11 do výstupního registru s indexem 14.

4.1.2 Podprogramy

Pro účely experimentů se spirálovou trajektorií byl navržen koncept podprogramů, který je popsán dále. V experimentech s přímkovou trajektorií je z těchto podprogramů použit jen podprogram main.

Každý program, který reprezentuje kandidátní řešení, je pro vykonávání v průběhu simulace rozdělen na 3 podprogramy. Tyto podprogramy se nazývají init, main a event.

Podprogram init se vykoná pouze na začátku simulace a všechny instrukce jsou provedeny ve stejném simulačním čase a poté je po 1 sekundě spuštěn podprogram main. Účelem podprogramu init je nastavení počátečního natočení kloubů modelu.

Podprogram main je v průběhu simulace vykonáván v nekonečné smyčce. Instrukce v tomto podprogramu se nevykonají všechny v nulovém čase, ale vykonávají se s časovým rozestupem 0.33 sekundy mezi každou z instrukcí. Jedná se o nejdélší a nejdůležitější podprogram.

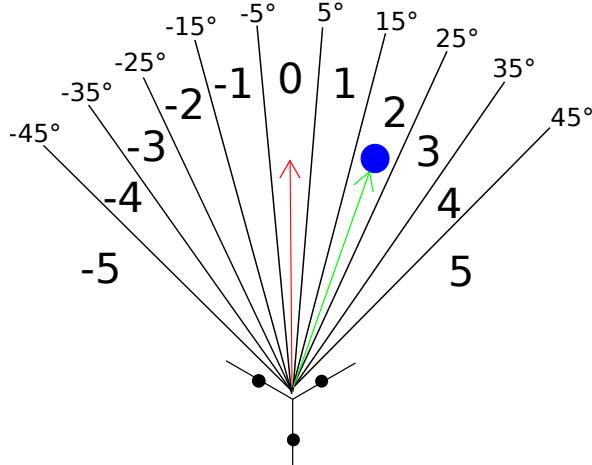
Podprogram event se vykoná v situaci, kdy se model přiblíží do určené vzdálenosti od referenčního bodu, avšak pro každý referenční bod pouze jednou. Stejně jako u podprogramu init jsou instrukce provedeny ve stejném simulačním čase a poté je po 1 sekundě spuštěn podprogram main. Účelem podprogramu event je změna natočení kloubů modelu jako příprava k pohybu k následujícímu referenčnímu bodu.

4.1.3 Hodnoty vstupních registrů

Vstupní registry jsou použity pouze pro experimenty se spirálovou trajektorií. Hodnoty, které se ukládají do vstupních registrů, vychází z informace o směru k následujícímu referenčnímu bodu. Informace o směru je opět vyjádřena číslem od -5 do 5 a vypočítává se následujícím mechanismem.

Prostor kolem modelu je rozdělen na 11 kruhových výsečí, kde každá výseč je ohodnocena číslem od -5 do 5, viz obrázek 4.2. Informace o směru je rovna ohodnocení výseče, ve které se nachází další referenční bod. Tato informace je v nezměněné podobě vložena do prvního vstupního registru. Do druhého registru je vložena hodnota s převráceným znamén-

kem. Tyto vstupní hodnoty mohou sloužit programům ke korekci směru, kterým se robot pohybuje, v závislosti na poloze dalšího bodu, ke kterému se má tento robot přiblížit.



Obrázek 4.2: Způsob výpočtu hodnot vstupních registrů. Model (na obrázku dole) směruje směrem nahoru a další referenční bod (modře) se nachází ve výšce ohodnocené číslem 2. Hodnoty vstupních registrů budou tedy čísla 2 a -2.

4.1.4 Výstupní registry

Každý z výstupních registrů interpretu odpovídá jednomu kloubu modelu. Hodnoty z výstupního registru jsou z interpretu čteny a převádí se na ovládací signály, které se předávají simulátoru. Hodnota (ovládací signál) v sobě obsahuje dvě informace. První z nich je dána znaménkem a určuje, kterým směrem bude aplikována síla v kloubu. Druhá informace je dána velikostí hodnoty a určuje velikost této síly.

4.1.5 Simulátor

Simulátor poskytuje řadu funkcí, které umožňují řídit běh simulace. Tyto funkce se poté používají v uživatelském programu, ve kterém nutné z těchto funkcí sestavit algoritmus spojité simulace (ukázkové uživatelské programy jsou součástí distribuce simulátoru). Uživatel má plnou kontrolu nad během simulace, a to tím, že volá knihovní funkci `mj_step()`, která v simulaci pokročí definovaným krokem. Ve smyčce algoritmu spojité simulace jsou vloženy funkce zajišťující běh interpretu, výpočet fitness funkce, mapování výstupních hodnot

z interpretu na ovládací signály a získávání pozic objektů z dat simulace. Ve zjednodušené formě je algoritmus simulace znázorněn na algoritmu 3.

Algoritmus 3: Algoritmus simulátoru

```

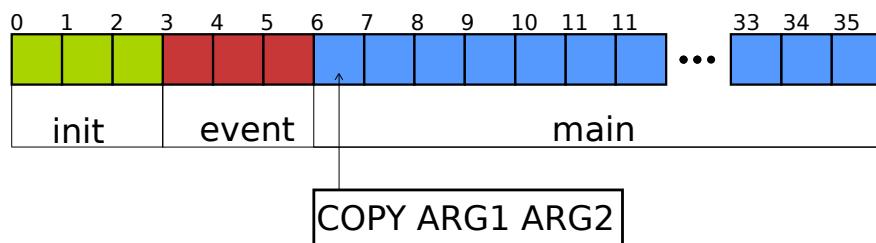
načti model ze souboru;
nastav čas simulace  $t = 0$ ;
vykonej celý program init;
while ( $t < \text{délka simulace}$ ) do
    proved další kroku simulace - mj_step();
    if (došlo k převrácení modelu) then
        | ukonči simulaci;
    end
    vypočti vstupní hodnoty interpretu;
    vykonej instrukci z programu main;
    použij výstupní hodnoty interpretu pro řízení modelu;
    zaznamenej vzdálenosti od jednotlivých referenčních bodů;
    if (došlo k přiblížení k následujícímu referenčnímu bodu) then
        | vykonej celý program event;
    end
    přičti k času  $t$  časový krok simulace;
end
vypočti fitness hodnotu;
```

4.2 Instance evolučního algoritmu

4.2.1 Reprezentace

Použitá reprezentace pracuje s genomem fixní délky, která se liší pro každý model a experiment. Tato délka bude uvedena u každého z experimentů v následující kapitole. Tato reprezentace kóduje každou instrukci jako n-tici, ve které je prvním prvkem název instrukce a poté následují její argumenty. Aktuálně jediná použitá instrukce je složena ze svého názvu (COPY, nebo i starší název SRE) a dvou celých čísel, které označují zdrojový a cílový registr.

Tato n-tice je použita jako hodnota jednoho genu. Pro experimenty se spirálovou trajektorii je celý genotyp pro účely křížení rozdělen na 3 části, které korespondují s podprogramy, viz obrázek 4.3. Pro experimenty s přímkovou trajektorií je použit jen podprogram main.



Obrázek 4.3: Rodzdělení genotypu na části pro podprogramy. Je zde znázorněn genotyp délky 36 genů, který je rozdělen po 3 genech pro podprogram init a event a 30 genů pro podprogram main. Je zde také znázorněna n-tice, která reprezentuje instrukci.

4.2.2 Fitness funkce

Fitness hodnota kandidátního programu se počítá následující způsobem. V průběhu simulace se pro každý referenční bod i zaznamenává nejmenší vzdálenost D_i mezi tímto bodem a modelem. $D_i = \min F(R, P_i)$, kde F je funkce vypočítávající vzdálenost, R je pozice modelu v prostoru, P_i je pozice referenčního bodu v prostoru a $i = 1, \dots, N$, kde N je počet referenčních bodů.

Po dokončení simulace je pro každý referenční bod je vypočteno skóre následujícím způsobem:

$$S_i = \begin{cases} t - D_i & \text{pokud } D_i \leq t \\ 0 & \text{jinak} \end{cases}$$

Parametr t je minimální vzdálenost, ve které musí být model od referenčního bodu, aby se skóre pro tento bod počítalo. Výsledná fitness hodnota je poté vypočtena jako suma skóre pro všechny body:

$$f = \sum_{i=1}^N S_i$$

Na základě rozměru modelu a celé scény v simulátoru byla zvolena hodnota parametru $t = 40$.

Motivací k použití tohoto systému byla myšlenka, že robotický model v průběhu simulace získává odměnu (skóre), které se poté GA snaží maximalizovat. Opačným přístupem by mohlo být použití metody nejmenších čtverců. Při tomto přístupu by GA minimalizoval minimální vzdálenost od referenčních bodů, do které se model v průběhu simulace dostal.

4.2.3 Populace a selekce

Pro evoluci programů byla vybrána varianta steady-state genetického algoritmu (algoritmus 4) s velikostí populace 1000 nebo 400 jedinců. Operátor selekce je implementován jako turnaj velikosti $k = 2$. Nejlepší jedinec v aktuální populaci a jeho mutant jsou zkopirováni do následující populace.

