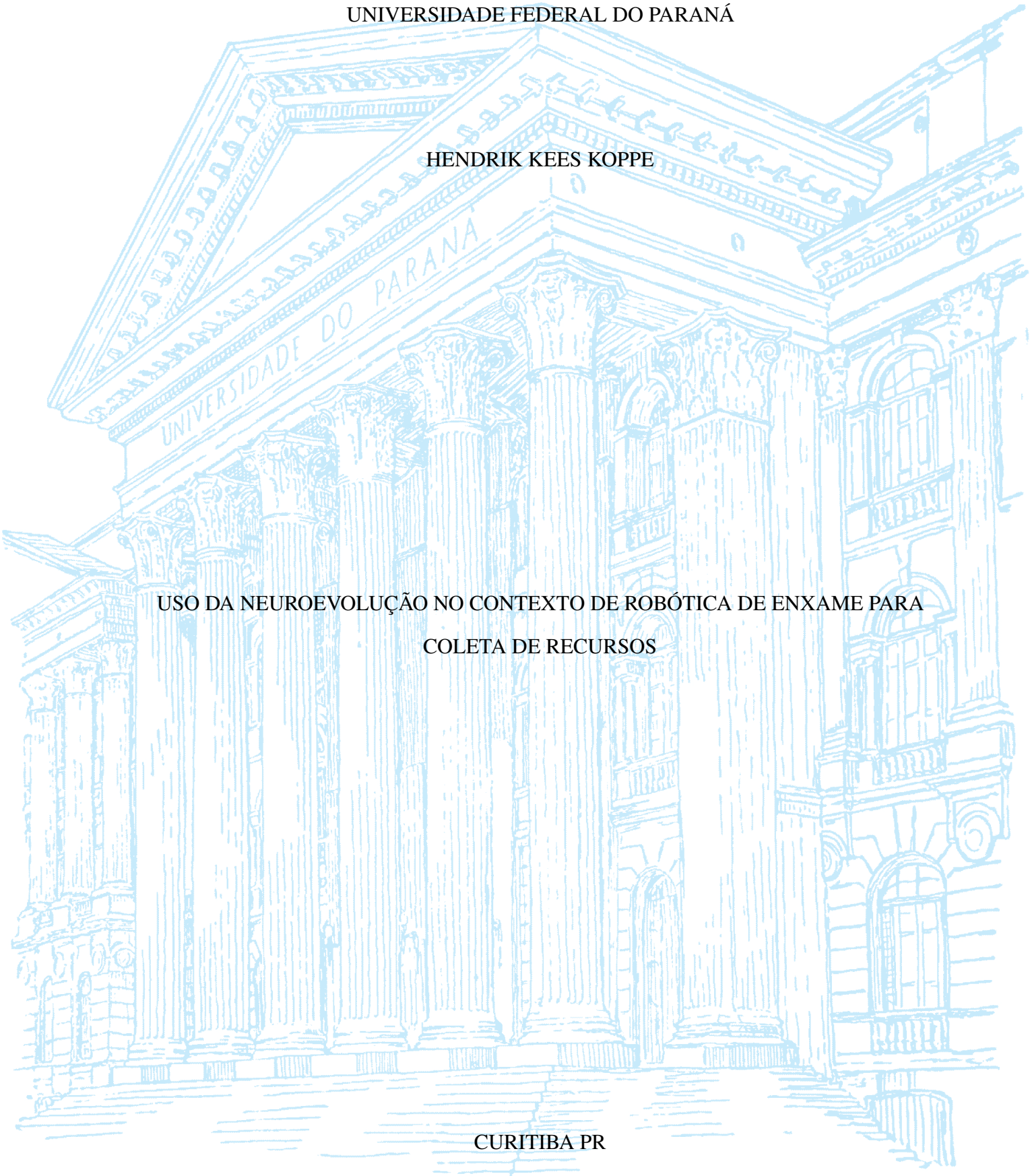
UNIVERSIDADE FEDERAL DO PARANÁ

HENDRIK KEES KOPPE



USO DA NEUROEVOLUÇÃO NO CONTEXTO DE ROBÓTICA DE ENXAME PARA COLETA DE RECURSOS

CURITIBA PR

2025

HENDRIK KEES KOPPE

USO DA NEUROEVOLUÇÃO NO CONTEXTO DE ROBÓTICA DE ENXAME PARA COLETA DE RECURSOS

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Computação*.

Orientador: Eduardo Jaques Spinosa.

CURITIBA PR

2025

Para minha mãe, minha maior apoiadora na jornada da graduação.

AGRADECIMENTOS

Gostaria de agradecer a todos que estiveram presentes na minha vida durante esses anos da graduação, a todos os meus familiares, amigos e colegas que tornaram esse período tão desafiador menos cansativo e muito mais prazeroso. Sem vocês não teria conseguido chegar onde cheguei hoje, nem ser quem eu me tornei.

RESUMO

Este trabalho investiga o uso de técnicas da neuroevolução aplicadas à Robótica de Enxame em um contexto de coleta de recursos, utilizando um simulador realista de sistemas robóticos multirrobô. A proposta consiste em utilizar algoritmos genéticos para evoluir redes neurais artificiais, que servem como controladores homogêneos de um enxame de robôs. A motivação da pesquisa surge da busca por métodos autônomos, robustos e adaptáveis para enxames robóticos, que dispensam a necessidade de regras explícitas codificadas manualmente. Para isso, foram combinados conceitos de redes neurais computacionais, algoritmos evolutivos e inteligência de enxame.

A metodologia compreende o desenvolvimento de um controlador robótico único para todos os indivíduos do enxame utilizando uma rede neural gerada por um algoritmo genético. O ambiente simulado contempla um ninho e uma fonte de recursos, para que os robôs realizem o forrageamento. A função de aptidão foi baseada em comportamentos inerentes ao forrageamento, como atingir a fonte de recursos, retornar ao ninho com recursos, e evitar períodos de inatividade. As redes neurais sofreram operações genéticas de seleção, recombinação e mutação, com variações na quantidade de gerações, tamanho das populações e quantidade de obstáculos dos mapas, para averiguar impactos desses fatores.

A implementação do controlador foi realizada no simulador *ARGoS3*, em *C++*, utilizando o modelo de robô *e-puck*. As redes neurais foram implementadas utilizando a biblioteca *Dlib*, e o algoritmo genético foi desenvolvido em *Python*. A lógica de comunicação entre os indivíduos foi baseada no sensor *Range and Bearing* do robô, permitindo que indivíduos compartilhassem informações sobre seus estados e localização de recursos de maneira descentralizada.

Foram conduzidos experimentos com diferentes estratégias evolutivas e em múltiplos cenários. Os resultados demonstraram que, ao longo das gerações, os robôs desenvolveram comportamentos emergentes de cooperação, otimizando a coleta de recursos. Foi observado também que características como elitismo e diversidade populacional influenciam significativamente na qualidade da solução final.

Palavras-chave: Robótica de Enxame. Algoritmos Evolutivos. Neuroevolução.

ABSTRACT

This work investigates the use of neuroevolution techniques applied to Swarm Robotics in a resource foraging context, using a realistic simulator for multi-robot systems. The proposed approach consists of using genetic algorithms to evolve artificial neural networks that serve as homogeneous controllers for a swarm of robots. The motivation for this research lies in the search for autonomous, robust, and adaptable methods for robotic swarms, eliminating the need for manually coded explicit rules. To achieve this, concepts from computational neural networks, evolutionary algorithms, and swarm intelligence were combined.

The methodology involves the development of a single robotic controller, shared by all swarm individuals, based on a neural network evolved through a genetic algorithm. The simulated environment includes a nest and a resource source, enabling the robots to perform foraging tasks. The fitness function was based on behaviors inherent to foraging, such as reaching the resource source, returning to the nest with resources, and avoiding idle periods. The neural networks underwent genetic operations including selection, recombination, and mutation, with variations in the number of generations, population size, and the number of obstacles in the maps, in order to evaluate the impact of these factors.

The controller was implemented in the *ARGoS3* simulator using *C++*, with the *e-puck* robot model. The neural networks were implemented using the *Dlib* library, and the genetic algorithm was developed in *Python*. Communication among individuals was based on the robot's *Range and Bearing* sensor, allowing robots to share information about their states and resource locations in a decentralized manner.

Experiments were conducted with different evolutionary strategies and across multiple scenarios. The results showed that, over successive generations, the robots developed emergent cooperative behaviors, optimizing the resource foraging process. It was also observed that characteristics such as elitism and population diversity significantly influenced the quality of the final solution.

Keywords: Swarm Robotics. Evolutionary Algorithms. Neuroevolution.

LISTA DE FIGURAS

LISTA DE ACRÔNIMOS

CNN	*Convolutional Neural Network*
RNN	*Recurrent neural networks*
NEAT	*NeuroEvolution of Augmenting Topologies*
HyperNEAT	*Hypercube-based NeuroEvolution of Augmenting Topologies*
ACO	*Ant Colony Optimization*
ABC	*Artificial Bee Colony*
PSO	*Particle Swarm Optimization*
RAB	*Range and Bearing*

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo, será apresentada uma introdução geral de todo o projeto desenvolvido, com uma contextualização do assunto, informações sobre o objetivo que se pretende atingir, a motivação, a relevância do tema, e o que foi proposto para atingir o objetivo, além da estrutura que este documento apresenta.

1.1 CONTEXTUALIZAÇÃO

A robótica de enxame, área que compreende o uso de diversos robôs com um mesmo controlador, simulando enxames de animais sociais, como formigas e abelhas, para resolver problemas complexos de maneira descentralizada pela colaboração entre os robôs, é uma abordagem que vem se mostrando cada vez mais eficiente em aplicações dos mais diversos campos da atuação, desde o uso de *drones* autônomos em lavouras para monitoramento mais eficiente, até logística de grandes armazéns por meio de robôs especializados.

Existem ainda muitas abordagens a serem exploradas nesse campo, uma vez que, é uma área relativamente nova, e que está nos primórdios quando ao seu uso para soluções de problemas no mundo real. Uma dessas abordagens a serem exploradas, é o uso de controladores robóticos baseados em redes neurais artificiais, já que permitiria desenvolver um controlador extremamente personalizado para determinada tarefa sem necessidade de programas complexos. Isso porque as redes neurais desenvolvem comportamentos totalmente personalizados para maximizar a solução de um problema, sem necessidade de uma programação humana específica para cada contexto em que o enxame for inserido.

1.2 OBJETIVOS

O objetivo que se pretende atingir neste trabalho é aplicar um algoritmo genético para evoluir redes neurais, que serão utilizadas dentro de controladores robóticos de um enxame de robôs virtuais, em um simulador realista de robótica, a fim de resolver um problema de coleta de recursos de forma eficiente.

Além do objetivo principal, tem-se um objetivo secundário, que é o desenvolvimento de uma interface que permita a comunicação entre um código que executaria um algoritmo genético, e o simulador robótico, que também precisaria ser integrado com uma biblioteca de redes neurais artificiais.

Ou seja, além do objetivo de desenvolver uma rede neural computacional que resolva o problema de coletas de recursos, por meio de um algoritmo genético, também foi necessário resolver o problema de integrar diversos programas e bibliotecas, a fim de permitir a solução do objetivo principal.

