UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

PEDRO HENRIQUE BERGAMO BERTOLLI

OTIMIZAÇÃO DO COMPORTAMENTO DE AGENTES UTILIZANDO SIMULATED ANNEALING

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2018

PEDRO HENRIQUE BERGAMO BERTOLLI

OTIMIZAÇÃO DO COMPORTAMENTO DE AGENTES UTILIZANDO SIMULATED ANNEALING

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. André Koscianski

PONTA GROSSA

2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa

Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Bacharelado em Ciência da Computação

TERMO DE APROVAÇÃO

OTIMIZAÇÃO DO COMPORTAMENTO DE AGENTES UTILIZANDO SIMULATED ANNEALING

por

PEDRO HENRIQUE BERGAMO BERTOLLI

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 12 de junho de 2018 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. André Koscianski
Orientador

_____
Prof. Dr. André Pinz Borges
Membro titular

_____
Prof. Dr. Augusto Foronda
Membro titular

_____
Profª. Dra. Helyane Bronoski Borges
Responsável pelo Trabalho de Conclusão de Curso

_____
Prof. MSc. Saulo Jorge Beltrão de Queiroz
Coordenador do curso

- A Folha de Aprovação assinada encontra-se arquivada na Secretaria Acadêmica -

RESUMO

BERTOLLI, Pedro Henrique Bergamo. **Otimização do Comportamento de Agentes Utilizando Simulated Annealing**. 2018. 50 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2018.

Este trabalho aborda conceitos de Inteligência Artificial com foco em agentes inteligentes e técnicas de otimização como programação genética e simulated annealing. Através do simulador de batalhas Robocode, foi possível implementar uma solução que visa gerar automaticamente o melhor comportamento de um agente reativo utilizando as técnicas demonstradas. Cada agente é representado por um robô tanque no simulador cujo objetivo principal definido é derrotar um oponente determinado número de vezes consecutivas para que seja considerado vencedor. Para encontrar o robô apto a resolver o problema proposto, duas implementações de código foram realizadas sendo possível realizar testes e analisar os diferentes resultados obtidos.

Palavras-chave: Otimização. Agentes. Comportamento. Técnicas. Robocode.

ABSTRACT

BERTOLLI, Pedro Henrique Bergamo. **Agents Behavior Optimization Using Simulated Annealing**. 2018. 50 f. Work of Conclusion Course (Graduation in Computer Science) – Federal Technology University - Paraná. Ponta Grossa, 2018.

This work deals with concepts of Artificial Intelligence focusing on intelligent agents and optimization techniques such as genetic programming and simulated annealing. Through the Robocode battle simulator, it was possible to implement a solution that aims to automatically generate the best behavior of a reactive agent using the demonstrated techniques. Each agent is represented by a tank robot in the simulator whose main objective is to defeat an opponent determined number of consecutive times to be considered a winner. In order to find the robot able to solve the proposed problem, two code implementations were performed, being possible to perform tests and analyze the different results obtained.

Keywords: Optimization. Agents. Behavior. Techniques. Robocode.

LISTA DE ILUSTRAÇÕES

Figura 1 - Representação de um Agente 10
Figura 2 - Arquitetura baseada em 4 estados mentais.....14
Figura 3 - Arquitetura BDI 15
Figura 4 - Funcionamento da Arquitetura Reativas.....16
Figura 5 - Camada Horizontal de um Arquitetura Híbrida 17
Figura 6 - Camada Vertical de uma passagem (1) e duas passagens (2) 18
Figura 7 - Árvore de Sintaxe.....22
Figura 8 - Método da Seleção Proporcional (Roleta) 25
Figura 9 - Operação de crossover entre indivíduos de uma população 27
Figura 10 - Operação de mutação em um indivíduo de uma população 27
Figura 11 - Fluxo Recozimento Simulado 29
Figura 12 - Interface Gráfica e IDE do Robocode 30
Figura 13 - Estrutura de um Robô Tanque.....32
Figura 14 - Código de um JuniorRobot 33
Figura 15 - Parametrização de Build.....35
Figura 16 - Parametrização de Batalha e argumentos da VM.....35
Figura 17 - Diagrama UML do código desenvolvido.....37
Figura 18 - Pseudocódigo do método renovaGeração() 39
Figura 19 - Código de funcionamento do SpinBot.....41

Gráfico 1 - Histórico de Evolução de Fitness Primeiro Teste……42
Gráfico 2 - Histórico de Evolução de Fitness Segundo Teste…….....43

Quadro 1 - Principais Métodos e Eventos Utilizados pelo Robocode ……..31

LISTA DE SIGLAS

IA	Inteligência Artificial
PG	Programação Genética
CE	Computação Evolucionária
BDI	*Belief-Desire-Intention*

SUMÁRIO

1 INTRODUÇÃO 7
1.2 OBJETIVOS 8
1.3 JUSTIFICATIVA 9
1.4 ORGANIZAÇÃO DO TRABALHO 9
2 AGENTES INTELIGENTES 10
2.1 CATEGORIAS DE AGENTES 11
2.2 ARQUITETURA DE AGENTES 13
2.2.1 Arquiteturas Deliberativas ou Cognitivas 13
2.2.1.1 Arquiteturas funcionais 13
2.2.1.2 Arquiteturas baseadas em estados mentais 14
2.2.1.2.1 Arquiteturas BDI (Belief-Desire-Intention) 14
2.2.2 Arquiteturas Reativas 15
2.2.3 Arquiteturas Híbridas 16
2.3 AMBIENTE 18
2.4 APLICAÇÃO 19
3 TÉCNICAS DE OTIMIZAÇÃO 21
3.1 PROGRAMAÇÃO GENÉTICA 21
3.1.1 Funcionamento 22
3.1.1.1 População inicial 23
3.1.1.2 Avaliação de aptidão (fitness) e seleção 24
3.1.1.3 Operadores genéticos 25
3.1.1.4 Nova população e critério de parada 27
3.2 RECOZIMENTO SIMULADO (SIMULATED ANNEALING) 28
4 SIMULADOR ROBOCODE 30
4.1 FUNCIONAMENTO 30
5 DESENVOLVIMENTO 34
5.1 GERAÇÃO AUTOMÁTICA DE AGENTES 36
5.2 TESTES 40
5.3 RESULTADOS 41
6 CONSIDERAÇÕES FINAIS 44
REFERÊNCIAS 45

1 INTRODUÇÃO

A Inteligência Artificial (IA) é uma área da ciência da computação que procura elaborar sistemas autônomos (OSTETTO; SANTOS, 2011). Alguns destes sistemas autônomos são denominados agentes e embora não tenham uma definição correta de comum acordo entre autores, sua principal capacidade está em sua autonomia de responder a estímulos em determinados ambientes e cumprir objetivos.

Ao projetar um agente deve se estabelecer a performance desejada a alcançar, o ambiente em que as tarefas serão realizadas e seus sensores e atuadores responsáveis respectivamente por interpretar suas percepções e agir sobre o ambiente.

Entretanto, agentes inteligentes nem sempre tomam a melhor decisão em relação ao ambiente em que se encontram e seu objetivo final estabelecido. Sua escolha em determinada situação pode ser considerada ótima em relação ao momento e satisfazer uma regra, porém esse conjunto de escolhas pode resultar em um caminho maior ou diferente do esperado, aumentando a complexidade de solução do problema.

O estudo da otimização de resolução de problemas com agentes inteligentes está presente em vários ramos da IA, como por exemplo a Computação Evolucionária (CE). A Programação Genética (PG) é uma frente de pesquisa da CE e possui seu modelo para soluções de problemas inspirado na Seleção Natural (DARWIN, 1859) onde a sobrevivência dos indivíduos é determinada pela sua capacidade de se adaptar ao meio ambiente em que vive (KOZA, 1992). Indivíduos adaptados ao ambiente tem mais chances de sobreviver, tornando maior a probabilidade de transmitir seus genes para a nova geração que por sua vez, também será exposta ao ambiente gerando indivíduos melhor adaptados e com maiores chances de passar adiante seu material genético. Na PG cada indivíduo representa um programa de computador que será avaliado de acordo com o resultado de sua execução. Em consequência, o torna eficiente na solução de diversos problemas em diferentes áreas do conhecimento desde mineração de dados e biologia molecular até o projeto de circuito digitais e inúmeras tarefas envolvendo otimização (O'NEIL; RYAN, 2000).

Outra técnica de otimização conhecida na inteligência artificial é o *simulated annealing* que possui como base conceitos fundamentados na prática da indústria

metalúrgica. Esse processo consiste no aquecimento de um metal sólido e o resfriamento gradativo do mesmo. Isso significa que quando um metal apresenta temperatura muita alta, suas móleculas se encontram com uma energia interna muita alta e ao resfriar gradativamente a energia interna se reduz aos poucos de modo a formar uma estrutura estável.

Da mesma forma funciona o algoritmo: a cada iteração o sistema busca a solução que está mais perto de seu objetivo e realiza pequenas modificações em seus parâmetros até que econtrar uma condição que satisfaça o problema.

A utilização de simuladores virtuais para representação de problemas em diversas áreas serve como auxílio para o estudo de novas técnicas e demonstração de resultados obtidos de forma mais próxima da realidade. O simulador Robocode é um simulador de batalhas e por meio dele é possível programar robôs com um conjunto de ações e comandos com o objetivo de sempre derrotar seu oponente.

Através dessa ferramenta, foi desenvolvido um código capaz de gerar automaticamente agentes utilizando uma combinação entre programação genética e simulated annealing de modo a configurar seu comportamento e vencer um número determinado de batalhas.

.

1.2 OBJETIVOS

O objetivo geral do trabalho é desenvolver, através do simulador Robocode, a automação do comportamento de um agente cujo objetivo é derrotar seu oponente.

Os objetivos específicos são:

- Utilizar a combinação de técnicas de otimização estudadas para configurar os movimentos de um robô;
- Desenvolver uma solução parar gerar um robô capaz de derrotar o oponente 10 vezes consecutivas;
- Analisar resultados obtidos.

1.3 JUSTIFICATIVA

Agentes inteligentes estão presentes em nosso cotidiano com grande frequência, mesmo que imperceptivelmente. Não se restringem apenas a problemas de estudo computacional e estão presentes desde o entretenimento digital até sistemas de automação de alta complexidade. Sua aplicação pode ser encontrada em diversas áreas como agricultura, medicina, transportes, tarefas militares e diversos outros serviços.

Erros por parte de sistemas autônomos podem resultar em uma situação inapropriada ou até mesmo fatal, como por exemplo, uma tomada de decisão errada de um carro autônomo que provoca um acidente. O estudo e desenvolvimento da otimização do comportamento de agentes é um processo importante onde as tarefas dadas a estes tendem a ser a ser executadas de forma a evitar erros.

Observando a utilização de agentes em nosso meio e tendo uma visão de que a otimização pode contribuir para a qualidade dos serviços oferecidos, surgiu a ideia da criação de uma aplicação que procura otimizar o comportamento de agentes. O estudo dessa técnica permite que seja aplicada em outras situações de diversas áreas do conhecimento cujo objetivo é encontrar uma solução considerada ótima para o problema proposto.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em seis capítulos. O capítulo 2 apresenta os conceitos sobre agentes, tais como seus tipos, arquiteturas e aplicações.

O Capítulo 3 aborda técnicas de otimização que são utilizadas em combinação para desenvolvimento do projeto.

O Capítulo 4 descreve o simulador Robocode e parte de seu funcionamento como ideia para desenvolvimento do trabalho.

O Capítulo 5 apresenta o desenvolvimento juntamente com os testes e resultados obtidos e por último o capítulo 6 com as considerações finais.

2 AGENTES INTELIGENTES

Embora existam muitos debates e discussões entre pesquisadores da área sobre a verdadeira definição de agente, o termo passou a ser empregado para descrever softwares que envolvam simples controles e/ou regulação, como por exemplo um termostato (REIS, 2003).

