Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: SOFTVÉROVÉ INŽINIERSTVO

Bc. Marek Hlaváč

POUŽITIE GRAMATICKÉHO ROJA V RIADENÍ ROBOTICKÝCH TANKOV PRE HRU ROBOCODE

Diplomový projekt II

Vedúci diplomového projektu: prof. RNDr. Jiří Pospíchal, DrSc.
December, 2012

Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: SOFTVÉROVÉ INŽINIERSTVO

Autor: Marek Hlaváč
Diplomový projekt: Použitie gramatického roja v riadení robotických tankov pre hru Robocode
Vedenie diplomového projektu: Prof. RNDr. Jiří Pospíchal, DrSc.
December, 2012

Diplomový projekt je zameraný na vytvorenie robotického tanku pre hru Robocode, ktorý prispôsobuje svoje správanie v dynamickom prostredí. Cieľ práce spočíva v použití gramatického roja za účelom evolvovať komplexné správanie robota, ktoré by bolo konkurencieschopné voči existujúcim robotom. Gramatický roj kombinuje gramatickú evolúciu a optimalizáciu rojom častíc za účelom nájsť optimálne riešenia pre dynamické problémy, ktoré sú reprezentované vhodne zvolenou gramatikou. Gramatika definuje povolenú množinu operácií, ktorými je správanie robota ohraničené.

Práca obsahuje podrobnú analýzu použitých metód a existujúcich riešení. Detailne sa venuje návrhu gramatiky, možnostiam modifikácie optimalizačných metód, kvantifikácii kvality riešení a architektúre softvérového prototypu. Následne je rozobratá etapa implementácie, ktorá sa venuje postupu práce a komunikácie medzi jednotlivými softvérovými modulmi. Posledné časti sa týkajú výsledkov testovania nájdených robotov v súbojoch s existujúcimi robotmi. Tento spôsob umožňuje určiť úroveň ich konkurencieschopnosti, odhaliť slabiny a opísať možnosti budúceho vylepšenia.

Annotation

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree Course: SOFTWARE ENGINEERING

Author: Marek Hlaváč
Diploma project: Using grammatical swarm robotic control in tanks for the game Robocode
Supervisor: Prof. RNDr. Jiří Pospíchal, DrSc.
2012, December

Master thesis is focused on creating a robotic tank for Robocode game which adapts behavior in dynamic environments. The main objective of the work lies in the usage of grammatical swarm to evolve complex robot behavior that is based on improving robots competitiveness against existing robots. Grammatical swarm combines grammatical evolution and particle swarm optimization to find optimal solutions for dynamic problems which are represented by properly chosen grammar. The grammar defines a set of allowed operations that bounds the options of robot`s behavior.

The document contains a detailed analysis of applied methods and existing solutions to the problem. Several chapters discussed the grammar design, possible modification of optimization methods, quality quantification of solutions and software prototype architecture. Implementation phase describes the work process and communication between individual software modules. Last section is focused on the testing results of designed robots based on battles with existing robots. This method allows to determine their level of competitiveness, identify weaknesses and describe opportunities for future improvements.

Obsah

1 Úvod

Umelej inteligencii sa v softvérovom inžinierstve venuje čoraz viac pozornosti, pretože si nachádza široké uplatnenie v mnohých softvérových systémoch a disponuje obrovským potenciálom nasadzovania v budúcnosti. Jednou z oblastí, ktorou sa odborníci aktívne zaoberajú sú optimalizačné úlohy. Cieľom úloh je aplikácia vhodnej optimalizácie vzhľadom na vlastnosti hľadaného riešenia.

Rozsiahly stavový priestor je hlavným problémom mnohých úloh, ktorých cieľom je vytváranie inteligentných umelých agentov. Agenty sú špecifické tým, že prispôsobujú svoju činnosť okolitému prostrediu za účelom splnenia zadaných cieľov. Pri riešení špecifických úloh musí agent získať znalosť, ktorá predstavuje vhodnú postupnosť krokov smerujúcich k riešeniu. Pri komplexnejších úlohách môže byť postupné prehľadávanie stavového priestoru nemožné z hľadiska výpočtových prostriedkov. Ďalšou komplikáciou je dynamický charakter úloh, ktorý spôsobuje meniace vlastnosti problému v čase.

V optimalizačných úlohách tohto druhu sa veľmi často pristupuje k použitiu evolučných výpočtových techník s využitím adaptívnych prístupov riešenia. Z hľadiska adaptívnych umelých agentov sú medzi najrozšírenejšie oblasti aplikácie evolučných techník robotika a herný priemysel. Dôvodom sú pozitívne výsledky nasadenia, flexibilné možnosti aplikácie riešení a jednoduché testovanie vyvinutých riešení.

Primárnym cieľom práce je vytvorenie návrhu softvérového prototypu umelého agenta prispôsobujúceho svoje správanie vonkajším vplyvom pomocou špecifikovanej evolučnej techniky. Správanie agenta je opísané zadefinovanou gramatikou, ktorá sa prostredníctvom gramatickej evolúcie v kombinácii s optimalizáciou rojom častíc vyvíja a umožňuje tak vytvárať lepšie a komplexnejšie riešenia agentov.

Druhá časť práce obsahuje postup implementácie robotického tanku pre programovaciu hru Robocode. V hre súťažia proti sebe umelé agenty, pričom ich cieľom je získať čo najviac víťazstiev v súbojoch s inými agentmi. Týmto spôsobom je možné vo fáze testovania kvantifikovať kvalitu riešenia na základe porovnania výsledkov s existujúcimi riešeniami a tým pádom overiť jeho efektívnosť a použiteľnosť.

2 Analýza problému

2.1 Opis problémovej oblasti

Pri tvorbe inteligentného agenta je nutné vykonať dôkladnú analýzu vlastností cieľového správania agenta a pravidiel, ktoré musia byť pri hľadaní dodržané. Následne vyvstáva otázka ako vhodne navrhnúť postup hľadania cieľového správania s tým zámerom, aby realizoval zadané ciele správne a efektívne. V prípade, že môžeme niektorý z aspektov v hľadaní optimálneho riešenia vylepšiť, tak je možné transformovať úlohu na optimalizačný problém a metódu vylepšenia nazývať optimalizačnou metódou. Optimalizačný problém je následne možné zadefinovať prostredníctvom:

- cieľovej funkcie opisujúcej problém, ktorý sa snažíme minimalizovať alebo maximalizovať,
- množiny parametrov vstupujúcich do problému,
- množiny ohraničení problému.

Pomocou týchto vlastností sme schopní vytvoriť optimalizačnú metódu, ktorá prehľadáva stavový priestor so zámerom nájsť optimálne riešenie z kandidátskych (aktuálne nájdených) riešení na základe priradenia hodnôt z definičného oboru cieľovej funkcie do oboru hodnôt tak, aby boli všetky ohraničenia problému splnené a funkcia minimalizovaná. Optimalizačné problémy je možné triediť na základe:

- počtu premenných ovplyvňujúcich cieľovú funkciu,
- typu hodnôt premenných (diskrétne, kombinatoriálne...),
- priebehom cieľovej funkcie (lineárny, kvadratický, nelineárny...),
- typu ohraničení problému,
- počtu lokálnych a globálnych optím,
- počtu optimalizačných kritérií (počtu a typu cieľových funkcií).

Pri vytváraní inteligentného agenta musíme zvoliť vhodný návrh optimalizačnej metódy vzhľadom na optimalizáciu hľadania vhodného správania agenta v komplexnom stavovom priestore kandidátskych riešení, pričom treba brať do úvahy dynamický charakter úlohy, ktorý predurčuje agenta k tomu, aby bol schopný prispôsobovať svoje správanie aktuálnemu stavu prostredia.

2.2 Evolučné výpočtové techniky

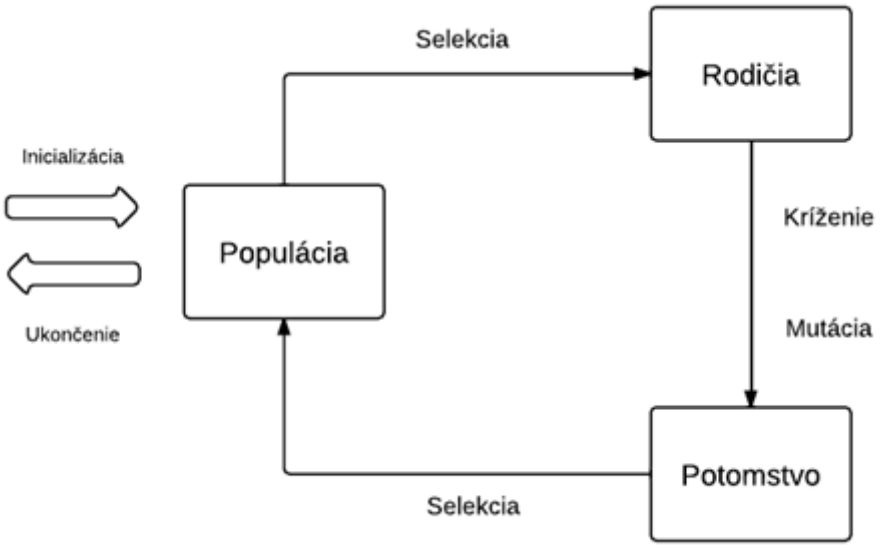
Evolučné výpočtové techniky (EC - Evolutionary Computation) sú techniky, ktoré sú založené na iteratívnom opakovaní metódy hľadania optimálneho riešenia, ktorej cieľom je zlepšovanie aktuálneho riešenia prostredníctvom výberu z populácie kandidátskych riešení. Populácia je vytváraná pomocou stochastických výberových metód so zámerom nájdenia optimálneho riešenia. Procesy hľadania sú často inšpirované biologickými procesmi evolúcie, pretože evolúcia dokáže produkovať vysoko optimálne výsledky.

2.2.1 Evolučné algoritmy

Evolučné algoritmy (EA - Evolutionary Algorithms) zahrňujú veľký počet techník, ktoré majú rovnaký základ v biologickej evolúcii, ale odlišujú sa v implementácii. Patria medzi ne genetické algoritmy, genetické programovanie, evolučné programovanie, evolučné stratégie a ďalšie.

Princíp činnosti [1] evolučných algoritmov (Obr. 1) spočíva v práci s populáciou kandidátskych riešení, ktoré sa evolvovaním vylepšujú za účelom nájdenia optimálneho riešenia. Algoritmus začína so vstupnou populáciou jedincov, ktorí predstavujú kandidátske riešenia. Kvalita riešení je vypočítaná pomocou zvolenej fitnes funkcie, ktorá kvantifikuje úroveň jednotlivých riešení. Na jej základe je následne realizovaný výber najlepších riešení. Tieto riešenia budú slúžiť ako rodičovské jedince pre ďalšiu generáciu prostredníctvom zvolenej metódy selekcie. Medzi najpoužívanejšie metódy selekcie patrí:

- náhodná selekcia - jedinci sú vyberaní bez ohľadu na fitnes,
- proporcionálna selekcia (ruleta) - pravdepodobnosť výberu jedinca je priamoúmerná vzhľadom na jeho fitnes,
- selekcia turnajom - jedinci sú náhodne zoskupený do turnajov určitej veľkosti, v ktorých sú vybrané najlepšie riešenia,
- selekcia na základe poradia – jedinci sú vyberaní na základe poradia určeného veľkosťou fitnes,
- elitarizmus – najlepších jedincov ponechávame v populácii nepozmenených.



Obr. 1. Model princípu práce evolučných algoritmov.

Pri tvorbe nového potomstva v jednotlivých generáciách sa používajú dva postupy odvodené z biologických mechanizmov:

1. kríženie - krížením dvoch a viacerých rodičov vznikne nový potomok,
2. mutácia - zmena aplikovaná na nových potomkoch.

Kombinovanie metódy selekcie a operátorov kríženia a mutácie vedie k postupnému zlepšovaniu fitnes v jednotlivých generáciách. Evolúcia môže byť v tomto prípade označená ako proces adaptácie vzhľadom na fitnes funkciu. Nutné je spomenúť aj stochastický charakter evolučných procesov, ktorý spočíva najmä pri vytváraní nových riešení pomocou kríženia a mutácie, keďže ich princíp spočíva na náhodných procesoch. Pri selekcii je to podobné, avšak popri stochastických metódach selekcie je možné použiť aj deterministické metódy.

2.2.2 Gramatická evolúcia

Gramatická evolúcia (GE – Grammatical Evolution) je jeden z najmladších prístupov evolučných výpočtových techník, ktorý je inšpirovaný molekulárnou biológiou a formálnymi gramatikami. Cieľ GE spočíva vo vylepšení gramatík, ktoré sa používajú v genetickom programovaní za účelom riešiť problémy v dynamických prostrediach.

GE je modulárna technika, ktorá má flexibilné možnosti použitia pretože umožňuje aplikovať rôzne vyhľadávacie stratégie a meniť správanie pri zmene zadefinovanej

gramatiky. Explicitne zadefinovaná gramatika umožňuje pomocou GE ľahko generovať riešenia v iných jazykoch. GE bola úspešne použitá pri generovaní riešení pre viaceré jazyky, ako napríklad Java, C/C++, Lisp a iné.

2.2.2.1 Dynamické problémy

Dynamické problémy môžu byť zadefinované [2] prostredníctvom porovnania rozdielov medzi postupom hľadania riešenia v statických a dynamických problémov. V statických problémoch evolvujeme populáciu riešení, až kým nie sú uspokojené všetky ukončujúce kritéria hľadania. Na druhej strane, v dynamických problémoch sa môžu meniť vlastnosti prostredia alebo fitnes funkcie v prebiehajúcom čase, čo vedie k dynamickým zmenám kritérií hľadania. Práve variabilný časový charakter je pozorovaný v mnohých reálnych problémov. Pre GE sa v tomto prípade otvárajú rozsiahle možnosti použitia.

Pri riešení dynamických problémov je nutné si vybrať jeden z dvoch prístupov ako riešiť problematiku súvisiacu so zmenou kritérií hľadania. Prvou je úplná výmena populácie po zaznamenanej zmene a druhou je prispôsobenie aktuálnej populácie tejto zmene. Ak nie je zmena kritická, tak je vhodnejšie prispôsobovať populáciu, pretože sa jedná o efektívnejší spôsob. Vytvorené riešenia sú takýmto spôsobom oveľa robustnejšie ako v prípade statických problémov.

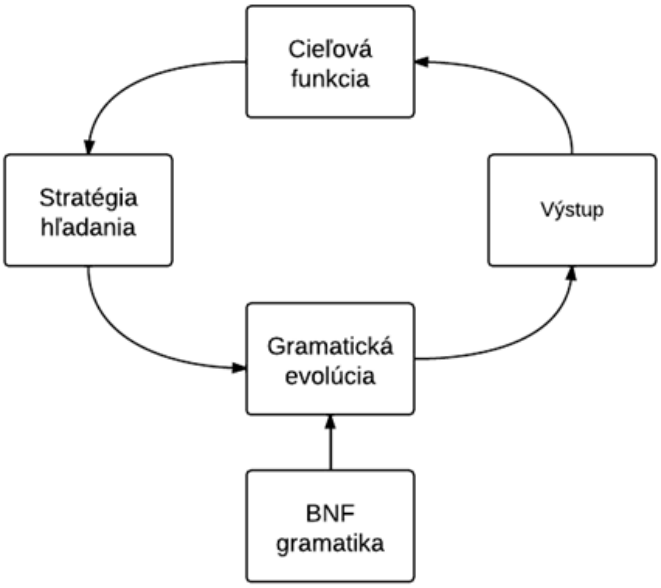
Ďalším problémom je rozsiahle množstvo druhov dynamických problémov, ktoré môžu mať rozličné ohraničenia a zmena v čase môže byť vykonávaná akýmkoľvek spôsobom. V prípade viacerých dimenzií, ktoré ovplyvňujú hľadané riešenie sa tak cieľom použitého algoritmu stáva sledovanie optimálneho stavu na úkor hľadania najlepších riešení problému v každom stave. Spôsobuje to, že hľadáme riešenie, ktoré musí mať stabilný charakter pri zmene podmienok. Finálna fitnes v tomto prípade nie je najvyššie zaznamenaná počas priebehu vykonávania algoritmu, ale na jej kvantifikáciu sa musia zvoliť dômyselnejšie techniky.

2.2.2.2 Princíp činnosti

Gramatická evolúcia je podobná genetickému programovaniu v tom, že používa evolučný proces za účelom automatického generovania počítačových programov variabilnej dĺžky. Na rozdiel od genetického programovania používa populáciu genotypov, ktoré sú reprezentované binárnymi alebo číselnými reťazcami [3]. Tie sú následne transformované do fenotypu funkčného programu cez proces transformácie genotypu na fenotyp. Transformácia je realizovaná prostredníctvom použitia gramatiky založenej na Backus Naurovej Forme (BNF), ktorá špecifikuje jazyk vyprodukovaného riešenia. Zámerom procesu transformácie je

oddelenie priestoru hľadania a priestoru riešení. Genotypy sú vyvíjané bez znalosti ekvivalentného fenotypu.

Modularita GE umožňuje ľahkú zmenu gramatiky, stratégie prehľadávania a cieľovej funkcie, ktoré môžu byť do GE zasunuté ako externé komponenty (Obr. 2). Modularita GE tak umožňuje použitie vhodnej stratégie prehľadávania na základe vlastností riešeného problému a gramatiky, ktorá sa použije na evolvovanie riešenia pomocou vlastného slovníka.



Obr. 2. Prehľad činnosti modulárneho návrhu GE.

2.2.2.3 Generovanie fenotypu

Riešenie problému prostredníctvom GE vyžaduje zadefinovanie BNF gramatiky, ktorá špecifikuje syntax fenotypov programov, ktoré budú produkované GE [2]. BNF gramatika je tvorená štvoricou:

- množinou neterminálnych symbolov N,
- množinou terminálnych symbolov T,
- množinou produkčných pravidiel P,
- počiatočným neterminálnym symbolom S.