1.3 MOTIVAÇÃO

A motivação para a busca de atingir este objetivo é a exploração de novas abordagens no campo da robótica de enxame, que podem levar a um maior desenvolvimento da área com controladores robóticos mais simples, porém robustos, que poderiam ser desenvolvidos com baixa interferência humana e alta personalização para os contextos específicos em que os enxames robóticos estão inseridos.

1.4 ESTRUTURA DO DOCUMENTO

Este documento foi estruturado em capítulos, cada um deles contendo os seguintes temas:

- Fundamentação Teórica: nesse capítulo, serão introduzidas as áreas da Computação que foram aplicadas neste trabalho, dentre elas a Robótica de Enxame e a área de Algoritmos Evolutivos, apresentando conceitos essenciais para o entendimento dos outros capítulos do texto, e introduzindo o leitor de forma teórica ao assunto.

- Abordagem Proposta: nesse capítulo, será apresentada a proposta para a solução do problema, explicando em detalhes a estrutura do algoritmo evolutivo, a integração dele com o simulador de robótica, além da integração do controlador robótico e as redes neurais. Ademais, serão apresentados alguns detalhes relacionados às simulações que se pretende realizar.

- Implementação: nesse capítulo, serão apresentadas as ferramentas utilizadas e as minúcias relacionadas à implementação dos programas necessários para que a abordagem proposta fosse executada com sucesso, como os detalhes quanto ao simulador, o robô escolhido para a resolução do problema, a forma com que foi realizada a comunicação entre os indivíduos, e a comunicação deles para com o ambiente em que eles estão inseridos.

- Experimentos: nesse capítulo, serão demonstrados os experimentos realizados com base na implementação, como mudanças de abordagens evolutivas e nos cenários que os robôs precisariam enfrentar, além da interpretação do impacto dessas alterações nas soluções obtidas pelas redes neurais artificiais.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão apresentadas as principais áreas da Computação aplicadas neste trabalho, além de alguns termos importantes que serão essenciais para o entendimento do texto. Os principais conceitos abordados serão as Redes Neurais Computacionais, além da classe dos Algoritmos Evolutivos e suas subclasses Algoritmos Genéticos e Neuroevolução, além de termos como Inteligência de Enxame e Robótica de Enxame.

2.1 REDES NEURAIS COMPUTACIONAIS

Redes Neurais Computacionais (Haykin, 2009) são algoritmos inspirados no funcionamento do cérebro humano. Elas consistem em unidades chamadas de neurônios artificiais, que, interconectadas, são capazes de processar informações, identificar padrões e tomar decisões com base em dados de entrada. Assim como cérebros biológicos, essas redes precisam ser treinadas para desenvolver certos comportamentos. O treinamento pode ocorrer de forma supervisionada, em que os dados fornecidos são acompanhados de respostas corretas (rótulos), permitindo que a rede aprenda padrões associados às saídas desejadas. Outra abordagem é o aprendizado por reforço, no qual a rede interage com um ambiente, recebendo feedbacks na forma de recompensas ou penalidades, ajustando-se progressivamente para maximizar seu desempenho.

As redes neurais são organizadas em camadas, como na Figura 2.1: uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Cada camada é composta por neurônios artificiais conectados por sinapses simuladas. Cada conexão possui um peso, que representa a importância da entrada correspondente. Além disso, cada neurônio possui um viés, também conhecido como *bias*, que ajusta seu limiar de ativação. Os sinais recebidos são processados por meio de uma função de ativação, como a sigmoide e ReLU, que define se o neurônio será ativado e com qual intensidade.

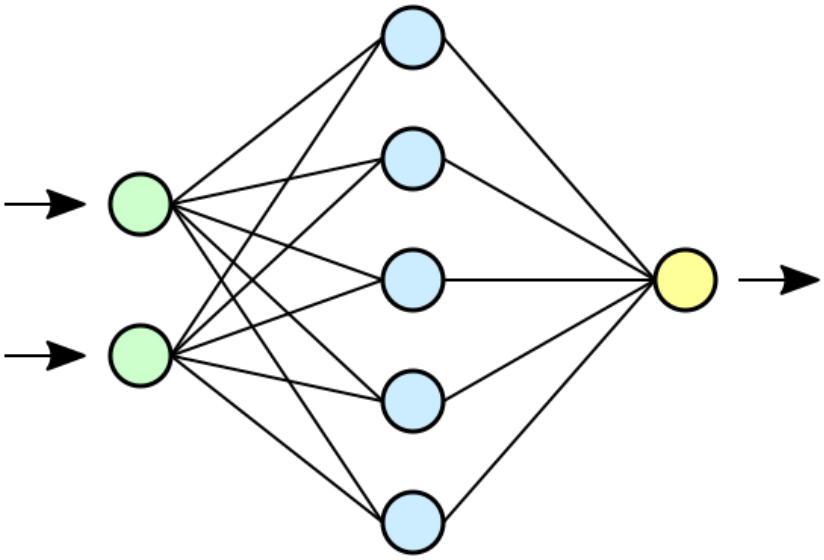


Figura 2.1: Ilustração representando uma rede neural, em verde é possível identificar os neurônios de entrada, em azul, os neurônios de uma camada oculta, e em amarelo, os neurônios de saída.

A aprendizagem ocorre por meio de algoritmos como retropropagação do erro (*backpropagation*), que ajusta os pesos e vieses com base no erro entre a saída esperada e a saída obtida, geralmente utilizando métodos de otimização como o gradiente descendente. Isso permite que a rede aprenda gradualmente a mapear entradas para saídas corretas, refinando seus parâmetros a cada iteração.

Um exemplo clássico de treinamento supervisionado é a identificação de algarismos manuscritos. Com uma base de dados suficientemente ampla e rotulada, é possível treinar uma rede neural para reconhecer, com certa acurácia, que número está representado em uma imagem nunca processada pela rede. Além desse exemplo, redes neurais são bastante utilizadas em tarefas como diagnóstico médico por imagem, processamento de linguagem natural, sistemas de recomendação e filtragem de spam.

Já no aprendizado por reforço, um exemplo comum é o treinamento de agentes inteligentes em jogos eletrônicos, onde o objetivo é maximizar a pontuação ou vencer adversários. Outro exemplo é o uso em robótica, como na estabilização de robôs, onde sensores fornecem *feedbacks* contínuos sobre o equilíbrio, e a rede neural aprende a ajustar os movimentos para manter a estabilidade do robô.

Além das redes neurais tradicionais (conhecidas como *feedforward*), existem outras arquiteturas relevantes, como as CNNs (*Convolutional neural network*), utilizadas principalmente em visão computacional, e as RNNs (*Recurrent neural networks*), mais adequadas para processar dados sequenciais.

2.2 ALGORITMOS EVOLUTIVOS

Algoritmos Evolutivos são uma família de algoritmos da Inteligência Artificial que utilizam a teoria da seleção natural de Darwin como inspiração para a solução de problemas computacionais (Bartz-Beielstein et al., 2014). A motivação dessa inspiração vem da filosofia de que, da mesma forma que a Biologia foi capaz de superar inúmeros problemas complexos por meio do processo evolutivo, a Computação poderia fazer o mesmo com algoritmos que, de certo modo, emulassem a evolução.

Nessa classe de algoritmo, inicialmente é gerada uma população de possíveis soluções, e cada uma delas é avaliadas conforme sua qualidade e corretude pela função de aptidão, também conhecida como função de *fitness*. Então, os melhores indivíduos são selecionados e recombinados para gerar a próxima geração de soluções, que passará pelo mesmo processo de avaliação pela função de *fitness*, além de uma seleção conforme as aptidões e recombinação para a geração de novas populações. Esse processo é repetido até que o algoritmo chegue em uma solução satisfatória ou um critério de parada predefinido seja atingido.

Esse tipo de algoritmo é utilizado principalmente para a solução de problemas de otimização de sistemas, como no gerenciamento de tráfego, para geração de código, como no caso da Programação Genética e para a busca de soluções para problemas combinatórios complexos.

2.2.1 Algoritmos Genéticos

Algoritmos Genéricos são uma classe de Algoritmos Evolutivos que utilizam representações genéticas para cada solução particular e operações genéticas para manipulação dessas soluções. Essas operações podem ser seleção, recombinação (*crossover*) e mutação. Nesse modelo de algoritmo, o código ou programa evolui e muta por meio da evolução da população ao longo de várias gerações para se tornar cada vez mais apto e adaptado a executar uma ou mais tarefas específicas.

Nesses algoritmos, as soluções são codificadas como representações genéticas, geralmente por meio de uma lista de valores, na maioria das vezes binários, similar à representação de genomas biológicos.

Além disso, essas soluções sofrem influência dos operadores genéticos, que tem inspiração biológica, como sobrevivência dos mais aptos, que é representada pela seleção, a reprodução, representada pelo *crossover* e a mutação biológica, que é simulada pela mutação do genoma computacional.

Na Seleção, ocorre a decisão de quais indivíduos devem ter seus genomas passados para a próxima geração. Esse processo pode ocorrer de várias maneiras, como ranqueamento, roleta, torneio e elitismo.

- Seleção por Roleta: no método de seleção por roleta, a probabilidade de seleção de um determinado genoma é proporcional à *fitness* dele. O método possui esse nome pois funciona da mesma forma que uma roleta física, onde a probabilidade da escolha de cada solução é proporcional a *fitness* dela. Esse método, embora seja o mais simples de implementar, pode não ser ideal quando se busca diversidade na população, uma vez que um indivíduo com aptidão muito superior aos demais costuma tornar as probabilidades de seleção de outras soluções muito baixas.

- Seleção por Torneio: a seleção via torneio ocorre pela organização aleatória dos indivíduos em grupos, que irão "competir"entre si. O indivíduo com maior *fitness* do grupo ganha o torneio e passa para a próxima fase, esse processo se repete até que todas as soluções tenham competido. Então é realizado o mesmo processo de organizar as soluções em grupos e realizar competições até que reste somente um grupo final de genomas, onde o cromossomo vencedor irá ser o selecionado para sofrer a recombinação.

- Elitismo: neste método de seleção, um ou mais códigos genéticos com as melhores aptidões são levados para a próxima geração sem nenhuma alteração, ou seja, sem passar pelo posterior *crossover* com outra solução, para preservar suas características originais. Esse método de seleção possui como principal objetivo manter o genoma dos melhores indivíduos da última geração, para evitar que as melhores soluções para o problema sejam perdidas ao longo do processo evolutivo. Embora seja um algoritmo que pretende acelerar o processo evolutivo, é um método de seleção que reduz a variabilidade genética da população, por exigir que os melhores indivíduos sejam copiados sem alterações em seus cromossomos, o que pode levar a comportamentos de convergência precoce.

- Seleção por Ranqueamento: nessa forma de seleção, indivíduos com as melhores notas serão selecionados com maior probabilidade que os outros, porém, o que é utilizado para fazer essa seleção não é diretamente a *fitness* do indivíduo, como nos outros métodos, mas sim sua posição em um *ranking* de todas as soluções da geração. Essa forma de seleção permite que indivíduos com aptidões não tão altas tenham maior chance de passar seus genomas para a próxima geração, principalmente nos casos em que as primeiras soluções do ranking possuam notas muito melhores as de posição mais baixa, o que seria improvável em métodos de seleção por roleta, por exemplos.