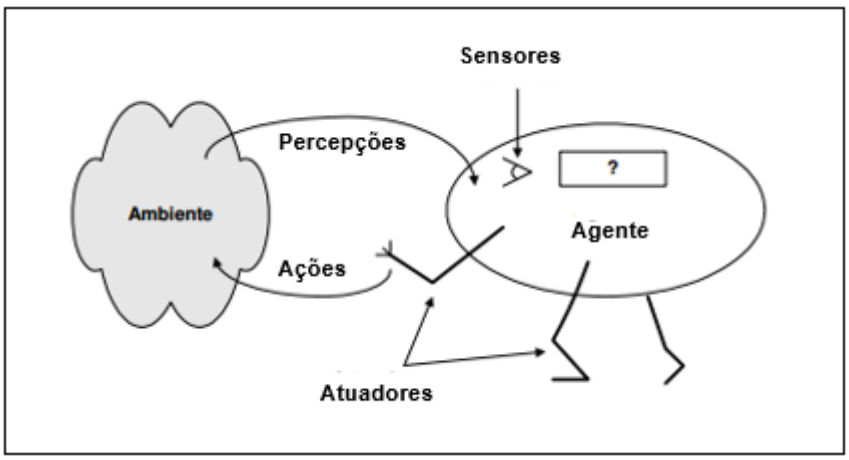
Os conceitos abordados nos trabalhos de Wooldridge e Jennings (1995), Franklin e Graesser (1997) e Russel e Norvig (1995) chegam a um consenso de que a autonomia é essencial em um agente, onde cada um possui sua própria existência e não depende da existência de outros.

Graesser (1997) faz uma lista de atributos necessárias a um agente, entre elas estão: autonomia, flexibilidade, adaptabilidade comunicatividade e mobilidade.

Norvig (1995) define agente como uma entidade autônoma que percebe seu ambiente através de sensores e executa suas ações através de atuadores. O mesmo descreve o exemplo de um robô apirador de pó situado em uma sala de dois ambientes e deve por meio de seus sensores (câmeras, sensores de sujeira) perceber o meio em que se encontra (ambiente) e executar suas ações básicas: que são mover ou limpar a sujeira, onde tais ações são realizadas por meio de seus atuadores (rodas, motor de sucção).

Para Wooldridge (1995) um agente é um hardware ou um software que satisfaz as propriedades de autonomia, reatividade, pró-atividade e habilidade social

Figura 1 - Representação de um Agente



Fonte: Norvig (1995)

Um agente inteligente é aquele que toma uma decisão correta de acordo com suas percepções e faz com que seu sucesso dependa de uma medida de desempenho estabelecida (RUSSEL; NORVIG, 1995).

O exemplo do robô aspirador de pó de Norvig (1995) pode ser incrementado para obter uma melhor visão desse conceito. O aspirador de pó encontra-se em um determinado cômodo de uma casa e seu objetivo é aspirar toda sujeira que encontrar e gastar o mínimo de energia possível (media de desempenho). Através de seus sensores o aspirador percebe o ambiente em que se encontra e qual ação deve tomar: se em sua determinada posição o local está sujo então a sujeira deve ser aspirada, se não, continua a se movimentar. De acordo com sua percepção de mundo, deve selecionar uma ação que venha maximizar seu desempenho que é limpar toda sujeira com o mínimo de energia.

O desempenho satisfatório depende do comportamento obtido através da sequência de ações geradas no ambiente, segundo Ronaldo Filho (FILHO, 2008) a racionalidade depende, em qualquer instante, de quatro fatores:

- A medida de desempenho define o critério de sucesso;
- Conhecimento anterior que o agente tem do ambientes;
- As ações que o agente pode executar;
- A sequência de percepções do agente até o momento

Muitas vezes a racionalidade é confundida com onisciência (CORREA FILHO, 1994). Um agente racional apresenta limitações pois seu objetivo é maximizar o desempenho esperado para uma determinada situação, diferentemente de chegar a perfeição (desempenho real) (RUSSEL; NORVIG, 1995). Um agente onisciente busca esse resultado real de suas ações e pode agir de acordo com ele (RUSSEL; NORVIG, 1995), porém é algo impossível no mundo real onde acontecimentos de probabilidade ínfima podem ocorrer e fazer com que a perfeição ou objetivo não sejam alcançados.

2.1 CATEGORIAS DE AGENTES

Como agentes possuem autonomia em suas ações, suas habilidades são variadas em relação a percepção, planejamento e tomada de decisões. Os agentes

são categorizados quanto ao nível de capacidade de resolução de problemas (WOOLDRIDGE; JENNINGS, 1995):

Reativos: Reagem a alterações no ambiente ou a mensagem de outros agentes. Os agentes reativos não apresentam uma memória de ações tomadas e também não planejam suas ações futuras, reagem somente a regras e planos em um modelo estímulo-resposta. Representa o mesmo tipo de agente desenvolvido neste trabalho.

Norvig (1995) divide a categoria de agentes reativos em quatro tipos:

- Reativos simples: O agente percebe o ambiente e retorna uma ação que está em seu conjunto de regras já estabelecido. Para cada estado uma regra é definida e resulta em uma nova ação.

- Reativos Baseados em Modelo: Mantém registro do ambiente em que atua, tanto de suas percepções como de suas ações executadas. Em seu funcionamento o agente percebe o estado em que se encontra e o atualiza, em seguida define suas ações através do conjunto de regras e novamente atualiza seu estado.

- Reativos Baseados em Metas: Além de possuir o registro de seus estados, possui uma meta que descreve um estado desejável a ser atingido (GIRARDI, 2004) em que pode fazer combinações de tal com suas possíveis ações a serem tomadas.

- Reativos Baseados na Utilidade: Uma medida de utilidade deve ser estabelecida de modo a permitir uma comparação entre vários estados buscando um maior nível de satisfatoriedade para o estado desejável.

Deliberativos: Possuem habilidade de raciocínio sobre suas ações e crenças e podem executar planos de ações. Os agentes deliberativos, também conhecidos como cognitivos ou intencionais, executam continuamente três funções: percebem as condições dinâmicas do ambiente, agem alterando as condições do ambiente e raciocinam de modo a interpretar percepções, resolver problemas, fazer inferências e determinar ações (HAYES-ROTH, 1995). Suas ações passadas são utilizadas para planejar ações futuras e apresentam uma elevada complexidade computacional, em lado oposto dos agentes reativos.

2.2 ARQUITETURA DE AGENTES

Assim como existem várias arquiteturas de software, também existem várias arquiteturas de agentes com certas características que permitem a avaliação de sua qualidade e eficácia. A arquitetura interna de uma agente está associada a própria definição e mecanismos de decisão do agente que determinam e influenciam sua atuação (SHHEIBIA, 2001).

2.2.1 Arquiteturas Deliberativas ou Cognitivas

Como descrito na seção 2.2, os agentes deliberativos possuem uma representação simbólica do mundo onde suas decisões são feitas por meio do raciocínio lógico utilizando regras (heurísticas) que controlam o comportamento do agente.

A arquitetura de um agente cognitivo possui: conhecimento, percepção, comunicação, raciocínio e decisão (DEMAZEAU, 1995).

A arquitetura de agentes deliberativos pode ser classificada como funcional ou baseada em estados mentais (OLIVEIRA, 1996).

2.2.1.1 Arquiteturas funcionais

Na arquitetura funcional o agente é composto por módulos responsáveis para cada uma de suas funcionalidades necessárias para sua operação (OLIVEIRA, 1996). Tal arquitetura pode ser dividida em três partes (STEINER, 1996):

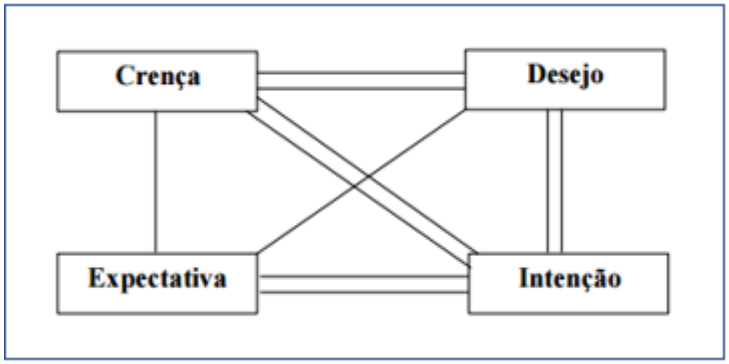
- Cabeça: controla as ações; corresponde a capacidades reativas e racionais;
- Comunicador: implementa a capacidade de comunicação do agente;
- Corpo: Observa o ambiente e executa as ações.

2.2.1.2 Arquiteturas baseadas em estados mentais

As arquiteturas baseadas em estados mentais consideram o agente como forma de um conjunto composto por crenças, escolhas e capacidades.

O comportamento autônomo do agente é baseado nestes estados mentais, onde a decisão a ser tomada deve ser executada a fim de satisfazer algum de seus objetivos de modo que mudanças no ambiente determinam mudanças nas suas crenças ou objetivos (JUCHEM; BASTOS, 2001).

Figura 2 - Arquitetura baseada em 4 estados mentais



Fonte: Fonte: Juchem e Bastos (2001)

Dentro da arquitetura baseada em estados mentais, encontra-se a arquitetura BDI (*Belief-Desire-Intention*).

2.2.1.2.1 Arquiteturas BDI (Belief-Desire-Intention)

Essa arquitetura considera que os agentes possuem três estados mentais: crenças, desejos e intenções.

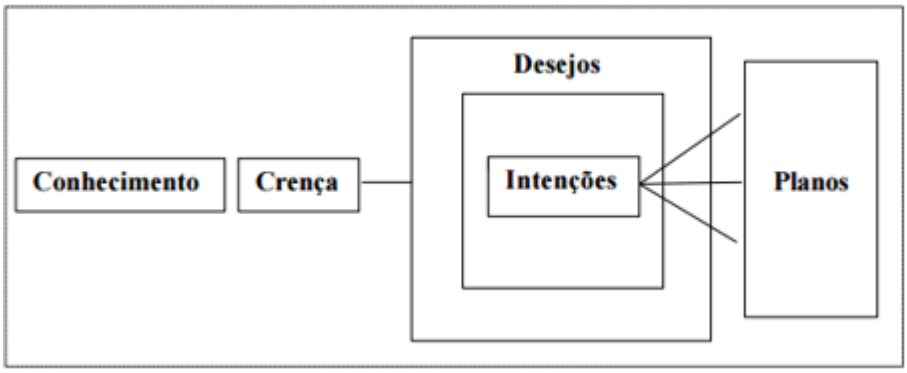
As crenças de um agente consistem na percepção que ele possui sobre o ambiente em que se encontra (GIRARDI, 2004), usados para expressar suas expectativas sobre seus possíveis estados futuros. As crenças de um agente podem ser vistas como um componente informativo do sistema (JUCHEM; BASTOS, 2001).

A motivação para agir e realizar metas são representadas pelos desejos de um agente (SHHEIBIA, 2001).

As intenções são um subconjunto de desejos e quando o agente resolve seguir uma meta específica, ela se torna uma intenção (GIRARDI, 2004).

A figura a seguir ilustra os componentes principais de uma arquitetura BDI.