Algoritmus 4: Steady state genetický algoritmus
--

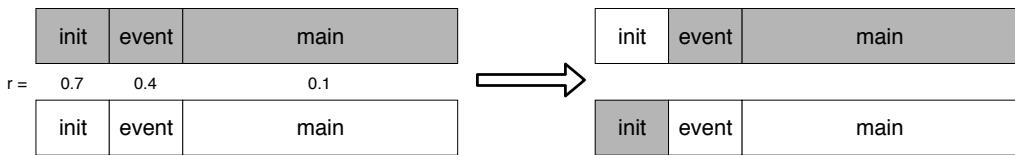
```
// v je velikost populace P;
nastav čas t = 0;
Inicializuj všechny chromozomy v P(t) náhodnými alelami;
do
    vypočítej fitness hodnoty v P(t);
    do
        vyber dva jedince z rodičovské populace P(t);
        vytvoř dva potomky použitím operátoru křížení;
        aplikuj operátor mutace na oba potomky;
        vlož tyto dva potomky do dočasné populace T;
    while (v populaci T je méně než v/2 jedinců);
    přepiš k nejhorších jedinců v P(t) jedinci z T vlož jedince z T do P(t);
    odstraň k nejhorších jedinců z P(t);
    t = t + 1;
while (není splněna ukončující podmínka);
```

4.2.4 Křížení

Pro účely této práce byl implementován vlastní operátor křížení, který je založen na uniformním křížení. Křížení je aplikováno s pravděpodobností $p_r = 0.8$. Narozdí od uniformního křížení, které pracuje na úrovni genů, křížení použité zde pracuje na úrovni podprogramů. V genomu jsou tedy 3 části, které se mohou mezi rodiči vyměňovat, viz obrázek 4.4.

Pro každý gen se vygeneruje náhodné číslo r z rovnoměrného rozložení na intervalu $\langle 0, 1 \rangle$. Číslo r se poté porovná s parametrem $p = 0.5$, a pokud je $r < p$, tak se použije podprogram z prvního rodiče, jinak ze druhého. Druhý potomek je vytvořen z podprogramů, které nebyly vybrány do prvního potomka

Tento operátor je použit pouze v experimentech se spriálovou trajektorií. V experimentech s přímkovou trajektorií je použito prosté jednobodové křížení s pravděpodobností $p_r = 0.8$, pracující na úrovni instrukcí.



Obrázek 4.4: Vlastní uniformní křížení. Zde, při křížení rodičů došlo k výměně podprogramu init.

Tento operátor se používá v experimentech se spirálovou trajektorií. Pro experimenty s přímkovou trajektorií se používá jednobodové křížení, které pracuje na úrovni jednotlivých instrukcí.

4.2.5 Mutace

Operátor mutace je nastaven s $p_m = 1$ a vždy mutuje jen jeden gen. Tento operátor je implementován následovně. Náhodně se vybere jedna instrukce z genotypu. S 20 % pravděpodobností je tato instrukce vymazána a je nahrazena novou, náhodně vygenerovanou. S 80 % pravděpodobností je tato instrukce upravena a to tak, že je zde 50 % pravděpodobnost na mutaci prvního, nebo druhého argumentu. Mutace argumentu je implementována přepsáním původní hodnoty nově vygenerovanou hodnotou.

4.2.6 Ukončovací podmínka

Jako ukončovací podmínka byla zvoleno ukončení evoluce po provedení daného počtu generací. Tento počet se liší pro jednotlivé experimenty, proto budou hodnoty uvedeny v popisu konfigurace GA v kapitole 5.

Kapitola 5

Experimenty

Pro účely experimentálního ověření funkčnosti použitých postupů byly navrženy 2 experimenty, které byly provedeny s každým modelem, tedy celkem 4 experimenty.

V sekci 5.1 je představen experiment s přímkovou trajektorií pro oba modely, který je následován experimentem se spirálovou trajektorií v sekci 5.2, rovněž pro oba modely.

5.1 Přímka

Tento experiment měl za cíl ověřit, zda-li je použitý způsob řízení modelu funkční. Úlohou modelu bylo pohybovat se po přímce, na které leželo 7 referenčních bodů.

Programy zde použité byly složeny jen z podprogramu main, který neměl k dispozici informace o směru k dalšímu referenčnímu bodu. Délka simulace byla nastavena na 80s.

5.1.1 Model trojnožky

V tomto experimentu byly použity následující parametry:

- 1000 jedinců v populaci,
- ukončení běhu evoluce po 200 generacích,
- 18 genů v genotypu.

Snímek scény simulátoru pro tento experiment je na obrázku 5.1.



Obrázek 5.1: Scéna pro experiment s trojnožkou na přímce

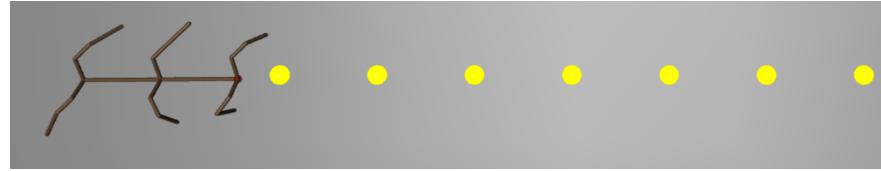
5.1.2 Model mravence

V tomto experimentu byly použity následující parametry:

- 400 jedinců v populaci
- ukončení běhu evoluce po 120 generacích

- 72 genů v genotypu

Snímek scény simulátoru pro tento experiment je na obrázku 5.2.



Obrázek 5.2: Scéna pro experiment s mravencem na přímce

5.2 Spirála

Tento experiment měl za cíl ověřit, zda-li je použitý způsob řízení modelu schopný pohybu i po netriviální trajektorii. Úlohou modelu bylo pohybovat se po spirále, na které leželo 9 referenčních bodů. Pro účely tohoto experimentu bylo zavedeno použití podprogramů a vstupních informací pro program.

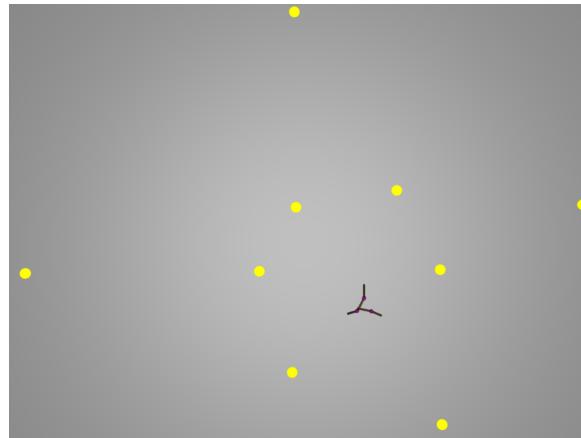
Délka simulace pro tyto experimenty byla nastavena na 600s a bylo zde použito podprogramové uniformní křížení s pravděpodobností $p_r = 0.8$, popsané v podsekci 4.2.4. V populaci bylo 1000 jedinců a každý evoluční běh byl ukončen po 300 generacích.

5.2.1 Model trojnožky

V tomto experimentu byly použity následující parametry:

- 36 genů v genotypu
- délka podprogramů init a event jsou 3 instrukce

Snímek scény simulátoru pro tento experiment je na obrázku 5.3.



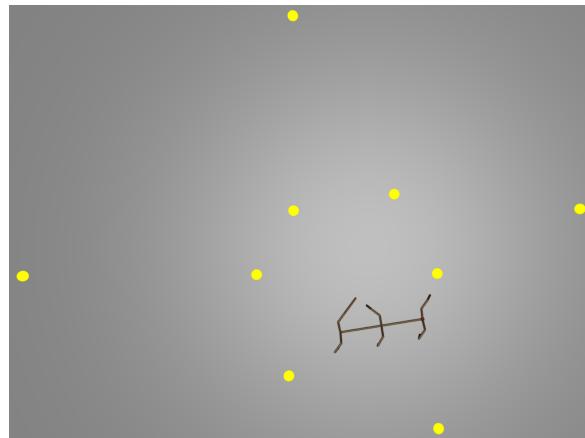
Obrázek 5.3: Scéna pro experiment s trojnožkou na spirále

5.2.2 Model mravence

V tomto experimentu byly použity následující parametry:

- 100 genů v genotypu
- délka podprogramů init a event je 12 instrukcí

Snímek scény simulátoru pro tento experiment je na obrázku 5.4.



Obrázek 5.4: Scéna pro experiment s mravencem na spirále

Kapitola 6

Výsledky experimentů

Pro každý experiment bylo provedeno 20 evolučních běhů, ze kterých byly zaznamenány fitness hodnoty všech jedinců. Pro každý evoluční běh byly z těchto hodnot vykresleny 2 typy grafů:

- vývoj maximální a průměrné fitness hodnoty jedinců v závislosti na generaci,
- vývoj fitness hodnot pro vybrané generace.

Pro každý experiment byla poté vykreslena sada histogramů, které ukazují rozložení fitness hodnot jedinců napříč všemi evolučními běhy.

V této kapitole je pro každý experiment prezentován graf vývoje průměrné a maximální fitness hodnoty v populaci a také vývoj fitness hodnot pro vybrané generace pro nejlepší evoluční běh. Veškeré další grafy, které byly vygenerovány, je možné nalézt v přílohách na přiloženém médiu.

Jedním z parametrů evolučních běhů byla délka simulace, která byla provedena pro každé kandidátní řešení. Na obrázcích, které zobrazují trajektorii nalezených řešení, je trajektorie vyznačena červenou a zelenou barvou. Tato trajektorie byla vykreslena až při ručním vyhodnocování experimentů po skončení všech evolučních běhů. Při tomto ručním vyhodnocování nebyl nastaven časový limit simulace a tudíž se mohlo projevit chování modelu v delším časovém horizontu. Červená barva označuje body v prostoru, do kterých se model dostal, něž vypršel čas přidělený na simulaci daného kandidátního řešení v evolučním běhu. Zelená barva označuje body v prostoru, do kterých se už model nestihl v rámci evolučních běhů dostat. Model tedy nebyl za pohyb po této (zelené) trajektorii nijak hodnocen a jeho pohyb tak zcela závisí na schopnosti kontroléru navigovat model k dalšímu referenčnímu bodu. Zelená trajektorie tedy může být použita k hodnocení kvality nalezeného řešení v prostředích, na které toto řešení nebylo učeno.