Nasledujúca gramatika je vytvorená ako príklad pre ukážku procesu mapovania a ďalších techník gramatickej evolúcie.

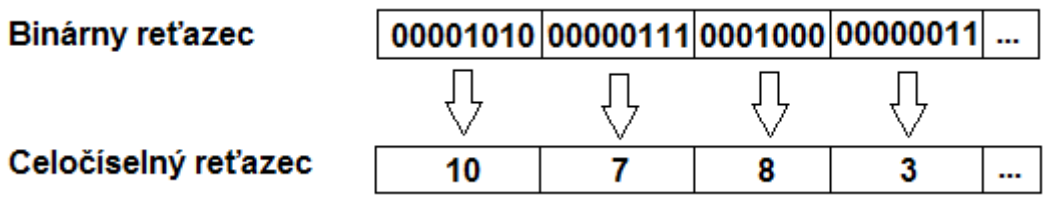
```
N = {<vyraz>, <operator>, <operand>, <premenna>}
T = {1, 2, 3, 4, +, -, /, *, x, y}
S = {<vyraz>}
```

```
P:
<vyraz>::=  <vyraz><operator><vyraz>          (0)
          | <operand>                         (1)
<operator>::=  +                              (0)
             | -                              (1)
             | /                              (2)
             | *                              (3)
<operand>::=   1                              (0)
             | 2                              (1)
             | 3                              (2)
             | 4                              (3)
             | <premenna>                     (4)
<premenna>::=  x                              (0)
             | y                              (1)
```

Gramatická evolúcia aplikuje pre vstupnú gramatiku výraz:

```
Pravidlo = c % r,
```

ktorý vyberie jednu z možností produkčných pravidiel P pre konkrétny neterminálny symbol z N. Parameter *c* je genetický kód a *r* je počet dostupných produkčných pravidiel, ktoré sú prístupné pre aktuálny neterminálny symbol. Obr. 3 a Tab. 1 opisujú jeden z možných scenárov procesu transformácie. Prvým krokom je prevod binárneho reťazca na celočíselný reťazec, pričom je použitých 8 bitov pre zakódovanie genetického kódu. Čísla v tomto reťazci postupne označujú výber produkčných pravidiel z BNF gramatiky, ktoré následne transformujú prvý neterminálny symbol v aktuálnej sekvencii podľa vybraného pravidla na základe kódu v reťazci, až kým nie sú nahradené všetky neterminálne symboly a teda vytvorený program pozostávajúci zo sekvencie terminálnych symbolov.



Obr.3. Ukážka reprezentácie genotypu.

Tab.1. Ukážka transformácie genotypu na fenotyp.

Kód	Selekcia	Pravidlo	Stav sekvencie
-	-	-	<vyraz>
10	10 % 2 = 0	<vyraz><operator><vyraz>	<vyraz><operator><vyraz>
7	7 % 2 = 1	<operand>	<operand><operator><vyraz>
8	8 % 5 = 3	4	4<operator><vyraz>
3	3 % 4 = 3	*	4*<vyraz>
4	4 % 2 = 0	<vyraz><operator><vyraz>	4*<vyraz><operator><vyraz>
7	7 % 2 = 1	<operand>	4*<operand><operator><vyraz>
9	9 % 5 = 4	<premenna>	4*<premenna><operator><vyraz>
5	5 % 2 = 1	y	4*y<operator><vyraz>
12	12 % 4 = 0	+	4*y+<vyraz>
1	1 % 2 = 1	<operand>	4*y+<operand>
10	10 % 5 = 0	1	**4*y+1**

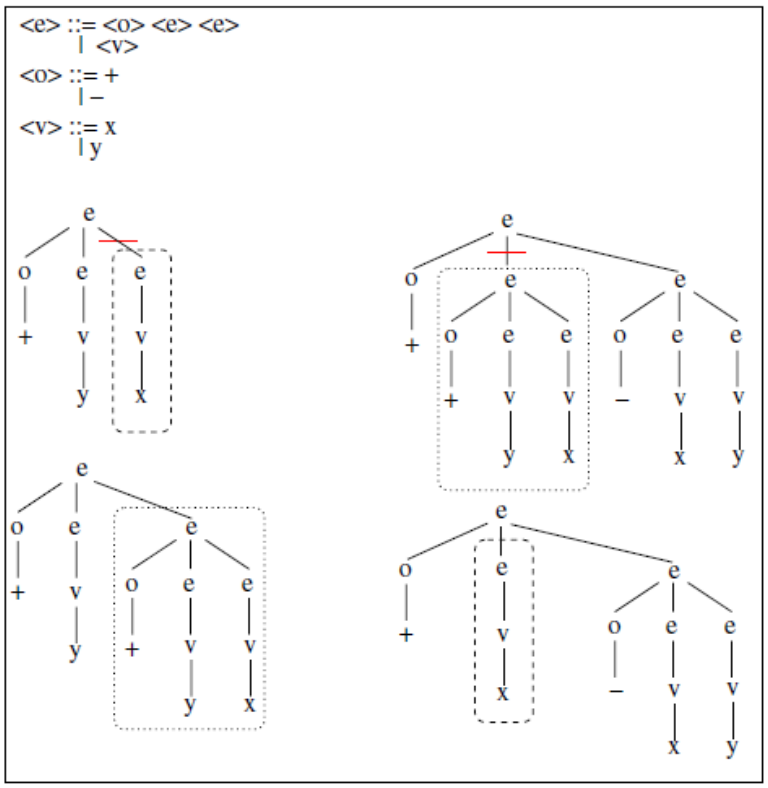
Proces transformácie môže byť ukončený jedným z troch spôsobov:

1. Kompletný program je vytvorený ešte pred ukončením sekvencie genotypu.
2. Pri predčasnom konci sekvencie genotypu sa pokračuje znova od začiatku sekvencie genotypu, pričom transformácia musí byť ukončená pod zadefinovaný maximálny prah, ktorý predstavuje najväčší možný počet transformácií.
3. Ak sa prekročí povolený limit transformácií pravidiel a v sekvencií sa stále nachádzajú neterminálne symboly, tak je transformácia pozastavená a jedinec je ocenený najhoršou fitnes.

2.2.2.4 Mutácia a kríženie

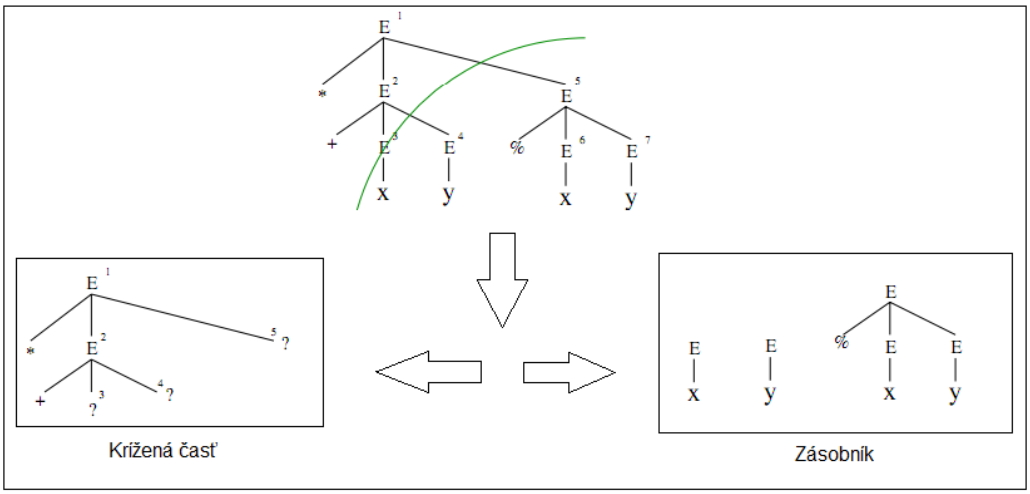
Mutácia je realizovaná prostredníctvom výmeny bitu v bitovom reťazci alebo hodnoty v celočíselným reťazcom na inú náhodnú hodnotu. Keďže mutácia sa vykonáva na genotype, tak môže nastať prípad, že po vykonaní mutácie bude jej efekt neutrálny. Neutrálna mutácia sa prejavuje na strane fenotypu, keď sa dva rôzne genotypy transformujú do rovnakého fenotypu.

Kríženie je vždy jednobodové a realizuje sa medzi dvoma sekciami genetického kódu dvoch rodičov [4], ktorý je zobrazený na Obr. 4. Pre ilustráciu je postup transformácie znázornený ako strom, v ktorom sú prepojené jednotlivé zmeny symbolov.



Obr.4. Aplikácia jednobodového kríženia.

Problém kríženia spočíva v efekte nenaplnenia (angl. ripple effect), ktorý je možné sledovať na strane fenotypu po označení časti genetického kódu, ktorá sa bude krížiť. V tomto kroku sa môže kontext genetického materiálu jedného rodiča líšiť od kontextu druhého rodiča. Riešením problému je vytvorenie dodatočného zásobníka, ktorý si bude pamätať orezané vetvy stromu [5]. V prípade nutnosti sa použijú vetvy zo zásobníku na doplnenie neurčeného kontextu nového jedinca, ktorý vznikol po krížení (Obr. 5).



Obr. 5. Ukážka stavu kríženia pri predchádzaní efektu nenaplnenia.

2.3 Rojová inteligencia

V kontexte hľadania riešenia v stavovom priestore je možné použiť techniku hľadania na základe rojov. Rojom (angl. swarm) môžeme označiť spolupracujúcu skupinu jedincov, ktorí sa snažia nájsť riešenie na určitý problém. Detailnejšia definícia je označenie roja ako dynamickej skupiny agentov, ktorá medzi sebou komunikuje priamou alebo nepriamou cestou, pričom ich konanie je obmedzené lokálnym prostredím, v ktorom sa aktuálne nachádzajú [6]. Pod lokálnym prostredím rozumieme špecifickú časť stavového priestoru riešenia problému vzhľadom na možný globálny priestor riešenia problému.

Interakcie medzi agentmi vedú k rôznym stratégiám distribuovaných a kolektívnych riešení problémov. Riešenie problémov prostredníctvom konkrétnych stratégií môžeme v kontexte roja označiť ako rojovú inteligenciu (SI - Swarm Intelligence), ktorá je podmienená interakciami medzi jedincami roja. Podstatu inteligencie roja je možné sledovať v situáciách, ktorých cieľom je priniesť úžitok všetkým jeho jedincom na základe ich vzájomnej kooperácie, pričom jedinci nepoznajú globálny stav prostredia, v ktorom sa nachádzajú. Jedinci sa snažia konať v rámci skupinovej interakcie so zámerom splnenia ich globálneho cieľa prostredníctvom výmeny lokálne dostupných informácií, ktoré sú postupne posielané naprieč celou skupinou, čo v konečnom dôsledku znamená, že efektivita hľadania riešenia problému je vyššia pri hľadaní viacerých kooperatívnych jednotlivcov ako keby ho mal riešiť jedinec sám. Na základe tohto princípu je možné SI označiť aj ako kolektívnu inteligenciu.

SI dala podnet vzniku novým výpočtovým metódam a technikám [7], ktoré sú označované pod pojmom výpočtová rojová inteligencia (CSI - Computational Swarm Intelligence). CSI je postavená na základe znalostí z SI a ústi do konkrétnych typov algoritmických riešení pre komplexné problémy. Hlavnou inšpiráciou výpočtových modelov sú rojové systémy nachádzajúce sa v prírode, ako napríklad mraveniská, včelie úle, húfy rýb a kŕdle vtákov. V týchto biologických systémoch majú jedinci relatívne jednoduché správanie, ale z globálneho hľadiska je ich kolektívne správanie veľmi komplexné. Správanie jedincov je definované trojicou pravidiel:

1. Vyhýbanie sa kolízii – cieľom je vyhnúť sa kolízii s blízkymi jedincami v roji.
2. Prispôsobovanie rýchlosti – cieľom je prispôsobiť rýchlosť pohybu susedným jedincom.
3. Zoskupovanie – každý jedinec sa snaží udržať čo najbližšie pri susedných jedincoch.

Komplexné správanie sa vyvíja v čase na základe interakcií medzi jedincami. Výsledné správanie roja nie je vlastníctvom žiadneho jedinca a zvyčajne sa nedá predurčiť alebo vydedukovať zo základného správania jedincov, ale je možné ho označiť ako kolektívne správanie podnietené sociálnym aspektom jedincov, ktoré formuje a ovplyvňuje roj. Na druhej strane, je možné sledovať cyklický vplyv medzi správaním roju a jedincami. Roj ovplyvňuje podmienky, podľa ktorých jedinci konajú a jedinci spätne vykonávajú akcie, ktoré ovplyvňujú roj. Na základe tohto faktu je nutné si uvedomiť, že interakcia a kooperácia je základným činiteľom SI.

Interakcia medzi jedincami udržiava skúsenosť a znalosť získanú z okolitého prostredia, čo zaručuje dynamické prispôsobovanie sa novým podmienkam. Interakciu je v biologických rojových systémoch možné rozdeliť do dvoch typov:

- priama interakcia – fyzický, audiovizuálny alebo iný percepčný kontakt,
- nepriama interakcia – lokálne zmeny prostredia.

Základnými princípmi SI sú [8]:

1. Princíp blízkosti (proximity) – skupina jedincov by mala byť schopná vykonávať jednoduché priestorové a časové výpočty, ktoré predstavujú priame vykonanie aktivity vychádzajúce zo správania, pričom zámer je maximalizácia úžitku pre skupinu.
2. Kvalitatívny princíp – skupina jedincov by mala byť schopná reagovať na faktory v prostredí.
3. Princíp rôznorodých reakcií – skupina jedincov by nemala odpovedať na aktivity vždy rovnakými spôsobmi.
4. Princíp stability – skupina jedincov by nemala meniť svoje správanie za každým, keď sa zmení prostredie.
5. Princíp adaptability – skupina jedincov musí byť schopná zmeniť svoje správanie, keď je výsledok zmeny pozitívny.

Nasledujúca kapitola sa venuje jednej z najrozšírenejších techník, a to optimalizácii rojom častíc, ktorá vychádza z rojovej inteligencie. Obsah kapitoly je spracovaný podľa [7], ktorý pokrýva všetky aspekty tejto techniky.

2.3.1 Optimalizácia rojom častíc

Optimalizácia rojom častíc (PSO - Particle Swarm Optimization) je postavená na základe sociálno-psychologického modelu vplyvu a učenia sa. Jedinci v roji častíc majú zadefinované

jednoduché správanie – opakovať správanie najúspešnejšieho jedinca spomedzi susedných jedincov.

Algoritmus PSO pracuje s rojom častíc, z ktorých každá častica predstavuje možné riešenie. V analógii s evolučnými algoritmami je možné povedať, že roj predstavuje populáciu a častice jedincov. Častice prechádzajú viacrozmerným priestorom hľadania, kde pozícia každej častice je vypočítaná na základe vlastnej skúsenosti a skúseností susedných častíc.

PSO sa drží princípov rojovej inteligencie, ktoré sú zadefinované v kapitole 2.2.2. Princíp blízkosti je adresovaný viacrozmernými priestorovými výpočtami, ktoré sú vykonávané v niekoľkých krokoch. Roj reaguje na faktor kvality prostredia najlepšou osobnou pozíciou jedinca a najlepšou pozíciou susedov. Prostredníctvom výmeny odpovedí medzi najlepšími pozíciami s inými susedmi je zachovaný princíp rôznorodých reakcií. Princíp stability je tvorený zmenou stavu v prípadoch vylepšenia najlepšej lokálnej alebo globálnej pozície. Adaptívny princíp sa preukazuje pri podmienenej zmene stavu na základe najlepšej lokálnej a globálnej pozície.

2.3.1.1 Model PSO

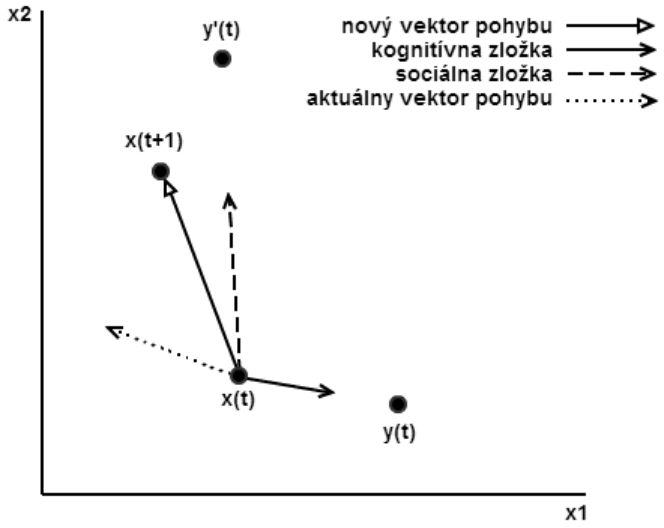
Pozícia častice je pre každú iteráciu výpočtu podmienená jej rýchlosťou $v_i(t)$ a aktuálnou pozíciou $x_i(t)$. Výpočet novej pozície $x_i(t+1)$ je realizovaný nasledovne:

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2.1}$$

kde vektor rýchlosti zabezpečuje proces optimalizácie a reflektuje znalosti získané na základe vlastnej a cudzej skúsenosti (Obr. 6). Vlastná znalosť častice sa označuje ako kognitívna zložka a cudzia znalosť získaná od inej častice ako sociálna zložka rovnice rýchlosti.