Figura 3 - Arquitetura BDI



Fonte: Ronaldo Filho 2008

2.2.2 Arquiteturas Reativas

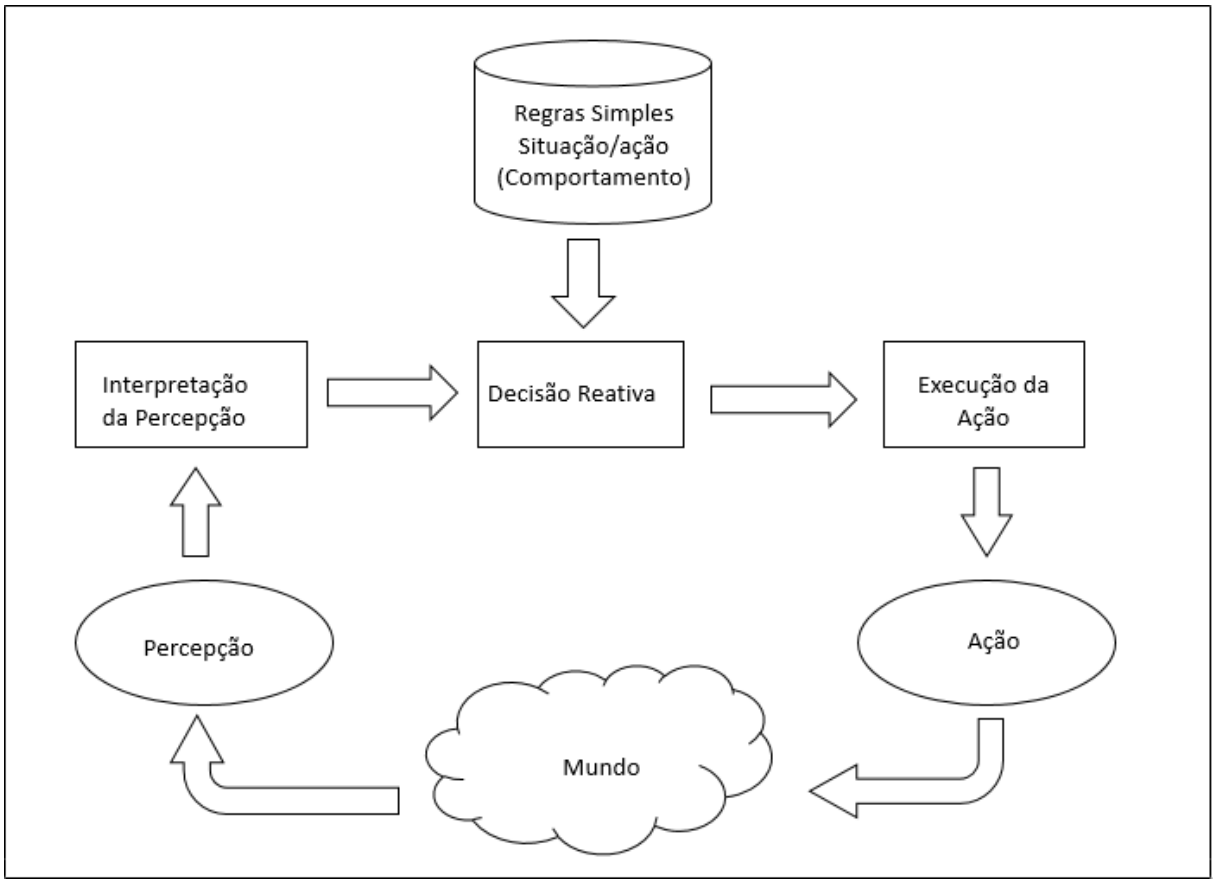
Uma arquitetura reativa é definida pela resposta do agente em relação a estímulos do ambiente captados por seus sensores (CORREA FILHO, 1994). O agente não tem conhecimento do ambiente e não utiliza raciocínio lógico, mas espera-se que sejam tomadas decisões inteligentes a partir dessas ações e reações.

Uma definição menos rigorosa feita por Moffat (1993) é que um agente é dito reativo se não necessariamente planeja tudo, mas pode apenas algumas vezes reagir a estímulos.

Existe ainda a abordagem de arquiteturas puramente reativas, onde os agentes reagem apenas a estimulos bem definidos por meio de um conjunto de ações pré-programadas, sem esboçar qualquer mecanismo de análise da situação (SHHEIBIA, 2001).

As arquiteturas reativas possuem simplicidade, baixo custo de processamento e boa robustez contra falhas (WOOLDRIDGE; JENNINGS, 1995), mas também tem desvantagens evidentes que impossibilitam os agentes de executar planos a longo prazo.

Figura 4 - Funcionamento da Arquitetura Reativas



Fonte: Adaptado de Wooldridge (1995)

2.2.3 Arquiteturas Híbridas

Como o próprio nome diz, as arquiteturas Híbridas combinam características das duas principais arquiteturas anteriores. Tanto a arquitetura Deliberativa quanto a Reativa possuem suas limitações (WOOLDRIDGE; JENNINGS, 1995), a primeira é muitas vezes incapaz de realizar uma ação imediata a um estímulo devido ao seu mecanismo complexo de raciocínio simbólico e a segunda apresenta limitação significativa em relação a implementação de um comportamento orientado por objetivos.

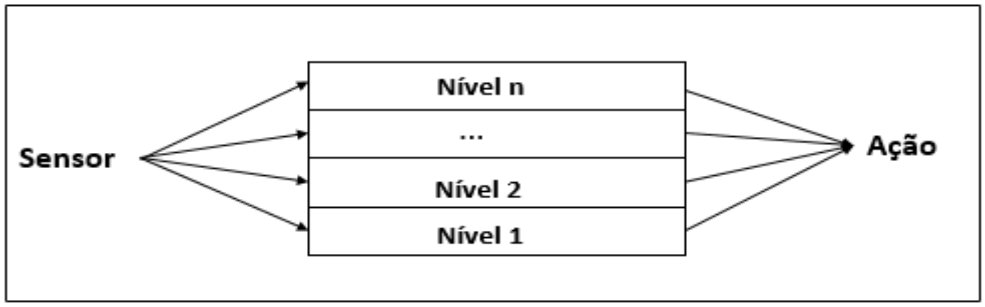
A arquitetura Híbrida estrutura um agente em dois subsistemas: um deliberativo que contém um modelo do ambiente capaz de desenvolver planos e tomar decisões pelo seu raciocínio e um reativo, que age a apenas em situações no ambiente sem precisar raciocinar (BERNSTEIN, 1982). A combinação destas virtudes resulta em uma arquitetura composta por níveis de modo a dispor hierarquicamente estes tipos e definir suas prioridades.

Outra possibilidade é a divisão em camadas, sendo constituída por:

- Camada Horizontal: Cada camada está diretamente conectada ao sensor de entrada e saída, atua como um agente e apresenta soluções em relação as ações a serem tomadas.
- Camada Vertical: Os sensores de entrada e saída estão localizados no máximo em uma camada

Uma grande vantagem da camada horizontal é o seu conceito de simplicidade: um agente com n tipos de comportamentos diferentes é estruturado em n camadas diferentes (WOOLDRIDGE; JENNINGS, 1995). Porém como todas camadas estão gerando sugestões de ações, uma função mediadora é inclusa para decidir qual camada tem controle sobre o agente.

Figura 5 - Camada Horizontal de um Arquitetura Híbrida

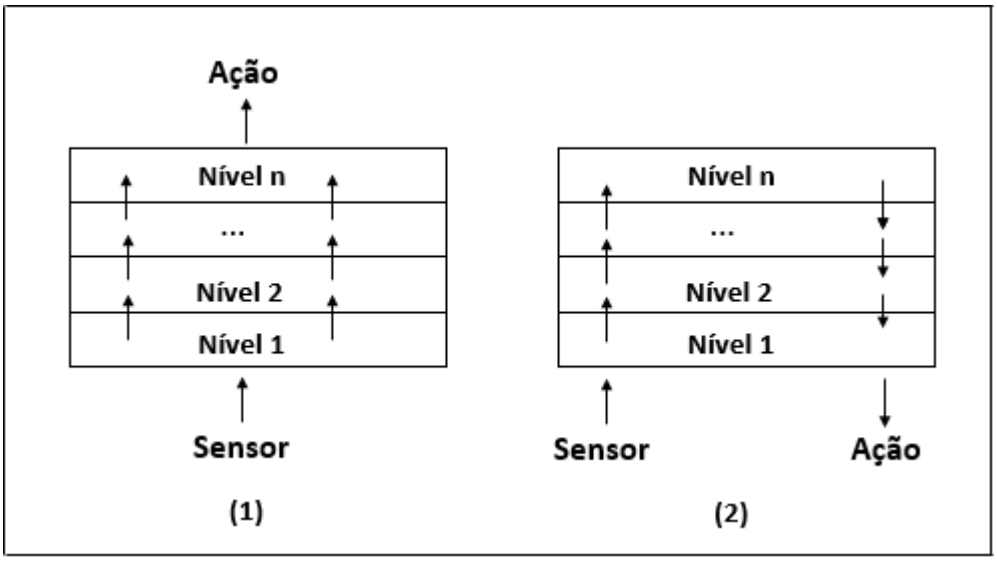


Fonte: Adaptado de Wooldridge (1995)

As camadas verticais podem ser de uma ou duas passagens, elas ajudam em parte a tomada de decisão dos agentes. Nas camadas de uma passagem, o controle flui sequencialmente através de cada camada, até que a camada final gera a ação de saída (GIRARDI, 2004). Nas arquiteturas de duas passagens a informação flui até chegar a última camada e o controle flui novamente para baixo.

Desse modo, para que a camada vertical de duas passagens tome uma decisão, o controle deve passar entre cada uma das diferentes camadas o que demonstra pouca flexibilidade pois se houver falha em uma das camadas o agente terá problemas na execução de sua ação.

Figura 6 - Camada Vertical de uma passagem (1) e duas passagens (2)



Fonte: Adaptado de Wooldridge (1995)

2.3 AMBIENTE

Para o projeto e construção de um agente inteligente é preciso definir o ambiente em que essas tarefas serão realizadas.

A percepção de ambiente de um agente determina quais serão sua ações a serem tomadas, um agente fora do escopo de um ambiente do qual foi planejado provavelmente não conseguirá cumprir com seus objetivos pois para cada situação em seu meio definido existe um tipo de ação a ser tomada.

O ambiente de tarefas de um agente pode ser especificado por quatro elementos representado pela sigla PEAS (*Performance, Environment, Actuators, Sensors*), ou seja, desempenho, ambiente, atuadores e sensores (RUSSELL; NORVIG; INTELLIGENCE, 1995).

Os ambientes aparecem em várias formas, são eles os citados pelo autor Ronaldo Filho (2008):

- Completamente observável versus parcialmente observável: Um ambiente completamente observável é aquele em que os sensores captam todos aspectos relevantes do ambiente, por outro lado um ambiente parcialmente observável apresenta deficiência em sua total percepção que pode ser causado por ruídos ou deficiências em sua observação.
- Determinístico versus Estocástico: Se o próximo estado é determinado pelo

estado atual e pela ação do agente ele é classificado como determinístico, diferentemente de um ambiente estocástico.

- Episódico versus Sequencial: Em um ambiente episódico a percepção e ação dependem exclusivamente do momento (episódio). Em ambientes sequenciais, as ações atuais podem afetar ações futuras.
- Estático versus Dinâmico: Se o ambiente pode mudar enquanto o agente está atuando, ele é um ambiente dinâmico para o agente, caso contrário ele é estático.
- Discreto versus Contínuo: O ambiente discreto possui um número finito de estados distintos, como por exemplo um jogo de xadrez que possui um número fixo de movimentos possíveis a cada jogada enquanto o contínuo muda suas percepções e ações em um espectro continuo de valores.

2.4 APLICAÇÃO

Agentes inteligentes estão presentes em nosso dia a dia com grande frequência, seja no meio de entretenimento com jogos digitais ou até em sistemas de automação mais complexos onde não envolvem apenas problemas de estudo computacional mas sim sua aplicação direta em áreas como medicina, transportes, agricultura, simulações, controle de robôs e diversos outros serviços.

A simulação com base em agentes inteligentes tem sido cada vez mais utilizada ao longo dos tempos, não só pelas suas características, mas também pelas potencialidades do seu uso quando aplicado nas diferentes áreas em que é possível sua inclusão (FARINHA, 2013).

Para realização de simulações, a construção de ferramentas preparadas para simular cenários e situações evoluiu com a necessidade de representação em diversas áreas do conhecimento. Dentre estas ferramentas de simulação estão o RoboCup Soccer (ROBOCUP, 2010) cujo objetivo é programar agentes para vencer uma partida de futebol; o RCR (ROBOCUP RESCUE, 2006) para resolver problemas de resgate de vítimas em catástrofes. Existe também o Robocode (LARSEN, 2010) que é um simulador de batalhas entre agentes e serviu como ideia para a confecção deste trabalho.