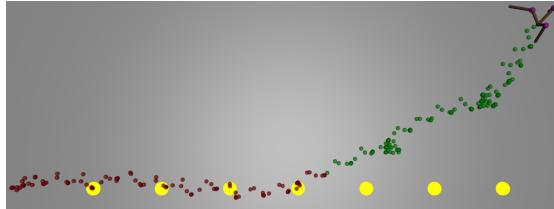
6.1 Přímka

6.1.1 Model trojnožky na přímce

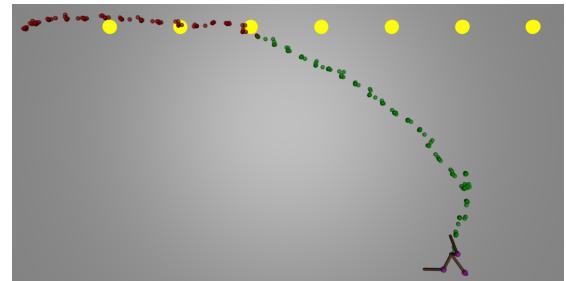
Trajektorii řešení s nejvyšší hodnotou fitness, které bylo v průběhu evolučních běhů pro tento experiment nalezeno, můžeme vidět na obrázku 6.1. Tuto trajektorii můžeme srovnat s jinou, která vznikla řízením neoptimálním řešením, viz obrázek 6.2.

Na těchto trajektoriích je možné vidět, že kontrolér po překročení času simulace, který byl nastaven při trénování, nebyl schopný řídit model tak, aby pokračoval ve stanovené

trajektorii. Toto je způsobeno tím, že kontrolér nemá k dispozici žádné vstupní informace o svém prostředí. Výsledné řešení je tedy schopné ovládat model jen na trajektorii, na které bylo trénováno, a to pouze po dobu, která byla definována simulačním časem při trénování.

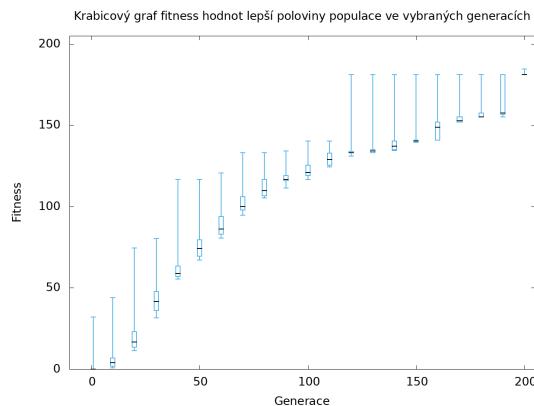


Obrázek 6.1: Trajektorie nejlepšího řešení trojnožky na přímce

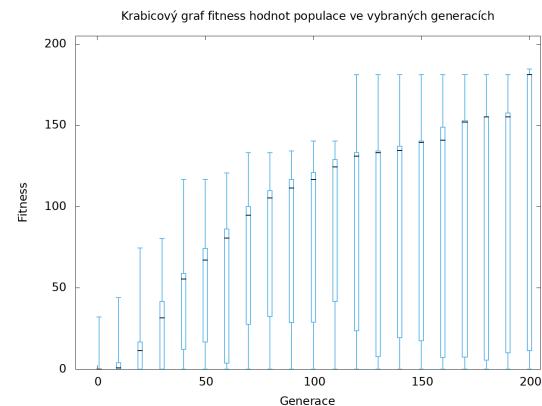


Obrázek 6.2: Trajektorie neoptimálního řešení trojnožky na přímce

Na obrázcích 6.3 a 6.4 jsou vykresleny grafy, ukazující maximální a minimální hodnoty fitness, stejně jako její kvartily. První z grafů je vykreslen pro lepší polovinu populace, druhý je vykreslen pro celou populaci.



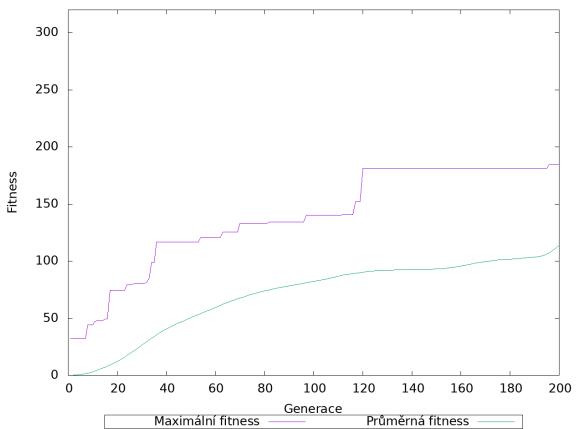
Obrázek 6.3: Vývoj fitness hodnot lepší poloviny populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu trojnožky na přímce



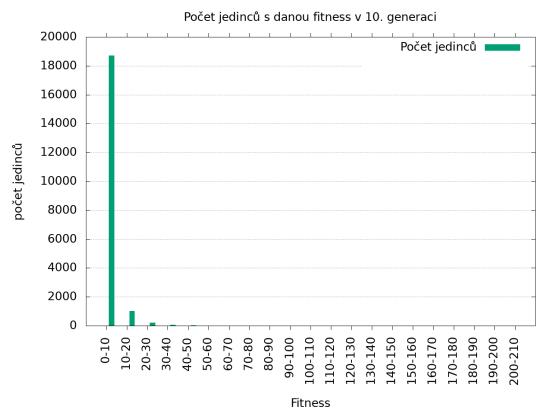
Obrázek 6.4: Vývoj fitness hodnot populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu trojnožky na přímce

Obrázek 6.5 zobrazuje průběh maximální a průměrné fitness hodnoty v jednotlivých generacích pro nejlepší běh evoluce. Obrázky 6.6 až 6.10 zobrazují agregovaná data všech 20 simulačních běhů pro tento experiment. Jsou zde vykresleny počty jedinců, jejichž fitness spadala do daného rozsahu hodnot.

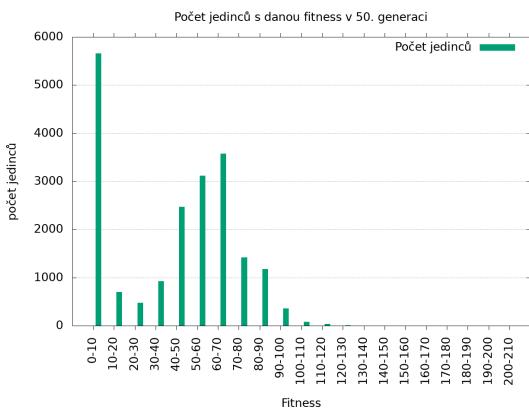
Můžeme si všimnout, že na histogramech i vývojích fitness hodnot je vidět velký počet jedinců, kteří mají velmi malou fitness v rozmezí 0 až 10 bodů. To je způsobeno skutečností, že model trojnožky je výjimečně nestabilní a tak i zdánlivě malá změna, jinak dobré fungujícího programu, může mít za následek převrácení modelu. Pokud v simulaci došlo k převrácení modelu, bylo toto kandidátní řešení ohodnoceno fitness hodnotou 0 a simulace byla přerušena.



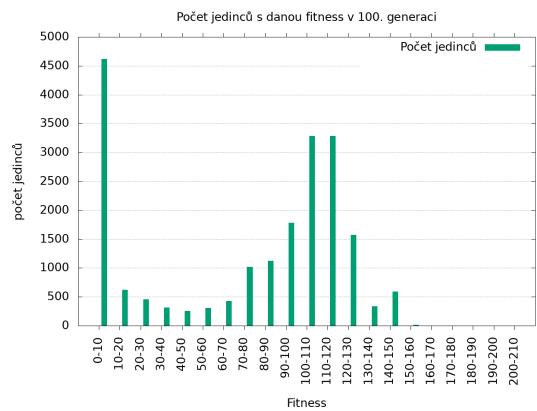
Obrázek 6.5: Průběh evoluce, která vedla k nalezení nejlepšího jedince pro experiment s trojnožkou na přímce



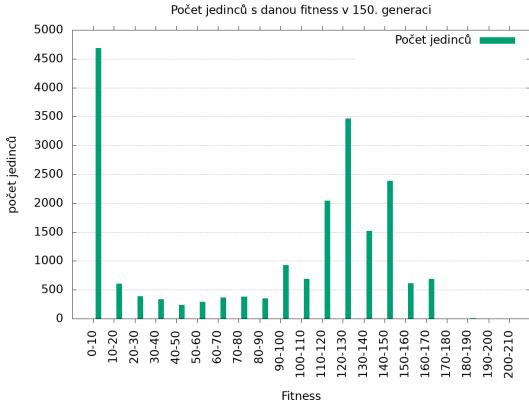
Obrázek 6.6: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 10. generaci



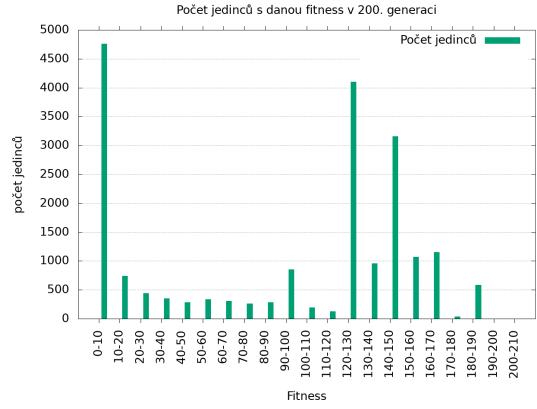
Obrázek 6.7: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 50. generaci



Obrázek 6.8: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 100. generaci



Obrázek 6.9: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 150. generaci



Obrázek 6.10: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 200. generaci

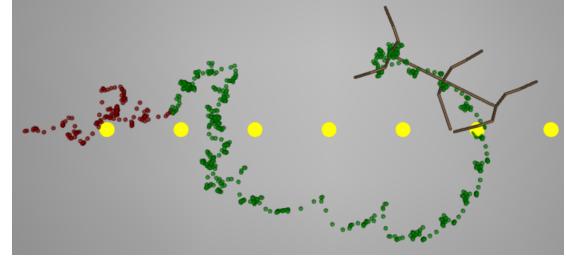
6.1.2 Model mravence na přímce

Trajektorii řešení s nejvyšší hodnotou fitness, které bylo v průběhu evolučních běhů pro tento experiment nalezeno, můžeme vidět na obrázku 6.11. Tuto trajektorii můžeme srovnat s jinou, která vznikla řízením neoptimálním řešením, viz obrázek 6.12.