Stav a správanie častice môžeme definovať prostredníctvom:

1. vektoru rýchlosti $v_i(t)$ – ktorý predstavuje vnútorný stav častice v aktuálnom čase,
2. kognitívnej zložky – ktorá tvorí základ tendencie častice vracať sa k miestam, ktoré ich uspokojovali najviac,
3. sociálnej zložky – ktorá predstavuje spoločenskú normu a pravidlá, ktoré sa snažia častice udržať.

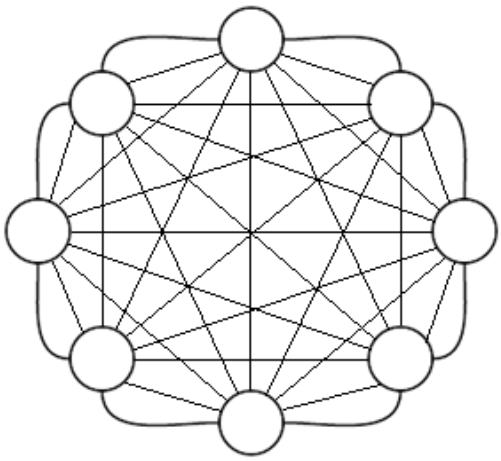


Obr. 6. Ukážka vplyvu zložiek a aktuálneho stavu na vektor pohybu častice v dvojdimenzionálnom priestore.

Model PSO je rozdelený na dva hlavné algoritmy Global Best PSO (gbest PSO) a Local Best PSO (lbest PSO), ktoré sú závislé na kognitívnom a sociálnom komponente vektora rýchlosti. Postup činnosti algoritmov je veľmi podobný, ale líšia sa v topológii sociálnej sieti častíc a v rovniciach použitých na výpočet rýchlosti a najlepších pozícii.

2.3.1.2 Global Best PSO

V gbest PSO susedí každá častica s ostatnými časticami v roji. Topológiu vzťahov v tejto sieti častíc je možné vykresliť prostredníctvom hviezdicovej topológie (Obr. 7), pre ktorú platí, že častice si aktualizujú sociálnu zložku na základe najlepšej pozície v roji. Keďže prepojenosť častíc je úplná, tak dôsledkom je rýchlejšia konvergencia k riešeniu na úkor zníženej rozmanitosti riešení.



Obr. 7. Hviezdicová topológia siete častíc v gbest PSO.

V tomto prípade je výpočet rýchlosti častice v nasledujúcom kroku realizovaný nasledovne:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_1(t)\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_2(t)\left[\hat{y}_j(t) - x_{ij}(t)\right] \quad (2.2)$$

kde $v_{ij}(t)$ je rýchlosť častice i pre dimenzie $j = 1, \dots, n$ v čase t, $x_{ij}(t)$ je pozícia častice i v dimenzii j v čase t, c_1 a c_2 sú pozitívne akceleračné konštanty určujúce veľkosť príspevku kognitívneho a sociálneho komponentu a $r_{1j}(t), r_{2j}(t) \sim U(0,1)$ sú náhodné hodnoty v rozsahu [0,1] získané z rovnomerného rozdelenia. Tieto náhodné hodnoty reprezentujú stochastickú časť algoritmu.

Najlepšia pozícia častice získaná na základe vlastnej skúsenosti y_i pre časticu i je najlepšia navštívená pozícia od začiatku algoritmu. Pri riešení problému maximalizácie úžitku vypočítame najlepšiu pozíciu v ďalšom kroku nasledovne:

$$y_{ij}(t+1) = \begin{cases} y_i(t), & ak\ f\left(x_i(t+1)\right) \le f(y_i(t)) \\ x_i(t+1), & ak\ f\left(x_i(t+1)\right) > f(y_i(t)) \end{cases} \quad (2.3)$$

kde f je fitnes funkcia. Podobne ako v evolučných algoritmoch, tak aj v PSO je kvalita častice (resp. kvalita riešenia) kvantifikovaná fitnes funkciou.

Globálne najlepšia pozícia v roji je vybratá ako maximálna hodnota spomedzi všetkých častíc v roji:

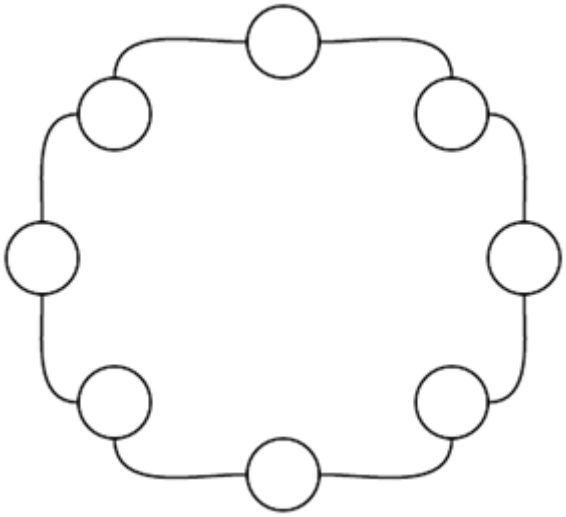
$$\hat{y}(t) = \max\{f(x_0(t), \dots, f(x_n(t))\} \quad (2.4)$$

Pseudokód algoritmu gbest PSO pre roj R:

```
repeat
    for each castica i = 1, ..., R.length do
        if f(R.x_i) > f(R.y_i) then
            R.y_i = R.x_i
        end
        if f(R.y_i) > f(R.ŷ) then
            R.ŷ = R.y_i
        end
    end
    for each castica i = 1, ..., R.length do
        vypocitaj rychlost pomocou rovnice 2.2
        vypocitaj poziciu pomocou rovnice 2.1
    end
until podmienka ukoncenia
```

2.3.1.3 Local Best PSO

Pre lbest PSO platí, že okolia susedov každej častice sú menšie ako v gbest PSO. Topológia siete vzťahov medzi časticami je prstencová (Obr. 8). Na rozdiel od gbest PSO je lbest PSO menej prepojený, čo spôsobuje zvýšenú rozmanitosť riešení. Väčšia rozmanitosť znižuje šancu na uviaznutie v lokálnom minime, avšak na úkor pomalšej konvergencie k riešeniu.



Obr. 8. Prstencová topológia siete častíc v lbest PSO.

Rýchlosť je vypočítaná podobne ako v prípade gbest PSO:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_1(t)\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_2(t)\left[\hat{y}_{ij}(t) - x_{ij}(t)\right] \quad (2.5)$$

s tým rozdielom, že $\hat{y}_{ij}$ určuje najlepšiu pozíciu, ktorá bola nájdená medzi susednými časticami i pre dimenziu j. Lokálne najlepšia pozícia $\hat{y}_i$ je teda najlepšia pozícia nájdená v množine susedov častice.

Dôležitou súčasťou metódy lbest PSO je dodatočné pridelenie informácie o susedoch pre každú časticu, pretože častice túto informáciu nevlastnia. Hlavným dôvodom je abstrahovanie výpočtu vzdialeností medzi každou dvojicou častíc za účelom zistenia susedského vzťahu. Druhým dôvodom je postupné rozšírenie najlepších riešení k všetkým časticiam bez ohľadu na ich aktuálnu pozíciu v priestore hľadania.

Pseudokód algoritmu lbest PSO pre roj R:

```
repeat
    for each castica i = 1, …, R.length do
        if f(R.x_i) > f(R.y_i) then
            R.y_i = R.x_i
        end
        if f(R.y_i) > f(R.ŷ_i) then
            R.ŷ_i = R.y_i
        end
    end
    for each castica i = 1, …, R.length do
        vypocitaj rychlost pomocou rovnice 2.5
        vypocitaj poziciu pomocou rovnice 2.1
    end
until podmienka ukoncenia
```

2.3.1.4 Nastavenia a parametre

Dôležitým krokom vykonávania algoritmu je inicializácia počiatočných pozícií častíc. Efektivita PSO je silne ovplyvnená práve inicializáciou, preto je odporúčané, aby boli častice rovnomerne rozprestreté v priestore hľadania. Ak sú niektoré regióny priestoru hľadania nepokryté už v inicializácii, tak je veľmi nízka pravdepodobnosť, že PSO tieto regióny odhalí. Rovnomerné rozdelenie počiatočných pozícií častíc po celom priestore hľadania je možné určiť pomocou:

$$x(t=0) = x_{min,j} + r_j\left(x_{max,j} - x_{min,j}\right), \forall j = 1, \dots, n \tag{2.6}$$

kde x_{max} a x_{min} sú ohraničenia konkrétnej dimenzie, n je počet dimenzií priestoru a $r_j \sim U(0,1)$ sú náhodné hodnoty z rozsahu [0,1] z rovnomerného rozdelenia. Počiatočný vektor pohybu musí byť pri inicializácii nenulový, aby častice dokázali preskúmať čo najviac priestoru bez vplyvu jednotlivých zložiek.

Ďalším dôležitým nastavením PSO je podmienka ukončenia, pri ktorej má algoritmus ukončiť svoje vykonávanie. Pri výbere spôsobu ukončenia je nutné vziať do úvahy fakt, aby ukončovacia podmienka nezastavila PSO predčasne, keď sa podarí nájsť suboptimálne riešenie a aby predchádzala pretrénovaniu.

V reálnych aplikáciách boli použité nasledovné typy ukončení:

- ukončenie po dosiahnutí zadaného počtu iterácií – používa sa v konjunkcii s iným typom ukončenia alebo pri meraní hľadania pre konkrétnej časové dĺžky,

- ukončenie po nájdení akceptovateľného riešenia – akceptovateľné riešenie je definované výškou prahu, ktorá po prekročení ukončí vykonávanie algoritmu,
- ukončenie pri nezlepšovaní riešenia v určitom časovom úseku – je nutná parametrizácia dĺžky časového úseku a prahu priemernej odchýlky, ktorá musí byť prekonaná, aby algoritmus pokračoval,
- ukončenie pri neakceptovateľnom priemere priestoru hľadania – výpočet je realizovaný pomerom aktuálneho priemeru roja (najväčšia vzdialenosť akejkoľvek pozície častice od najlepšej globálnej pozície) k iniciálnemu priemeru roja v priestore, pričom je nutné nastaviť veľkosť hodnoty prahu, pri ktorom nastáva ukončenie,
- ukončenie pri neakceptovateľnom stúpaní cieľovej funkcie – výpočet je realizovaný prostredníctvom najlepších hodnôt v predchádzajúcom a aktuálnom kroku, pričom ukončenie spočíva v neprekročení prahu, ktorý predstavuje zlepšovanie riešenia v čase.

Základným krokom k riešeniu problémov prostredníctvom PSO je nastavenie parametrov, ktoré s nimi súvisia. Rôzne variácie modelov PSO môžu vlastniť špecifické parametre, avšak v základnom PSO modeli sú zadefinované nasledovné parametre:

- veľkosť roja – počet častíc v roji,
- veľkosť susedného okolia – priamoúmerne ovplyvňuje počet interakcií, ktoré sú vykonávané medzi časticami, pričom menšie susedné okolia sú menej náchylné na uviaznutie v lokálnom minime,
- počet iterácií – súvisí s ukončovacou podmienkou,
- akceleračné koeficienty – nastavenie vplyvu kognitívnej a sociálnej zložky pri výpočte pohybu častíc,
- stochastické parametre – náhodné vplyvy zložiek častice.

2.3.1.5 Modifikácie

I keď sa základný model PSO použil pri riešení klasických optimalizačných problémov, tak jeho problém spočíva v nekonzistencii pri konvergovaní k dobrým riešeniam. Z tohto dôvodu sú vytvorené modifikácie základného modelu za účelom zrýchlenia konvergencie a skvalitnenia nájdených riešení.

Jeden z aspektov, ktorý určuje efektivitu a správnosť optimalizačného algoritmu je vzťah medzi prieskumom a vykorisťovaním hľadania. Prieskum je schopnosť algoritmu hľadať cez

rôzne oblasti priestoru hľadania za účelom nájsť optimálne riešenie. Vykorisťovanie určuje schopnosť koncentrovať hľadanie v okolí sľubných oblastí za účelom zlepšenia kandidátskych riešení. Dobrý optimalizačný algoritmus vhodne synchronizuje tieto protichodné ciele. V PSO sa pre tento účel používa modifikácia usmernenia pohybu (velocity clamping), ktorého spôsobuje obmedzenie pohybu častíc pomocou ohraničení rýchlosti pohybu na základe:

$$v_{ij}(t+1) = \begin{cases} v'_{ij}(t), & ak\ v'_{ij}(t+1) < V_{max,j} \\ V_{max,j}, & ak\ v'_{ij}(t+1) \geq V_{max,j} \end{cases} \quad (2.7)$$

kde $V_{max,j}$ predstavuje maximálne ohraničenie rýchlosti častice v dimenzii j. Rýchlosť je teda obmedzená podobne ako pozícia častice vo vzorci 2.3. Problémom tejto modifikácie je optimálne nastavenie ohraničenia, ktoré by nepodnecovalo pomalý alebo rýchly pohyb, resp. nezvýhodňovalo prieskum alebo vykorisťovanie.

Ako alternatíva pre modifikáciu usmernenia pohybu sa používa modifikácia váhou zotrvačnosti (inertia weight). Váha je dodatočný parameter, ktorý vlastní každá častica. Predstavuje veľkosť s akou sa bude pohybovať (resp. zotrvávať) častica v aktuálnom vektore pohybu. V dôsledku dopĺňa vzorec 2.2 nasledovným spôsobom:

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1(t)\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_2(t)\left[\hat{y}_j(t) - x_{ij}(t)\right] \quad (2.8)$$

kde w reprezentuje váhu zotrvačnosti. Podobne ako pri predchádzajúcej modifikácii, tak aj tu je problém s nastavením optimálnej váhy, ktorá by nezvýhodňovala prieskum priestoru hľadania pred vykorisťovaním alebo naopak. Časté je použitie dynamickej zmeny veľkosti váhy v čase. Dynamická zmena môže byť realizovaná niekoľkými spôsobmi:

- náhodné nastavenie váhy v každej iterácii, napr. prostredníctvom Gaussovho rozdelenia,
- lineárne klesajúcou hodnotou váhy, ktorá začína vo vysokej počiatočnej hodnote a lineárne klesá do minimálnej hodnoty váhy (správanie podobné ako pri simulovanom žíhaní),
- nelineárne klesajúcou hodnotu váhy, ktorá za začína vo vysokej počiatočnej hodnote a rýchlosť klesania je podmienená pridaním dodatočných konštánt (správanie podobné ako pri simulovanom žíhaní),
- váha adaptujúca sa na základe fuzzy logiky, ktorá definuje konkrétne pravidlá (napr. s využitím hodnôt fitnes a iných), ktoré určujú aktuálny stav váhy,
- rastúca hodnota váhy správajúca sa opačne ako pri klesajúcich hodnotách.

2.3.1.6 Porovnanie PSO a EC

Všeobecne je PSO považovaná za techniku spadajúcu do EC, pretože s nimi zdieľa niekoľko významných čŕt. Avšak kvôli značným rozdielom je PSO v tejto práci separovaná od evolučných výpočtových techník a zaradená do rojovej inteligencie. Rozdiely a podobnosti sú opísané v nasledujúcich odsekoch.

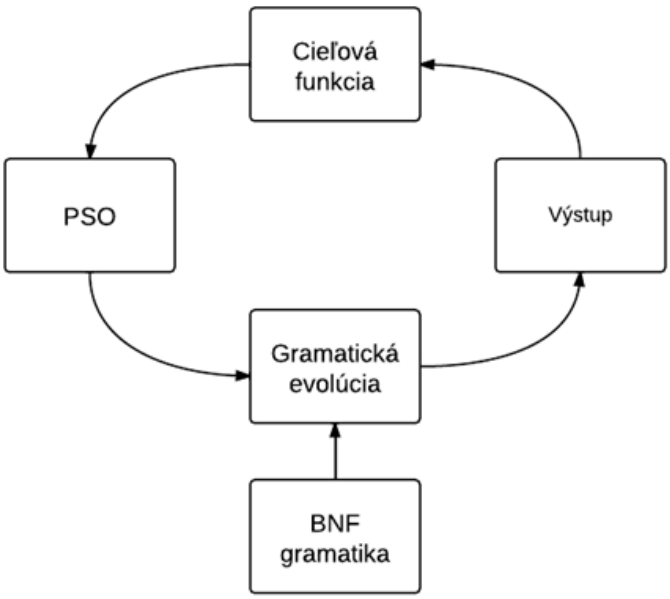
Oba prístupy sú založené na stochastickej optimalizácii, ktorá vychádza z populácie riešení. Riešenia sú získavané transformáciou populácií, ktoré sú inšpirované prírodnými fenoménmi s využitím špecifických operátorov. Hlavným rozdielom medzi PSO a EC je v tom, že zatiaľ čo v EA jedinci prehľadávajú priestor sami, tak v PSO je prehľadávanie založené na priamych sociálnych interakciách s inými jedincami, ktorí už vlastnia určitú znalosť prostredia. Ďalším dôležitým rozdielom je, že PSO má pamäť predchádzajúcich dobrých riešení, ktoré ovplyvňujú následný pohyb v priestore hľadania.

Nevýhodou PSO je z hľadiska reprezentácie nutnosť vytvorenia reprezentácie vlastností jedincov prostredníctvom vektorov s pohyblivou desatinnou čiarkou, keďže pohyb nie je diskrétny, ale spojitý. Oba prístupy používajú na kvantifikovanie kvality nájdených riešení fitnes funkciu. Hlavný rozdiel je v tom, že v EA je hľadanie priamo podmienené aktuálnou hodnotu fitnes funkcie, zatiaľ čo v PSO je hľadanie podmienené vlastnou skúsenosťou a skúsenosťou susedov. Fitnes funkcia je v PSO použitá len na kvantifikovanie kvality riešení, nie na transformáciu populácie, ako v prípade EA.