Crossover, ou Recombinação, é o operador genético fortemente inspirado na Biologia em que se utiliza duas ou mais soluções diferentes e combina-se diferentes partes delas para desenvolver

uma nova. Algo análogo ao que ocorre na reprodução sexuada. Existem diferentes técnicas de *crossover*, que divergem na quantidade de pontos de corte em que ocorrerá a recombinação entre progenitores. As principais delas sendo *crossover* de um ponto, k pontos, e *crossover* uniforme.

- *Crossover* de um ponto: a recombinação dos progenitores ocorre em um ponto específico do código genético, de forma que o trecho cromossômico antes do corte seja de um progenitor e o trecho depois seja de outro.
- *Crossover* de k pontos: o processo é similar a recombinação de 1, porém ocorre com um valor k, maior ou igual a 2, que é definido por quem está projetando o algoritmo. O processo de selecionar trechos de código de cada progenitor alternadamente pode ser realizado com quantos pontos forem necessários, desde que seja um valor menor ou igual a quantidade de elementos no genoma, já que é uma modalidade de operador generalista.
- *Crossover* Uniforme: nessa modalidade de *crossover*, ao invés de selecionar pontos de corte, cada elemento do genoma é selecionado de forma aleatória com igual probabilidade entre os genitores, de forma que, na média, o filho tenha uma quantidade proporcional do código genético dos seus progenitores, essa modalidade de recombinação é interessante quando se pretende que os genomas filhos tenham proporcionalmente características de seus antepassados, o que não costuma ocorrer nos *crossovers* por pontos de corte.

Mutação é um operador genético que tem por objetivo manter alta a diversidade genética da população e evitar estagnações em soluções locais, por meio de mutações aleatorizadas em determinados cromossomos da geração atual. Existem diferentes tipos de mutação, podendo ser inversões de bits, em casos de cromossomos binários, ou mutações aleatórias com Distribuição Gaussiana, para cromossomos que sejam valores reais ou inteiros.

- Inversão de bits: na mutação por inversão de bits, que é exclusiva para códigos genéticos representados de forma binária, cada elemento do cromossomo tem a chance aleatória de ser invertido, ou seja, se seu valor é 0, passa a ser 1, e se é 1 passa a ser 0. Nessa modalidade de operador, para cada elemento, é randomizado se a inversão deve ou não ocorrer, e se for decidido que sim, basta inverter o bit atual. Esse é o operador de mutação mais simples e rápido, porém, possui a desvantagem de funcionar somente para representações binárias.
- Mutação com Distribuição Gaussiana: para valores não binários, é possível aplicar o mesmo padrão da Inversão de Bits, percorrendo-se os elementos do genoma e verificando-se se deve ou não ocorrer mutação. Porém, nos casos positivos, ao invés de inverter o bit, é aleatorizado um novo valor com base em uma distribuição gaussiana, de forma que a mutação possua mais chance de se manter suave, mas ainda assim, permitindo trocas para valores mais extremos em alguns casos. Devido a essa particularidade da distribuição gaussiana de possibilitar variações mais extremas em casos específicos, existem variações desse algoritmo de mutação em que, caso o valor aleatório de mutação ultrapasse um limiar de variação, o valor é descartado e recalculado, de forma que as mutações se mantenham controladas. Esse estilo de mutação permite pequenos ajustes nas soluções, que podem otimizar, ou não a pontuação da solução atual. Por se tratar de uma função com variações mais controladas, é ideal para pequenos ajustes nas soluções.

2.2.2 Neuroevolução

A neuroevolução é uma aplicação específica de Algoritmos Evolutivos, que tem por objetivo projetar, ajustar e otimizar uma ou mais Redes Neurais por meio da evolução genética da rede. Na neuroevolução, inicialmente são geradas redes com valores de pesos e vieses aleatórios, podendo incluir também a topologia da rede, que irão passar por todo o processo de um Algoritmo Genético. As redes neurais passam por seleção, recombinação e mutação, por diversas gerações, uma vez que esses valores atuam como o genoma da solução.

A neuroevolução é particularmente útil em contextos onde o gradiente de erro, isto é, a taxa de variação da função de erro em relação aos pesos da rede, não está disponível, como em ambientes simulados com recompensas não recorrentes, ou ainda quando se busca automatizar a construção da arquitetura da rede neural, algo inviável com métodos tradicionais baseados no gradiente de erro. Essa abordagem é bastante utilizada em robótica autônoma, especialmente em tarefas de navegação e comportamentos coletivos em robótica de enxame, onde os agentes são avaliados com base no desempenho global de sua atuação, e não em saídas rotuladas específicas.

Um dos algoritmos mais conhecidos de neuroevolução é o NEAT (*NeuroEvolution of Augmenting Topologies*), que permite evolução da topologia da rede junto com os pesos dela, além do HyperNEAT (*Hypercube-based NeuroEvolution of Augmenting Topologies*), evolução do NEAT que permite a geração de redes de grande escala com padrões espaciais, sendo particularmente útil em aplicações com estruturas regulares, como controle de múltiplos atuadores em robótica.

2.3 INTELIGÊNCIA DE ENXAME

Inteligência de enxame é uma sub-área da Inteligência Artificial que se baseia em comportamentos coletivos naturais, de formigas, abelhas, cupins e outros animais sociais com baixa complexidade individual, mas com comportamentos coletivos complexos para desenvolver algoritmos que envoluam um código centralizado executado em várias instâncias de forma que essas instâncias consigam cooperar entre si e atingir resultados complexos (Beni e Wang, 1993).

A inteligência de enxame vem sendo explorada cada vez mais em aplicações que demandam soluções descentralizadas, adaptativas e robustas, especialmente em ambientes dinâmicos. Essa abordagem se destaca por formar sistemas de múltiplos agentes simples, que, por meio de interações locais, conseguem resolver tarefas complexas de forma eficiente e escalável.

Um ponto de destaque desse modelo de algoritmo é a tolerância a falhas e alta adaptação, uma vez que, caso um indivíduo do grupo seja removido ou não tenha sucesso em sua missão, todos os outros podem continuar operando normalmente sem grandes perdas de desempenho. Isso é especialmente útil para ambientes adversos, onde imprevistos podem fazer com que alguns indivíduos não consigam seguir com suas tarefas. Isso ocorre devido as características inerentes do algoritmo, que, por ser descentralizado e retornar resultados por meio de comportamentos emergentes dos indivíduos como um todo, não dependem das capacidades individuais de agentes específicos.

Alguns algoritmos de inteligência de enxame são:

- ACO (*Ant Colony Optimization*): inspirado no comportamento de forrageamento de formigas, em que trilhas de feromônio são usadas para guiar outras formigas até fontes de alimento. Esse algoritmo é bastante utilizado em problemas de otimização, como o Problema do Caixeiro Viajante, roteamento de veículos e escalonamento de tarefas.

- ABC (*Artificial Bee Colony*): baseado na coleta de néctar por abelhas melíferas, é eficaz em problemas de otimização e ajuste de parâmetros em modelos.

- PSO (*Particle Swarm Optimization*): se baseia nos movimentos coordenados de pássaros e peixes, sendo muito utilizado em problemas de otimização, e ajustes finos em redes neurais.

2.4 ROBÓTICA DE ENXAME

Robótica de enxame é uma sub-área da robótica que tem por objetivo implementar a inteligência de enxame em diversos robôs com baixo poder computacional e autonomia total.

A robótica de enxame já é utilizada em logística de armazéns. Nesses ambientes, enxames de robôs são utilizados para movimentar mercadorias e otimizar os espaços de armazenamento, com o intuito de economizar espaço e poupar tempo dos trabalhadores (Trianni et al., 2016).

Também vem sendo utilizada para exploração de áreas de risco, particularmente em áreas de desastres, por meio de enxames que minimizam o tempo necessário para essa tarefa. Além do uso na agricultura, em que, simulando revoadas de pássaros, *drones* conseguem aplicar inseticidas e fertilizantes de forma otimizada (Cheraghi et al., 2021).

2.5 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados os principais conceitos e áreas da Computação relacionados à Computação Bioinspirada, com destaque para Redes Neurais, Robótica de Enxame e Algoritmos Genéticos. Esses tópicos fornecem a base teórica necessária para compreender os métodos e abordagens utilizados no desenvolvimento deste trabalho, especialmente no que diz respeito à modelagem de comportamentos coletivos inteligentes e ao aprendizado de agentes. No próximo capítulo, serão apresentadas as abordagens e técnicas propostas para o desenvolvimento deste trabalho.

3 ABORDAGEM PROPOSTA

Neste capítulo, será apresentada a abordagem proposta para este trabalho, com foco na aplicação de técnicas de neuroevolução no contexto da robótica de enxame, especificamente para a resolução do problema de forrageamento. A proposta envolve o uso de um simulador de robótica realista para avaliar o desempenho de redes neurais evoluídas por um algoritmo genético, que são utilizadas como controladores homogêneos para todos os indivíduos do enxame.

Serão detalhados neste capítulo os principais componentes da abordagem, como a lógica do algoritmo evolutivo, a definição da função de aptidão, o modelo do mapa da simulação, a topologia da rede neural utilizada, os operadores genéticos aplicados e os requisitos do simulador escolhido.

3.1 VISÃO GERAL

O objetivo deste trabalho é simular a neuroevolução de uma rede neural no contexto da robótica de enxame em um cenário de forrageamento (coleta de recursos), por meio de cópias idênticas de um único controlador robótico para todos os indivíduos de um enxame, implementado em um simulador realista de robótica. Para isso, a abordagem proposta foi desenvolver um algoritmo evolutivo que consiga criar diversas redes neurais aleatórias, que componham uma geração de enxames, e cada uma dessas redes neurais seja implementada de forma similar em cada controlador dos indivíduos do enxame. Então, após a simulação em um ambiente com uma área de ninho, e uma área de fonte de alimento, cada indivíduo recebe uma nota, com base na função de *fitness* do problema, e esse controlador externo soma as notas obtidas por cada indivíduo pós-simulação para compor a nota total do enxame. Por meio dessa nota, é possível identificar quais enxames foram os mais aptos.

Por fim, com as simulações realizadas e as notas calculadas, é realizado um algoritmo de roleta para selecionar dois genitores para procriar e, com a seleção aleatórias entre os pesos e vieses das redes genitoras, gerar uma nova rede neural que compõe parte da próxima geração. Essa rede então sofrer uma mutação em todos os seus elementos por meio de uma função de aleatoriedade, mutação essa que tem por objetivo evitar a convergência prematura das soluções para um ótimo local e inserir uma maior variedade genética na população, por meio da inserção de redes neurais mutantes nas gerações, que tragam soluções alternativas. E então o algoritmo se repete, até que todas as simulações de todas as gerações terminem de executar. Por último, a rede neural com maior pontuação de aptidão da última geração é considerada a solução final para o problema. Na Figura 3.1 é possível verificar de forma gráfica o diagrama desse funcionamento.