Na trajektorii nejlepšího řešení lze vidět, že toto řešení bylo schopné pokračovat ve stanovené trajektorii i po překročení času simulace, který byl nastaven při trénování. Ve srovnání s trojnožkou je tento výsledek lepší, ale i zde je to dílo náhody – tedy řešení není schopné navigace v neznámém prostředí.

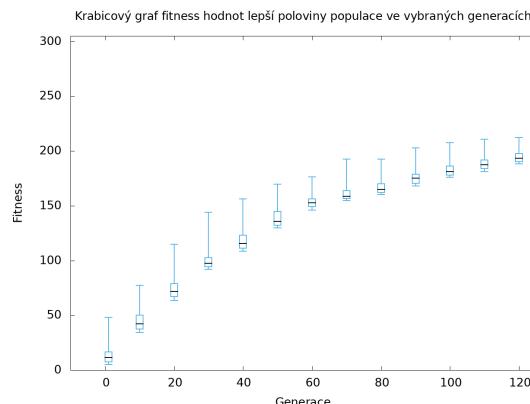


Obrázek 6.11: Trajektorie nejlepšího řešení mravence na přímce

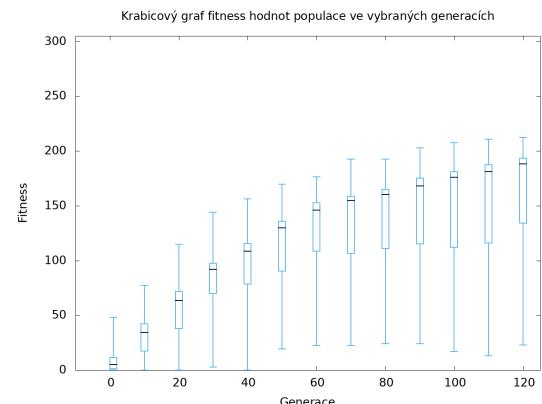


Obrázek 6.12: Trajektorie neoptimálního řešení mravence na přímce

Na obrázcích 6.13 a 6.14 jsou vykresleny grafy, ukazující maximální a minimální hodnoty fitness, stejně jako její kvartily. První z grafů je vykreslen pro lepší polovinu populace, druhý je vykreslen pro celou populaci.

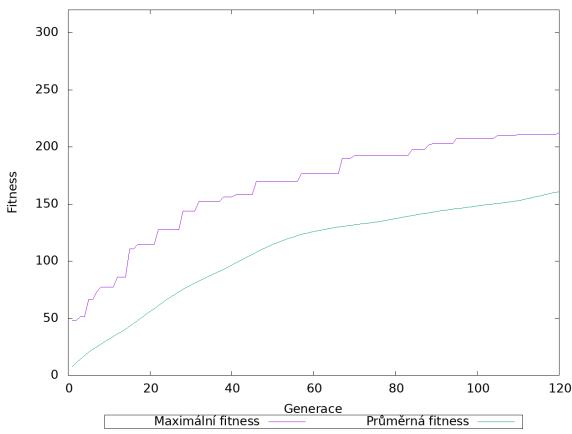


Obrázek 6.13: Vývoj fitness hodnot lepší poloviny populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu mravence na přímce

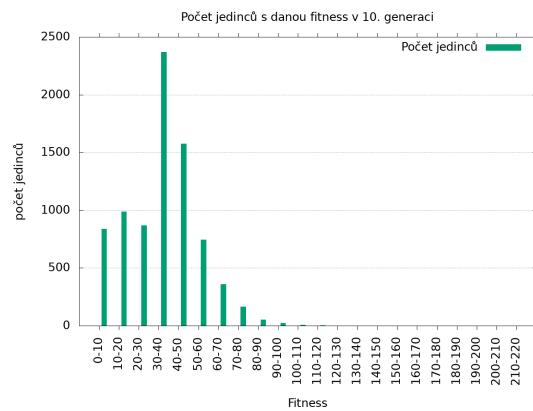


Obrázek 6.14: Vývoj fitness hodnot populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu mravence na přímce

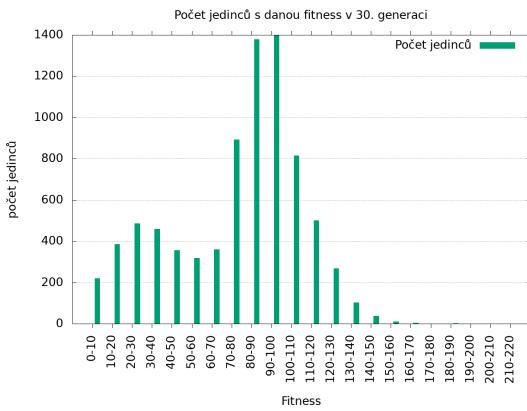
Obrázek 6.15 zobrazuje průběh maximální a průměrné fitness hodnoty v jednotlivých generacích pro nejlepší běh evoluce. Obrázky 6.16 až 6.20 zobrazují agregovaná data všech 20 simulačních běhů pro tento experiment. Jsou zde vykresleny počty jedinců, jejichž fitness spadala do daného rozsahu hodnot.



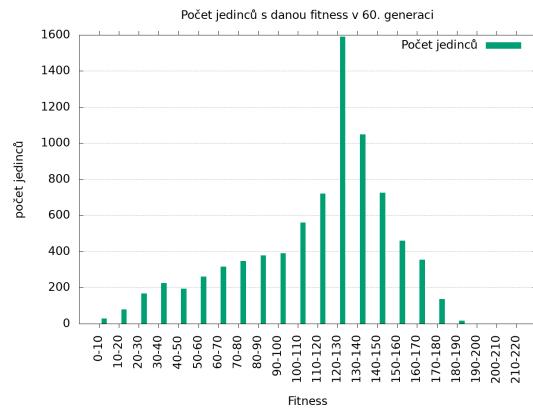
Obrázek 6.15: Průběh evoluce, která vedla k nalezení nejlepšího jedince pro experiment s mravencem na přímce



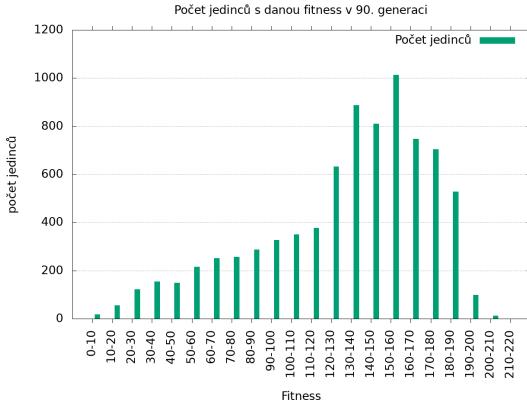
Obrázek 6.16: Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 10. generaci



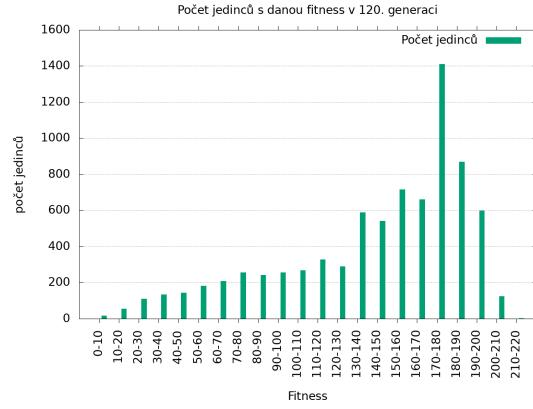
Obrázek 6.17: Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 30. generaci



Obrázek 6.18: Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 60. generaci



Obrázek 6.19: Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 90. generaci



Obrázek 6.20: Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 120. generaci

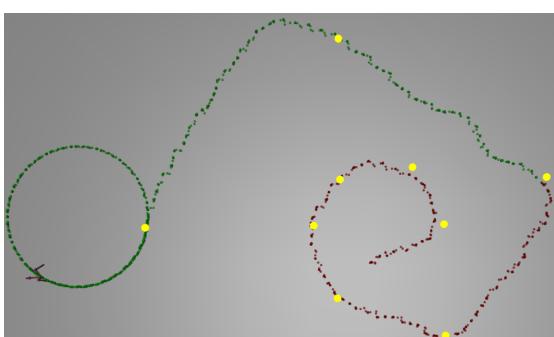
6.2 Spirála

6.2.1 Model trojnožky na spirále

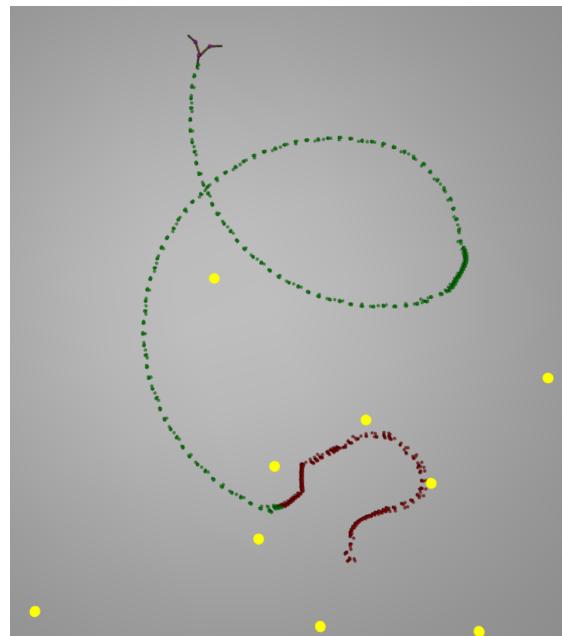
Trajektorii řešení s nejvyšší hodnotou fitness, které bylo v průběhu evolučních běhů pro tento experiment nalezeno, můžeme vidět na obrázku 6.21. Tuto trajektorii můžeme srovnat s jinou, která vznikla řízením neoptimálním řešením, viz obrázek 6.22.