Mutácia v EA je rovnocenná so stochastickými parametrami r_1 a r_2, ktoré sú vložené do výpočtu nových pozícií častíc v PSO. Avšak kríženie nie je rovnocenné so žiadnou technikou v PSO, i keď do určitej miery je možné považovať pohyb častíc ku globálne najlepšej pozícii za takúto techniku. V konečnom dôsledku je možné tvrdiť, že PSO má slabý selektívny mechanizmus, zatiaľ čo cieľom EA je selekcia najlepších riešení.

2.4 Gramatický roj

Gramatický roj je technika, ktorá prepája optimalizáciu rojom častíc (PSO) s gramatickou evolúciou (GE) [9]. Cieľom gramatického roja je získať z oboch techník to najlepšie pri riešení dynamických problémov (Obr. 9). Pri kombinácii GE s PSO je nutné vyriešiť niekoľko problémov realizácie výpočtu, pretože kontext jednotlivých techník to nedovoľuje.



Obr. 9. Princíp činnosti gramatického roja.

Prvý problém súvisí s reprezentáciou jedinca, ktorý v gramatickej evolúcii predstavuje binárny, resp. celočíselný reťazec pevnej dĺžky. Keďže PSO hľadá riešenie v spojitom priestore, tak je nutné vykonávať zaokrúhlenie hodnoty pozície častice po každej vykonanej iterácii. Z použitej gramatiky je následne možné vytvoriť riešenie programu na základe zaokrúhlených celočíselných hodnôt.

Ďalší problém súvisí taktiež s reprezentáciou jedinca. Jedná sa o maximálnu celočíselnú hodnotu, ktorú môže jedinec v reťazci vlastniť. Maximálna hodnota musí byť použitá pri prehľadávaní priestoru pomocou PSO ako maximálna hodnota pre pozíciu a rýchlosť, ktorú môže častica nadobúdať, aby sa predchádzalo nežiaducim problémom s prepočítavaním nepovolených hodnôt, resp. pretekaním hodnôt do nepovolenej množiny hodnôt.

Implicitným problémom GE, ktorý pretrváva aj v gramatickom roji je voľba podmienky ukončenia transformácie. Ideálny spôsob je nastaviť maximálny prah transformácie, ktorý predstavuje zadefinovanie počtu možných prekladov neterminálnych znakov v sekvencii, ktoré sa vykonávajú pri transformácií genotypu na fenotyp. Problém má dopad na správanie PSO, pretože môžu nastať situácie, keď sa prekročí prah počtu transformácií a riešenie sa stane neakceptovateľným (dostane najnižšiu fitnes). To môže vytvoriť nevhodný región priestoru hľadania, pričom nie nutne všetky riešenia v tomto regióne môžu byť neakceptovateľné. Ideálne by bol stav, keby sa dokázali všetky riešenia preložiť do funkčného programu.

Spojenie PSO s GE nesie so sebou neriešiteľný problém. Jedná sa o absenciu kríženia medzi dvoma jedincami. Aj keď princíp funkčnosti PSO môže so sebou niesť znaky kríženia, tak sa nejedná o plnohodnotné nahradenia procesu kríženia. Dôsledkom absencie kríženia je nutné zadefinovanie fixnej dĺžky reprezentácie vektorov v PSO, pretože variabilitu dĺžky nie je možné dosiahnuť. Prehľadávaný priestor v PSO musí mať fixný počet dimenzií, aby nenastali problémy s konzistenciou priestoru prehľadávania a taktiež neexistuje spôsob ako implicitne meniť dĺžku bez procesu kríženia na strane GE.

PSO, ktorý je použitý v gramatickom roji je modifikáciou gbest PSO z Kap. 2.3.1.5. Prvá modifikácia spočíva v použití váh zotrvačností, pričom sa jedná o použitie metódy s lineárnym klesaním hodnoty váhy podľa nasledujúceho výpočtu:

$$w = w_{max} - \frac{w_{max} - w_{min}}{iter_{max}} * iter \tag{2.9}$$

kde w_{max} a w_{min} sú počiatočná a minimálna hodnota váhy nadobudnutá v poslednej iterácii, $iter_{max}$je počet iterácii algoritmu a $iter$ je aktuálna iterácia. Druhá modifikácia je usmernenie pohybu, ktorá určuje maximálnu rýchlosť akú môže častica nadobudnúť.

Dôležitým krokom aplikácie gramatického roja je parametrizácia, ktorá súvisí s voľbou hodnôt jednotlivých parametrov za účelom zabezpečenia optimálneho správania sa algoritmu. Medzi parametre patria:

- počet dimenzií prehľadávaného priestoru v PSO,
- hodnoty x_{min} a x_{max}, ktoré ohraničujú pohyb v dimenzii, pričom x_{max} by mal zodpovedať maximálnej hodnote, ktorú môže nadobudnúť jedinec v genotype,
- maximálna možná hodnota rýchlosti V_{max} v PSO, ktorá by mala byť rovná hodnote x_{max},
- akceleračné koeficienty PSO c_1 a c_2,
- stochastické parametre PSO r_1 a r_2,
- počiatočnú a minimálnu hodnotu váhy zotrvačnosti w_{max} a w_{min},
- počet iterácií algoritmu $iter_{max}$.

2.5 Robocode

Robocode je multiplatformová open-source programovacia hra, ktorej cieľom je simulácia bojov robotických tankov v bojovej aréne. Ľudský hráč je programátor, ktorý naprogramuje správanie robota, pričom na prebiehajúce zápasy nemá vplyv. Cieľom hry je vytvorenie robota s umelou inteligenciou, ktorá bude schopná reagovať na udalosti vyskytujúce sa v súbojoch s inými robotmi za účelom porazenia protivníka.

Robocode beží na platforme Java a roboti sú písaný v tomto programovacom jazyku. Súboje sú simulované prostredníctvom vstavaného grafického prostredia. Hráči si môžu svojich robotov otestovať proti základnej sade robotov, ktorí využívajú konkrétne typy jednoduchých stratégií. Finálny výsledok je určený na základe počtu víťazstiev.

Dôležitou možnosťou odskúšania robotov spočíva v online ligách (Tab. 2), kde hráči môžu poslať svojich robotov a tí sú testovaní proti iným robotom. Pozícia je určená na základe skóre, ktoré sa určí z výsledkov proti jednotlivým robotom, pričom výpočet skóre je realizovaný na princípe šachových turnajov.

Tab. 2. Online ligy.

Názov ligy	Opis ligy
MegaBots RoboRumble	1 proti 1 bez obmedzení
MiniBots RoboRumble	1 proti 1 s dĺžkou programov menej ako 1500 znakov
MicroBots RoboRumble	1 proti 1 s dĺžkou programov menej ako 750 znakov
NanoBots RoboRumble	1 proti 1 s dĺžkou programov menej ako 250 znakov
MegaBots MeleeRumble	10 tankov proti sebe bez obmedzení
MiniBots MeleeRumble	10 tankov s dĺžkou programov menej ako 1500 znakov
MicroBots MeleeRumble	10 tankov s dĺžkou programov menej ako 750 znakov
NanoBots MeleeRumble	10 tankov s dĺžkou programov menej ako 250 znakov
TeamRumble	2 tímy proti sebe vo veľkostí 2 tankov bez obmedzení

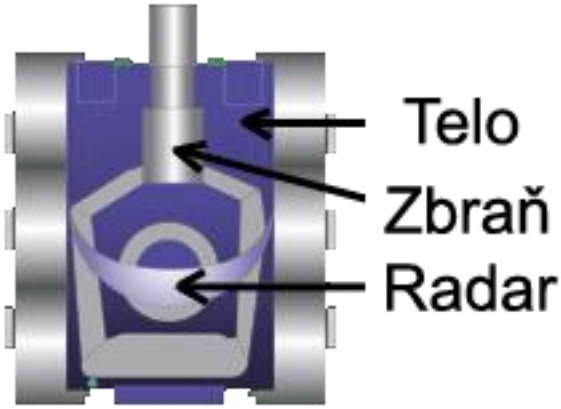
2.5.1 Robot

Robot je napísaný ako udalosťami riadený Java program. Princíp činnosti spočíva v cyklickom opakovaní hlavného tela programu robota kontrolujúce správanie tanku, ktoré môže byť prerušené udalosťami posielanými zo simulačného prostredia [10]. Program robota obsahuje špecifické funkcie, ktoré sú volané pri zachytení prislúchajúcich udalostí, na ktoré sú zaregistrované. Tab. 3 obsahuje zoznam všetkých udalostí, ktoré môžu byť zaznamenané.

Tab. 3. Zoznam udalostí.

Názov udalosti	Opis udalosti
onBulletHit	zasiahnutie nepriateľa
onBulletHitBullet	vzájomné zasiahnutie striel
onBulletMissed	zasiahnutie steny bojovej arény
onHitByBullet	zasiahnutie nepriateľom
onHitRobot	kolízia s tankom
onHitWall	kolízia so stenou
onScannedRobot	spozorovanie robota radarom

Robot sa skladá z 3 častí (Obr. 10) a jeho správanie môže byť definované akciami, ktoré sú v Tab. 4. Akčné členy, ktoré zabezpečujú vykonávanie akcií sú pohyb tanku, rotácia tela tanku, rotácia zbrane, rotácia radaru a streľba.



Obr. 10. Anatómia robotického tanku.

Tab. 4. Zoznam akcií robotického tanku.

Názov akcie s parametrom	Opis akcie
ahead(double)	pohyb vpred o zadanú vzdialenosť
back(double)	pohyb vzad o zadanú vzdialenosť
doNothing()	vynechanie akcie (preskočenie kroku simulácie)
fire(double)	streľba so zadanou silou
resume()	povolenie zastaveného pohybu
setAdjustGunForRobotTurn(bool)	natavenie závislosti pohybu zbrane od tela tanku
setAdjustRadarForGunTurn(bool)	nastavenie závislosti pohybu radaru od zbrane
setAdjustRadarForRobotTurn(bool)	nastavenie závislosti pohybu radaru od tela tanku
stop()	zastavenie pohybu agenta
turnGunLeft()	otočenie zbrane doľava
turnGunRight()	otočenie zbrane doprava
turnLeft()	otočenia tela tanku doľava
turnRadarLeft()	otočenie radaru doľava
turnRadarRight()	otočenie radaru doprava
turnRight()	otočenie tela tanku doprava

Výsledok zápasu je vypočítaný na základe počtu súbojov, ktoré sa vykonávajú v rade. Súboj je ukončený, keď zostane v aréne len jeden tank. Tanky sú zničené, ak úroveň energie tanku klesne na nulu. Energia môže počas súboja stúpať a klesať, čo závisí od nasledujúcich pravidiel:

- tank získa energiu, ak zasiahne strelou iný tank,
- tank stratí energiu, ak je zasiahnutý strelou,
- tank stratí energiu, ak je v kolízii s iným tankom alebo stenou,
- tank stratí energiu, ak vykoná streľbu, pričom úroveň straty je proporcionálna úrovne sily strely.

Streľba je obmedzená zahrievaním hlavne, ktorej teplota rastie proporcionálne silou streľby. Teplota časom klesá a streľba je umožnená, až keď má teplota nulovú hodnotu.

Počas hry je možné získavať informácie prostredníctvom indikátorov stavu tanku. Tie sa dajú následne použiť pri vykonávaní programu robota a zvýšiť tak komplexnosť jeho správania. Zoznam indikátorov je opísaný v Tab. 5.

Tab. 5. Zoznam indikátorov tanku.

Názov indikátoru	Opis indikátoru
battlefieldHeight	výška arény
battlefieldWidth	šírka arény
energy	aktuálna energia tanku
gunHeading	smer natočenia zbrane
gunHeat	teplota hlavne
heading	smer natočenia tela tanku
radarHeading	natočenie radaru
time	čas kola
velocity	aktuálna rýchlosť tanku
x	horizontálna pozícia tanku v aréne
y	vertikálna pozícia tanku v aréne

2.6 Analýza existujúcich riešení

Robocode obsahuje vlastný repozitár, do ktorého umiestňujú autori svojich robotov. Veľké množstvo z nich obsahuje zdrojový kód programu, vlastnosti stratégií a techník použitých pri tvorbe, dosiahnuté výsledky a ďalšie dodatočné informácie. Pre účely práce bolo cieľom nájsť robotov, ktorí vznikli prostredníctvom automatického generovania programu robota evolučnými výpočtovými technikami. Ako podklad pre odborne zdokumentované riešenia

v nasledujúcich kapitolách boli použité informácie z práce [11], ktorá opisuje prvé pokusy automatickej tvorby programov robotov.

2.6.1 GP-Bot

GP-Bot bol vytvorený za účelom porovnania stratégií získaných evolučnými výpočtovými technikami proti stratégiám, ktoré boli vytvorené manuálne [12]. GP-Bot sa úspešne zúčastnil ligy pre kategóriu HaikuBots, ktorá bola zameraná na vytvorenie robota, v ktorom sú súboje realizované jeden proti jednému, pričom program robota je obmedzený na 4 riadky kódu. Dôvod voľby kategórie tohto typu je generovanie riešení genetickým programovaním, ktoré produkuje dlhé riadky kódu. Na základe tohto faktu boli ostatné typy kategórii zamietnuté ako nevyhovujúce.

Princíp evolúcie robota spočíva v použití populácie jedincov, ktorí sú reprezentovaní pomocou LISP výrazov pozostávajúcich z funkcií a terminálnych znakov. Použitými funkciami sú logické a aritmetické výrazy, ktoré majú vstupné parametre a vracajú numerickú hodnotu. Terminálne znaky pozostávajú z matematických funkcií, hodnôt stavu robota získaných z indikátorov a konštánt. Evolúcia riešenia spočíva v použití genetického programovania.

Pri vývoji boli použité rôzne konfigurácie evolúcie, ako napríklad STGP (Strongly Typed Genetic Programming), kde sa typy vstupných parametrov a terminálnych znakov líšia a ADF (Automatically Define Functions), ktorá podporuje evolúciu podmnožín riešení. V konečnom dôsledku sa tieto techniky vynechali, pretože ich efekt nebol použiteľný pre hru Robocode.

Architektúra programu musela byť prispôsobená obmedzeniam veľkosti programu robota, takže sa muselo upustiť od niektorých funkcionalít robota. Ako prvé sa vynechalo používanie rotácie radaru, ktorý sa otáčal zároveň s hlavňou tanku. Druhou výnimkou bola streľba, ktorá bola zadefinovaná ako numerická konštanta vyskytujúca sa kdekoľvek v kóde. Hlavný cyklus programu obsahoval jeden riadok kódu, ktorý otáčal hlavňou tanku v jednom smere za účelom zaznamenania protivníka radarom. Zvyšné tri riadky kontrolovali jeden akčný člen, ktorého vstup bol získaný evolúciou. Štruktúra programu vyzerala nasledovne:

```
while(true){
  turnGunRight(value); // hodnota otocenia je maximalna mozna
}

onScannedRobot(){
  moveTank(<GP#1>); // posunutie tanku smerom k evolvovanej hodnote
  turnRight(<GP#2>); // otocenie tanku smerom k evolvovanej hodnote
  turnGunRight(<GP#3>); // otocenie hlavne smerom k evolvovanej hodnote
}
```

Fitnes jednotlivých riešení bola vypočítaná na základe dvoch prístupov:

1. Výber protivníkov bol realizovaný zo sady existujúcich tankov, pričom pre testovanie každého jedinca v populácii boli z tejto sady náhodne vybraté 3 tanky, na ktorých sa realizovalo testovanie.
2. Výpočet fitnes funkcie bol realizované na základe výpočtu aký sa používal v turnaji: $F = \frac{S_P}{S_P+S_A}$, kde F je výsledné pomerové skóre, S_P je skóre získané robotom a S_A náhodne vybraným protivníkom. Konečná hodnota fitnes je určená priemernou hodnotou získaného skóre so všetkými protivníkmi.

Keďže výpočet fitnes evolučného algoritmu so zadefinovanými parametrami podľa Tab. 6. bol výpočtovo náročný, tak sa použilo distribuované počítanie fitnes hodnôt medzi viac ako 20 počítačov. Výber najlepšieho riešenia po ukončení evolučného algoritmu spomedzi všetkých riešení spočíval v ich otestovaní proti skupine 12 rozličných robotov. Na základe výsledkov sa vybralo najvhodnejšie riešenie, ktoré sa označilo ako GP-Bot.

Tab. 6. Parametre evolučného algoritmu.

Parameter	Hodnota
populácia	256 jedincov
podmienka ukončenia	zastavenie pri nezlepšovaní fitnes funkcie
mutácia	zmena podstromu v uzle alebo liste
kríženie	výmena podstromov dvoch riešení
selekcia	turnaj o veľkosti 5
elitizmus	2 najlepší jedinci postupovali bez modifikácie

Výsledkom GP-Bota v súťaži HaikuBots bolo 3. miesto, pričom programy ostatných robotov boli manuálne vytvorené. Z všeobecného hľadiska je GP-Bot úspešný, pretože:

- je konkurencie schopný v súboji s manuálne vytvorenými robotmi,
- učenie GP-Bota je podmienené súbojmi s inými typmi robotov,
- umožňuje rozšírenie správania pre iné kategórie turnajov.

2.6.2 Koevolúcia robotov s využitím SCALP

SCALP (Spatial Co-Evolution in Age Layered Plans) je používaný ako evolučné prostredie za účelom podpory koevolúcie. SCALP pozostáva z niekoľkých oddelených vrstiev v priestore

a využíva koncept starnutia v jednotlivých vrstvách. SCALP vrstva pozostáva z mriežky prepojených uzlov so 4 priamo susediacimi a 4 diagonálne susediacimi uzlami [13].

V každom uzle spoločne žije hostiteľ a parazit. Pre každú generáciu je v každom uzle realizovaný boj medzi parazitom so susediacimi uzlami, v ktorých dostávajú obaja rôzne skóre. Myšlienka tejto implementácie spočíva v tom, že prostredníctvom koevolúcie môže predátor dobehnúť niektorú z koristí. Na druhej strane, pre korisť je menej podstatné, že dokáže predbehnúť viacej predátorov ako to, že dokáže predbehnúť viacej koristí.