3.2 MAPA DA SIMULAÇÃO

Para o mapa em que as simulações ocorreram, foi proposto um modelo simples, que possui uma área que serve como a base (ou ninho) do enxame robótico, de onde os robôs partem e para onde eles retornem após obter o recurso do forrageamento, além de uma área que é a fonte de recursos, para onde os robôs precisam ir para obter uma unidade do recurso que estão buscando.

As dimensões do mapa são de uma proporção que não exigisse uma busca exaustiva para que os robôs encontrassem a fonte de alimento, dessa forma, a duração da simulação pode ser menor, além de permitir funções de *fitness* mais simples para as redes neurais, que não

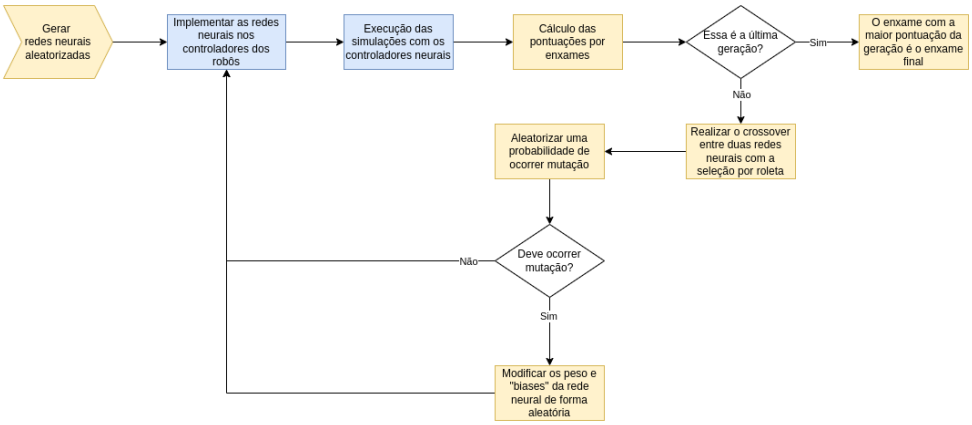


Figura 3.1: Fluxograma da execução do algoritmo. Em amarelo, ações realizadas pelo programa principal e em azul as realizadas pelo controlador do robô.

precisam englobar recompensas parciais afim de evitar treinamentos demorados sem resultados expressivos.

Para que os robôs pudessem identificar em que região estão, foi proposto um esquema de cores, em que, em branco são representadas regiões que não são nem o ninho e nem a fonte de recursos, em preto são a área do ninho, que precisam ser de um tamanho mínimo que pudesse comportar todos os robôs do enxame no início da simulação, e por último, em uma cor que é interpretada como uma cor intermediária entre preto e branco por sensores de cor ou por sensores de escala de cinza, é a fonte de recurso. Para facilitar a visualização, a cor intermediária escolhida foi uma cor que não fosse em escala de cinza, no caso, a cor verde. A imagem plana do mapa pode ser verificada na Figura 3.2

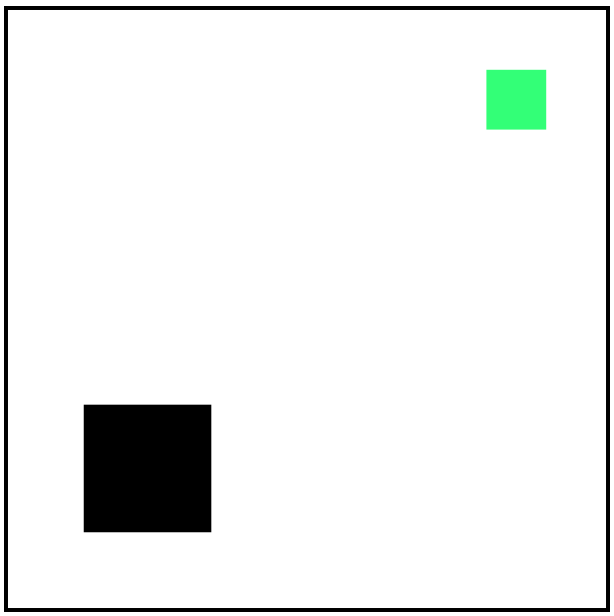


Figura 3.2: Figura representando o mapa da simulação, em preto está o ninho, que possui um tamanho maior para acomodar todos os robôs do enxame, e em verde, a fonte de recursos.

3.3 FUNÇÃO DE APTIDÃO

Para a função de aptidão foi proposto um modelo de pontuações individuais para cada robô, que são somadas ao final da execução da simulação, para formar a função de aptidão global do enxame. A função é baseada em comportamentos relevantes do robô na tarefa de obter recursos e retornar para o ninho, por isso, foi decidido que três comportamentos influenciam na aptidão do robô:

- Obter uma unidade de recurso: quando um robô, que antes não estava carregando um recurso chega à fonte de alimento, é considerado que ele obteve uma unidade de recurso, ou seja, atingiu um objetivo parcial da tarefa de forrageamento, e por isso, ele ganhará 3 pontos de aptidão.
- Voltar para o ninho com uma unidade de recurso: quando um indivíduo que estava carregando um recurso volta para o ninho, é considerado que ele cumpriu o objetivo principal do forrageamento. Ele volta a ficar sem recursos e ganhará 5 pontos de aptidão.
- Ficar parado ou com velocidade muito lenta por uma sequência de tempo: caso um indivíduo passe muito tempo parado, ou com uma velocidade muito baixa, seja por ter se chocado com outros robôs ou com as paredes, ou ainda pelo indivíduo não estar se movimentado o bastante para apresentar comportamento relevante, por decisão da própria rede neural, consideramos que ele não está contribuindo para o forrageamento, e por isso, o robô perde 1 ponto na *fitness*.

3.4 LÓGICA DE ESTADOS

Por conta da modelagem escolhida para a função de aptidão, e da natureza do problema que pretende-se resolver, foi proposta também uma lógica de máquina de estados finita para os indivíduos, que permite tornar mais simples a averiguação das pontuações de aptidão e facilita a comunicação entre os robôs quanto aos seus estados atuais. A lógica de estados funciona da seguinte forma:

Inicialmente, todo o enxame está no estado "No ninho", um estado que indica que o indivíduo não possui nenhum recurso com ele, e que ele está em uma posição do mapa que faz parte do ninho. Assim que os robôs começam a se movimentar para áreas fora do ninho, eles passam para o estado "Procurando alimento", em que eles ainda não carregam uma unidade de recurso e não chegaram a uma fonte de recursos. Os robôs ficarão nesse estado até entrarem em uma área de fonte de recursos ou voltarem para o ninho. Se o robô voltar para o ninho, ele volta ao estado "No ninho", se chegara na fonte de recursos, ele passa a carregar uma unidade de recurso e passa ao estado "Na fonte de alimento". Nesse momento, os robôs só podem passar para o estado "Voltando para o ninho", que representa que eles carregam um recurso e não estão nem na fonte de alimento e nem no ninho. A partir desse estado, o robô só pode voltar para o estado "Na fonte de alimento"ou para o estado "No ninho". Se ele voltar para o ninho, o robô deixa o recurso no ninho, pontua e volta para o estado inicial. Graficamente, essa máquina finita de estados é representada pela Figura 3.3

3.5 REQUISITOS PARA O SIMULADOR DE ROBÓTICA

A proposta deste projeto foi utilizar um simulador de robótica o mais realista possível, que possuísse física verossímil e utilizasse modelos de robôs reais, de forma que o controlador final pudesse ser implementado em um robô físico.

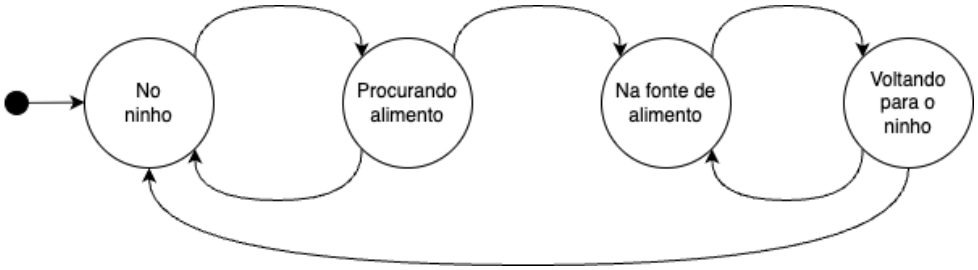


Figura 3.3: Diagrama de transição de estados do robô.

Foi considerado como pré-requisito, o simulador suportar simulações com diversos robôs, uma vez que, o trabalho busca formar enxames robóticos, com dez robôs sendo operados simultaneamente. Outro requisito foi possuir uma forma de executar as simulações por linha de comando e sem janela gráfica, de forma que a execução do simulador pudesse ser realizada por um código externo de forma automatizada, sem que fosse necessário intervenção humana.

3.6 TOPOLOGIA DA REDE NEURAL

Para a rede neural, foi proposta uma topologia que fosse eficiente para uma grande quantidade de entradas, uma vez que, as entradas seriam todos os sensores presentes no robô, e a saída de apenas dois valores, indicando a intensidade de ativação das rodas esquerda e direita do robô.

Para isso, foi escolhida uma rede *feedforward* com duas camadas ocultas com função de ativação ReLu, com 20 e 10 neurônios respectivamente, e uma camada de saída de 2 neurônios com função de ativação sigmoide, que restringe os valores de saída para valores entre 0 e 1, sendo 0, o motor desativado e 1 o motor ativado com potência máxima.

Foi decidido também normalizar as entradas dos sensores para que todos os valores ficassem no intervalo entre 0 e 1, para acelerar o treinamento e evitar que algumas variáveis dominassem o aprendizado da rede neural.

3.7 OPERADORES GENÉTICOS

Os operadores genéticos propostos para este trabalho foram, por questão de simplicidade de implementação, a seleção por meio de roleta, além do *crossover* uniforme e a mutação em todos os pesos e vieses. Para a seleção por roleta, foi decidido fazer uma normalização nas notas, para evitar que existissem notas negativas devido a soluções em que os robôs ficam parados por muito tempo. Para a recombinação, foi escolhido seguir pela abordagem mais clássica de utilizar dois progenitores por cruzamento.

3.8 CONSIDERAÇÕES FINAIS

Neste capítulo foi detalhada a abordagem metodológica adotada para alcançar o objetivo do trabalho, que consiste em simular a evolução de redes neurais por meio de algoritmos evolutivos, no contexto da robótica de enxame aplicada a um cenário de forrageamento. A proposta consiste em um controlador neural compartilhado entre todos os indivíduos do enxame, permitindo observar comportamentos coletivos emergentes ao longo das gerações.

Foram apresentadas as decisões relacionadas à estrutura do algoritmo, à função de aptidão, à topologia da rede neural e aos operadores genéticos utilizados. Essas escolhas

foram feitas com base na viabilidade computacional, na coerência com a literatura da área e na capacidade de promover o aprendizado pelas redes neurais em um cenário não supervisionado.

No próximo capítulo, serão apresentadas as implementações realizadas com base nesta abordagem, com detalhes quanto a implementação do algoritmo genético, também relativo ao modelo de robô escolhido, às bibliotecas e ao simulador utilizado.