No simulador Robocode, cada agente é representado por um robô tanque de guerra cujo objetivo é derrotar seu oponente durante uma batalha. O próprio jogador

é responsável por programar o conjunto de regras e definir uma estratégia de comportamentos para que seu robô seja vencedor. Neste trabalho, a configuração de movimentos do agente é realizada utilizando a combinação de técnicas de otimização apresentadas no capítulo seguinte.

3 TÉCNICAS DE OTIMIZAÇÃO

A área de Otimização, dentro da Computação, é bastante ampla e abrange variados problemas em matemática, engenharia, simulação, economia, para citar apenas algumas áreas. Neste trabalho, o objetivo de criar automaticamente um comportamento significa criar código, ou configurar código, de maneira que cada solução represente um agente diferente.

A otimização sempre se refere a maximização ou minimização de uma função que contém um conjunto de variáveis, como por exempo as coordenadas de movimentos realizados por um agente no simulador Robocode. Dentre estes métodos, encontram-se os determinísticos e os probabilísticos.

O foco do desenvolvimento deste projeto será na utilização de métodos probabilísticos. Os métodos de otimização probabilísticos utilizam dados e parâmetros variáveis como forma de se chegar ao objetivo

Neste capítulo serão apresentadas duas técnicas que foram consideradas importantes dentro do escopo do trabalho.

3.1 PROGRAMAÇÃO GENÉTICA

A Programação Genética (GP – *Genetic Programming*) foi desenvolvida por John Koza (1995) como uma extensão do estudo de Algoritmos Genéticos (*GA – Genetic Algorithm*) criados por John Holland (1984) e baseada nos princípios da seleção natural e evolução das espécies descritos por Darwin (1900).

Um indivíduo melhor adaptado ao ambiente tem mais chances de sobreviver, perpetuar seus genes e contribuir para a disseminação de suas características que foram essenciais para sua sobrevivência e desse modo, criar gerações cada vez mais adaptadas que suas anteriores (DARWIN, 1900).

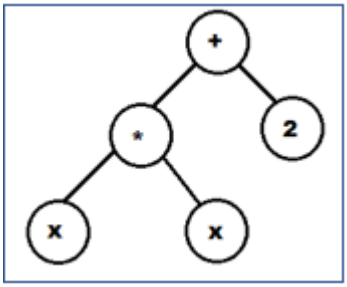
A PG pode ser definida como um método automatizado de gerar um programa de computador que apresente uma solução otimizada para um dado problema (SARAIVA, 2014). Cada programa é associado a um valor de mérito (*fitness*) que significa o quão bem ele realiza ou executa sua tarefa estabelecida, diretamente relacionado com sua capacidade de adaptação.

Normalmente os indivíduos são representados em forma de uma árvore de sintaxe abstrata, isto é, são formados por funções e terminais que constituem suas características e definem o comportamento no ambiente para o qual foi desenvolvido (MAIA JR; BIANCHI, 2001).

O conjunto de F (funções) aparece nos vértices internos da árvore e pode conter operadores aritméticos, lógicos, condicionais, funções matemáticas ou específicas de domínio do problema, enquanto o conjunto T (terminais) representa as folhas da árvore e é composto por variáveis ou constantes.

Como exemplo, a expressão matemática x*x+2 possui os conjuntos F e T, onde F = {*, +} e T = {x, 2}. Sua representação por meio de uma árvore de sintaxe abstrata é mostrada na imagem a seguir.

Figura 7 - Árvore de Sintaxe



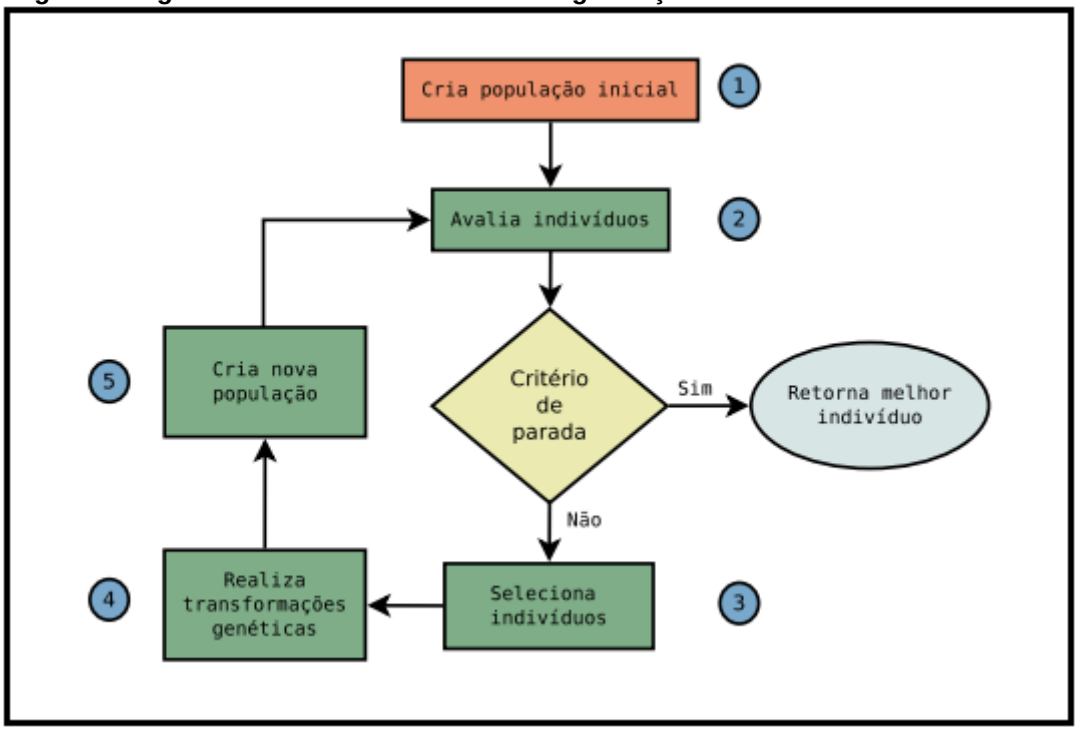
Fonte: Autoria Própria

3.1.1 Funcionamento

O algoritmo de PG é simples e pode ser descrito resumidamente em cinco etapas (SARAIVA, 2014) :

- Cria a população inicial;
- Determina o fitness de cada indivíduo (avaliação);
- Seleciona indivíduo;
- Realiza operações genéticas;
- Cria nova população e verifica se a condição de término foi atendida.

Figura - Diagrama de Funcionamento da Programação Genética



Fonte: Saraiva (2014)

3.1.1.1 População inicial

A população inicial é composta por árvores geradas aleatoriamente a partir dos conjuntos F e T. Inicialmente escolhe-se uma função pertencente a F e para cada argumento dessa função é escolhido um elemento do conjunto união de F e T (BRUHA, 2003). Este processo continua até que a árvore seja constituída apenas de símbolos terminais como nós-folha onde possui um valor limite de profundidade pré-estabelecido.

Para essa geração aleatória, Koza (1992) propõe três métodos:

Método *Full*: Cria árvores completas onde todas as folhas possuem a mesma distância até a raiz, sendo tal distância igual a profundidade máxima da árvore.

Método *Grow*: Cria árvores de profundidade variável em que a escolha dos nós é feita de forma aleatória entre funções e terminais. Caso o valor limite de profundidade seja atingido será selecionado um terminal ao invés de um nó.

Método *Ramped-half-and-half*: Combina os métodos Full e Grow e gera um número igual de árvores para cada profundidade, entre 2 e a profundidade máxima.

3.1.1.2 Avaliação de aptidão (fitness) e seleção

A função fitness representa a medida de adaptação de um indivíduo ao seu meio (GALASTRI, 2003) e retorna para cada um destes um valor real resultante entre o confronto dos valores de entrada e o valor de saída esperado. Quanto mais próximo o valor retornado estiver do valor de saída, melhor é o programa e maior será sua chance de ser selecionado para reproduzir.

A função de fitness é fundamental para resolução do problema pois é através dela que o programa irá se adaptar de forma a obter um resultado satisfatório, ou seja, essa função é responsável por definir o rumo que o algoritmo desenvolvido toma para criar suas soluções. Koza (1992) define quatro métodos para a avaliação de fitness:

- *Raw Fitness* (Aptidão Nata): Simples avaliação em cima dos valores de fitness obtidos, avaliando cada indivíduo de modo a formar um *ranking* de valores.

- *Standardized Fitness* (Aptidão Padronizada): Para o raw fitness os valores podem ser pequenos se comparados a erros ou grandes caso seja avaliado uma taxa de eficiência (KOZA, 1992). O *standardized* fitness garante uma padronização de métrica, onde quanto menor o valor numérico mais adaptado é o indivíduo. Em casos de avaliação de erros, tanto o *raw fitness* quanto o *standardized fitness* terão valores idênticos (SARAIVA, 2014), a não ser que que o *raw fitness* tenha valor 0, neste caso é subtraído um valor constante correpondente ao valor mínimo que o *raw fitness* pode atingir, sendo possível o *standardized fitness* atingir valor 0. Para casos de medidas de eficiência, ou o *raw fitness* já apresenta uma medida de sucesso ou o *standardized fitness* deve ser calculado a partir da subtração do *raw fitness* por um valor constante correspondente ao valor máximo que o mesmo pode atingir.

- *Adjusted Fitness* (Aptidão Ajustada): Obtido através do *standarized fitness*, seu valor estará sempre no intervalo [0,1], onde os melhores indivíduos são representados pelos maiores valores (ZUBEN, VON, [S.D.]).

- *Normalized Fitness* (Aptidão Normalizada): Possui valor contido no intervalo [0, 1] sendo que os maiores valores pertencem aos melhores indivíduos.

3.1.1.3 Operadores genéticos

Após a seleção dos indivíduos por meio da função de *fitness* os operadores genéticos tratam da criação de novos indivíduos derivados de indivíduos anteriores. Koza (1992) define três operadores principais: os operadores de Reprodução, Cruzamento (*Crossover*) e Mutação.

Reprodução: Um indivíduo é selecionado e feito uma cópia do mesmo para próxima geração sem alterações em seu código. Existem vários métodos de seleção, o mais popular é o método de seleção proporcional apresentado por John Holland (1984) e escolhido por Koza em seu primeiro livro (KOZA, 1992).

Esse método também é conhecido como método da roleta, onde cada indivíduo ocupa uma fatia proporcional ao seu fitness. Os indivíduos são selecionados como se um roleta estivesse sendo girada, aqueles que ocupam uma maior fatia tem mais chance de serem selecionados.

A Figura 10 a seguir mostra quatro indivíduos com suas respectivas funções de aptidão e suas proporções ocupadas em um gráfico de setores.

Figura 8 - Método da Seleção Proporcional (Roleta)

Item	Aptidão/ Fitness	Chance de Escolha
1	150	30%
2	50	10%
3	200	40%
4	100	20%
Total	**500**	**100%**

Fonte: Autoria Própria

Outros métodos também utilizados são (BLICKLE, 1996): Seleção por Torneio (*Tournament Selection*), Seleção por Truncamento (*Truncation Selection*), Seleção por Nivelamento Linear (*Linear Ranking Selection*) e Seleção por Nivelamento Exponencial (*Exponential Ranking Selection*).

A Seleção por Torneio é feita escolhendo-se indivíduos aleatoriamente e os colocando para um confronto. O indíviduo que se apresenta mais proximo da função de fitness estabelecida é selecionado para próxima geração.

A Seleção por Truncamento escolhe aleatoriamente o melhores indivíduos dentro de um conjunto e descarta o restante (GRITZ; HAHN, 1995).