Skóre hostiteľa je vypočítané prostredníctvom súčtu skóre z bojov proti parazitom, pričom parazit získava skóre na základe jeho najhoršieho boja. Po vyhodnotení každého uzla je možné pre hostiteľov a parazitov preniesť potomka do nového uzla v ďalšej generácii. 40% uzlov v novej generácii sú potomkami, ktorí majú jedného z rodičov na rovnakom mieste v predchádzajúcej generácii a ďalší je náhodne vybraný susediaci rodič, 10-20% riešení je zmutovaných.

Pre SCALP pozostávajúceho z vrstiev platí, že hostiteľ je testovaný aj proti parazitom, ktorý sa nachádzajú v uzle na nižšej vrstve a jeho susedným uzlom. Každá vrstva má určitý generačný limit, ktorý sa zvyšuje vyššími vrstvami. To v konečnom dôsledku spôsobuje, že nižšie vrstvy slúžia ako dodatočný genetický materiál pre riešenia na vyšších vrstvách. Na základe Robocode bola koevolúcia realizovaná spôsobom, že paraziti predstavovali manuálne napísané programy robotov a hostitelia boli automaticky vytvorené riešenia, ktoré im mali konkurovať.

Výsledkom koevolúcie s využitím SCALP bolo vybranie niekoľkých robotov s najlepším fitnes v jednotlivých generáciách za účelom otestovania ich úspešnosti bojovať so základnou množinou robotov. Roboti boli úspešní v súboji s väčšinou robotov s úspešnosťou nad 70%, pričom jediný problém bol s robotom Walls (jeho stratégia spočíva v pohybe blízkosti stien), ktorého nedokázali úspešne poraziť. Proti robotom, ktorí sú umiestnení v druhej polovici najlepšej tisícke robotov sa podarilo vytvoriť robota, ktorý ich porážal s úspešnosťou nad 50%, avšak proti robotom v prvej polovici sa nepodarilo vytvoriť konkurencie schopné riešenie.

3 Opis riešenia

3.1 Špecifikácia požiadaviek

Cieľ práce sa zameriava na splnenie dvoch hlavných požiadaviek. Prvou je použitie gramatického roja ako optimalizačnej evolučnej techniky za účelom úspešného vyriešenia definovaného problému, ktorý súvisí s programovacou hrou Robocode. Druhou požiadavkou, ktorá zároveň preverí úspešnosť splnenia prvej časti, je vyvinutie úspešného agenta v hre Robocode, ktorého konkurencieschopnosť by dosahovala úroveň analyzovaných agentov vytvorených pomocou iných optimalizačných evolučných techník (Kap. 2.6).

3.2 Návrh

Kapitola návrhu je venovaná opisu návrhu riešenia, ktorý je rozdelený do jednotlivých podkapitol na základe aspektu riešených problémov. Cieľom je poskytnúť základný pohľad na koncept riešenia s použitím vybraných algoritmov, ktorý bude úspešne riešiť stanovené požiadavky finálnej práce. Návrh postupne prechádza cez všetky úrovne riešenia, od komplexných problémov až po elementárne.

3.2.1 Architektúra

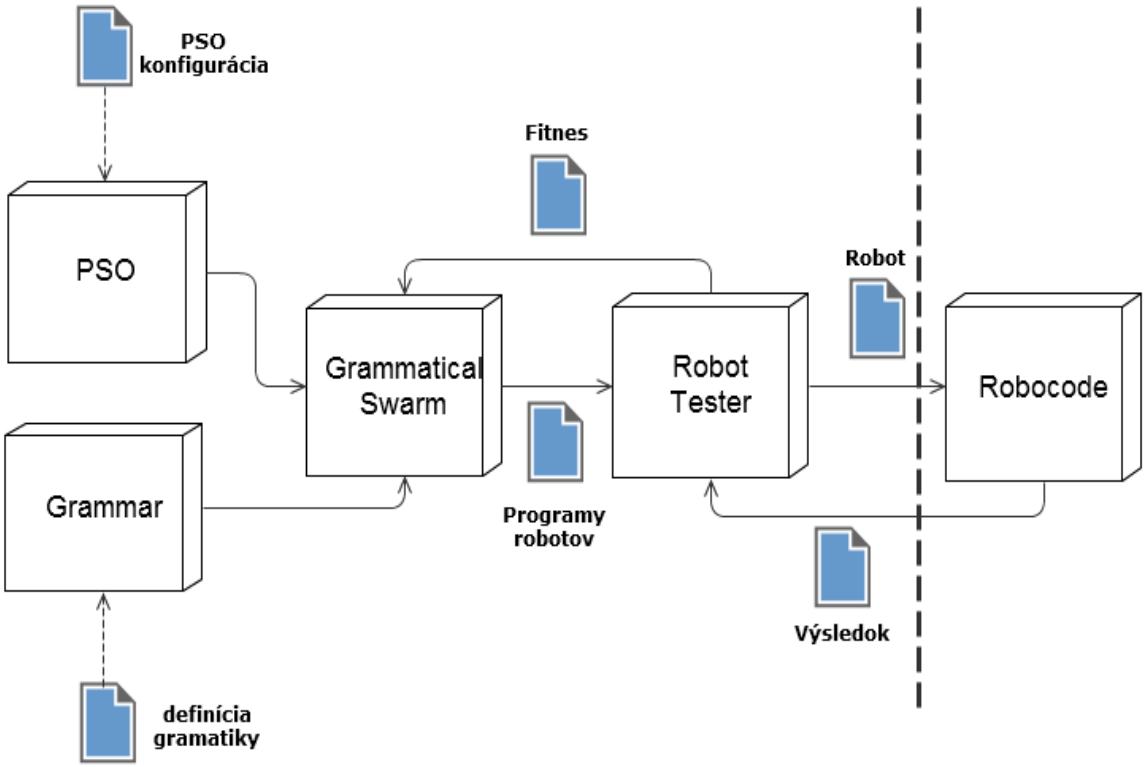
Prvým a základným problémom, ktorý je nutné vyriešiť je návrh architektúry softvérovej aplikácie riešiacej špecifikované požiadavky. Návrh architektúry musí spĺňať taktiež funkcionálne požiadavky týkajúce sa efektívnosti a rozšíriteľnosti. Efektívnosť aplikácie súvisí s rýchlosťou hľadania riešenia, keďže úroveň časovej zložitosti evolučných výpočtových techník je priamoúmerná komplexnosti riešených problémov (Kap. 3.3.1).

Pod rozšíriteľnosťou sa rozumie jednoduché pridávanie nových funkcionalít algoritmov za účelom jednoduchého doplnenia alebo úpravy určitého aspektu výpočtu (optimalizácie, experimenty). Rozšíriteľnosť je realizovaná pomocou modulárneho rozdelenia systému a definície spôsobu komunikácie medzi jednotlivými modulmi a externými systémami.

Výpočtové časti systému môžeme rozdeliť do nasledovných modulov:

1. PSO – zabezpečuje výpočet pre algoritmus optimalizácie rojom častíc,
2. Grammar – zabezpečuje prácu s gramatikou a jej definíciu,
3. Gramatical Swarm – zabezpečuje výpočet algoritmu gramatického roja,
4. Robot Tester – zabezpečuje výpočet fitnes hodnoty robotov.

Pre jednotlivé moduly je nutné definovať vstupné a výstupné údaje, ktoré si budú medzi sebou posielať. Výsledný návrh aplikácie je zobrazený na Obr. 11.



Obr. 11. Modulárny návrh finálnej aplikácie.

Na začiatku výpočtu je nutné pripraviť parametre algoritmov a definíciu gramatiky na základe informácií v príslušných konfiguračných súboroch. Po inicializácii sa spustí vykonávanie hlavného algoritmu gramatického roja, ktorý v jednotlivých iteráciách generuje programy robotov, ktoré sú následne priamo testované v súbojoch v externom prostredí hry Robocode. Výstupom je výsledok súbojov, ktorý sa použije pre výpočet fitnes hodnoty robota. Algoritmus následne pokračuje ďalšou iteráciou.

Znázornený návrh nám spĺňa obe funkcionálne požiadavky, pretože hlavný algoritmus je možné jednoducho rozšíriť, modifikovať alebo prispôsobiť novým potrebám a v testovacom module je možné vykonávať optimalizáciu časovej zložitosti výpočtu, teda je možné splniť aj požiadavku efektívnosti. V prípade potreby je taktiež možné nahradiť celý modul bez toho, aby sa muselo zasahovať do iných modulov, pričom jedinou potrebou je dodržanie vstupno-výstupného protokolu posielaných dát medzi príslušnými modulmi.

3.2.2 Robot

Pri návrhu správania robota si je nutné uvedomiť, že zložitosť správania je priamoúmerná výpočtovej zložitosti. Zložitosť správania je podmienená metódami generovania programu robotov, ktoré opisujú ich správanie v hre. To znamená, že súčasťou návrhu správania robota je výber metódy generovania finálneho programu. Pri výbere metódy generovania sme identifikovali prístupy, ktoré sú zobrazené v Tab. 7.

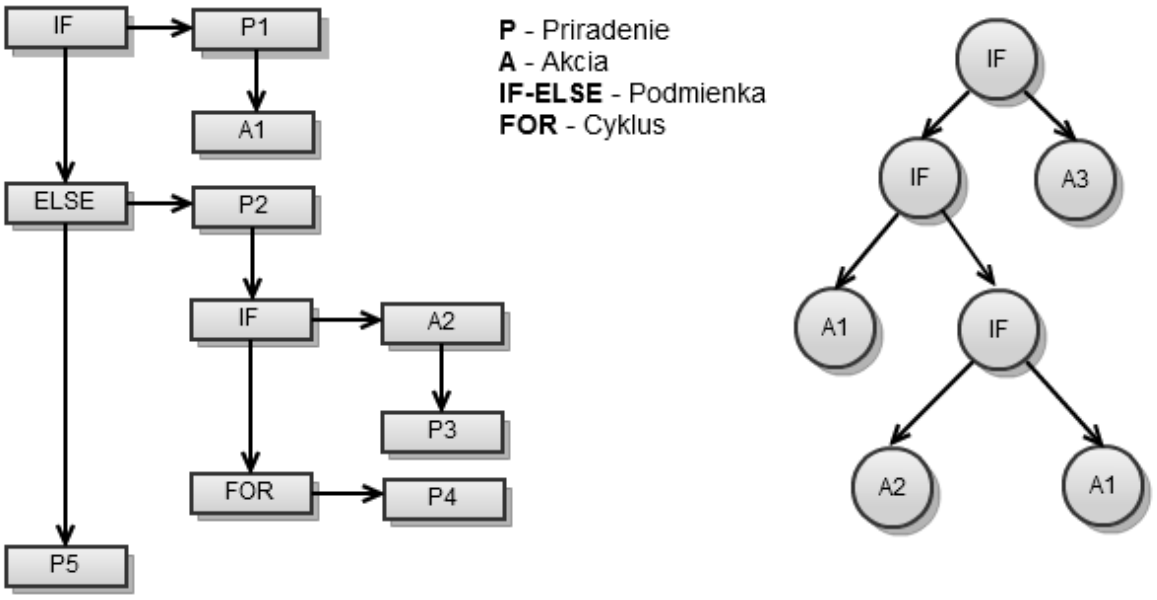
Tab. 7. Metóda generovania programu robota.

Metóda	Zložitosť správania	Výpočtová zložitosť	Informácia
úplná	vysoká	vysoká	cieľom je dosiahnuť správanie manuálne vytvorených robotov
úplná stromová	normálna	normálna	správanie je generované rozhodovacím stromom (if-else)
parciálna	vysoká	normálna (nie nutne)	generujeme len špecifikovanú časť robota
parciálna stromová	normálna	nízka	správanie je generované rozhodovacím stromom, ale je obmedzené počiatočnou štruktúrou programu robota

Manuálnu tvorbu programov robotov (resp. úplnú metódu generovania) je možné asociovať so skriptovaním umelej inteligencie v jednoduchých počítačových hrách. To znamená, že ak máme množinu povolených akcií, ktoré je možné v hre vykonať, tak výsledné správanie je možné vytvoriť rozhodovacím stromom (if-else), v ktorom uzly stromu obsahujú rozhodovacie prvky ovplyvňované stavom prostredia hry a listy povolené akcie.

Dôsledkom transformácie úplnej metódy generovania na úplnú stromovú metódu sme zmenšili stavový priestor možných riešení, tým pádom sa zníži aj výpočtová zložitosť riešenia na úkor abstrahovania častí syntaxe a logiky programov vyskytujúcich sa v programoch manuálne vytvorených robotov (vlastné stavové premenné, funkcie, cykly, objekty). Na Obr. 12 je ukážka programov vygenerovaných úplnou a úplnou stromovou metódou.

Pri riešení je potrebné zvoliť vhodnú metodológiu vývoja robotov, ktorá predstavuje taký spôsob vývoja robotov, aby sme minimalizovali chyby a čas vzniknutý pri ich vývoji. Vhodným prístupom je progresívny prístup, ktorý znamená, že vývoj je zameraný na postupnú tvorbu od jednoduchších robotov, až k zložitejším. Minimalizácia chýb je zabezpečená použitím techník pre jednoduchšie problémy, kde je ľahšie overenie ich správnosti a minimalizácia času spočíva v nižšej výpočtovej zložitosti.



Obr. 12. Model programov úplnej (vľavo) a úplnej stromovej metódy (vpravo).

Na základe progresívneho prístupu vývoja robotov je vhodné zvoliť parciálnu stromovú metódu generovania programov robotov. Výpočtová zložitosť tejto metódy je nízka, pretože stavový priestor možných riešení je ohraničený. Pri parciálnej metóde je ohraničením počiatočná štruktúra programu robota, ktorá presne definuje vyvíjané časti programu robota. To umožňuje usmernenie vývoju len na tie časti programu, ktoré chceme vylepšiť. Parciálna stromová metóda pracuje na rovnakom princípe ako parciálna metóda, ale nižšia výpočtová zložitosť je zabezpečená abstrahovaním časti syntaxe a logiky programov. V kontexte úplnej stromovej metódy to môže predstavovať vývoj len určitej vetvy stromu.

3.2.3 Gramatika

Cieľom návrhu gramatiky je vytvorenie gramatiky, ktorá by spĺňala tri hlavné požiadavky. Prvou je splnenie kritérií generovania programov robotov, ktoré zabezpečíme pomocou stromovej metódy (Kap. 3.2.2). Druhou je menšia veľkosť množiny rozvetvujúcich sa produkčných pravidiel, aby sa predišlo problému s generovaním veľmi hlbokých stromov. Treťou je vhodná voľba terminálnych akcií a neterminálnych funkcií vzhľadom na ich počet, aby sa dalo pomocou gramatiky úspešne vygenerovať komplexnejšie správanie robota. Podobne ako v prípade návrhu generovania programov robota je možné definovať viacero úrovní gramatiky na základe počtu pravidiel, terminálnych akcií a neterminálnych funkcií.

3.2.3.1 Základná gramatika

Vychádzajúc z analýzy Robocode prostredia v Kap. 2.5 a povolenej množiny akcií GP-Bota v Kap. 2.6.1 sa definovali funkcie a terminály, ktoré sa použijú ako základ pre vývoj jednoduchých robotov v BNF gramatike (Tab. 8). Generovanie programu je riešené pomocou parciálnej stromovej metódy, pričom základná štruktúra programu je rovnaká ako v prípade GP-Bota:

```
while(true){
  turnRadarLeft(value); // hodnota otocenia je maximalna mozna
}

onScannedRobot(){
  moveTank(<vyraz>); // posunutie tanku
  turnLeft(<vyraz>); // otocenie tanku
  turnGunLeft(<vyraz>); // otocenie hlavne
}
```

Tab. 8. Funkcie (neterminály) a terminály základnej gramatiky.