4 IMPLEMENTAÇÃO

Neste capítulo será apresentada a implementação dos programas que controlam a simulação robótica e a rede neural, além do algoritmo evolutivo. Ademais, serão aprofundados detalhes técnicos, referentes ao simulador, ao modelo do robô escolhido, além do processo de comunicação entre o simulador e a rede neural, e entre o simulador robótico e o algoritmo evolutivo, que foi desenvolvido inteiramente nesse projeto.

4.1 ARGOS3

Para a escolha do simulador de robótica, foi realizada uma extensa pesquisa do uso de simuladores de robótica no contexto específico da robótica de enxame, para decidir qual se adequaria melhor ao trabalho. Foram analisados diferentes simuladores, entre eles Webots (Michel, 2004), Enki (Magnenat et al., 2011), CoppeliaSim (Rohmer et al., 2013), Roborobo (Bredèche et al., 2013) e ARGoS (Pinciroli et al., 2012).

Após serem comparadas características relevantes, como a seleção de robôs disponíveis, serem ou não *softwares* livres, a facilidade de utilizar os simuladores, linguagens de programação disponíveis neles, possibilidade de executar sem janela gráfica e de forma automatizada, a possibilidade de centralizar os controladores neurais, além da eficiência em executar o simulador com múltiplos robôs ao mesmo tempo, foi decidido utilizar o simulador ARGoS em sua terceira versão. Dentre os simuladores analisados, o ARGoS foi o único desenvolvido visando o uso no contexto de sistemas multi-robôs, além disso, é um simulador *open-source* e eficiente, que permite o uso de robôs amplamente utilizados em pesquisas da área da robótica, como o *e-puck* (Mondada et al., 2009), e o *pi-puck* (Millard et al., 2017). Também preenche os requisitos de permitir execução em modo não gráfico, e por linha de comando.

Como afirmou Pinciroli et al., autores do artigo *ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems* (Pinciroli et al., 2012):

> O ARGoS foi projetado para simular experimentos complexos envolvendo grandes grupos de robôs de diferentes tipos. O ARGoS é o primeiro simulador multirrobô que é, ao mesmo tempo, eficiente (desempenho rápido com muitos robôs) e flexível (altamente personalizável para experimentos específicos). *(tradução livre)*

4.1.1 Estrutura de arquivos

Um experimento com o simulador ARGoS3 é estruturado em múltiplos arquivos que representam diferentes responsabilidades no projeto. A seguir, é descrita a organização de diretórios e arquivos relacionados a execução base do simulador, realizada no projeto, junto com explicações referentes a cada tipo de arquivo:

- **arenas/**
  Diretório que contém os arquivos no formato `.argos`, escritos em XML. Esses arquivos descrevem o ambiente simulado, como a posição inicial dos robôs, obstáculos, sensores e atuadores utilizados, parâmetros físicos do mundo (gravidade, limites de arena, etc.), duração do experimento, detalhes de visualização, e a associação com os controladores compilados. É possível definir diferentes arquivos de arena para diferentes experimentos e configurações, sem necessidade de recompilação dos controladores robóticos.

- **controllers/**
  Contém os arquivos-fonte em `C++` responsáveis pelos controladores dos robôs. Os controladores podem sobrescrever algumas funções nativas do ARGoS, permitindo execução de código ao inicializar, parar, reiniciar, ou a cada ciclo da simulação. Esses arquivos são compilados como bibliotecas que serão utilizadas durante a simulação.

- **build/**
  Diretório gerado automaticamente durante o processo de compilação. Ele contém os arquivos objeto (`.o`) e bibliotecas compartilhadas (`.so`) resultantes da compilação dos controladores.

- **CMakeLists.txt**
  Arquivo de configuração utilizado pelo `CMake`, que é o sistema de build recomendado pelo ARGoS. Ele especifica os arquivos-fonte, bibliotecas externas e *flags* de compilação. Esse arquivo gera automaticamente o ambiente de build adequado ao sistema operacional em que o simulador está sendo executado.

Essa separação em arquivos permite modularidade, reusabilidade de controladores e arenas, facilitando a manutenção e expansão do projeto. A simulação é iniciada por meio do comando `argos3 -c arenas/arena.argos`, que carrega o ambiente e executa os controladores compilados associados aos robôs definidos.

4.1.2 Funções nativas dos controladores

Como citado na seção anterior, o ARGoS possui algumas funções que podem ou devem ser implementadas afim de alterar o comportamento do robô ou realizar ações específicas em determinados momentos da simulação, sendo elas:

- `Init()`: função de implementação obrigatória, que é executada uma única vez no início da simulação, onde os sensores e atuadores do robô precisam ser inicializados para serem posteriormente utilizados, além disso, variáveis de controle e outras bibliotecas externas que exigem configurações iniciais podem ser inicializadas nessa função.

- `ControlStep()`: função de implementação obrigatória, que é executada a cada passo da simulação, sendo executada diversas vezes por segundo. Nessa função, pode ser feita a leitura dos sensores, que permitem ao robô perceber o ambiente a sua volta, o processamento da lógica inserida pelo desenvolvedor, e o acionamento dos atuadores do robô, que permitirão ao robô interagir com o ambiente da simulação.

- `Destroy()`: função de implementação opcional, que é executada ao encerramento da simulação, seja pelo fim do tempo de execução, pelo encerramento abrupto pelo usuário ou pelo sistema, ou ainda pela destruição do robô na simulação. Nessa função, costumam ser inseridas lógicas de limpeza de alocações de memória, fechamento de arquivos, além da gravação de registros e outras informações importantes em arquivos externos.

- `Reset()`: função de implementação opcional, que é executada ao reiniciar simulações realizadas de forma sucessivas, que tem como principal função limpar variáveis e dados salvos da simulação finalizada.

4.1.3 Robô escolhido

Para a escolha do robô cujo enxame é responsável pelo processo de forrageamento, foi levado em consideração sua relevância na literatura da área da robótica de enxame, assim como a variedade de sensores e atuadores. Por conta disso, o modelo escolhido foi o *e-puck* (Mondada et al., 2009), um robô amplamente utilizado em experimentos de controle descentralizado, agregação, e navegação autônoma.

O *e-puck* é um robô educacional e de pesquisa, com dimensões compactas e arquitetura simplificada, o que o torna ideal para simulações em larga escala. Ele possui dois motores independentes para locomoção, sensores infravermelhos de proximidade, sensores de luz, acelerômetro, giroscópio e uma câmera frontal. Seu *hardware* foi projetado para ser extensível, permitindo o acoplamento de módulos adicionais, como câmeras de resolução mais alta ou controladores embarcados mais potentes (como o *Raspberry Pi*, no caso do *pi-puck*).

No contexto do ARGoS3, o *e-puck* possui suporte completo e otimizado, com sensores e atuadores simulados fiéis aos reais e altamente configuráveis. Isso permite a criação de cenários fidedignos com dezenas ou centenas de indivíduos simulados em paralelo.

Além disso, o *e-puck* é frequentemente utilizado como modelo base em publicações relacionadas à neuroevolução e aprendizado de máquina embarcado, o que o torna compatível com o objetivo deste trabalho, que é estudar o impacto de técnicas evolutivas no comportamento coletivo em cenários de coleta de recursos (forrageamento).

Na Figura 4.1 é possível verificar os principais sensores e atuadores de um modelo físico do robô:

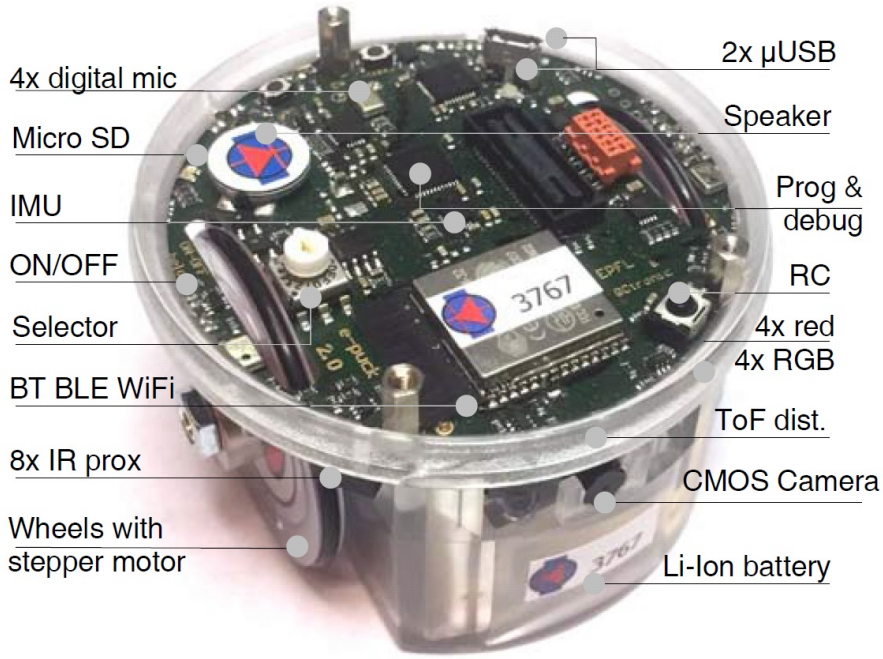


Figura 4.1: Imagem real de um e-puck com seus sensores e atuadores.

4.2 IMPLEMENTAÇÃO DO CONTROLADOR ROBÓTICO

A implementação do controlador consistiu em desenvolver um código capaz de ler um arquivo com os parâmetros da rede neural, inicializar a rede neural com esses valores, e a cada passo da simulação, alimentar a rede neural com todos os dados vindos dos sensores,

atualizar o estado atual do robô, de acordo com a posição dele, atuar nas duas rodas do robô e nos atuadores de comunicação entre o enxame, atualizando a pontuação total do indivíduo, se necessário. Embora pareça simples, a integração de bibliotecas de rede neural com o simulador, além do processo de selecionar os sensores e atuadores a serem usados, e de transformar a leitura dos sensores em informações úteis, exigiu bastante tentativa e erro.

Nos próximos tópicos, serão especificados os detalhes de implementação quanto a lógica de identificação da área atual em que o robô se encontra, a lógica de cálculo da pontuação, a lógica de comunicação entre os indivíduos do enxame, além da implementação da rede neural dentro do controlador do ARGoS3.

4.2.1 Lógica de identificação da área atual

Para permitir identificar qual o tipo da área em que o robô se encontra, foi utilizado o mapeamento do solo pelo robô por meio dos três sensores de solo presentes no e-puck (dois laterais e um frontal), que captam a intensidade da luz refletida pela superfície, podendo variar de 0, para nenhuma luz, até 1, para luz total.

É possível desenvolver um protocolo uma vez que, as áreas do mapa foram definidas de tal forma que, cada cor indica uma diferente área, sendo branco uma área de caminho, preto uma área de ninho, e verde (identificado como uma intensidade intermediaria de brilho pelo sensor) a fonte de alimento.