Na trajektorii nejlepšího jedince (obrázek 6.21) můžeme vidět velmi podobné chování, jako u experimentu s mravencem na spirále, které je blíže popsáno u tohoto experimentu v podsekci 6.2.2. Toto konkrétní řešení ovládá model tak, že bez informace o směru k dalšímu referenčnímu bodu se model pohybuje po téměř dokonalé kružnici a to ve směru hodinových ručiček. Tento pohyb je korigován pomocí vstupních hodnot programu tak, že je model schopný pohybu po přímce, nebo i po spirále, která vyžaduje pohyb proti směru hodinových ručiček.

Na této trajektorii můžeme také vidět, že kontrolér byl díky informaci o svém prostředí (směru k dalšímu referenčnímu bodu) schopen řídit model tak, že byl tento model schopný pokračovat v pohybu k dalšímu referenčnímu bodu, byl tedy schopný generalizace.



Obrázek 6.21: Trajektorie nejlepšího řešení trojnožky na spirále

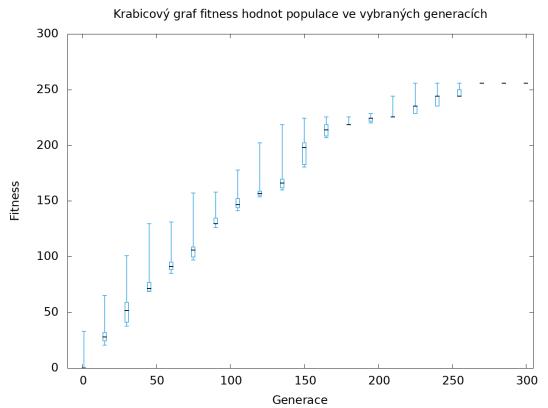


Obrázek 6.22: Trajektorie neoptimálního řešení trojnožky na spirále

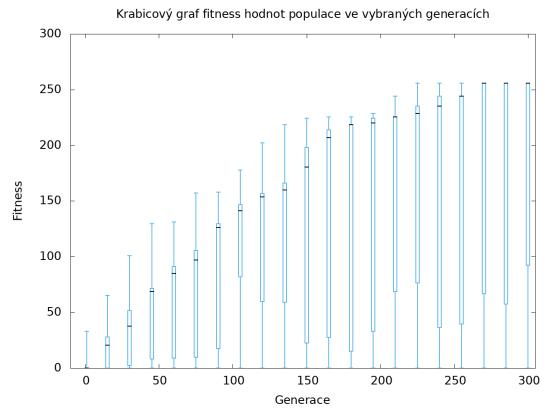
Na obrázcích 6.23 a 6.24 jsou vykresleny grafy, ukazující maximální a minimální hodnoty fitness, stejně jako její kvartily. První z grafů je vykreslen pro lepší polovinu populace, druhý je vykreslen pro celou populaci.

Obrázek 6.25 zobrazuje průběh maximální a průměrné fitness hodnoty v jednotlivých generacích pro nejlepší běh evoluce. Obrázky 6.26 až 6.30 zobrazují agregovaná data všech 20 simulačních běhů pro tento experiment. Jsou zde vykresleny počty jedinců, jejichž fitness spadala do daného rozsahu hodnot. Na těchto grafech je možné opět pozorovat velké množství jedinců s nízkou fitness, stejně jako ve výsledcích experimentu s trojnožkou na přímce.

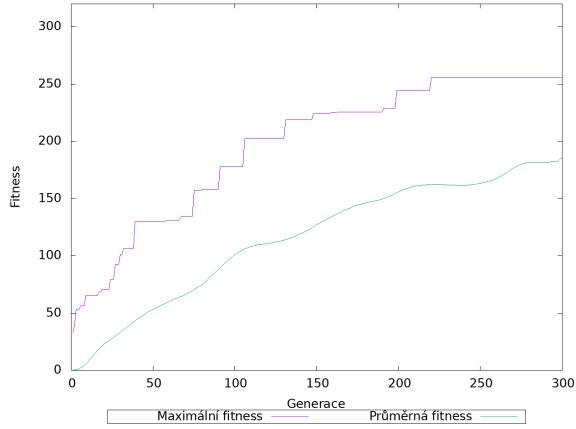
Podobně jako v experimentu s mravencem na spirále, i zde řešení s nejlepší hodnotou fitness nevykazovalo nejlepší chování při ručním vyhodnocování. Toto lze pozorovat například



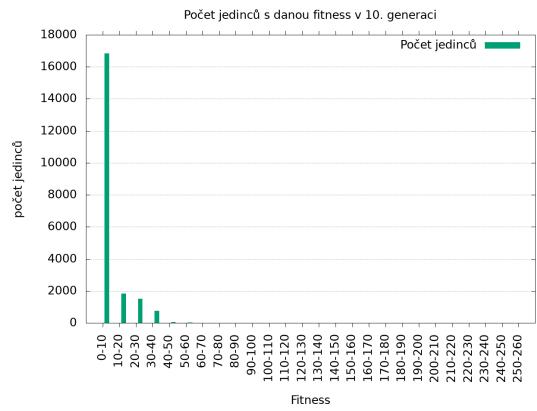
Obrázek 6.23: Vývoj fitness hodnot lepší poloviny populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu trojnožky na spirále



Obrázek 6.24: Vývoj fitness hodnot populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu trojnožky na spirále

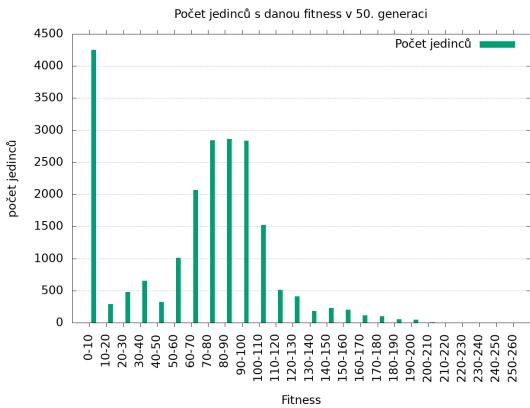


Obrázek 6.25: Průběh evoluce, která vedla k nalezení nejlepšího jedince pro experiment s trojnožkou na spirále

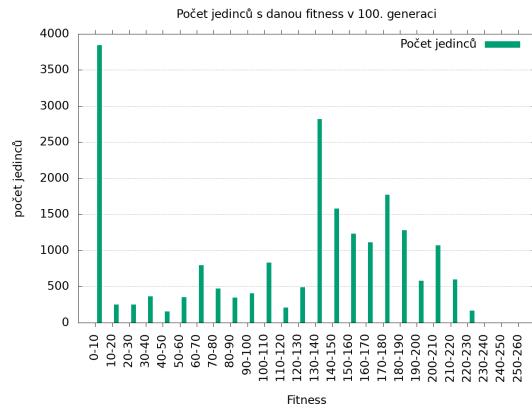


Obrázek 6.26: Histogram celkového počtu jedinců ve všech bězích trojnožky na spirále v 10. generaci

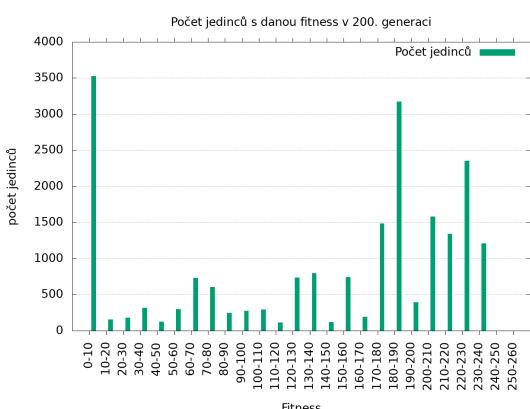
na čtvercové trajektorii, kde můžeme porovnat trajektorii nejlepšího řešení 6.31 s trajektorií druhého nejlepšího řešení 6.32. Na jiných trajektoriích měla většina řešení problémy se stabilitou modelu a docházelo k převrácení. To je dáno tím, že samotný model trojnožky je nestabilní a tak např. při nutnosti prudce změnit směr pohybu může jednoduše dojít k převrácení.



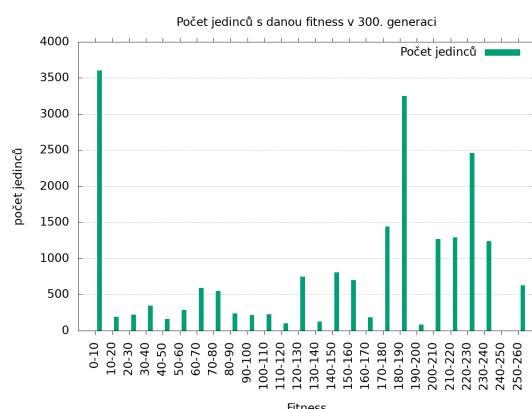
Obrázek 6.27: Histogram celkového počtu jedinců ve všech bězích trojnožky na spirále v 50. generaci



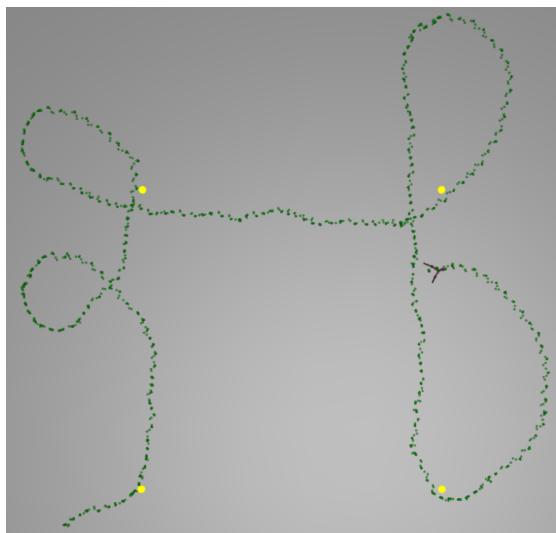
Obrázek 6.28: Histogram celkového počtu jedinců ve všech bězích trojnožky na spirále v 100. generaci