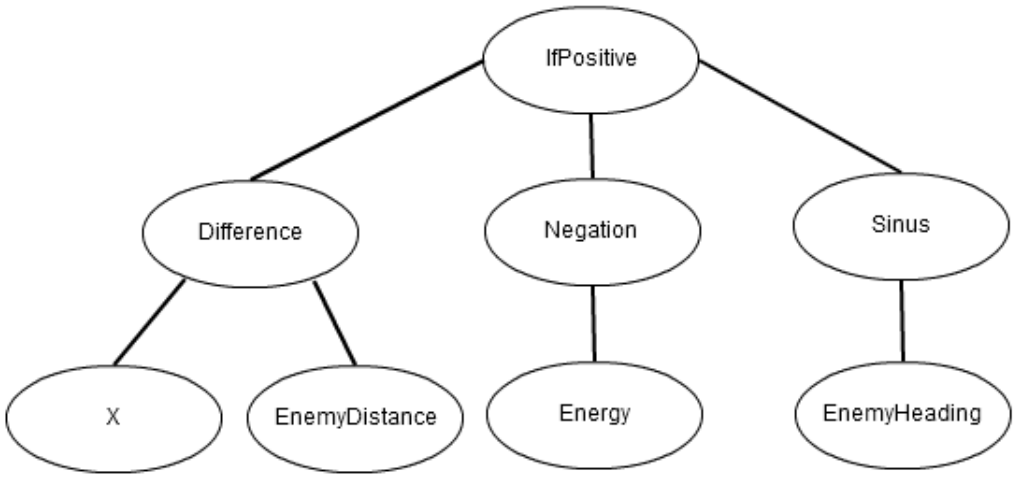
Názov	Typ	Informácia
Energy()	terminál	vráti aktuálnu energiu tanku
EnemyEnergy()	terminál	vráti energiu nepriateľského tanku
Heading()	terminál	vráti natočenie tanku
EnemyHeading()	terminál	vráti natočenie nepriateľského tanku
X()	terminál	vráti aktuálnu horizontálny polohu
Y()	terminál	vráti aktuálnu vertikálnu polohu
Width()	terminál	vráti veľkosť bojiska
EnemyDistance()	terminál	vráti vzdialenosť nepriateľského tanku
Fire(x)	funkcia	vykoná streľbu, ak je $x > 0$
IfGreater(x, y, left, right)	funkcia	ak je $x > y$, tak vráti výraz *left*, inak *right*
IfPositive(x, left, right)	funkcia	ak je $x > 0$, tak vráti výraz *left*, inak *right*
Sum(x, y)	funkcia	vráti súčet *x* a *y*
Difference(x, y)	funkcia	vráti rozdiel *x* a *y*
Abs(x)	funkcia	vráti absolútnu hodnotu *x*
Negation(x)	funkcia	vráti negáciu *x* (vynásobi hodnotu -1)
Sinus(x)	funkcia	vráti sínus *x*

Formálny zápis gramatiky vyzerá nasledovne (kvôli ukážke nie sú vyjadrené terminálne symboly definícií funkcií):

```
G = {N, T, P, S}

N = {<vyraz>, <strelba>, <if>, <if-nula>, <sucet>, <rozdiel>,
<abs>, <negacia>, <sin>}

T = {energia, e_energia, smer, e_smer, x, y, velkost,
vzdialenost}

S = {<vyraz>}
```

```
P:
<vyraz>::=  <strelba> | <if> | <if-nula> | <sucet> | <rozdiel> |
             <abs> | <negacia> | <sin> | energia | e_energia |
             smer | e_smer | x | y | velkost | vzdialenost
<strelba>::=  strelba(<vyraz>)
<if>::=  if(<vyraz>, <vyraz>, <vyraz>, <vyraz>)
<if-nula>::=  if-nula(<vyraz>, <vyraz>, <vyraz>)
<sucet>::=  sucet(<vyraz>, <vyraz>)
<rozdiel>::= rozdiel(<vyraz>, <vyraz>)
<abs>::= abs(<vyraz>)
<negacia>::= negacia(<vyraz>)
<sin>::= sin(<vyraz>)
```

Výsledná gramatika obsahuje základné funkcie, ktoré je možné použiť pri generovaní programu robota. Na Obr. 13. je ukážka jednoduchej gramatiky a stromovej metódy generovania.



Obr. 13. Ukážka vygenerovaného výrazu jednoduchej gramatiky stromovou metódou.

3.2.3.2 Rozšírená gramatika

Základnú gramatika je možné jednoducho rozšíriť o nové terminály a funkcie. V prípade rozšírenia je potrebné sledovať pomer počtu rozhodovacích a ostatných elementov, aby nenastala situácia, že rozvetvovanie stromu bude minimálne (bude nastávať len s malou pravdepodobnosťou) alebo nekonečné (bude ohodnotené nulovou fitnes).

Rozšírenie môže byť realizované pomocou:

- stochastických funkcií – náhodné generovanie čísel z intervalu,

- matematických funkcií – násobenie, delenie, modulo a iné,
- numerických konštánt – hodnoty nemenné počas generovania gramatiky,
- goniometrických funkcií – sínus, kosínus a ich inverzné funkcie,
- funkcií prostredia – rýchlosť nepriateľa, natočenie zbrane nepriateľa a iné,
- pomocných funkcií – maximálne možné hodnoty, vzdialenosť od steny a iné.

3.2.4 Optimalizácia

Návrh optimalizácie predstavuje hlavnú časť riešenia, ktorá sa skladá z vyhľadávacej metódy v stavovom priestore možných riešení a komunikácie s ostatnými časťami aplikácie. Optimalizačný algoritmus je reprezentovaný postupnosťou ucelených krokov, ktoré sú vykonávané za účelom nájdenia takých robotov, ktorý by boli konkurencieschopní v boji proti základnej sade robotov poskytnutou hrou Robocode a inými robotmi. Kapitola sa skladá z návrhu základného algoritmu gramatického roja a jeho možnými rozšíreniami.

3.2.4.1 Gramatický roj

Gramatický roj je technika, ktorá je totožná s gramatickou evolúciou, ale ako metódu prehľadávania stavového priestoru používa optimalizáciu rojom častíc (PSO). Kap. 2.4 obsahuje vysvetlenie a teoretické základy ohľadom činnosti práce gramatického roja. Nasledujúce riadky sa venujú niektorým aspektom použitia gramatického roja, od elementárnych rozhodnutí tvorby algoritmu cez parametrizáciou jeho hodnôt a vysvetlením konfigurácie jednotlivých parametrov, až po opísanie hlavných princípov činnosť algoritmu a detailne rozobratie jeho jednotlivých krokov.

Gramatický roj používa ako vyhľadávaciu funkciu PSO, ktorá môže vyhľadávať na základe globálne najlepšej pozície (gbest) alebo najlepšej hodnoty suseda (lbest). Podľa [9] sa ukázal gramatický roj úspešný už pri riešení problémov s nižším počtom častíc, pričom nižší počet častíc je predpokladom aj pri riešení nášho problému. Dôsledkom tohto tvrdenia je, že použitie lbest vyhľadávania nie sú vyhovujúce, pretože definícia veľkosti susedstva pre veľký počet dimenzií a menší počet častíc nemusí byť vhodne nastavená, čo spôsobí, že susedné častice sa nemusia ovplyvňovať a teda budú konvergovať veľmi rýchlo do lokálnych miním. Preto je odporúčané použiť gbest vyhľadávanie.

Tab. 9 obsahuje zoznam parametrov s odporúčaným intervalom hodnôt a vysvetlením, ktoré je potrebné vhodne nastaviť pred vykonávaním algoritmu. Intervaly hodnôt nemusia byť pevné, ale vyjadrujú hodnoty, ktoré boli nastavené pri riešení iných problémov týkajúcich sa klasického PSO alebo gramatického roja.

Tab. 9. Parametre gramatického roja.

Parameter	Hodnota	Vysvetlenie
veľkosť roja	30 – 100	počet častíc v roji
dĺžka vektora častice	50 – 150	počet dimenzií polohového vektora častice
dolné ohraničenie hodnôt polohového vektora	0	minimálna veľkosť hodnoty v polohovom vektore
horné ohraničenie hodnôt polohového vektora	255	maximálna veľkosť hodnoty v polohovom vektore
dolné ohraničenie hodnôt vektora rýchlosti	-32	minimálna veľkosť hodnoty vo vektore rýchlosti
horné ohraničenie hodnôt vektora rýchlosti	32	maximálna veľkosť hodnoty vo vektore rýchlosti
kognitívny koeficient	0.5 – 1	rýchlosť konvergencie častice k najlepšej lokálnej pozícii
sociálny koeficient	0.5 – 1	rýchlosť konvergencie častice k najlepšej globálnej pozícii
počiatočný koeficient zotrvačnosti	0.9 – 1	iniciálna hodnota váhy zotrvačnosti, ktorú má častica pridelenú v prvej iterácii
koncový koeficient zotrvačnosti	0.2 – 1	finálna hodnota váhy zotrvačnosti, ktorú nadobudne častice v poslednej iterácii
počet iterácií	100 – 10000	počet iterácií PSO algoritmu
maximálny počet transformácií prekladu	50 – 500	maximálna dovolená veľkosť stromu, ktorú je možné vygenerovať algoritmom

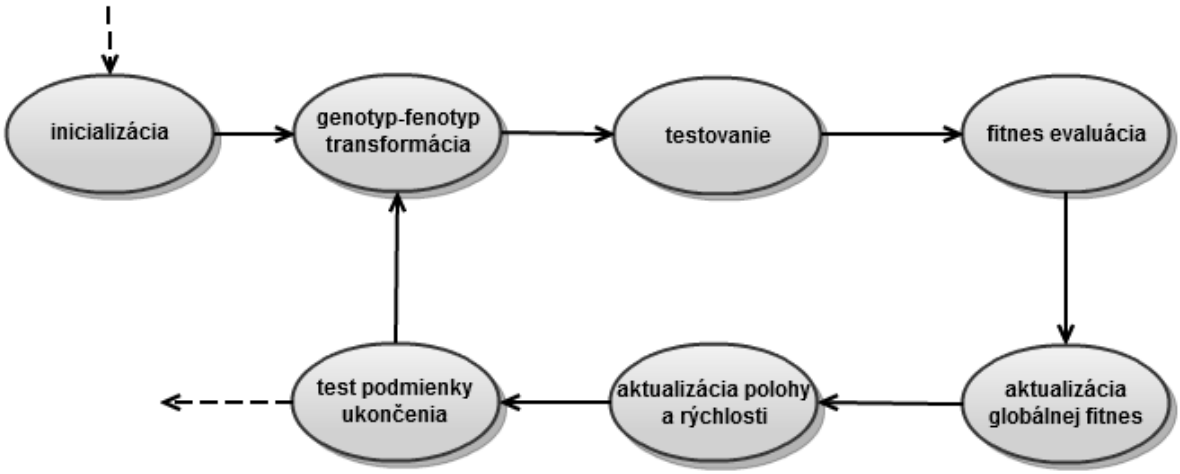
Polohový vektor častice v roji reprezentuje genotyp kandidátskeho riešenia. V gramatickom roji sa vykonáva transformácia genotypu na fenotyp prostredníctvom definovanej gramatiky. Preto je potrebné pri generovaní programu vyriešiť problém hĺbky rozvetvenia rozhodovacieho stromu, ktoré predstavuje finálne správanie kandidátskeho riešenia vo forme programu robota. Pre účely nášho riešenia je ideálne zvoliť opakované použitie vektoru častice (po spracovaní hodnoty v poslednej dimenzii sa pokračuje znova od začiatku), ktoré je obmedzené maximálnym počtom transformácií prekladu neterminálnych znakov za účelom vygenerovania akceptovateľného riešenia. V prípade prekročenia prahu sa riešenie stane neakceptovateľným a ohodnotí sa najnižšou fitnes.

Algoritmus obsahuje dve modifikácie, ktorých cieľom je vylepšenie prehľadávania priestoru možných riešení. Prvou je pridanie váhy zotrvačnosti, ktorá predstavuje veľkosť s akou sa bude zotrvávať častica v aktuálnom vektore pohybu. Druhou je ohraničenie veľkostí hodnôt vektoru rýchlosti, ktorá zabraňuje príliš rýchlemu pohybu častíc v priestore.

Keďže pri hľadaní riešenia nášho problému nepoznáme finálne riešenie a v konečnom dôsledku ich môže byť viacej, tak je algoritmus ukončený len v prípade, ak je splnená jedna z nasledujúcich podmienok:

1. počet iterácií presiahne maximálny počet iterácii,

2. riešenie sa nevyvíja (uviazne v lokálnom alebo globálnom minime).



Obr. 14. Postupnosť krokov optimalizácie gramatickým rojom.

Algoritmus optimalizácie gramatického roja je znázornený na Obr. 14 a pozostáva z nasledujúcich krokov:

1. *Inicializácia* – vygenerovanie náhodných hodnôt pre polohový vektor a vektor rýchlosti každej častice.
2. *Genotyp-fenotyp transformácia* – vytvorenie programov robotov na základe polohového vektoru častíc.
3. *Testovanie* – testovanie kvality robotov v súbojoch.
4. *Fitnes evaluácia* – určenie fitnes hodnoty každej častice na základe výsledkov testovania.
5. *Aktualizácia globálnej fitnes* – v prípade nutnosti sa určí nová globálna fitnes hodnota.
6. *Aktualizácia polohy a rýchlosti* – výpočet nového polohového vektoru a vektoru rýchlosti každej častice.
7. *Test podmienky ukončenia* – ak je podmienka splnená, tak sa ukončí vykonávanie algoritmu.

3.2.4.2 Modulárne generovanie

Modulárne generovanie (MG) je optimalizácia, ktorá rozširuje optimalizáciu gramatickým rojom. Cieľom modulárneho generovania je vývoj len určitej časti správania robota. V prípade generovania robotov pomocou parciálnej stromovej metódy ide o generovanie len

tých častí rozhodovacieho stromu, ktoré sú na začiatku algoritmu zvolené. Pre základnú gramatiku definovanú v Kap. 3.2.3.1 sa jedná o generovanie jednej z akcií robota:

1. posun tanku,
2. otočenie tanku,
3. otočenie hlavne tanku.

Základným predpokladom optimalizácie je príprava ostatných modulov programu robota, tak aby generovanie zlepšovalo kvalitu robota. To môžeme zabezpečiť prostredníctvom jednej z nasledujúcich metód, ktoré sú opísané v Tab. 10. Pre jednotlivé metódy môžeme definovať špecifický typ a formát testovania, to znamená, že môžeme priamo ovplyvňovať ciele vývoja a modelovanie správania robota.

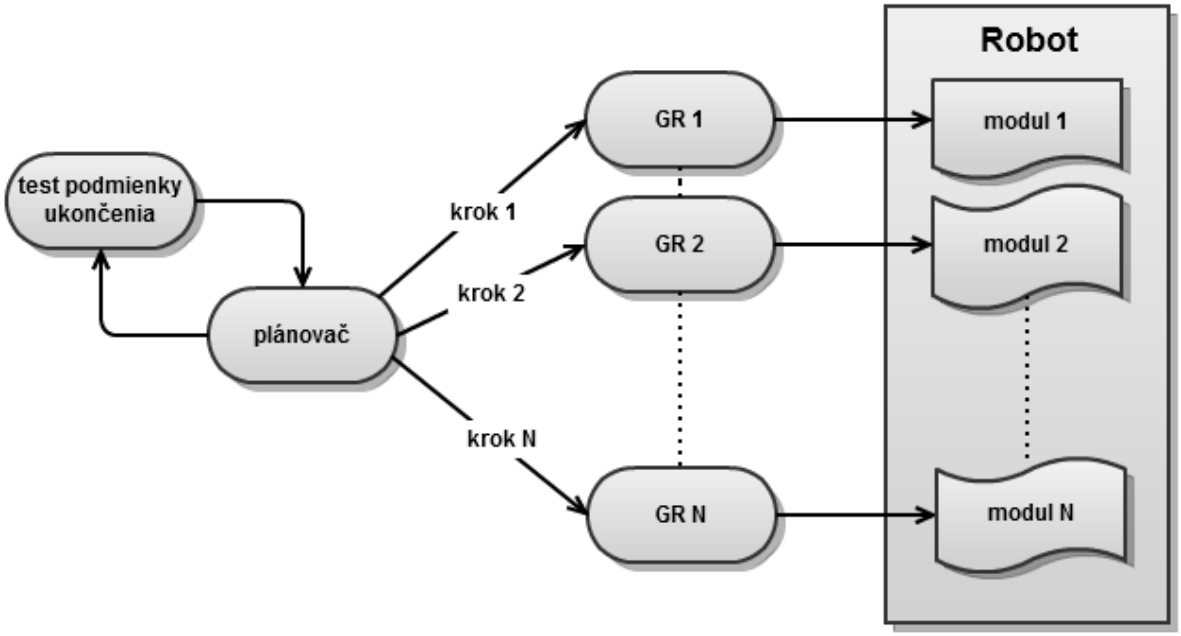
Tab. 10. Metódy MG.

Metóda	Výhoda	Nevýhoda	Princíp
cielené MG	pokračovanie vo vývoji robota	preddefinované správanie pre ostatné moduly robota	generujeme len konkrétnu časť robota, ostatné časti majú preddefinované správanie
striedavé MG	vývoj viacerých modulov robota zároveň	-	striedavo generujeme jednotlivé časti robota

3.2.4.3 Paralelné gramatické roje

Paralelné gramatické roje sú algoritmickým riešením optimalizácie pracujúcej na princípe striedavého MG. Základná myšlienka aplikácie striedavého MG je, že každý vyvíjaný modul musí byť reprezentovaný samostatným nezávislým gramatickým rojom. Algoritmus pre prácu s paralelnými gramatickými rojmi je plánovačom nad množinou gramatických rojov s tým, že jeho jedinou funkcionalitou bude prepínanie vývoja medzi jednotlivými modulmi (resp. rojmi).

Prepínanie práce medzi jednotlivými modulmi môže byť definované prostredníctvom viacerých metód. Pre účely riešenia nášho problému sme zvolili metódu postupného generovania s jednotkovým krokom (Obr. 15). Princíp tejto metódy spočíva v prepínaní vývoja modulov plánovačom, ktorý aktivuje v každom kroku iný gramatický roj. Jednotlivé gramatické roje vyvíjajú určitý modul finálneho robota a pre ostatné moduly používajú najlepšie riešenia rojov získané od začiatku behu algoritmu. Podmienka testu ukončenia algoritmu gramatických rojov je generalizovaná do plánovača.



Obr. 15. Postupné generovanie modulov s jednotkovým krokom.

3.2.5 Fitnes funkcia

Základnou súčasťou evolučných optimalizačných techník je správny návrh fitnes funkcie. V našom prípade by mala fitnes funkcia odzrkadľovať kvalitu robota, ktorú je možné definovať viacerými spôsobmi. V Tab. 11 sú uvedené jednotlivé hodnoty výsledkov súbojov poskytnutých hrou Robocode, ktoré je možné pri meraní použiť. Každá z hodnôt má definované výhody a nevýhody, s ktorými je potrebné byť oboznámený pri návrhu fitnes.

Tab. 11. Hodnoty merania fitnes funkcie.