A lógica aplicada foi, a cada ciclo da simulação, fazer a leitura dos três sensores, e considerar que a menor intensidade de todas seria a utilizada para análise. Com base nesse valor obtido, é comparada a menor intensidade dos sensores com os valores de luminosidade de cada região do mapa, para identificar em que tipo de área o robô está. Com a informação da área atual do robô, é possível aplicar a lógica de estados apresentada no Capítulo 3, atualizar o estado atual do robô, a nota, se ocorreu algum evento que desencadeie mudança na pontuação, e por último sinalizar por meio das luzes *leds* do robô se ele está carregando um alimento (*leds* amarelos), se ele está no ninho (*leds* azuis), ou ainda se ele está fora do ninho e sem alimento (*leds* verdes), como mostrado na Figura 4.2.

É importante notar que essa indicação visual por meio de luzes serve para facilitar depuração e identificação do estado atual dos robôs por observadores externos, uma vez que a comunicação entre os robôs quanto aos seus estados é feita por meio do sensor de *range and bearing*, que será detalhado mais a frente.

A seguir, é apresentado o pseudo-código da lógica de identificação da área atual da posição do robô:

Listing 4.1: Decisão de comportamento com base na leitura do solo

```
1  leituraChao = obterLeituraDoSoloMaisEscura()
2
3  se leituraChao indica NINHO então
4     definirLEDs(AZUL)
5     possuiComida = falso
6     statusAtual = NO_NINHO
7
8  senão se leituraChao indica FONTE_DE_ALIMENTO então
9     definirLEDs(AMARELO)
10    possuiComida = verdadeiro
11    statusAtual = NA_FONTE_ALIMENTO
12
13 senão
14    se possuiComida então
```

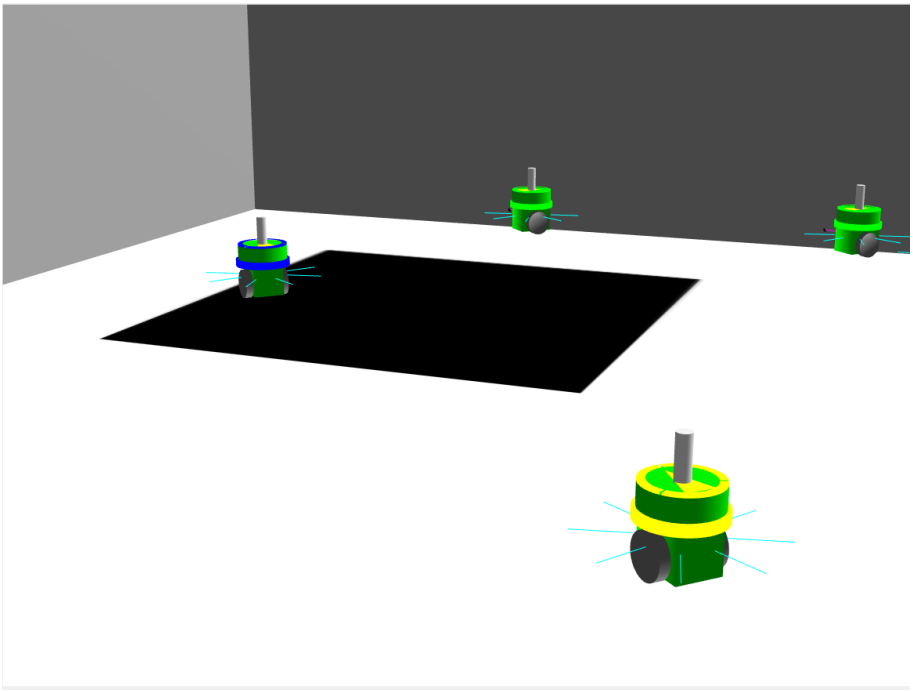


Figura 4.2: Quatro robôs exibindo as luzes referentes ao seu estado atual.

```
15        definirLEDs(AMARELO)
16        statusAtual = VOLTANDO_PARA_NINHO
17     senão
18        definirLEDs(VERDE)
19        statusAtual = PROCURANDO_ALIMENTOS
20     fim se
21  fim se
22
23  //código de cálculo de pontuação
```

4.2.2 Lógica de cálculo da pontuação

Para o cálculo da pontuação individual do robô, foi necessário implementar uma variável global `pontuacaoIndividuo`, que era inicializada com valor zero na chamada da função `Init`, e era atualizada a cada ciclo do controlador, na função `ControlStep`, dependendo dos pré-requisitos de atualização de pontuação serem preenchidos. Por fim, na chamada da função `Destroy`, um arquivo `pontuacao_id.txt` era gerado com a pontuação feita pelo indivíduo, em que o identificador era obtido de acordo com a numeração do robô no enxame.

A lógica de cálculo da pontuação também seguiu a documentada pelo Capítulo 3, porém, sofreu modificações durante o desenvolvimento quanto ao valor de cada ação, devido à baixa relevância de ações de coleta de recurso perante as pontuações perdidas por robôs que permaneciam sem se movimentar. Na proposta inicial, o robô ganhava 3 pontos ao obter um alimento, e 5 pontos ao voltar para o ninho com o alimento. Para surtir um efeito relevante, precisou-se alterar esses valores para 3000 e 5000 respectivamente, já que, durante a simulação, o robô poderia permanecer parado por milhares de ciclos.

Também foram definidas as constantes de ciclos parados para penalidade do indivíduo como 10, e a velocidade que foi utilizada para definir que o robô ficou sem se movimentar para 3,0 radianos por segundo. Para comparação, a velocidade máxima desenvolvida pelo *e-puck* é de aproximadamente 2 rotações por segundo, ou 12,57 radianos por segundo. Também foi utilizado

para considerar o robô parado, se os seus sensores frontais de proximidade identificassem obstáculos com intensidade 0,95, sendo 0 nenhum obstáculo próximo, e 1, um obstáculo muito próximo.

Dessa forma, a lógica de cálculo da pontuação a cada ciclo pode ser verificada no pseudo-código a seguir:

Listing 4.2: Cálculo de pontuação

```
//código de identificação de área atual

se status mudou então
   se statusAtual indica NA_FONTE_ALIMENTO
   e ultimoStatus indica PROCURANDO_ALIMENTOS então
      pontuacaoIndividuo = pontuacaoIndividuo + 3000;
   senão se statusAtual indica NO_NINHO
   e ultimoStatus indica VOLTANDO_PARA_NINHO então
      pontuacaoIndividuo = pontuacaoIndividuo + 5000;
   fim se
fim se

// ...

se (velocidadeEsquera < VELOCIDADE_PARADO
ou velocidadeDireita < VELOCIDADE_PARADO)
ou (sensorFrente1 > DISTANCIA_OBSTACULO
ou sensorFrente2 > DISTANCIA_OBSTACULO) então
   segundosParado = segundosParado + 1
senão
   segundosParado = 0
fim se

se segundosParado > LIMITE_TEMPO_PARADO então
   pontuacaoIndividuo = pontuacaoIndividuo - 1
   segundosParado = 0
fim se
```

4.2.3 Lógica de comunicação com outros robôs do enxame

Para a seleção da melhor lógica de comunicação com os outros robôs, foi necessário testar diferentes combinações de atuadores e sensores. Inicialmente, pretendia-se utilizar os atuadores de *led* para o envio do status atual do robô, por meio da intensidade da luz emitida pelas diferentes cores das lâmpadas como identificador do estado do indivíduo, junto dos sensores de luminosidade para captar esses dados. Porém, essa primeira abordagem demonstrou possuir algumas falhas, que tornaram essa aproximação inviável.

Durante os testes, identificou-se que, embora as lâmpadas emitissem intensidades luminosas diferentes de acordo com a cor selecionada para seus atuadores, conforme a distância entre o robô interceptando a luminosidade emitida pelo sensor de luz e o robô emissário, a quantidade de luz registrada no sensor poderia variar muito, podendo atingir níveis similares ao de outros estados. Isso se mostrou ainda mais problemático em cenários com muitos robôs próximos emitindo diferentes cores de luzes, caso em que os robôs possuíam dificuldade de identificar os estados corretos, por não terem a capacidade de identificar que o que estava sendo captado não era a luz de somente um robô, mas de vários. Por fim, verificou-se que, ao se aproximar de obstáculos e dos limites da arena, os robôs passavam a identificar menos luz, já que os obstáculos não são luminosos, o que causava falsos positivos para diferentes estados.

Também verificou-se a possibilidade de seguir utilizando os atuadores *led*, porém utilizando a câmera nativa do *e-puck* para identificação dos estados. Essa abordagem também não trouxe bons frutos, pois, embora resolvesse o problema dos indivíduos do enxame não conseguirem identificar mais de um robô ao mesmo tempo, assim como os problemas relacionados a mudanças de intensidade luminosa serem confundidos com diferentes estados, a câmera simulada do robô não permite a identificação de mais de três indivíduos ao mesmo tempo, além de ser um sensor único frontal no robô com um baixo campo de visão, o que impede que o indivíduo consiga ter uma noção mais completa do enxame e obter informações globais dos comportamentos de seus companheiros, restringindo-o a coletar informação de três robôs, no máximo, desde que eles estivessem em uma posição específica que favorecesse isso.

Por fim, foi escolhida uma abordagem que não envolvesse atuadores e sensores que dependessem muito do estado do meio externo para serem propagados: o uso do sensor e atuador *Range and Bearing* (RAB). Esse sensor e atuador são dispositivos que utilizam um meio de propagação sem fio, implementados fisicamente como ondas infravermelhas ou ondas de rádio, que permitem o envio e recebimento de até 4 *bytes* de informação, e no momento da interceptação, também permitem identificar o ângulo (*range*) e a distância (*bearing*) de onde essa informação foi enviada.

Essa forma de comunicação se mostrou muito menos propensa a ruídos advindos do ambiente, além de permitir trocas de informação com riqueza de detalhes, que permitiam a todos os robôs coletarem informações relevantes dos indivíduos do seu enxame com precisão quanto à localização de onde essa informação veio. Isso se mostra bastante rico quando analisado, por exemplo, no caso em que um dos robôs encontra a fonte de alimento na simulação: assim que um indivíduo chega à fonte de alimento, ele passa a enviar para os outros robôs do enxame que seu status agora é "Na fonte de alimento", e por meio do ângulo e distância de onde veio essa informação, todos os outros robôs podem identificar a localização dessa região a partir do seu referencial atual, e se mover nessa direção para obter o recurso.

Embora existisse a capacidade de envio de 4 *bytes* de informação, decidiu-se manter o envio somente do estado atual do robô, caso ele carregasse alimento, estivesse no ninho ou na fonte de alimento, de forma que, se tentasse reproduzir os comportamentos identificados em colônias biológicas durante o forrageamento, e evitar uma sobrecarga de informações irrelevantes na camada de entrada da rede neural.

4.2.4 Rede Neural

Para a implementação da rede neural dentro do controlador do ARGoS3, foi preciso importar uma biblioteca da linguagem `C++`, a mesma linguagem de programação utilizada no restante do código do controlador, que fosse leve, para não impactar na performance das simulações, *header-only*, ou seja, não exigisse instalação complexa em cada máquina em que o experimento fosse realizado, somente a importação da biblioteca no cabeçalho do código, mas completa e flexível, para permitir a população de pesos e vieses de forma personalizada para cada neurônio da rede.