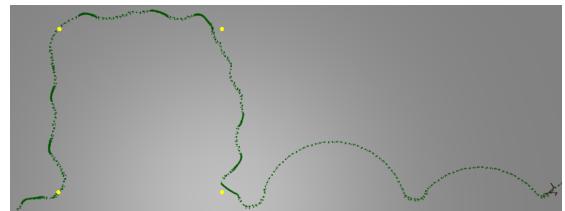
Obrázek 6.29: Histogram celkového počtu jedinců ve všech bězích trojnožky na spirále v 200. generaci



Obrázek 6.30: Histogram celkového počtu jedinců ve všech bězích trojnožky na spirále v 300. generaci



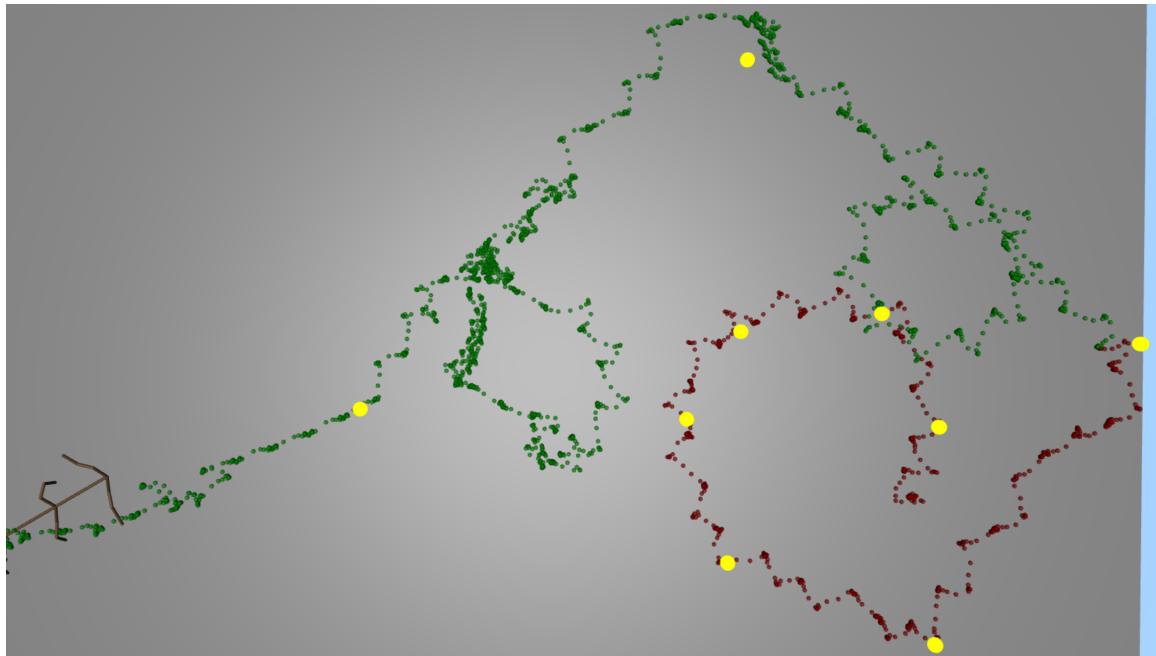
Obrázek 6.31: Nejznámá testovací trajektorie č.1 pro kontrolér trojnožky – nejlepší řešení



Obrázek 6.32: Nejznámá testovací trajektorie č.1 pro kontrolér trojnožky – druhé nejlepší řešení

6.2.2 Model mravence na spirále

Trajektorii řešení s nejvyšší hodnotou fitness, které bylo v průběhu evolučních běhů pro tento experiment nalezeno, můžeme vidět na obrázku 6.33. Toto řešení je asi o 7 bodů fitness lepší, než-li druhé nejlepší řešení, jehož trajektorie je vidět na obrázku 6.34. Při porovnání trajektorií nejlepšího a druhého nejlepšího řešení lze usoudit, že zde došlo k přílišné specializaci pouze na část spirály a že toto řešení již není schopné správně pokračovat po zbytku spirály. Nejlepší řešení má tedy o pár bodů fitness více, ale není tak dobré na zbytku trajektorie, kterou už model nestihl v přiděleném čase na simulaci dokončit.



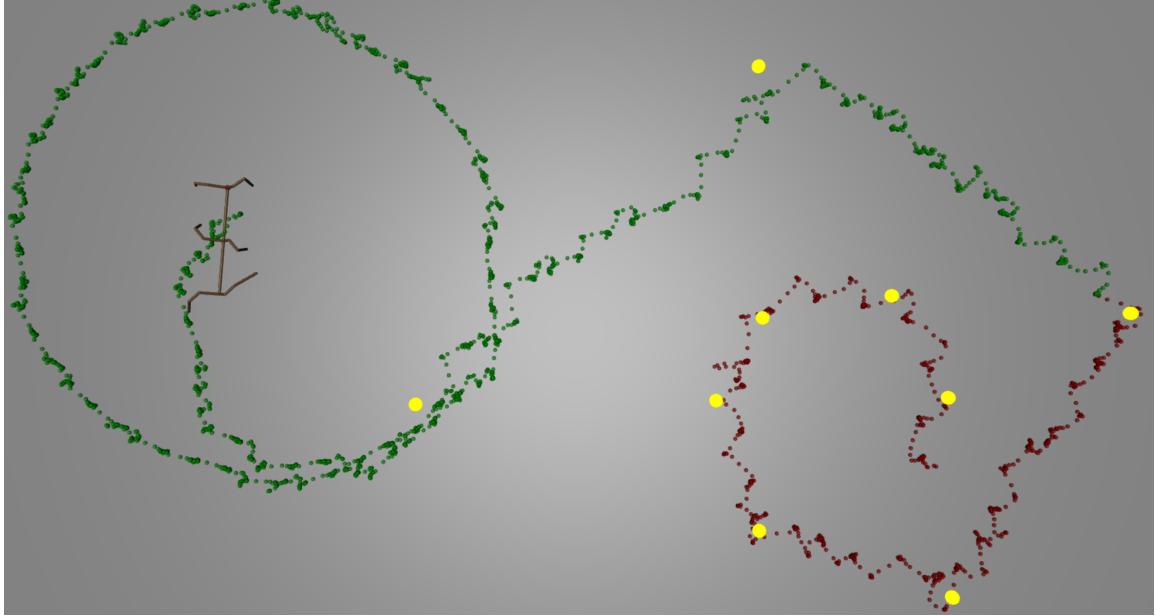
Obrázek 6.33: Trajektorie řešení s největší fitness mravence na spirále

U trajektorie druhého nejlepšího řešení (obrázek 6.34) se na chvíli pozastavíme. Na této trajektorie lze pozorovat několik zajímavostí. V průběhu evolučních běhů bylo toto řešení schopné se přiblížit v přiděleném čase k prvním 7 referenčním bodům (červená trajektorie). Při ručním vyhodnocování řešení bylo toto řešení schopno pokračovat po stanovené trajektorii (zelená část), která byla zhruba definována referenčními body, které od sebe byly relativně velmi vzdálené.

Požadovaná spirálová trajektorie nebyla definována dostatečným počtem bodů, výsledné řešení tedy mezi těmito body zvolilo nejkratší možnou cestu. Po přiblížení k poslednímu referenčnímu bodu již toto řešení nedostávalo validní informaci o směru k dalšímu referenčnímu bodu, protože zde již další referenční bod nebyl. Místo toho tato informace pro program znamenala to, že další referenční bod se vyskytuje přímo před modelem. Model tedy opsal téměř dokonalou kružnici a v tomto pohybu pokračoval až do ukončení simulace.

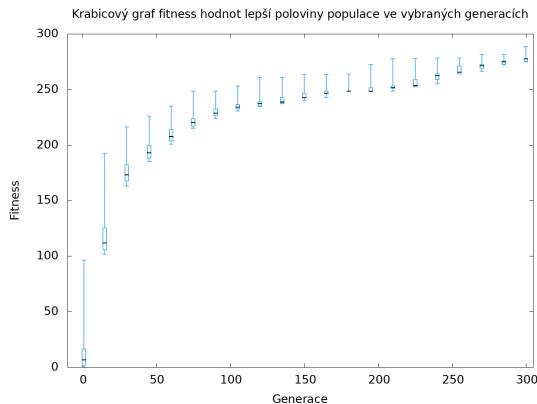
Z tohoto chování lze vypozorovat způsob, jakým tento konkrétní kontrolér řídí model (ačkoli stejné chování bylo pozorováno i ve výsledcích experimentů s trojnožkou na spirále). Tento kontrolér, bez informací o směru k dalšímu referenčnímu bodu, má za následek pohyb modelu po kružnici ve směru hodinových ručiček. Tento pohyb je poté s použitím informací o směru k dalšímu referenčnímu bodu korigován tak, že se je model schopný pohybovat po přímce, nebo i dokonce po oblouku opečného směru, tedy proti směru hodinových ručiček.

Je tedy zajímavé, že pro potřeby pohybu po spirále, po které se má model pohybovat proti směru hodinových ručiček, se vyvinul kontrolér, který provádí přesně opačný pohyb.

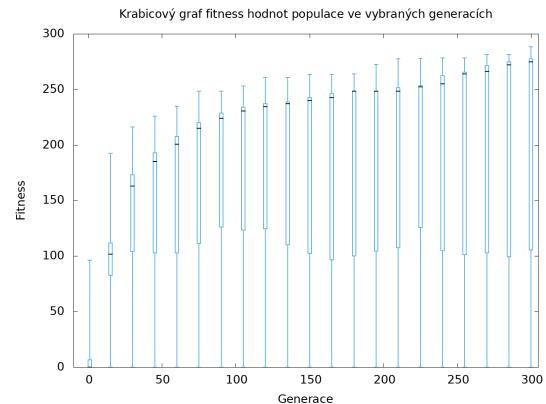


Obrázek 6.34: Trajektorie nejlepšího řešení mravence na spirále

Na obrázcích 6.35 a 6.36 jsou vykresleny grafy, ukazující maximální a minimální hodnoty fitness, stejně jako její kvartily. První z grafů je vykreslen pro lepší polovinu populace, druhý je vykreslen pro celou populaci.