Hodnota	Výhoda	Nevýhoda
umiestnenie (rank)	jednoduché hodnotenie	ignoruje špecifické aspekty hodnotenia
skóre (score)	komplexné hodnotenie	môže byť náchylná na špecifické chyby
percentuálne skóre	komplexné hodnotenie	môže byť náchylná na špecifické chyby
prežitie (survival)	špecifické hodnotenie	zvýhodňuje silných robotov
poškodenie (bullet damage)	špecifické hodnotenie	pri dlhých a vyrovnaných súbojoch môže byť skreslená
nárazové poškodenie (ram damage)	špecifické hodnotenie	pri dlhých a vyrovnaných súbojoch môže byť skreslená

Výber konkrétnej metódy musí byť podložený zámerom, ktorý chceme na robotoch sledovať. Ideálne je použitie kombinácie metód, aby sme minimalizovali chybu merania fitnes funkcie.

Ďalším aspektom, ktorý je nutné zahrnúť do výpočtu a testovania robotov je postupný vývoj týchto robotov, ktorý vyplýva z použitia evolučnej optimalizačnej techniky. Preto je vhodné navrhnúť také riešenie, ktoré by testovalo robotov proti dynamickej množine nepriateľských robotov na základe ich kvality. Taktiež je nutné zobrať do úvahy samotný problém, ktorý sa snažíme riešiť, pretože hľadáme konkurencieschopného robota, pričom konkurencieschopnosť môže byť definovaná viacerými spôsobmi. Tab. 12 obsahuje identifikované prístupy testovania proti množine robotov.

Tab. 12. Metódy testovania robotov.

Metóda	Princíp	Výhoda	Nevýhoda
jednoduchá	testovanie proti konkrétnemu robotovi	jednoduchosť, rýchlosť	náchylnosť na chyby, špecializácia
komplexná	testovanie proti pevnej množine robotov	komplexnosť	zložitosť
progresívna	postupné odomykanie robotov na základe kvality	postupné učenie, nižšia zložitosť	definícia kvality
dynamická	množina robotov sa mení v čase	overenie komplexnosti riešenia	náhodný charakter konvergencie riešení
cielená	testovanie proti množine robotov za účelom vyvinúť konkrétnu časť riešenia	vhodné pre MG	definícia dosiahnutia cieľov

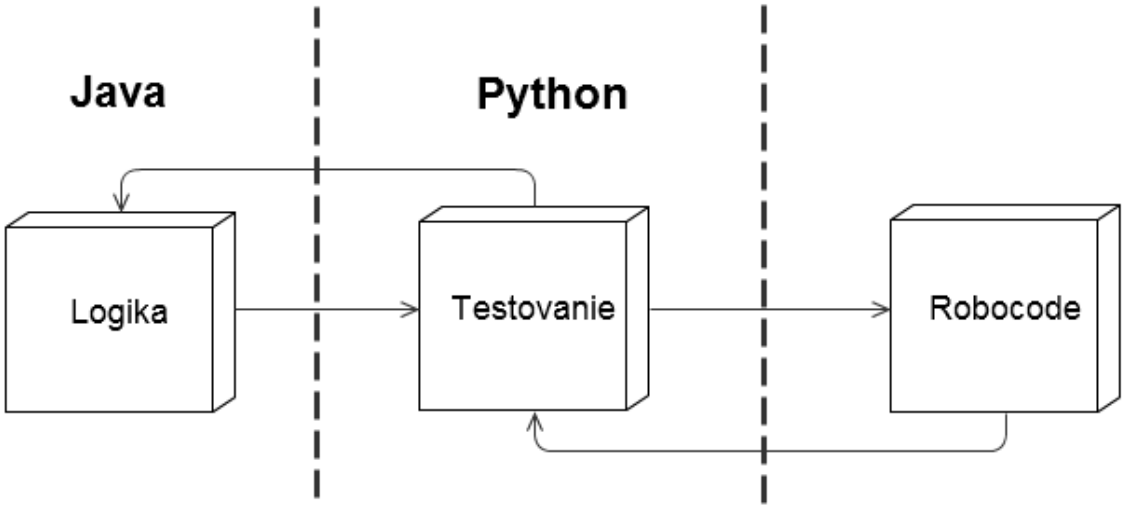
Pri výbere metódy testovania je hlavným rozhodnutím zameranie robota, to znamená, že či je našim cieľom vyvinúť špecializované alebo komplexné riešenie. Úroveň jednotlivých metód sa pohybuje medzi týmito dvoma typmi riešenia, pričom obe strany majú určité úskalia.

Problémom komplexnosti riešenia je, že kvalita robota klesá veľkosťou testovacej množiny nepriateľských robotov, pretože vývoj kvalitných úzko špecializovaných čŕt správania robota (konkrétnych techník) je relatívne pomalý. Na druhej strane, komplexné riešenie je robustné, a preto je vyššia šanca, že uspeje aj proti novým robotom, ktorý pracujú na podobnej mechanike. Problém úzkej špecializácie riešenia je evidentný a súvisí s tým, že správanie robota bude vytvorené len na jeden účel, čo je problematické pri testovaní proti testovacej množine skladajúcej sa z viacerých robotov pracujúcich na rozdielnej mechanike.

Ďalší problém súvisí s organizáciou hľadania riešenia v hre Robocode, pretože hra umožňuje definíciu počtu súbojov. Počet súbojov musí mať dostatočne veľkú hodnotu nato, aby sme minimalizovali chybu výsledku (čím viac súbojov, tým je výsledok presnejší) a dostatočne malú hodnotu nato, aby výpočet nebol príliš zložitý.

4 Implementácia

Implementácia aplikácie je realizovaná v programovacích jazykoch Java a Python. Algoritmické moduly systému sú implementované v Jave (PSO, gramatický roj) a podporný modul pre paralelný výpočet, testovanie a komunikáciu s externým prostredím Robocode v Pythone (Obr. 16). Hlavným dôvodom je skúsenosť s paralelizáciou výpočtu pomocou Pythonu, teda išlo o efektívnejšiu implementáciu z pohľadu časových prostriedkov.



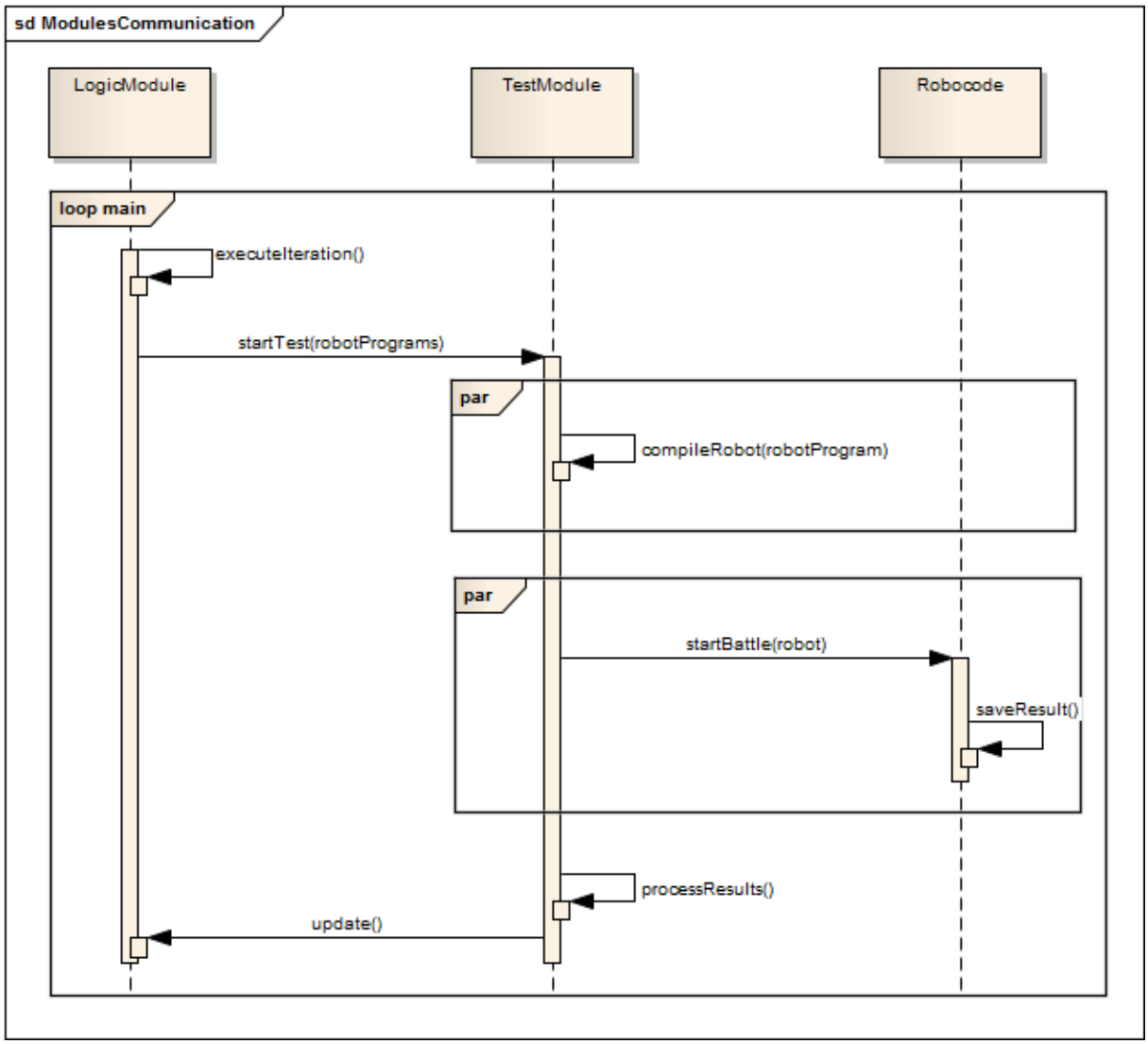
Obr. 16. Komunikácia hlavných modulov systému.

Hlavné problémy implementácie riešenia môžeme rozdeliť do nasledovných kategórií:

- overenie správnej činnosti algoritmov – konvergovanie algoritmov k riešeniu,
- paralelizácia – urýchlenie výpočtu pomocou paralelných výpočtových techník,
- logovanie – zápis výsledkov a ich vizualizácia.

Konvergovanie algoritmov k riešeniu sa podarilo úspešne vykonať pri jednoduchých príkladoch pre PSO a generovanie gramatiky. Pre komplexné problémy nie je možná automatická kontrola, takže je nutné ju vykonávať manuálne krokovaním cez jednotlivé logy iterácií.

Základný princíp paralelizácie je zobrazený na Obr. 17, ktorý spočíva v paralelizovaní kompilácie robotov a následného procesu testovania robotov v Robocode, kde je pre každého robota vyčlenené jedno vlákno (angl. thread). Dôsledkom optimalizácie bolo zníženie času kompilácie približne o 45% a času trvania súbojov približne o 65%.



Obr. 17. Sekvenčný diagram komunikácie modulov a paralelizácie výpočtu.

Programy robotov, výsledky súbojov a logy výpočtov sú zapisované do špeciálne navrhnutej hierarchickej štruktúry priečinkov tak, aby bolo jednoduché spracovávanie výpočtových informácií. Testovací priečinok obsahuje priečinky s informáciami každej iterácie algoritmu s detailnými informáciami ohľadom iterácie. Iterácia obsahuje priečinok pre každého robota (resp. časticu), kde sú uložené detailné informácie robotov. Takýmto spôsobom bolo možné jednoduché vytvorenie grafického používateľského rozhrania, ktoré bolo zamerané na textovú a grafickú vizualizáciu výsledkov.

5 Testovanie

Testovanie implementovanej aplikácie prebiehalo v dvoch fázach počas samotného vývoja. Výsledkom oboch testovaní je dvojica robotov s najlepšou fitnes hodnotou, ktorí sa počas výpočtov vyvinuli a neboli prekonaní až do ukončenia výpočtu. Každé z vyvinutých riešení má označenie podľa mechaniky, ktorú používa v súbojoch. Dôležitou časťou testovania je výber množiny nepriateľských robotov, ktorí sa použijú pri vývoji alebo overení kvality finálneho riešenia. Zoznam robotov s ktorými sa počas vývoja robotov pracovalo je v Tab. 13.

Tab. 13. Zoznam nepriateľských robotov.

Robot	Princíp činnosti
Target	neškodný robot, ktorý sa používa ako terč pri vývoji v prvých iteráciách
MyFirstRobot	jednoduchý pohyb a streľba pri zaznamenaní radarom
RamFire	útok nárazmi do nepriateľa a streľba z blízka
Crazy	náhodný pohyb
Tracker	postupné zameriavanie s približovaním sa k nepriateľovi a streľba z blízka
VelociRobot	dynamická zmena rýchlosti a jednoduchá streľba
Fire	pohyb aktivovaný až po zásahu nepriateľa
SpinBot	pohyb v kruhoch a streľba na nepriateľa po zaznamenaní radarom
Walls	pohyb pozdĺž stien
TrackFire	postupné zameriavania a rýchla streľba
GP-Bot	robot vyvinutý prostredníctvom genetického algoritmu (kap. 2.6.1)
HarperBot	robot vyvinutý prostredníctvom koevolúcie (kap. 2.6.2)

5.1 Rumbler

Hlavným cieľom vývoja, pri ktorom sa podarilo nájsť riešenie bolo overenie správnosti práce optimalizácie pomocou gramatického roja, správnosť generovania programov na základe triviálnej gramatiky s unárnymi operátormi, správnosť komunikácie s Robocode a spracovania výsledkov za účelom korektného výpočtu fitnes proti jednoduchému robotovi. Testovanie bolo prekvapením, pretože sa podarilo vyvinúť riešenie (Rumbler), ktoré porazilo robota MyFirstRobot s viac ako 95% úspešnosťou. I keď strom riešenia má minimálnu veľkosť, tak sa potvrdzuje tvrdenie, že aj jednoduchý robot môže byť silný. Rumbler má pridelené označenie na základe jeho metódy útoku, teda postupný útok nárazom.

Tab. 14. Opis parametrizácie a metód použitých pri vývoji Rumblera.

Názov	Opis
algoritmus hľadania	gramatický roj
veľkosť roja	100 častíc
veľkosť vektora častice	100 dimenzií
rozmer priestoru	255
maximálna rýchlosť častice	64
sociálny a kognitívny koeficient	1.0
váha zotrvačnosti	od 0.1 do 0
gramatika	triviálna bez podmienených výrazov
metóda výpočtu fitnes	jednoduchá na základe percentuálneho skóre z 10 kôl (kap. 3.2.5)
nepriateľské roboty	MyFirstRobot
iterácia nálezu	37
veľkosť programu (stromu)	6
princíp činnosti	každým zaznamenaním nepriateľa sa k nemu natočí a posunie, následne strieľa
nedostatky vývoja	krátky program robota, triviálna gramatiky (unárne operátory) a jednoduchá metóda výpočtu fitnes

5.2 Taurus

Pri vývoji Taura sa použila už úplná základná gramatika, menší roj na základe empirického výskumu podľa [9] a progresívna metóda výpočtu fitnes, ktorá odomyká robotom nové zápasy proti silnejším súperom pomocou získaných výsledkov s predchádzajúcimi súpermi.

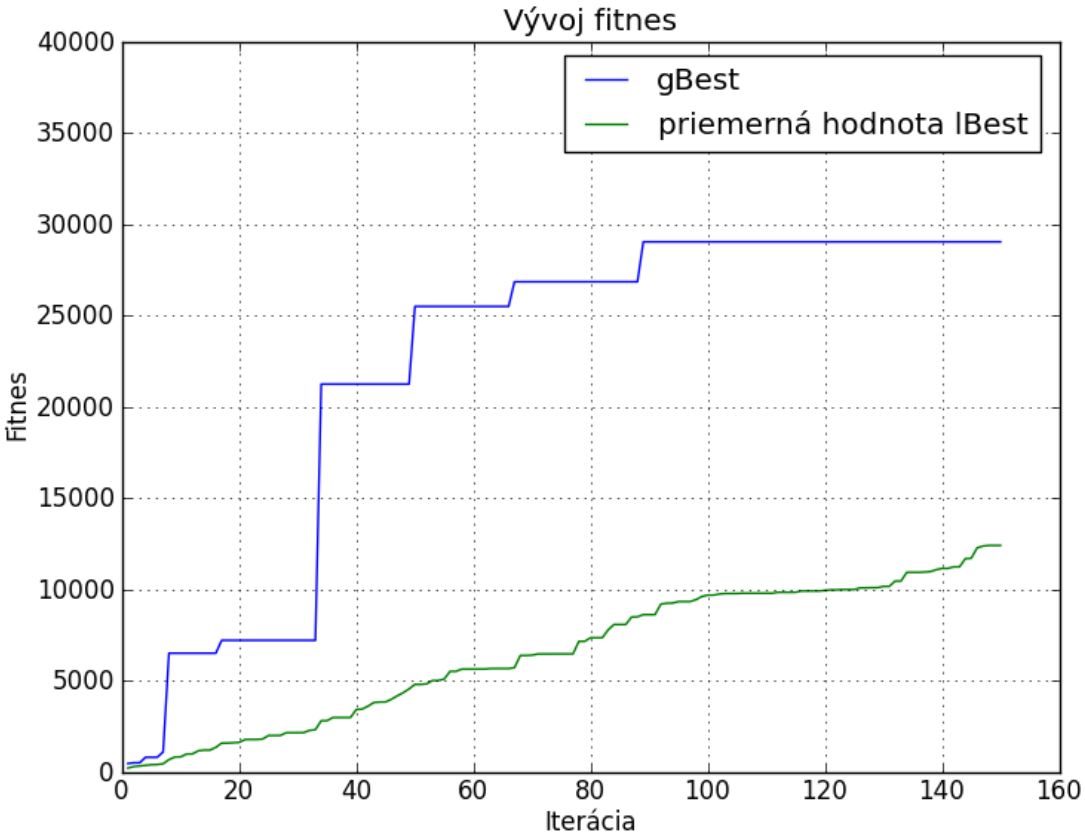
Tab. 15. Opis parametrizácie a metód použitých pri vývoji Taura.