A biblioteca escolhida para isso foi a biblioteca Dlib (King, 2009), uma biblioteca de código aberto escrita totalmente em `C++`, amplamente utilizada na literatura em aprendizado de máquina, visão computacional e outras áreas relacionadas à inteligência artificial.

Para a integração da rede neural com o controlador robótico, foi adicionado à função `Init` do controlador, o código que cria a rede neural, rede essa que consiste em uma arquitetura com três camadas densas: a primeira camada recebe os dados de entrada e os processa por 20 neurônios com função de ativação `ReLU`; em seguida, uma segunda camada oculta composta por 10 neurônios também com ativação `ReLU` refina os padrões extraídos; por fim, uma camada de

saída com 2 neurônios e ativação `sigmoide` gera os comandos de controle contínuos (valores entre 0 e 1), representando as velocidades desejadas para as rodas esquerda e direita.

Então, na função `ControlStep`, que é executada a cada ciclo da execução, a rede recebia como entrada os seguintes valores normalizados entre 0 e 1: os valores dos 8 sensores de proximidade do *e-puck*, os ângulos, distâncias e estados atuais de todos os outros robôs do enxame, que tenham enviado seus dados para o indivíduo, e por último, o status atual do robô. Para um enxame de 10 robôs, o tamanho de enxame utilizado nos testes, foram até 39 entradas para a rede neural, que com base nas entradas da rede, retornaria dois valores, também normalizados, que representam a intensidade de ativação dos motores esquerdo e direito do robô, que então, seriam acionados.

4.3 IMPLEMENTAÇÃO DA ARENA

Para a implementação da arena conforme a proposta apresentada no Capítulo 3, foi necessário fazer alguns ajustes no arquivo `.argos`, que representa a arena da simulação. Dentre eles, foram alterados os campos de duração do experimento e *seed* de aleatoriedade. Para a duração do experimento, foi definida uma duração fixa de 200 ciclos, sendo que a cada segundo são executados 10 ciclos, de forma que a execução de cada experimento dure 20 segundos, se executado de forma gráfica. Além disso, foi adicionada uma semente de aleatoriedade fixa, para permitir a reprodutibilidade dos experimentos.

Para implementar as áreas de diferentes cores no chão da arena, foi utilizado um arquivo de imagem no formato `.png`, criado com um software de edição gráfica. Esse arquivo foi referenciado no campo `<floor>` do arquivo, cobrindo toda a área do solo da arena. Essa imagem continha áreas codificadas por cores, representando zonas como o ninho (preto), a fonte de alimento (verde) e o restante da arena (branco), que podiam ser detectadas pelos sensores de chão dos robôs.

A arena foi delimitada por quatro paredes (norte, sul, leste e oeste), representadas por caixas tridimensionais fixas com altura suficiente para conter os robôs, impedindo que saíssem da área de simulação.

Dentro da arena, foram distribuídos automaticamente 10 robôs do tipo *e-puck*, utilizando o bloco `<distribute>`, com posições e orientações geradas de maneira uniforme em uma região específica da arena (entre as coordenadas $(-1.1, 0.5)$ e $(-0.5, 1.1)$, que eram as coordenadas do ninho).

Além disso, foram configurados os sensores e atuadores necessários para o comportamento dos robôs, como o atuador diferencial das rodas, LEDs, e o módulo de comunicação por *range and bearing*. Do lado dos sensores, foram ativados os sensores de proximidade, sensores de solo e o receptor de *range and bearing*. A renderização das simulações pôde ser ativada opcionalmente por meio da seção `<visualization>`, que era removida ao executar o programa genético, e adicionada para execução manual.

4.4 IMPLEMENTAÇÃO DO PROGRAMA GENÉTICO

O programa genético desenvolvido em `Python` é responsável por gerenciar todo o experimento, desde a geração das primeiras redes, execução das simulações, cálculo das notas por enxame, seleção, recombinação, mutação, até a escrita dos registros em um arquivo de texto, e criação de um gráfico de evolução das melhores, piores e notas médias por geração.

É importante notar que todo o algoritmo genético foi desenvolvido especialmente para este presente trabalho, não utilizando nenhuma biblioteca externa para isso, e garantindo que os operadores genéticos fossem totalmente personalizados para o problema atual.

A seguir, é descrita em detalhes cada parte do algoritmo.

4.4.1 Funções principais

- `criarPrimeiraGeracao()`: inicializa a geração zero criando 10 redes neurais aleatórias. Cada rede é gerada por meio de um script externo `gerarEntradaInicial.py`, que grava os parâmetros iniciais em arquivos na pasta `params_rede`.

- `buildarProjeto()`: compila o controlador `C++` do ARGoS3, executando `cmake` e `make` no diretório `build`. Isso garante que o simulador utilize sempre o código atualizado.

- `rodarExperimento()`: invoca o ARGoS3 para executar a simulação utilizando o arquivo `.argos`. Os logs são redirecionados para `/dev/null` para não poluir o terminal.

- `calcularNotaEnxame(i)`: soma as pontuações individuais dos 10 robôs do enxame, lendo arquivos em `pontuacoes/pontuacao_epuckX.txt`. Remove esses arquivos após leitura e registra a pontuação coletiva do enxame, atualizando os melhores resultados globais.

- `normalizarNotas()`: lê as pontuações das 10 redes dessa geração, normaliza os valores para garantir que não existam notas negativas, e retorna o vetor de notas processadas para seleção.

- `selecionarMelhoresEnxamesRoleta(notas)`: realiza a seleção via roleta proporcional às notas. Retorna um par de índices correspondentes aos dois enxames escolhidos para gerar um novo filho.

- `crossover(notas)`: inicia uma nova geração, selecionando pares de pais por roleta e combinando parâmetros linha a linha. Cada filho nasce com probabilidade de mutação (10%), caso em que seus parâmetros recebem pequenas alterações aleatórias variando entre +0,10 e -0,10.

- `rodarExperimentoGeracao(finalizar)`: executa todos os experimentos da geração atual. Para cada indivíduo (rede), renomeia seu arquivo de parâmetros para o nome esperado pelo controlador, dispara a simulação, calcula a nota e atualiza estatísticas de maior, menor e média da geração. Ao final, chama a função de cruzamento ou, se for a última geração, salva o melhor enxame como `enxame_final.txt`.

- `limparPastas()`: prepara o ambiente de execução apagando e recriando as pastas `params_rede` e `pontuacoes`, garantindo que não existam resultados residuais entre execuções.

- `imprimir(mensagem)`: imprime mensagens no console, e, opcionalmente, também em arquivo de log `execucao.txt` quando o programa é iniciado com o parâmetro `-s`.

- `plotarGrafico()`: ao término do experimento, gera e salva um gráfico ilustrando a evolução, por geração, das notas mínima, média e máxima. Utiliza a biblioteca gráfica `matplotlib` (Hunter, 2007) para gerá-lo.

4.4.2 Fluxo do algoritmo

O fluxo principal do algoritmo é o seguinte:

1. Define uma semente de aleatoriedade para garantir reprodutibilidade do experimento.
2. Compila o controlador com `buildarProjeto()`.
3. Limpa ambientes anteriores com `limparPastas()`.
4. Gera a primeira população com `criarPrimeiraGeracao()`.
5. Para cada geração:
   (a) Executa os experimentos, um para cada rede da geração.
   (b) Normaliza as notas e realiza seleção, *crossover* e possível mutação.
   (c) Se for a última geração, salva o melhor enxame em `enxame_final.txt`.
6. Ao final, a função `plotarGrafico()` é chamada para visualização dos resultados.

4.4.3 Código de geração da primeira geração de redes neurais

Para criar os indivíduos da geração inicial, foi desenvolvido um script em `Python`, chamado `gerarEntradaInicial.py`. Esse script é responsável por gerar os pesos e os vieses da rede neural de cada robô do enxame de forma aleatória, respeitando os limites estabelecidos para variação dos valores.

A rede neural utilizada possui três camadas totalmente conectadas, sendo elas:

- Uma primeira camada oculta de 20 neurônios, que recebe os dados dos sensores dos robôs
- Uma segunda camada oculta de 10 neurônios, para refinar os dados
- Uma camada de saída, com 2 neurônios de saída

Os pesos de cada conexão são inicializados com valores aleatórios entre −0,5 e +0,5, e os vieses variam entre −1,0 e +1,0.

Esse *script* é utilizado diretamente pela função `criarPrimeiraGeracao()` no programa principal. Para cada indivíduo, o program gera um arquivo contendo os parâmetros (pesos e vieses) da rede, que será carregado posteriormente pelo controlador de cada robô durante a simulação.

4.5 CONSIDERAÇÕES FINAIS

Neste capítulo, foram apresentados todos os detalhes de implementação dos programas, que envolveram o desenvolvimento de um controlador robótico com redes neurais em `C++`, além de um algoritmo genético em `Python`. No próximo capítulo, serão apresentados os experimentos realizados a fim de verificar impactos relacionados a variações de abordagem no algoritmo genético.

5 EXPERIMENTOS

Neste capítulo serão apresentadas análises comparativas de experimentos referentes a mudanças de abordagens no algoritmo genético e no ambiente físico da simulação, a fim de verificar o impacto de diferentes parâmetros e variáveis na evolução e resultado final dos enxames. Dentre os experimentos realizados, tem-se mudanças na implementação da mutação, variação no tamanho e quantidade de gerações, mudanças de arenas, e o impacto do elitismo na evolução das maiores notas dos enxames.

Para isso, foram gerados gráficos comparativos, em que são exibidas as notas do melhor e pior enxame, em verde e azul, respectivamente, além da pontuação média dos enxames de cada geração, em laranja, para facilitar a comparação.

5.1 IMPACTO DA IMPLEMENTAÇÃO DA MUTAÇÃO

Com o objetivo de avaliar o impacto de diferentes estratégias de mutação na evolução das redes neurais, foram conduzidos três experimentos, mantendo-se constantes todos os parâmetros e as sementes de aleatoriedade. A única variação entre as execuções foi o tipo de mutação empregado, que alternou entre três abordagens: mutação linear, mutação gaussiana tradicional e uma variação denominada mutação gaussiana reversa, projetada para simular o comportamento oposto ao da distribuição gaussiana padrão. Os detalhes de cada uma dessas abordagens são apresentados a seguir.

5.1.1 Mutação Linear

Na abordagem linear, a seleção dos enxames a serem mutados segue o mesmo critério das demais configurações. No entanto, a alteração dos parâmetros das redes ocorre por meio de uma variação uniforme: cada valor pode ser incrementado ou decrementado em até 0,1. Por exemplo, um neurônio com ativação 0,35 pode, após a mutação, assumir qualquer valor entre 0,25 e 0,45, com probabilidade uniforme. Essa abordagem é a mais simples de implementar, por utilizar-se de uma função de aleatoriedade tradicional, e apresenta um bom balanço entre soluções com baixas e altas variações entre os valores mutantes.

Após a execução do experimento, obteve-se o gráfico da Figura 5.1. Nota-se que houve um crescimento, porém, com pontos de convergência prematura em ótimos locais, o que demonstra que a mutação não foi forte o bastante para levar uma solução de um ótimo local para uma pontuação melhor.