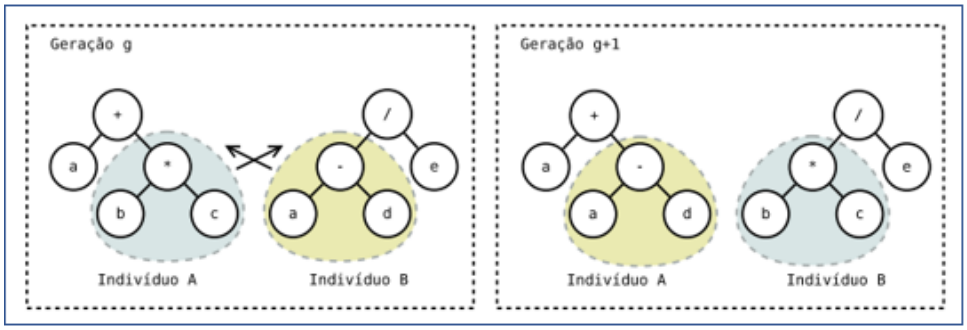
Para eleminar as desvantagens do uso de seleção desproporcional, a Seleção por Nivelamento Linear sugere que os indivíduos sejam ordenados pelo valor de fitness (ZUBEN, VON, [S.D.]). Ao melhor é atribuído o nível 1 e os demais também são classificados em relação a níveis, onde possuem menores probabilidades de seleção que seus antecessores.

A Seleção por Nivelamento Exponencial difere da Linear apenas pelo fato de apresentar probabilidades exponencialmente ponderadas (BLICKLE, 1996), quanto mais próximo do valor 1 menor é sua exponencialidade.

*Crossove*r: Essa operação aumenta a variedade genética na população ao inserir novos elementos na mesma (GALASTRI, 2003). Dois indivíduos são selecionados e recombinados para gerar outros dois: os melhores indivíduos de cada grupo são selecionados como pais e escolhe-se aleatoriamente um ponto de secção na árvore para serem trocados e resultar em dois novos filhos.

Os novos indivíduos podem obter os melhores fragmentos dos seus pais e, portanto, podem superá-los e prover uma melhor solução para o problema (SARAIVA, 2014).

Figura 9 - Operação de crossover entre indivíduos de uma população



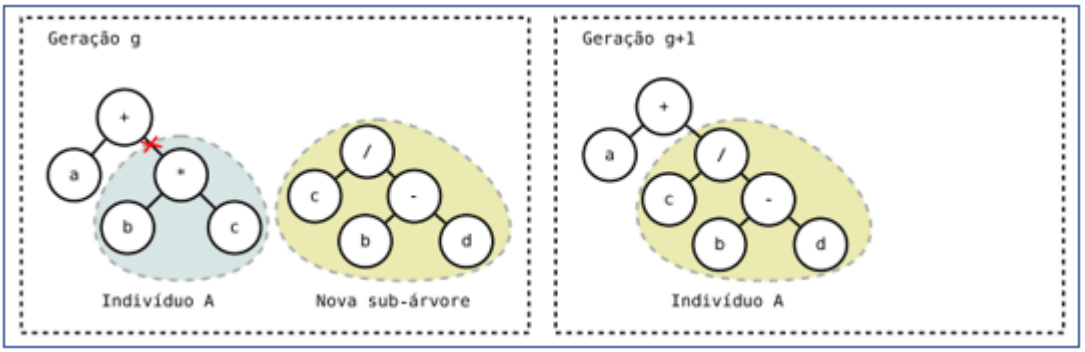
Fonte: Saraiva (2014)

O conjunto de funções terminais devem ser compatíveis para que o cruzamento seja possível, o nome que satisfaz essa propriedade é definido como função de Fechamento ou Closure por Koza (1995).

Mutação: Acontece com uma probabilidade muito baixa e permite uma variabilidade na população. Pode ser exemplificado como uma característica especial de um indivíduo numa população. Tal mutação pode até ser responsável por criar novos indivíduos com genes considerados importantes no processo de evolução.

Uma provável diminuição no valor de aptidão desse indivíduo é esperada, pois esse novo material genético ainda não foi testado.

Figura 10 - Operação de mutação em um indivíduo de uma população



Fonte: Saraiva (2014)

3.1.1.4 Nova população e critério de parada

Com todo esse processo de operações genéticas descrito anteriormente, as novas gerações tendem a ser cada vez melhores que as anteriores, perpetuando os genes essenciais para a sobrevivência até então.

O laço de repetição do processo evolutivo é infinito e necessita ser interrompido de alguma maneira. Para Koza (1992) há duas maneiras de fazer isso. A primeira é limitar o número máximo de gerações, muito utilizado em problemas que não esperam resultado exato, tais como ajustes de curvas, ou para problemas que não se sabe determinar o resultado final (GALASTRI, 2003).

A segunda maneira é encontrar um indivíduo que satisfaça perfeitamente a função objetivo, utilizado em problemas que possuem uma resposta exata.

3.2 RECOZIMENTO SIMULADO (SIMULATED ANNEALING)

O Recozimento Simulado é um método de otimização global que tem seus conceitos fundamentados na prática da indústria metalúrgica, no qual consiste em um resfriamento lento e gradativo de sólidos metálicos em temperaturas muito altas.

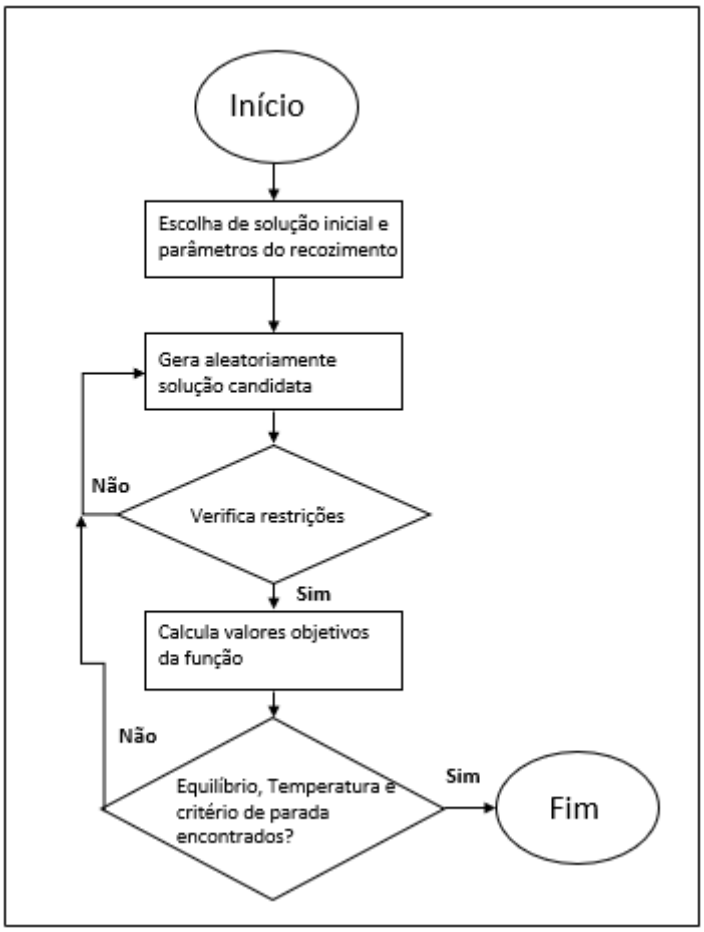
Quando o metal encontra-se com a temperatura alta suas móleculas encontram-se em um estado de agitação, fazendo com que a energia interna seja muito grande. Ao resfriá-lo gradativamente as móleculas perdem aos poucos a quantidade de energia, fazendo com que se solidifique de modo a manter a estrutura estável.

Noronha (2000) descreve o estado resfriado de um problema combinatório quando não há mais possibilidade de se achar uma melhor solução continuando a exploração. Novas soluções encontradas podem ou não serem aceitas.

Este procedimento tem similaridade com a otimização por melhoria iterativa, como algoritmos genéticos, onde o ponto ótimo será aquele com menor variação de energia, ou seja, o ponto em que o sistema se encontra estável.

A Figura 11, demonstra o fluxo dos processos realizados pelo método do Recozimento Simulado até a obtenção dos valores ótimos esperados.

Figura 11 - Fluxo Recozimento Simulado



Fonte: Adaptado de Noronha (2000)

À medida em que o algoritmo de otimização é executado, ele reduz a “temperatura” do sistema. Isso significa que as mutações vão se tornando mais suaves, até um ponto em que, se a temperatura chegar a zero, não haveria mais mutação nenhuma. A justificativa para o funcionamento do algoritmo é começar fazendo uma busca de várias alternativas muito diferentes entre si e, com o tempo, passar a fazer pequenos ajustes em soluções que se apresentam como boas.

As ‘mutações’ aplicadas não necessariamente precisam seguir ideias de algoritmos genéticos, como crossover e recombinação, embora nada também impeça seu uso.

4 SIMULADOR ROBOCODE

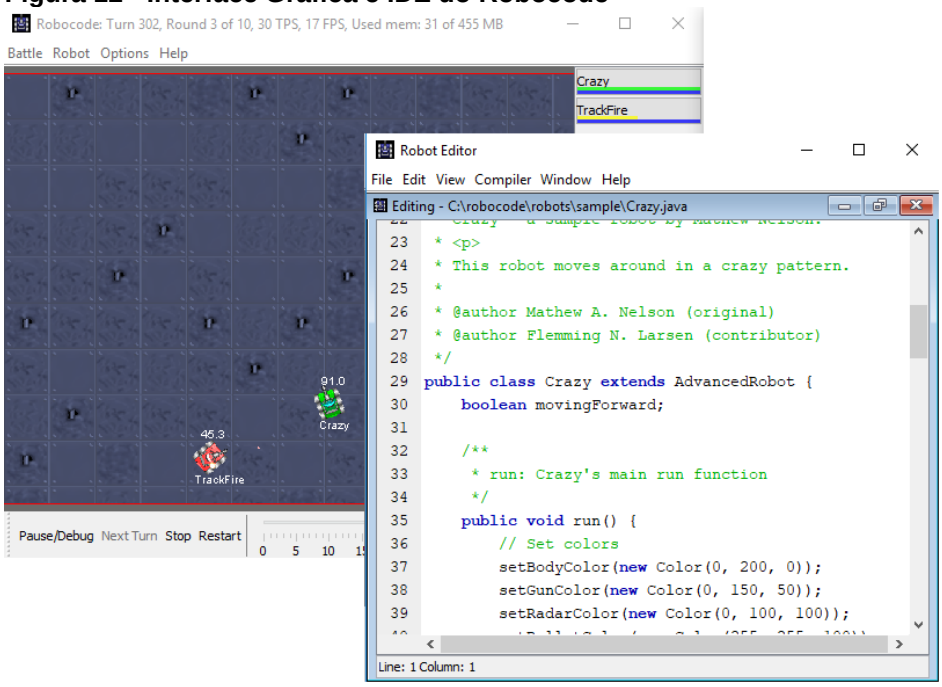
O Robocode é um jogo de programação que foi inicialmente desenvolvido para o aprendizado da linguagem Java e posteriormente outras linguagens como C# e Scala.

O objetivo do jogo é programar um robô tanque de batalha para competir contra outros robôs em uma arena. O jogador é o programador que faz o desenvolvimento de robôs autônomos simulados, definindo o conjunto de regras para que os robôs operem disputas virtuais (ROBOCODE, 2013).

4.1 FUNCIONAMENTO

O Robocode possui uma IDE (Integrated Development Environment) própria para codificação ou modificação de robôs e uma interface onde é possível executar e visualizar os combates.

Figura 12 - Interface Gráfica e IDE do Robocode



Fonte: Autoria Própria

Cada robô possui o método de execução run() que inicia sempre junto a um novo turno e é responsável pelas principais ações do agente. Dentro deste método

existem instruções responsáveis pela estratégia dos robôs, onde são executadas até que um evento seja capturado (ALAIBA & ROTARU, 2008).

Os eventos são responsáveis por rastrear e captar informações do ambiente como: colisões, detecção de inimigo, início e fim de turnos, danos e etc. São chamados quando algo específico acontece ao decorrer do combate.

O Quadro a seguir demonstra os principais métodos e eventos utilizados pelo Robocode.