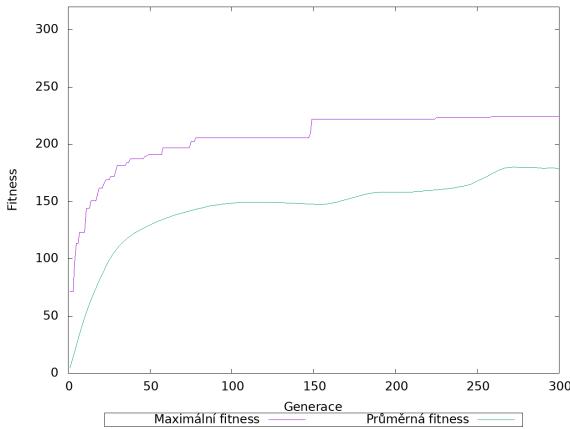


Obrázek 6.35: Vývoj fitness hodnot lepší poloviny populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu mravence na spirále

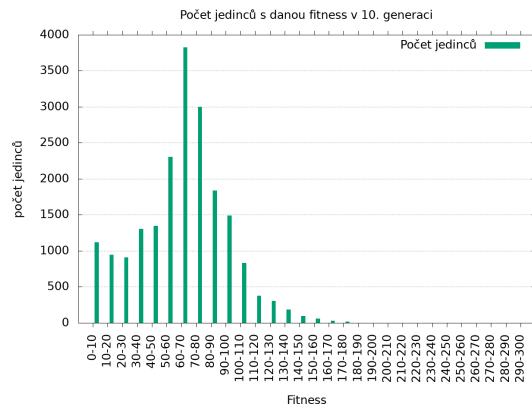


Obrázek 6.36: Vývoj fitness hodnot populace evolučního běhu, který vedl k nalezení nejlepšího jedince experimentu mravence na spirále

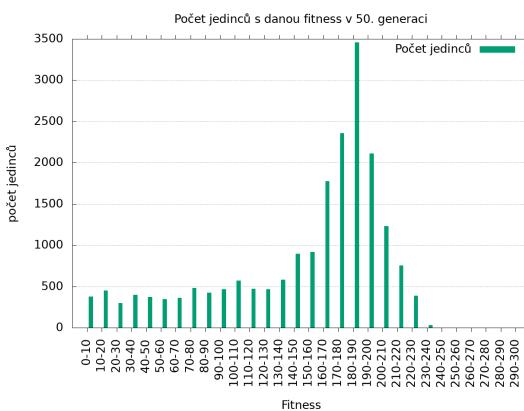
Obrázek 6.37 zobrazuje průběh maximální a průměrné fitness hodnoty v jednotlivých generacích pro nejlepší běh evoluce. Obrázky 6.38 až 6.42 zobrazují agregovaná data všech 20 simulačních běhů pro tento experiment. Jsou zde vykresleny počty jedinců, jejichž fitness spadala do daného rozsahu hodnot.



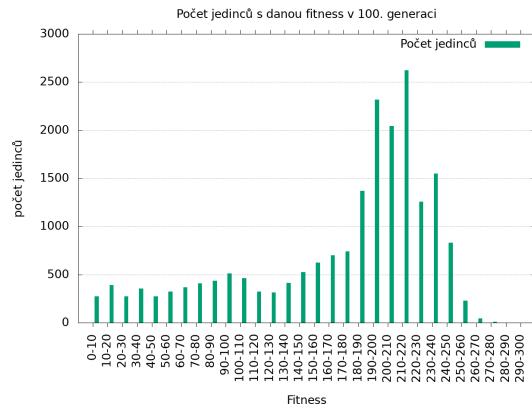
Obrázek 6.37: Průběh evoluce, která vedla k nalezení nejlepšího jedince pro experiment s mravencem na spirále



Obrázek 6.38: Histogram celkového počtu jedinců ve všech bězích mravence na spirále v 10. generaci

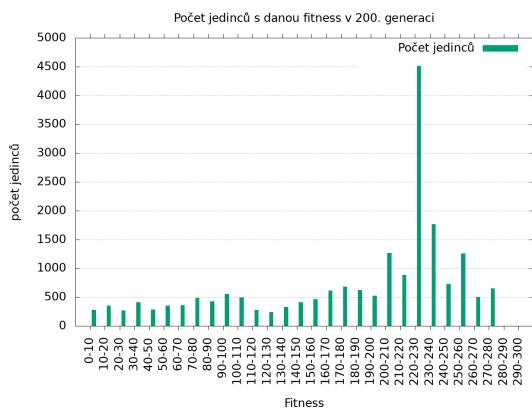


Obrázek 6.39: Histogram celkového počtu jedinců ve všech bězích mravence na spirále v 50. generaci

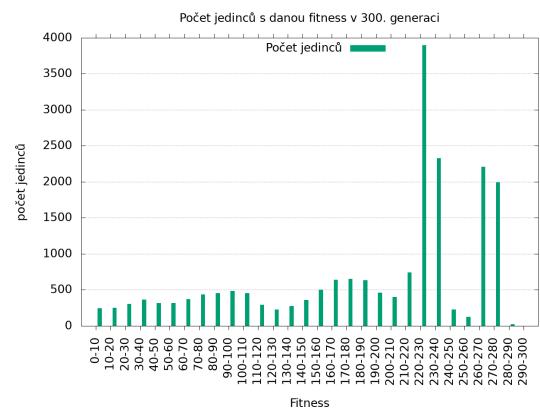


Obrázek 6.40: Histogram celkového počtu jedinců ve všech bězích mravence na spirále v 100. generaci

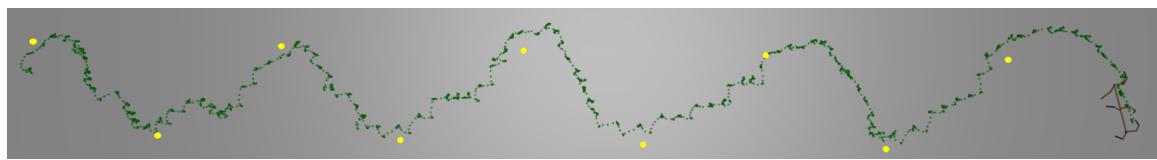
Nalezené řešení bylo ověřeno na několika dalších trajektoriích, které jsou znázorněny na obrázcích 6.43 až 6.45. Z těchto trajektorií lze vypozorovat, že schopnost daného řešení pohybovat se po neznámé trajektorii se různí. Trajektorii vycházející z funkce cosinus (obrázek 6.43), zvládlo řešení běž problému. Další trajektorii vyžadující relativně rychlé otočení modelu (obrázek 6.44), toto řešení také zvládlo bez větších problémů. Můžeme zde pozorovat, že daný kontrolér má větší poloměr otáčení při zátáčení vpravo, než-li vlevo – např. mezi druhým a třetím referenčním bodem, nebo při zatáčení k poslednímu referenčnímu bodu. Poslední trajektorii, která definovala čtverec (obrázek 6.45), toto řešení sice dokončilo, ale velmi neefektivním způsobem. Ukázalo se, že toto řešení bylo do jisté míry schopné generalizace, ale tato schopnost nebyla dokonalá.



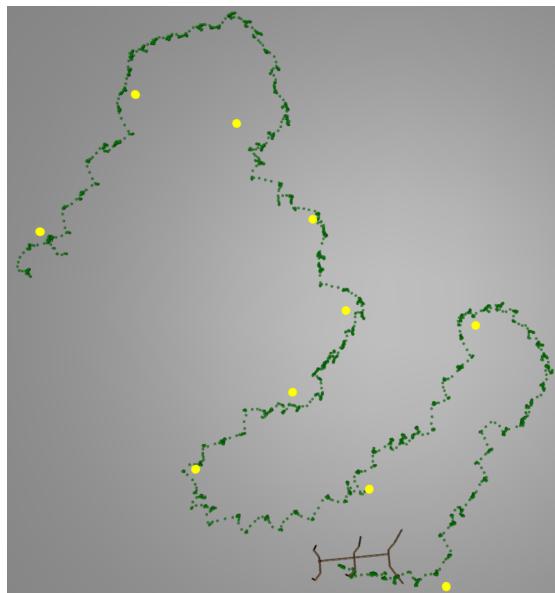
Obrázek 6.41: Histogram celkového počtu jedinců ve všech bězích mravence na spirále v 200. generaci



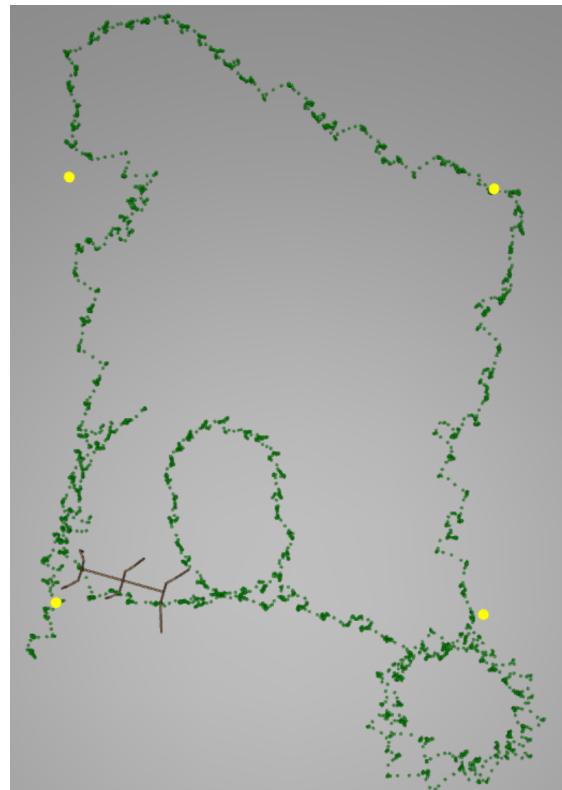
Obrázek 6.42: Histogram celkového počtu jedinců ve všech bězích mravence na spirále v 300. generaci