Názov	Opis
algoritmus hľadania	gramatický roj
metóda generovania programu	úplná stromová metóda (kap. 3.2.3)
veľkosť roja	30 častíc
veľkosť vektora častice	100 dimenzií
rozmer stavového priestoru	255
maximálna rýchlosť častice	32
sociálny a kognitívny koeficient	1.0
váha zotrvačnosti	od 0.9 do 0.4
gramatika	základná (kap. 3.2.3.1)
metóda výpočtu fitnes	progresívna na základe totálneho skóre z 10 kôl (kap. 3.2.5)
nepriateľské roboty	Target, MyFirstRobot, Rumbler, Crazy, Tracker, Walls
iterácia nálezu	88
veľkosť programu (stromu)	59
princíp činnosti	v prípade zaznamenania nepriateľa radarom sa natočí k nemu, následne strieľa a posúva sa až k stene
nedostatky vývoja	nerovnomerné generovanie stromu správania

Princíp činnosti Taura pracuje na podobnej mechanike ako v prípade Rumblera, čo môže byť čiastočne spôsobené aj tým, že Rumbler sa použil v testovacej množine ako nepriateľský robot. Taurus bol vyvinutý v 88. iterácii algoritmu, pričom lepšie riešenie sa nepodarilo nájsť až po 150. iteráciu, v ktorom bolo hľadanie ukončené.

Na Obr. 18 si môžeme všimnúť, že trend globálnej fitnes je skokový. To je jedným z charakteristických znakov evolučných optimalizačných techník. Ďalším znakom je postupné zlepšovanie priemernej fitnes, ktorá je v rámci možností PSO vyjadrená priemernou hodnotou najlepšej lokálnej fitnes všetkých častíc (aritmetický priemer lokálne najlepšej hodnoty všetkých častíc).

Fitnes hodnota je vypočítaná na základe získaného skóre proti robotom v testovacej množine. Ak bol robot úspešný v súboji so slabšími robotmi (jeho skóre prekročilo definovaný limit pre ďalšiu úroveň), tak sa mu odomkol zápas s nasledujúcim silnejším súperom. Finálna fitnes robota je súčtom skóre, ktoré získal vo všetkých zápasoch.



Obr. 18. Vývoj globálnej a priemernej lokálnej fitnes pri vývoji Taura.

5.3 Porovnanie výsledkov

Kapitola sa zaoberá porovnaním výsledkov a štatistík analyzovaných a vyvinutých robotov. Tab. 16. obsahuje výsledky dosiahnuté v súboji proti základnej množine robotov (označenie [Z]) a vybranej množine robotov podľa [13], pričom tieto výsledky sú priamo použité v tabuľke (HarperBot), keďže analyzovaný robot nie je voľne prístupný. Ostatní roboti boli otestovaní proti rovnakej množine robotov v súboji s počtom kôl 100. Jednotlivé výsledky obsahujú aj počet víťazstiev z týchto 100 kôl, keďže totálne skóre môže byť skresľujúce.

Tab. 16. Porovnanie výsledkov vyvinutých a existujúcich robotov.

Súper	Rumbler (V)	Taurus (V)	HarperBot	GP-Bot (V)
MyFirstRobot [Z]	83% (97-3)	67% (66-34)	-	92% (89-11)
RamFire [Z]	48% (52-48)	54% (67-33)	67%	86% (97-3)
Crazy [Z]	50% (32-68)	55% (56-44)	86%	86% (77-23)
Tracker [Z]	42% (27-73)	49% (42-58)	83%	87% (96-4)
VelociRobot [Z]	50% (54-46)	46% (47-53)	74%	84% (84-16)
Fire [Z]	35% (7-93)	48% (33-67)	-	92% (93-7)
SpinBot [Z]	40% (26-74)	44% (33-67)	72%	84% (88-12)
Walls [Z]	3% (0-100)	20% (7-93)	65%	49% (19-81)
TrackFire [Z]	28% (0-100)	26% (0-100)	64%	81% (94-6)
GuessFactor	11% (0-100)	12% (1-99)	51%	54% (60-40)
Peryton	13% (0-100)	9% (0-100)	55%	52% (58-42)
Sparrow	7% (0-100)	3% (0-100)	25%	39% (28-72)
Duelist	8% (0-100)	6% (1-99)	19%	36% (28-72)
Tron	1% (0-100)	2% (0-100)	8%	3% (0-100)
Aspid	6% (0-100)	4% (0-100)	8%	15% (0-100)
Cigaret	6% (0-100)	5% (0-100)	15%	14% (1-99)
Priemer	**26,9% (18-82)**	**28,1% (22-78)**	**49,3%**	**59,6% (57-43)**

Tab.17. obsahuje porovnanie kvality vyvinutých a prístupných robotov, ktorý boli vyvinutý evolučnou optimalizačnou technikou.

Tab. 17. Porovnanie kvality vyvinutých a existujúcich robotov medzi sebou.

Súper	Rumbler (V)	Taurus (V)	GP-Bot (V)
Rumbler	-	54% (62-38)	87% (99-1)
Taurus	46% (38-62)	-	88% (97-3)
GP-Bot	13% (1-99)	12% (3-97)	-

6 Zhodnotenie

Cez prvý semester sa podarilo úspešne spracovať kapitolu analýzy, ktorá sa zaoberá vysvetlením evolučných optimalizačných techník zameraných na riešenie dynamických problémov. Postupne opisuje princípy evolučných algoritmov, gramatickej evolúcie, rojovej inteligencie, optimalizácie rojom častíc a gramatického roja. Taktiež sa venuje prostrediu Robocode a vysvetľuje jeho základné vlastnosti a princípy.

V druhom semestri sa vytvoril prototyp prostredia zameraného na vývoj robotov pomocou gramatického roja. Pomocou modulárnej implementácie je možná jednoduchá úprava logiky hlavného algoritmu a možnosť optimalizácie výpočtového procesu. Taktiež sa podarilo implementovať jednoduchú paralelizáciu, ktorá urýchľuje výpočtový proces. V rámci dokumentu boli spracované kapitoly návrhu, implementácie a testovania, v ktorých sú identifikované metódy a prístupy riešenia pre zadaný problém. Prostredníctvom prototypu prostredia boli vyvinuté dva roboty, ktoré síce nedosahujú kvality analyzovaných robotov, ale sú dôkazom toho, že algoritmická a technologická realizácia riešenia je použiteľná a dá sa na nej ďalej stavať.

6.1 Plán práce

Tab. 18 obsahuje zoznam funkcionalít a rozšírení zoradených podľa priority, ktoré sa plánujú v nasledujúcom semestri vykonať. Opísané sú postupne v nasledujúcich odsekoch.

Tab. 18. Plán práce pre nasledujúci semester.

Úloha	Funkcionalita
1.	rozšírenie základnej gramatiky
2.	implementácia modulárneho generovania a paralelného gramatického roja
3.	experiment s cieleným vývojom modulov robota
4.	experiment zameraný na porovnanie metód testovania
5.	vyvinutie robota porážajúceho základných robotov (s úspešnosťou aspoň 80%)
6.	vyvinutie robota porovnateľného kvalitou s analyzovanými robotmi
7.	návrh možností vylepšenia vyvinutého robota
8.	vytvorenie finálneho riešenia
9.	overenie kvality riešenia v online turnaji
10.	analýza možností generovania zložitejšieho správania zmenou gramatiky
11.	experiment s generovaním komplexného správania robotov
12.	distribuovanie výpočtového procesu medzi viacero počítačov

Rozšírenie základnej gramatiky spočíva v pridaní nových funkcií a konštánt podľa Kap. 3.2.3.2 za účelom zväčšenia priestoru možných riešení, ktorý zvýši možnosti generovania rozhodovacieho stromu, resp. zvýši pestrosť generovania správania robota.

Implementácia modulárneho generovania a paralelného gramatického roja je realizácia metód opísaných v Kap. 3.2.4.2 a Kap. 3.2.4.3, ktoré navzájom súvisia. Modulárne generovanie správania robota je prístup k vývoju robota zameraný na vývoj špecifickej časti, pričom paralelný gramatický roj je algoritmický spôsob realizácie modulárneho generovania.

Cieľom *experimentu s cieleným vývojom modulov robota* je vytvoriť algoritmické riešenie pre túto metódu a porovnať výsledky s metódou striedavého generovania, na ktorej je založený algoritmus paralelných gramatických rojov.

Ďalšou úlohou je *experiment zameraný na porovnanie metód testovania*, ktorý súvisí s porovnaním metód hodnotenia robotov počas vývoja (Kap. 3.2.5) a výberom najvhodnejšej fitnes funkcie za účelom rýchlejšieho vyvinutia úspešného robota.

Vývoj robota porážajúceho základných robotov je jednou zo základných požiadaviek zadania. Zámerom úlohy je vyvinutie robota, ktorý bude porážať základnú množinu robotov (Tab. 16) s priemernou percentuálnou mierou počtu víťazstiev aspoň 80%. Nadväzujúcou úlohou je *vyvinutie robota porovnateľného kvalitou s analyzovanými robotmi* (Kap. 2.6), kde je cieľom zlepšenie konkurencieschopnosti vyvinutého riešenia prostredníctvom analýzy jeho slabín a experimentovania s použitými metódami za účelom odstránenia týchto slabín.

Nasledujúce úlohy sa týkajú možného rozšírenia a súvisia s *návrhom možností vylepšenia vyvinutého robota*, ktoré by boli odskúšané pri *vytváraní finálneho riešenia (robota)*, pričom *kvalita riešenia by bola overená vo zvolených online turnajoch* podľa Tab. 2 v Kap. 2.5.

Nasledujúce úlohy sú nad rámec zadania. Zameriavajú sa na *analýzu možností generovania zložitejšieho správania zmenou gramatiky* a *experiment s generovaním komplexného správania robotov*. Cieľom týchto úloh je definovanie možných vylepšení a rozšírení, ktoré by viedli k vytvoreniu robotov kvalitatívne porovnateľných s manuálne vytvorenými robotmi. Nežiaducim dôsledkom bude zvýšenie výpočtovej zložitosti, na ktoré bude musieť použitá jedna z metód *distribuovaného výpočtového procesu medzi viacero počítačov*.

7 Použitá literatúra

[1] Eiben, A.E., Smith, J.E.: *Introduction to Evolutionary Computation*. Springer, Natural Computing Series, (2008), s. 15–24.

[2] Dempsey, I., O`Neill, M., Brabazon, A.: *Foundations in Grammatical Evolution for Dynamic Environments*. Springer, Studies in Computational Intelligence Series, (2009), s. 2–20.

[3] Hugosson, J. et al.: Genotype Representations in Grammatical Evolution. *Applied Soft Computing Journal,* (2010), Zv. 10, s. 36–43.

[4] O`Neill, M. et al.: Crossover in Grammatical Evolution. *Journal of Genetic Programming and Evolvable Machines,* (2003), Zv. 4, s. 67–84.

[5] Keijzer, M.: Ripple Crossover in Genetic Programming. *Proceedings of 4th European Conference on Genetic Programming (EUROGP2001),* Como, Italy, (2001), Zv. 2038, s. 74–86.

[6] Liu, Y., Passino, K.M.: *Swarm Intelligence: Literature Overview*. Technical report, Ohio State University, USA, (2000), Zv. 2015, s. 1–8.

[7] Engelbrecht, A.P. *Fundamentals of Computational Swarm Intelligence.* Wiley, (2005), s. 2–129.

[8] Millonas, M.M.: Swarms, Phase Transitions, and Collective Intelligence. *Proceedings of Artifical Life III.* Santa Fe Institute: Addison-Wesley, USA, (1992), s. 417–445.

[9] O`Neill, M., Brabazon, A.: Grammatical Swarm: The generation of programs. *Natural Computing: An International Journal,* (2006), Zv. 5, s. 443–462.

[10] Nelson, M.: *Robocode*. [Online] 11. 08. 2009. [Dátum: 06. 12. 2012.] http://robowiki.net/wiki/Robocode.

[11] Eisenstein, J.: *Evolving Robocode Tank Fighters*. Technical report AIM-2003-023 AI Lab, Massachusetts Institute of Technology, USA, (2003).

[12] Shichel, Y., Ziserman, E., Sipper, M.: GP-Robocode: Using Genetic Programming to Evolve Robocode Players. *Proceedings of 8th European Conference on Genetic Programming (EUROGP2005),* Lausanne, Switzerland, (2005), s.143–154.

[13] Harper, R.: Co-evolving robocode tanks. *GECCO '11 Proceedings of the 13th annual conference on Genetic and evolutionary computation*, ACM New York, USA, (2011), s. 1443–1450.

Príloha A: Inštalačná príručka

Pre korektné spustenie je nutné vykonávať inštaláciu na operačnom systéme Windows a mať nainštalované nasledujúce softvérové prostredia:

- Java 1.6.0 a viac
- Python 3.0 a viac
- Robocode 1.7.4.4 a viac (v priečinku *C:\Robocode*)

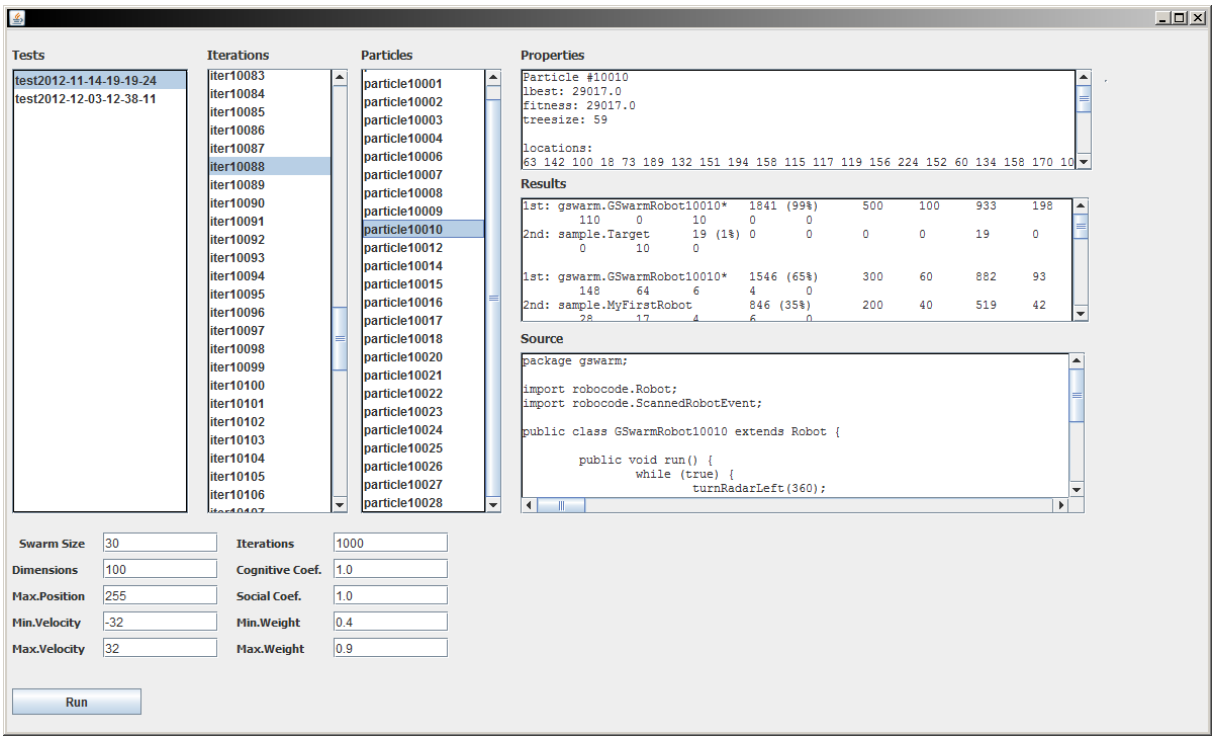
Inštaláciu a spustenie prostredia je možné vykonať aj prostredníctvom nasledujúcich krokov:

1. na elektronickom médiu otvoríme priečinok *install* a nainštalujeme potrebné prostredia cez nasledujúce inštalačné súbory:
    a. Java (32-bit) *jre-7u9-windows-i586.exe*, (64-bit) *jre-7u9-windows-x64.exe*
    b. Python (32-bit) *python-3.0.msi*, (64-bit) *python-3.0.amd64.msi*
    c. Robocode (32/64-bit) *robocode-1.7.4.4-setup.jar*
2. finálne prostredia spustíme pomocou *install/GSwarm0.4.jar*

Príloha B: Používateľská príručka

Po spustení aplikácie sa zobrazí používateľské rozhrania podobné ako na Obr. 19. Rozhranie pozostáva z nasledujúcich častí:

- *Tests* – menu s možnosťou prepínania medzi výsledkami testov,
- *Iterations* – menu s možnosťou prepínania medzi iteráciami v rámci zvoleného testu,
- *Particles* – menu s možnosťou prepínania medzi časticami v rámci zvolenej iterácie,
- *Properties* – zobrazuje informácie ohľadom aktívnej iterácie alebo častice,
- *Results* – zobrazuje výsledky súbojov robota pre zvolenú časticu,
- *Source* – zobrazuje vygenerovaný program robota,
- *Parameters* – nastavenie parametrov algoritmu podľa kap. 3.2.4.1,
- *Run* – spustenie algoritmu.



Obr. 19. Ukážka používateľského rozhrania.

Príloha C: Obsah elektronického média

Elektronické médium má nasledovnú štruktúru:

- /**docs** – obsahuje dokumentáciu diplomovej práce,
- /**install** – obsahuje inštalačné súbory prostredí a samotnú aplikáciu,
- /**robots** – obsahuje vyvinutých robotov,
    - /**extern** – obsahuje pokročilú množinu externých robotov,
- /**src** – obsahuje zdrojový kód projektu,
- /**tests** – obsahuje log súbory vykonaných testov.