Quadro 1 - Principais Métodos e Eventos Utilizados pelo Robocode

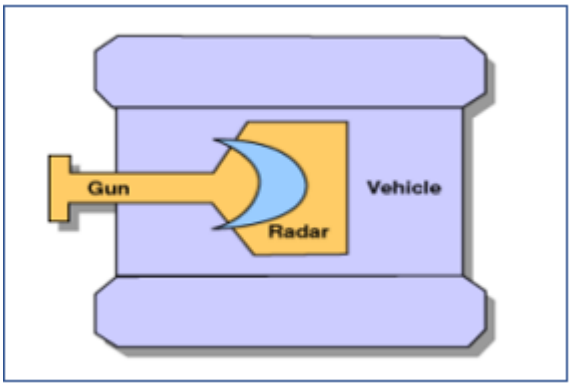
Tipo	Comando	Descrição
Método	ahead(double)	Movimenta para frente a distância passada por parâmetro.
Método	back(double)	Movimenta para trás a distância passada por parâmetro.
Método	turnRight/turnLeft(double)	Gira o robô para a direita e esquerda respectivamente de acordo com o valor em graus passado por parâmetro.
Evento	run	Executado quando um turno é iniciado.
Evento	onScannedRobot	Executado quando o radar detecta o inimigo.
Evento	onHitRobot	Executado quando há colisão entre robôs.
Evento	onHitWall	Executado quando há colisão nas paredes.
Evento	onHitBullet	Executando quando o robô é atingido por uma bala.

Fonte: Autoria Própria

A estrutura do tanque é composta por três elementos:

- Corpo: responsável pelo movimento e rotação do robô. Suas principais operações são: mover para frente, para trás, lado esquerdo, lado direito e parar;
- Canhão: realiza disparos e gira para os lados esquerdo e direito;
- Radar: localiza inimigos na arena e movimenta-se junto com o canhão ou de forma independente.

Figura 13 - Estrutura de um Robô Tanque



Fonte: http://robocode.sourceforge.net

O simulador possui uma lista de regras implementadas em relação a pontuação. Essas regras possuem algumas restrições:

Restrição de energia: Pode-se definir a quantidade de energia que será utilizada a cada disparo realizado, sendo proporcional a sua potência e recuperada quando um inimigo é atingido. Ao ser atingido por um inimigo o decréscimo da energia é quatro vezes maior que a potência do disparo que varia entre 1 e 2, caso a potência seja maior que 2 o cálculo de dano é feito pela seguinte expressão 2*(potência-1);

Restrição de Calor: O canhão só dispara quando sua temperatura estiver em zero. O calor gerado é proporcional a potência do disparo.

Cada robô implementado deve ser diretamente ou indiretamente herança de umas das classes definidas pelo Robocode. As três classes disponíveis são:

- JuniorRobot: É o tipo mais básico de robô. Seu modelo simplificado tem o propósito de ensinar conceitos de programação a iniciantes. Apresenta restrição de acesso a vários métodos e eventos de forma a ocultar as regras e cálculos complexos que o Robocode utiliza.
- Robot: Possui acesso a uma lista um pouco restrita de métodos e eventos. Permite programar robôs com comportamentos mais inteligentes que os da classe JuniorRobot.
- AdvancedRobot: É o mais avançado tipo de robô. Permite acesso a diversos recursos, como por exemplo: modificar métodos e eventos, escrever e ler arquivos e obter informações mais precisas sobre coordenadas em relação ao inimigo e campo de batalha.

A figura 14 apresenta o código de um robô que se estende da classe

JuniorRobot, o mesmo é chamado dentro do Robocode para executar suas ações.

Figura 14 - Código de um JuniorRobot

```
1   package sample;
2   import robocode.JuniorRobot;
3
4   public class MyFirstJuniorRobot extends JuniorRobot {
5
6       public void run() {
7
8           setColors(green, black, blue);
9
10          while (true) {
11              ahead(100);
12              turnGunRight(360);
13              back(100);
14              turnGunRight(360);
15          }
16      }
17
18      public void onScannedRobot() {
19          fire(1);
20      }
21
22      public void onHitByBullet() {
23          turnAheadLeft(100, 90);
24      }
25  }
```

Fonte: Adaptado de Mathew A. Nelson (2001)

Como pode ser visto na linha 6 do código, o método run() é chamado e dentro dele um loop com instruções é executado até que a batalha termine ou um evento seja capturado. Quando um evento é capturado, interrompe-se a execução do laço, executa-se o código do evento e retorna à execução de instruções do loop de onde parou. Entende-se como fim da batalha, a derrota de um robô.

É possível visualizar na linha 18 um evento que sempre atira com poder de fogo igual a 1 no inimigo quando o mesmo é localizado. Na linha 22 o robô se movimenta 100 pixels para frente e rotaciona 90 graus para esquerda quando percebe que foi atingido.

É possível manipular a velocidade de animação gráfica das batalhas de 0 a 1000 frames por segundo. Caso a janela da interface gráfica seja minimizada, a aplicação passa a ser executada em modo console exibindo apenas os logs da batalha.

5 DESENVOLVIMENTO

Esse capítulo descreve como se deu a implementação e como foram realizados testes do programa.

Para testar as técnicas descritas nos capítulos anteriores, foi desenvolvido código interligado ao próprio programa Robocode versão 1.9.3.2. Uma vez que toda execução do projeto depende desse ambiente, será explicado a seguir como é realizada a configuração do mesmo, visando deixar documentado para outras pessoas que decidam fazer testes similares.

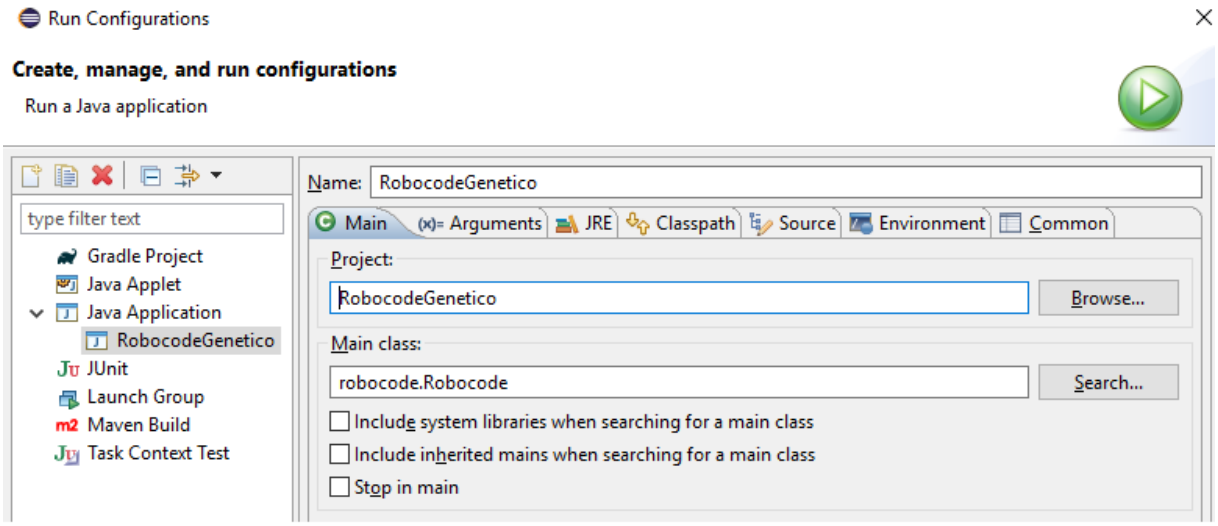
Um novo projeto Java foi criado no ambiente de desenvolvimento Eclipse e todo conteúdo do diretório do software Robocode foi copiado para o mesmo *workspace* do novo projeto.

As bibliotecas responsáveis pelo funcionamento do jogo foram adicionadas ao Java Build Path, de modo que o novo projeto tenha permissão a utilizar todos recursos originais como: interface gráfica, sons, configurações, regras e classes contendo métodos e eventos necessários para o desenvolvimento.

Para gerar uma batalha é necessário a criação de um arquivo de extensão .battle que contém propriedades indispensáveis para a batalha, como por exemplo: tamanho do campo e a localização (pacote) onde se encontra o novo robô e o adversário desejado.

Para executar o código no servidor local como uma aplicação Java é preciso configurar os parâmetros de Build apontando para classe principal do Robocode, conforme figura abaixo:

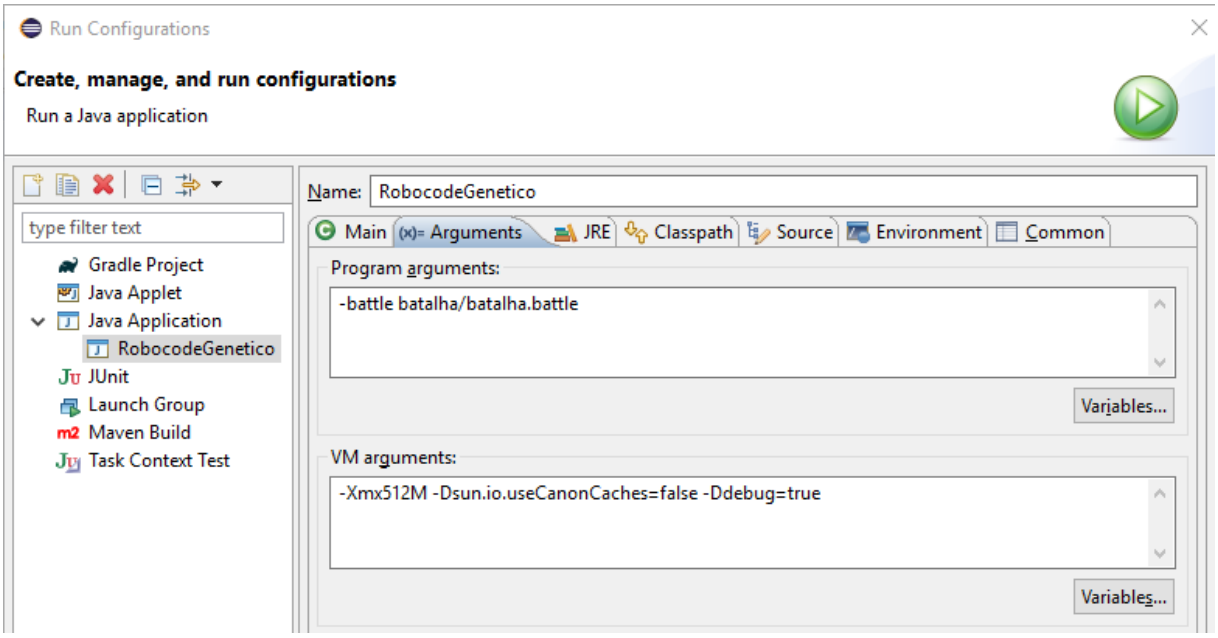
Figura 15 - Parametrização de Build



Fonte: Autoria Própria

É necessário também realizar a configuração dos argumentos referentes a batalha e Virtual Machine.

Figura 16 - Parametrização de Batalha e argumentos da VM



Fonte: Autoria Própria

Os agentes implementados no projeto correspondem ao modelo reativo simples, parecido com o definido por Moffat (1993). Embora o problema estudado, da batalha de robôs, abra possibilidade paras estudar agentes que realmente incorporem

técnicas de IA, neste projeto os algoritmos de Inteligência Artificial são executados off-line para dar origem a um robô que irá competir. Dessa maneira a execução do agente não é muito onerosa em termos de processamento durante a batalha, pois esse tempo foi gasto em separado, treinando um competidor. Outra possibilidade seria implementar agentes que, durante a batalha, aplicam técnicas de IA para “pensar” nas melhores soluções. Entretanto, essa solução poderia ser muito demorada e não foi definida como objetivo deste trabalho.

5.1 GERAÇÃO AUTOMÁTICA DE AGENTES

A forma mais geral da programação genética, apresentada no capítulo 3.1, consiste em gerar código fonte sem intervenção humana, aplicando aleatoriamente operações como mudança de operadores, troca de linhas, inserção e remoção de comandos.

Embora esse método possa gerar agentes com comportamentos interessantes, há uma probabilidade elevada de que seja criado muito código de baixa qualidade ou que não faça sentido, como loops infinitos, condições erradas (atacar quando o agente deveria fugir), etc.