Obrázek 6.43: Neznámá testovací trajektorie č.1 pro kontrolér mravence



Obrázek 6.44: Neznámá testovací trajektorie č.2 pro kontrolér mravence



Obrázek 6.45: Neznámá testovací trajektorie č.3 pro kontrolér mravence

Kapitola 7

Závěr

Tato práce se zabývala evolučním návrhem kontroléru pro robotický model v počítačové simulaci, který měl za úkol pohybovat se po stanovené trajektorii. K tomuto účelu byla použita metoda, založená na lineárním genetickém programování. Funkčnost této metody byla ověřena provedením 4 experimentů. Tyto experimenty ukázaly, že je tuto metodu možné použít pro evoluční návrh jednoduchých robotických kontrolérů. Pro všechny experimenty se podařilo najít kontrolér, který by odpovídal požadavkům, které byly dány konfigurací konkrétního experimentu. Konkrétně experimenty na spirále, ve kterých byly použity informace o směru k dalšímu referenčnímu bodu, ukázaly, že nalezená řešení jsou částečně schopná generalizace, tedy řídit robota i v prostředích, kterým nikdy nebyla vystavena – tato schopnost je nejlepší u kontroléru, který byl vyvinut pro experiment s mravencem na spirále.

Jako pokračování v této práci bych se chtěl zaměřit na zvýšení schopnosti generalizace kontrolérů. K tomu bych využil způsob, kdy by fitness funkce byla složena z více simulací na různých trajektoriích, jejichž výsledky se poté skládaly do výsledné fitness hodnoty. Pro urychlení evolučního běhu by se také dal využít přístup, kdy by se počáteční populace kandidátních řešení negenerovala náhodně, ale byla by založena na kandidátních řešeních, které byly schopné model ovládat na přímkové trajektorii. Tímto způsobem by při evoluci nebylo nutné od začátku hledat kontroléry, které jsou schopné řídit model tak, aby byl schopen chodit, ale bylo by možné se zaměřit na schopnost generalizace.

Literatura

- [1] Bongard, J. C.: Evolutionary Robotics. *Commun. ACM*, ročník 56, č. 8, Srpen 2013: s. 74–83, ISSN 0001-0782, doi:10.1145/2493883.
URL <http://doi.acm.org/10.1145/2493883>
- [2] Bräuer, M. F.; Banzhaf, W.: *Linear Genetic Programming*. Springer Publishing Company, Incorporated, první vydání, 2010, ISBN 1441940480, 9781441940483.
- [3] Eiben, A. E.; Smith, J. E.: *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, druhé vydání, 2015, ISBN 3662448734, 9783662448731.
- [4] Hodgins, J. K.: Three-dimensional human running. In *Proceedings of IEEE International Conference on Robotics and Automation*, ročník 4, Apr 1996, ISSN 1050-4729, s. 3271–3276 vol.4, doi:10.1109/ROBOT.1996.509211.
- [5] Holland, J. H.: *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992, ISBN 0262082136.
- [6] Hornby, G. S.: *Generative Representations for Evolutionary Design Automation*. Dizertační práce, Waltham, MA, USA, 2003, aAI3073875.
- [7] Koza, J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992, ISBN 0-262-11170-5.
- [8] Reil, T.; Husbands, P.: Evolution of central pattern generators for bipedal walking in a real-time physics environment. *IEEE Transactions on Evolutionary Computation*, ročník 6, č. 2, Apr 2002: s. 159–168, ISSN 1089-778X, doi:10.1109/4235.996015.
- [9] SYSWERDA, G.: Uniform crossover in genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, 1989: s. 2–9.
URL <https://ci.nii.ac.jp/naid/10000012509/en/>
- [10] Todorov, E.; Erez, T.; Tassa, Y.: MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012, ISSN 2153-0858, s. 5026–5033, doi:10.1109/IROS.2012.6386109.
- [11] Wolff, K.; Wahde, M.: Evolution of Biped Locomotion Using Linear Genetic Programming. [Online; navštívěno 20.03.2018].
URL https://www.intechopen.com/books/climbing_and_walking_robots_towards_new_applications/_evolution_of_biped_locomotion_using_linear_genetic_programming

Příloha A

Obsah přiloženého CD

```
cd
└── experimenty
    ├── mravenec_primka
    ├── mravenec_spirala
    ├── trojnozka_primka
    └── trojnozka_spirala
├── latex
└── zdrojove_kody
    ├── evo_framework
    └── simulator
```

Každý podadresář adresáře `experimenty` obsahuje grafy, přeložené binární soubory, soubor s modelem, nejlepší řešení a Makefile pro spuštění daného experimentu. Pro spuštění stačí v konkrétním podadresáři (např. `mravenec_primka`) spustit `make compute` nebo `make render`.

V adresáři `latex` jsou umístěny zdrojové soubory potřebné pro přeložení textu této práce.

V adresáři `zdrojove_kody` jsou umístěny zdrojové soubory aplikace, v podadresářích jsou poté zdrojové soubory aplikace se simulátorem a evolučním frameworkem.

Příloha B

Instalace a spuštění aplikace

Pro běh aplikace jsou potřeba: CMake, PHP 7.2 a simulátor MuJoCo.

Aplikace byla vyvíjena a testována na OS Linux (Arch Linux, distribuce Manjaro).

Návod pro instalaci simulátoru MuJoCo lze nalézt na oficiálních stránkách¹. Pro běh simulátoru je potřeba mít licenci, kterou lze získat na webu². Licence je vázaná na konkrétní počítač a je uložena v souboru mjkey.txt, který musí být umístěn ve stejném adresáři, jako binární soubory (je zde umístěn prázdný soubor, který je potřeba nahradit souborem s licencí). Úspěšnost instalace simulátoru MuJoCo lze ověřit pomocí přeložených programů a souboru Makefile v adresáři `sample`, který je součástí distribuce simulátoru.

Pokud fungují programy v adresáři `sample`, mělo by být možné spustit binární soubory v adresáři `experimenty`, nebo přeložit zdrojové kódy (postup dál). V adresáři `experimenty` jsou podadresáře pro každý experiment, obsahující binární soubory pro spuštění simulace (`compute`), který vypočítává fitness hodnotu, nebo pro vizualizaci výsledku (`render`). V každém podadresáři jsou také soubory, definující použitý model (`model.xml`) a soubor s instrukcemi (`individuals.txt`).

Pro zkompilování zdrojových kódů je připraven konfigurační soubor pro CMake. Pomocí CMake můžeme následujícími příkazey vygenerovat Makefile a spustit jej:

```
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug  
make
```

Pro konfiguraci základních parametrů programů `compute` a `render` je připraven soubor `config.h`, ve kterém je možné nastavit počet referenčních bodů a počet řízených kloubů modelu (např. při změně modelu nebo změně počtu referenčních bodů). Je zde také nastavení pro rychlosť vizualizace.

B.1 Program compute

Program `compute` je aplikace, obalující simulátor, sloužící k výpočtu fitness hodnot programů. Tento program pracuje s xml souborem, obsahujícím model, a se souborem `individuals.txt`, ze kterého čte jednotlivá kandidátní řešení. Tato kandidátní řešení jsou v souboru oddělena řádkem začínajícím hvězdičkou (viz soubor `experimenty/individuals.txt`). Program pro každé kandidátní řešení provede simulaci a do souboru `fitnesses.txt` vypíše fitness hodnoty pro každé kandidátní řešení v pořadí, v jakém byly v souboru `individuals.txt`.

¹<http://www.mujoco.org/book/programming.html#Introduction>

²<https://www.roboti.us/license.html>

B.2 Program render

Program `render` je aplikace, obalující simulátor, sloužící k vizualizaci konkrétního řešení. Tento program, stějně jako program `compute`, pracuje s xml souborem, obsahujícím model, a se souborem `individuals.txt`, ze kterého čte jedno kandidátní řešení. Program pro toto kandidátní řešení spustí simulaci, která je vizualizována pomocí OpenGL. V této simulaci je možné na model aplikovat různé síly (posuny, rotace).

B.3 Evoluční framework

Aplikace obsahující evoluční algoritmus je samostatná aplikace napsaná v jazyce PHP verze 7.2. Pro běh aplikace je tedy nutné mít nainstalovaný na počítači interpret jazyka PHP verze 7.2, dostupný např. na webu³, nebo v balíčovacím systému OS (doporučuji).

Tato aplikace se spouští příkazem:

```
php src/Run/run.php <EXPERIMENT>
```

Kde `<EXPERIMENT>` je název experimentu, který chceme spustit. Dostupné experimenty mají jména:

- trojnozka-primka
- trojnozka-spirala
- mravenec-spirala
- mravenec-primka

Po spuštění aplikace vypíše název experimentu, který spouští (to je ve skutečnosti název třídy, ve které je definované nastavení experimentu, jako např. počet jedinců nebo délka simulace). Aplikace při spuštění vytvoří v adresáři `data` nový adresář, který se jmeneje podle názvu experimentu a v něm podadresář, jehož název je aktuální datum a čas. Aplikace pro každou generaci v tomto adresáři vytvoří nový podadresář pro každou generaci, která je vyhodnocena. Tento podadresář obsahuje soubor `individuals.txt` a `fitnesses.txt`. Poté jsou po vyhodnocení všech jedinců vypsány statistiky o aktuální populaci, které se zároveň vypisují do souboru `log.txt`.

³<http://php.net/downloads.php>