Uma das maneiras de tratar esse problema é trabalhar com estruturas predefinidas em lugar de gerar todo o código a partir de instruções individuais.

Para Koza (1992), se uma estrutura particular é acreditada ou conhecida como importante, então pode-se modificar o sistema de PG para exigir que todos os indivíduos tenham essa estrutura.

A estrutura utilizada neste estudo não gera código e sim configurações, sendo possível descartar o uso de alguns passos presentes na PG, como técnicas de crossover e recombinação de material genético.

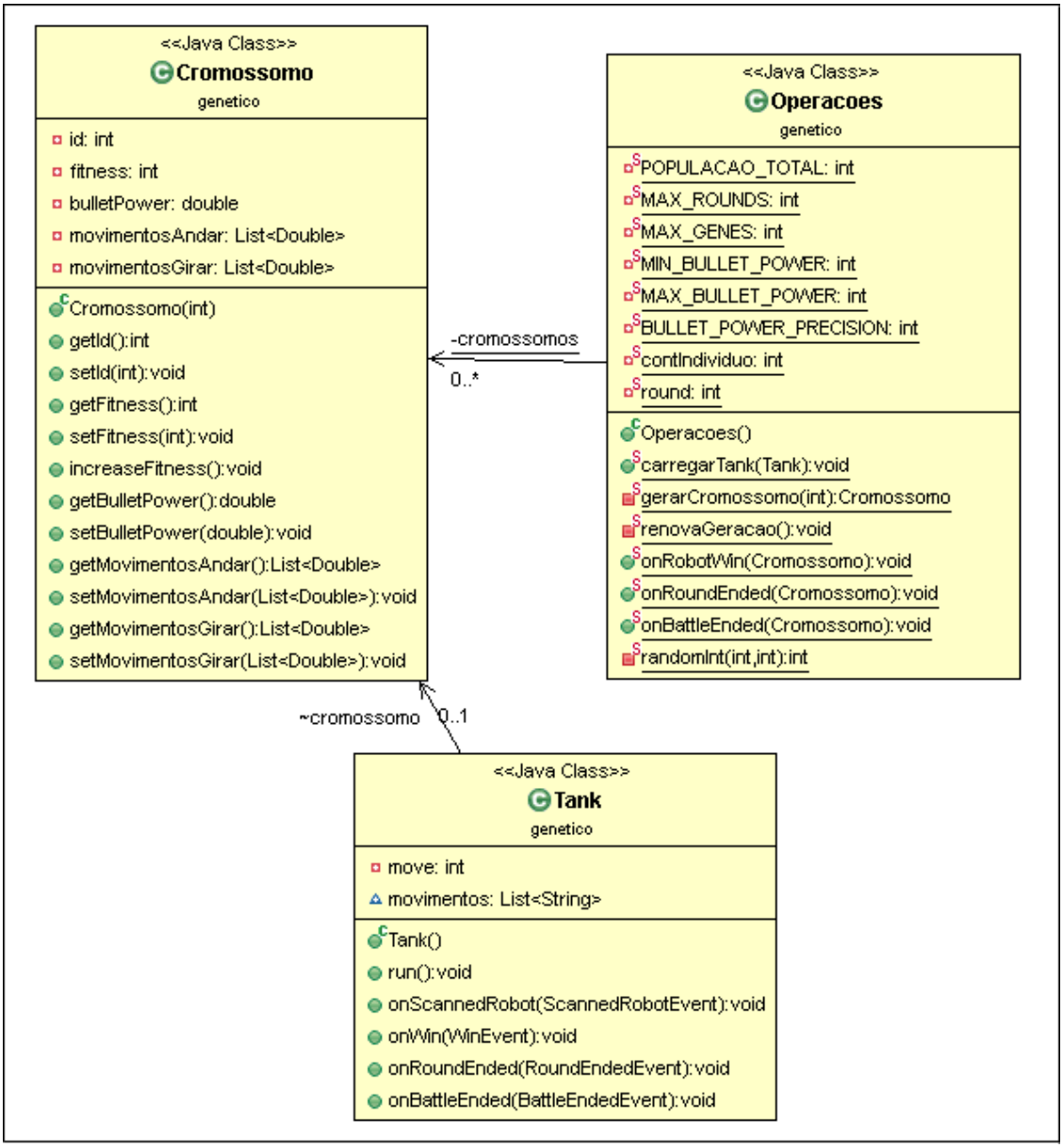
Para gerar um novo indivíduo, pequenas modificações nos genes são realizadas utilizando os conceitos de recozimento simulado. Essas mudanças representam reconfigurações de movimentos de forma a ajustar o comportamento do indivíduo à solução desejada.

Apesar de não utilizar os conceitos envolvendo operações de material genético, alguns termos específicos da PG foram utilizados no desenvolvimento deste

trabalho. Os cromossomos representam uma estrutura (objeto) composta por vários genes (atributos) os quais são responsáveis por definir as características particulares de cada indivíduo (robô).

O código desenvolvido é apresentado no diagrama UML a seguir:

Figura 17 - Diagrama UML do código desenvolvido



Fonte: Autoria Própria

A classe Cromossomo contém todos atributos que compõem um Robô (Tank): identificador (id), movimentos (movimentosAndar e movimentosGirar), poder de fogo (bulletPower) e valor de fitness (fitness).

A classe Operações é responsável por toda lógica executada dentro do método run() da classe Tank, onde a cada início de turno invoca o método carregarTank(), o qual é responsável por fazer a geração randômica inicial dos valores de movimentos e poder de fogo pelo método gerarCromossomo(). A cada fim de turno é verificado o resultado da batalha e acrescentado um ponto ao valor de fitness em caso de vitória, sendo estabelecido que cada robô deve batalhar 10 vezes para compor sua avaliação. Após todos indivíduos gerados batalharem, é verificado o valor de fitness apresentado. A função de fitness responsável por esta avaliação corresponde ao somatório do número de vitórias, possuindo neste caso, valor máximo igual a 10. Caso valor de fitness seja o máximo definido, significa que o indivíduo venceu todos combates e já satisfez o critério de parada, caso contrário, os cromossomos serão ordenados em ordem decrescente pelo seu fitness e uma nova população será gerada pelo método renovaGeracao().

Duas versões de código foram feitas para este método. A primeira utiliza apenas o robô de maior fitness para gerar uma quantidade fixa de mutações na população. A segunda implementação, um ajuste da primeira, gera uma quantidade variável de filhos de acordo com o valor de fitness de cada indivíduo.

Após implementação da primeira versão, verificou-se que a faixa de população a ser substituída era muito grande (mais da metade), praticamente não estavam acontecendo mutações e sim uma substituição parcial da população com base nas características do indivíduo mais forte. Como forma de corrigí-la, na segunda implementação algumas faixas de fitness foram definidas para criação dos filhos de cada indivíduo.

Para cada valor de fitness que o indivíduo apresenta ele possui um valor fixo de filhos que serão criados com base em suas características adicionando uma variação randômica entre -100 e 100 pixels para os movimentos de *ahead* e *back* e de -50 a 50 graus para rotações.

O pseudocódigo da correção pode ser visualizado na figura abaixo.

Figura 18 - Pseudocódigo do método renovaGeração()

```
INICIO
    int numeroFilhos;
    ENQUANTO (indiceRoboForte < indiceRoboFraco) FAÇA
        SE (indiceRoboForte.fitness > 6) numeroFilhos = 3;
        SENAO SE (6 <= indiceRoboForte.fitness >= 4) numeroFilhos = 2;
        SENAO numeroFilhos = 1;
        PARA k <- 1 ATÉ numeroFilhos FAÇA
            deleta indiceRoboFraco;
            gera filho de indiceRoboForte(Simulated Annealing);
            substitui indiceRoboFraco;
            -- indiceRoboFraco;
            SE (indiceRoboFraco = indiceRoboForte)
                break;
            FIM SE
        FIM PARA
        ++indiceRoboForte;
    FIM ENQUANTO
    reset robôs Fitness;
FIM ALGORITMO
```

Fonte: Autoria Própria

Os indivíduos mais fortes geram um número de filhos de acordo com seu valor de fitness. Durante esse processo de criação, os robôs mais fracos são substituídos por uma mutação dos mais fortes. Essa mutação é caracterizada por uma pequena variação randômica nos valores de movimentos, assim como é utilizado no método de Recozimento Simulado.

Cada vez que o método renovaGeracao() é chamado, indivíduos contendo características levemente modificadas dos mais fortes da iteração são criados. Este processo busca encontrar o resultado ótimo gradativamente de modo a manter o equilíbrio e estabilidade do sistema sem variações bruscas.

Após concluído esse processo de mutação, os valores de fitness são reiniciados, as batalhas retornam a acontecer e todo processo se repete novamente.

5.2 TESTES

Para realização dos testes, uma população total de 100 indivíduos foi gerada e cada um realizou 10 batalhas contra o robô SpinBot para compor a avaliação de fitness. Cada batalha tem o significado de um round, ou turno, e acaba somente quando um dos robôs é o vencedor.

O SpinBot, apesar de possuir um código simples de funcionamento é um dos robôs avançados disponíveis no simulador e apresenta um comportamento fixo definido. Ele se movimenta em círculos e sempre que obtém a localização do inimigo atira com o poder máximo de fogo. Se houver uma colisão, o SpinBot verifica se a causa foi seu movimento em direção ao oponente ou o contrário. Para o primeiro caso, seu comportamento é virar à direita com o objetivo de escapar ou continuar em frente ao inimigo para realizar disparos, este último se repete caso a colisão seja do adversário.

Figura 19 - Código de funcionamento do SpinBot

```
public class SpinBot extends AdvancedRobot {

    public void run() {

        setBodyColor(Color.blue);
        setGunColor(Color.blue);
        setRadarColor(Color.black);
        setScanColor(Color.yellow);

        while (true) {
            setTurnRight(10000);
            setMaxVelocity(5);
            ahead(10000);
        }
    }

    public void onScannedRobot(ScannedRobotEvent e) {
        fire(3);
    }

    public void onHitRobot(HitRobotEvent e) {
        if (e.getBearing() > -10 && e.getBearing() < 10) {
            fire(3);
        }
        if (e.isMyFault()) {
            turnRight(10);
        }
    }
}
```

Fonte: Adaptado de Mathew A. Nelson (2001)

A condição de parada realizada foi ao encontrar um indivíduo com valor de fitness igual a 10, ou seja, um indivíduo que ganhou 10 batalhas consecutivas contra o tanque SpinBot.

Foi usado um limite de 40 mil rounds, o equivalente ao máximo de 40 gerações caso nenhum indivíduo apto seja encontrado. As gerações se renovam sempre após todos robôs batalharem, ou seja, cada robô batalha 10 vezes para compor sua função de fitness. Numa população de 100 indivíduos isso equivale a 1000 batalhas.

5.3 RESULTADOS

Foram testadas as duas soluções implementadas para o algoritmo gerador de mutações. A primeira utilizando apenas o robô de maior valor de fitness para compor novos indivíduos e a segunda gerando uma quantidade de filhos para cada faixa de

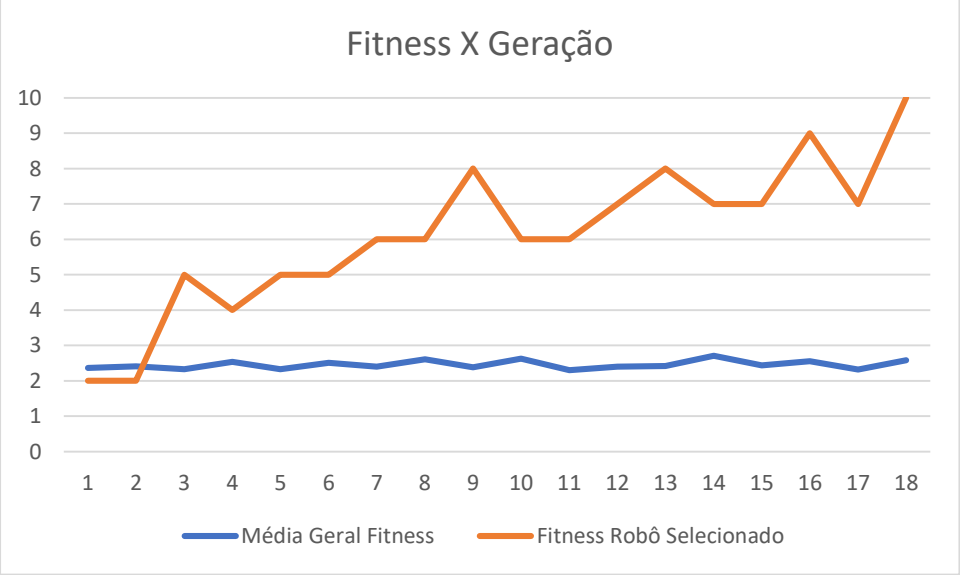
valor de fitness conforme descrito no capítulo 5 e podendo ser visualizado na Figura 17.

Para o primeiro caso obteve-se um indivíduo selecionado na 18ª geração, capaz de derrotar o oponente SpinBot em uma disputa de dez batalhas consecutivas.

Executando a aplicação em modo gráfico a previsão de término do programa seria de aproximadamente 1,5 minutos por batalha, ao total completando 18,75 dias, porém em modo console cerca de mil batalhas foram processadas por minuto.

No Gráfico 1 abaixo é possível acompanhar o histórico da evolução do ajuste de fitness do indivíduo selecionado e da média geral da população ao longo das 18 gerações.

Gráfico 1 - Histórico de Evolução de Fitness do Primeiro Teste

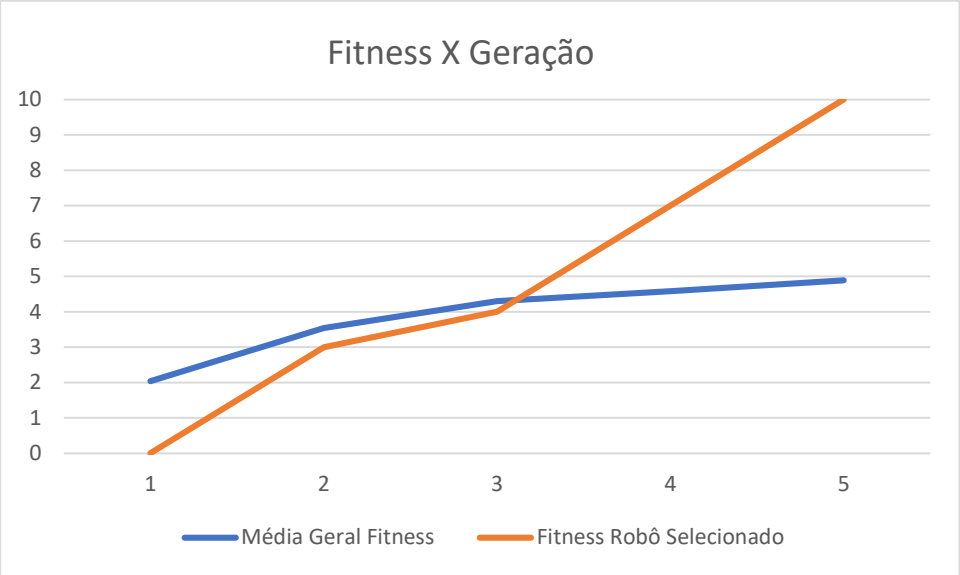


Fonte: Autoria Própria

Para segunda implementação a solução foi encontrada na 5ª geração, observando-se também uma melhora significativa da média geral de fitness da população como mostra o Gráfico 2 abaixo.

A redução no tempo de execução do programa caiu para 5 minutos em modo console e uma estimativa prevista de 7,5 dias caso executado em modo gráfico.

Gráfico 2 - Histórico de Evolução de Fitness do Segundo Teste



Fonte: Autoria Própria

6 CONSIDERAÇÕES FINAIS

O objetivo principal por trás do desenvolvimento do ambiente de simulação Robocode foi avaliar os resultados obtidos ao realizar mutações em combinação com a aplicação de uma metaheurística conhecida como Recozimento Simulado, além de propor uma nova ferramenta para estudo de técnicas de inteligência artificial em jogos.

Nos capítulos de 1 a 4 foi apresentado uma base teórica que envolve o contexto da aplicação desenvolvida, iniciando com o conceito sobre agentes e sua arquitetura, o simulador robocode e técnicas de otimização.

O capítulo 5 abordou a etapa de implementação do projeto, juntamente com seus testes e resultados obtidos.

Para os resultados, verificou-se que a segunda implementação do algoritmo gerador de agentes convergiu para solução ótima de forma aproximadamente 3 vezes mais rápida que a primeira versão do código. Tal otimização poder ser também verificada nos gráficos de média geral do fitness da população, onde a segunda versão manteve um crescimento constante equanto para o outro caso a média obteve-se em linha tênue sem melhoras significativas.

O Robocode mostrou-se uma boa alternativa ao desenvolvimento do trabalho, oferecendo bibliotecas e interfaces gráficas, o que permite ao aluno se concentrar de forma mais precisa na implementação de raciocínio dos agentes utilizando os conceitos de lógica, inteligência artificial e orientação a objeto de forma gradativa a entender o funcionamento do simulador.

Como trabalhos futuros, pode-se considerar aumentar a flexibilidade do gerador de código, podendo incluir mais comandos dos tanques e itens como loops e decisões, retirando-se o processo de variações randômicas e adicionando variações mais direcionadas e inteligentes para cada caso específico de tomada de decisão. Deve-se notar que ao incluir esses itens o problema cresce muito, não somente em relação a complexidade de código mas também em termos de processamento necessário para execução. Assim, outra possibilidade de estudo é analisar como reduzir esse número de combinações descartando apenas soluções de baixa qualidade.

REFERÊNCIAS

ALAIBA, V.; ROTARU, A. **Agent architecture for building Robocode players with SWI-Prolog**. Proceedings of the International Multiconference on Computer Science and Information Technology, 2008, p. 3-7.

BERNSTEIN, G. S. **Training behavior change agents: A conceptual review**. Behavior Therapy, 1982. v. 13, n. 1, p. 1–23.

BLICKLE, T. **Evolving compact solutions in genetic programming: A case study**. *In*: VOIGT, H.-M. *et al.* (Org.). Parallel Problem Solving from Nature — PPSN IV. Springer Berlin Heidelberg, 1996, p. 564–573.

BRUHA, I. **Some Enhancements in Genetic Learning: A Case Study on Initial Population**. *In*: ZHONG, N. *et al.* (Org.). Foundations of Intelligent Systems. Springer Berlin Heidelberg, 2003, p. 539–543.

CORREA FILHO, M. **A Arquitetura de Diálogos entre Agentes Cognitivos Distribuídos**. Rio de Janeiro: COPPE da UFRJ, 1994.

DARWIN, C. R. **On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life**. London: 1859.

DARWIN, C. **The origin of species, by Charles Darwin.** New York, Literary Classics, 1900.

DEMAZEAU, Y. **From Interactions To Collective Behaviour In Agent-Based Systems**.1995. p. 117–132.

FARINHA, P. M. L. **Modelos de simulação em MATSim aplicados à análise de sistemas de transporte**. dez. 2013.

FILHO, F. A. R. **Desenvolvimento de um Sistema 3D para Simulação de Comportamentos de Agentes Inteligentes**. 2008.

FRANKLIN, S.; GRAESSER, A. **Is it an Agent, or just a Program?**: A Taxonomy for Autonomous Agents. Intelligent agents III agent theories, architectures, and languages. [S.l.]: Springer, 1997, p. 21–35.

GALASTRI, L. A. **O USO DE PROGRAMAÇÃO GENÉTICA NA TOMADA DE DECISÃO EM JOGOS**. Blumenau: FURB, 2003.

GIRARDI, R. **Engenharia de Software baseada em Agentes.** 2004.

GRITZ, L.; HAHN, J. K. **Genetic programming for articulated figure motion**. The Journal of Visualization and Computer Animation, 1 jul. 1995. v. 6, n. 3, p. 129–142.

HAYES-ROTH, B. **An Architecture for Adaptive Intelligent Systems**. [S.l.]: [s.n.], 1995. V. 72.

HOLLAND, J. H. **Genetic Algorithms and Adaptation**. *In*: SELFRIDGE, O. G.; RISSLAND, E. L.; ARBIB, M. A. (Org.). **Adaptive Control of Ill-Defined Systems**. Springer US, 1984, p. 317–333.

JUCHEM, M.; BASTOS, M. R. **Engenharia de Sistemas Multiagentes: Uma Investigação sobre o Estado da Arte**. [S.l.], 2001. p. 12.

KIRKPATRICK, S.; GELATT JUNIOR, C.D.; VECCHI, M.P. **Optimization by simulated annealing**. Science, New York, 1983, v.220, p.671-680.

KOZA, J. R. **Genetic programing: on the programming of computers by means of natural selection**. 4. ed. MIT: [s.n.], 1992.

LARSEN, F. N. **Robocode: build the best – destroy the rest!**

MAIA JR, L. C.; BIANCHI, R. A. **USANDO PROGRAMAÇÃO GENÉTICA PARA EVOLUIR AGENTES JOGADORES DE FUTEBOL DE ROBÔS**. **Revista FEI,** 2001. Disponível em: <http://www.fei.edu.br/~rbianchi/publications/RevistaFEI2001.pdf>.

MOFFAT, D.; FRIJDA, N. H.; PHAF, R. H. **Prospects for Artificail Intelligence. Proceedings of AISB 93, Birmingham**. [S.l.]: [s.n.], 1993.

NORONHA, T.F. **Uma Abordagem sobre Estratégias Metaheurísticas.** 2000. Disponível em:<http://www.sbc.org.br/reic/edicoes/2001e1/cientificos/UmaAbordagemsobreEstrategiasM etaheuristicas.pdf>

OLIVEIRA, F. **Inteligência Artificial Distribuída**. Anais: Sociedade Brasileira de Computação, 1996.

O'NEIL, M.; RYAN, C. **Grammar based function definition in Grammatical Evolution**. Proceedings of the 5th Annual Conference in Genetic Programming. (GECCO 2000) ISBN 1558607080, 2000. p. 485–490.

OSTETTO, A. A.; SANTOS, F. Dos. **Extensão Swarm Intelligence para o Simulador Robocup Rescue**. 2011.

REIS, L. P. **Coordenação em Sistemas Multi-Agente**. [S.l.]: PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2003.

ROBOCUP. **RoboCup Soccer**. Disponível em: <http://www.robocup.org/robocup-soccer>.

ROBOCUP RESCUE. Tokyo. Disponível em: <http://www.robocuprescue.org>.

RUSSELL, S.; NORVIG, P.; **INTELLIGENCE, A. A modern approach**. Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs, 1995. v. 25.

SARAIVA, P. C. **Programação genética aplicada à busca de imagens**. 28 fev. 2014.

SHHEIBIA, T. A. A. El. **Controle de um Braço Robótico utilizando uma Abordagem de Agente Inteligente**. 2001.

SILVA, A.S.N., SAMPAIO, R.M. e ALVARENGA, G.B. **Uma Aplicação de Simulated Annealing para o Problema de Alocação de Salas**. Journal of Computer Science, 2005, vol.4, n.3, p.59-66.

STEINER, D. D. **Foundations of Distributed Artificial Intelligence**. *In*: O'HARE, G. M. P.; JENNINGS, N. R. (Org.). New York, NY, USA: John Wiley & Sons, Inc., 1996, p. 345–364.

WOOLDRIDGE, M.; JENNINGS, N. R. **Intelligent Agents: Theory and Practice.** Knowledge Engineering Review, 1995. v. 10, p. 115–152.

ZUBEN, I.-P. VON. **Programação genética (PG)**. Disponível em: <ftp://calhau.dca.fee.unicamp.br/pub/docs/vonzuben/ia707_01/topico8_01.pdf>.