Também é possível verificar que as notas mais baixas variaram bastante de valor, flutuando entre valores negativos e valores próximos das notas médias. Isso demonstra que a mutação linear teve um bom balanço entre mutações fortes, que poderiam levar as soluções para valores mais extremos de notas, seja positiva ou negativamente, e mutações fracas, que mantiveram as pontuações das redes neurais mutantes próximas da média. A nota do melhor enxame utilizando essa modalidade de mutação foi de mais de 50.000 pontos, uma nota que indica que o forrageamento foi completado com sucesso, porém não muitas vezes.

5.1.2 Mutação Gaussiana

Para a abordagem de mutação gaussiana, a função utilizada para mutação foi uma implementação própria da aleatoriedade gaussiana do módulo *random* da linguagem de progra-

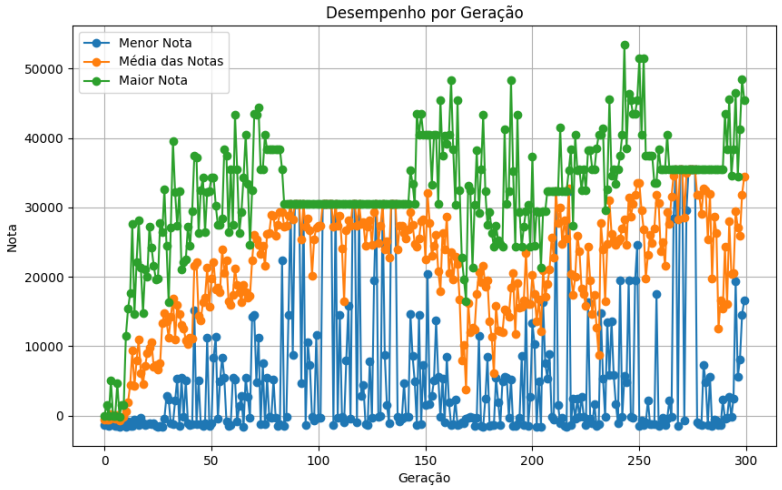


Figura 5.1: Resultado gráfico para 300 gerações de uma população com 10 redes neurais utilizando mutação linear.

mação Python, com valores variando entre -0,1 e +0,1. A função gaussiana, ou normal, faz com que a mutação tenha maior probabilidade de ser menos intensa, mantendo os valores mutados próximos dos originais, uma abordagem que promove pequenos ajustes nas redes neurais, com baixa probabilidade de levar a rede neural para soluções completamente diferentes das originais.

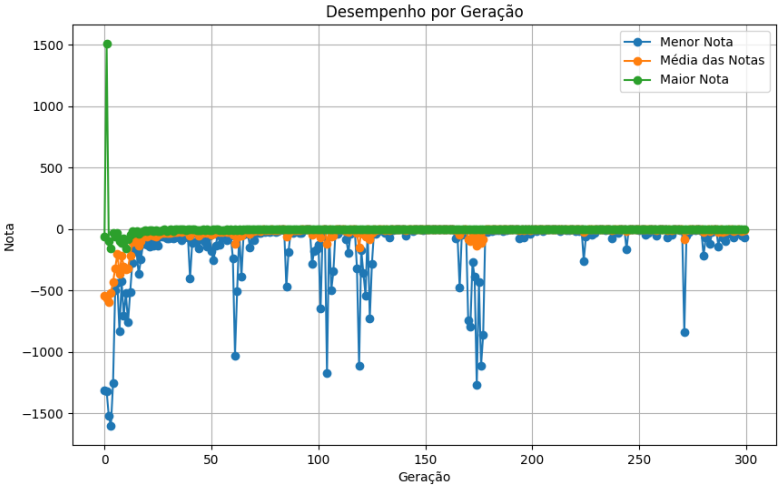


Figura 5.2: Resultado gráfico para 300 gerações de uma população com 10 redes neurais utilizando mutação gaussiana.

Pela análise do gráfico presente na Figura 5.2, percebe-se que a mutação não melhorou o desempenho das redes neurais. Embora os valores das notas máximas tenham permanecidos muito próximos por gerações, provavelmente por uma baixa variabilidade genética na população, a mutação gaussiana não conseguiu ajustar os parâmetros para que as redes saíssem de um ótimo local, como ocorreu com a mutação normal.

A nota mais alta entre os enxames foi de aproximadamente 1.500 pontos, um valor desenvolvido na geração inicial, quando a mutação não estava sendo aplicada ainda, e que indica que o enxame não conseguiu desenvolver os objetivos do forrageamento.

5.1.3 Mutação Gaussiana Reversa

Para simular um comportamento oposto ao da distribuição gaussiana tradicional, foi desenvolvida uma função própria de mutação, denominada mutação gaussiana reversa. Ao contrário da gaussiana padrão, que concentra as mutações em torno do valor central (valores próximos ao original), essa abordagem aumenta a probabilidade de alterações mais intensas, distantes do valor de origem, incentivando maior variabilidade genética.

A função foi implementada em Python utilizando uma distribuição *Beta*, que gera uma curva em forma de "U", concentrando a maioria dos valores nas extremidades do intervalo. O código utilizado é apresentado a seguir:

Listing 5.1: Função Gaussiana Reversa em Python

```
def inverso_gaussiano(centro, intensidade):
    # Gera valor entre [0, 1] com concentração nos extremos
    x = random.betavariate(0.5, 0.5)

    # Transforma para valores entre -1 e +1
    y = (x - 0.5) * 2

    # Escala pela intensidade e aplica ao centro
    return centro + intensidade * y * 10
```

A mutação gaussiana reversa, ao privilegiar mudanças mais expressivas, foi capaz de manter uma diversidade genética mais ampla durante as gerações, favorecendo a busca por soluções alternativas ao redor do espaço de busca, o que mostrou ser uma abordagem bastante promissora, atingindo valores de quase 120.000 pontos, mais que o dobro da mutação com função linear.

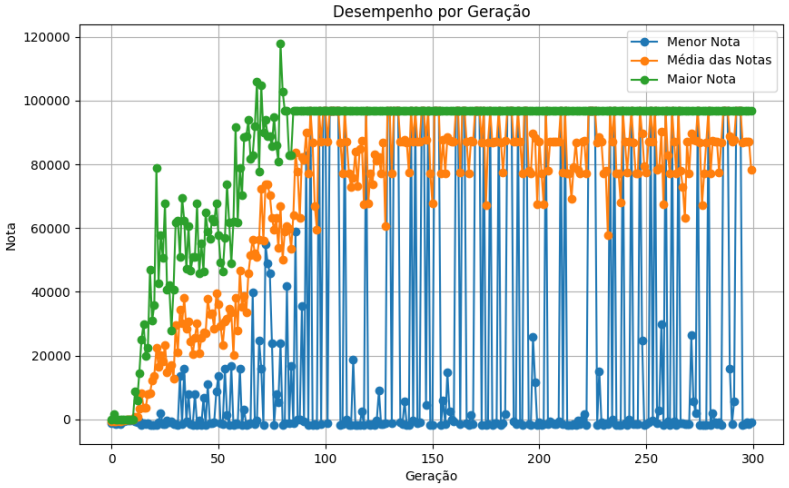


Figura 5.3: Resultado gráfico para 300 gerações uma população de 10 redes neurais utilizando mutação gaussiana reversa.

Pela análise do gráfico presente na Figura 5.3, é possível verificar como o crescimento da pontuação do melhor enxame foi veloz, atingindo valores acima da melhor solução encontrada utilizando a mutação linear em menos de 25 gerações. Porém, a partir da geração 100, houve uma estagnação em um ótimo local, muito provavelmente ocasionado pela falta de variedade genética, intensificada pela rápida convergência de soluções. Problema esse que poderia ser contornado aumentando a quantidade de redes neurais por geração. Mesmo com esse comportamento

indesejado, pela pontuação atingida pelo melhor enxame, é possível verificar que esse enxame conseguiu atingir tanto o objetivo parcial (atingir a fonte de recurso) quanto o final (retornar para o ninho com uma unidade de recurso) do forrageamento diversas vezes com sucesso.

5.1.4 Comparação entre Experimentos

A comparação entre os três tipos de mutação permite observar como a escolha da estratégia de alteração dos parâmetros influencia significativamente o desempenho evolutivo das redes neurais no contexto do forrageamento.

A mutação linear apresentou um desempenho intermediário: ainda que tenha resultado em uma solução capaz de atingir o objetivo final do ambiente (coletar e retornar com recursos), a progressão foi relativamente lenta, principalmente quando comparada a mutação gaussiana reversa, e em diversos momentos houve estagnações prematuras em ótimos locais. No entanto, sua distribuição uniforme promoveu um bom equilíbrio entre diversidade e estabilidade, o que explica a grande variação entre as notas mínimas e médias ao longo das gerações.

Por outro lado, a mutação gaussiana tradicional demonstrou-se ineficaz nesse cenário. Apesar de favorecer pequenas alterações e ajustes graduais nos pesos das redes neurais, a baixa intensidade das mutações limitou severamente a exploração do espaço de busca. Como consequência, os enxames não conseguiram sair de ótimos locais, e a melhor pontuação obtida se manteve praticamente constante desde as primeiras gerações, indicando uma convergência prematura sem evolução significativa.

Em contrapartida, a mutação gaussiana reversa se destacou pelo desempenho superior. Por incentivar mutações intensas, essa abordagem promoveu uma maior exploração do espaço de busca, permitindo que as redes alcançassem rapidamente soluções de alta qualidade. Ainda que tenha ocorrido uma estagnação a partir da geração 100, o patamar atingido foi muito mais elevado que nas outras abordagens, com valores próximos a 120.000 pontos. Esse resultado evidencia que a diversidade genética inicial promovida por mutações mais agressivas pode ser altamente benéfica, especialmente em estágios iniciais da evolução. Por outro lado, também ressalta a necessidade de mecanismos complementares, como aumento populacional ou técnicas de preservação de diversidade, para evitar perda prematura de variabilidade.

Portanto, é possível concluir que a mutação gaussiana reversa foi a abordagem mais eficaz entre as testadas, seguida da mutação linear. A mutação gaussiana tradicional, apesar de útil para ajustes finos, mostrou-se limitada para o cenário presente neste trabalho.

5.2 IMPACTO DO TAMANHO DA POPULAÇÃO E DA QUANTIDADE DE GERAÇÕES

Para verificar o impacto da variação na quantidade de gerações e no tamanho delas nas notas finais dos enxames, decidiu-se comparar uma mesma quantidade de simulações com a mesma semente de aleatoriedade, mesma função de variação para a mutação, nesse caso, a função gaussiana reversa, e mesmos mapas, porém, variando entre a quantidade de gerações e quantidade de enxames de diferentes redes neurais por geração, para que todos os experimentos utilizassem um poder computacional parecido, com os mesmos valores inicias e lógica do código genético, porém, variando o tamanho de populações e a quantidade de gerações.

Foram executadas três variações, comparando-se a mesma quantidade de 10.000 simulações:

- 1.000 gerações de uma população com 10 redes neurais
- 100 gerações de uma população com 100 redes neurais

- 10 gerações de uma população com 1.000 redes neurais

Gerando os seguintes resultados respectivamente:

5.2.1 1.000 gerações de uma população com 10 redes neurais

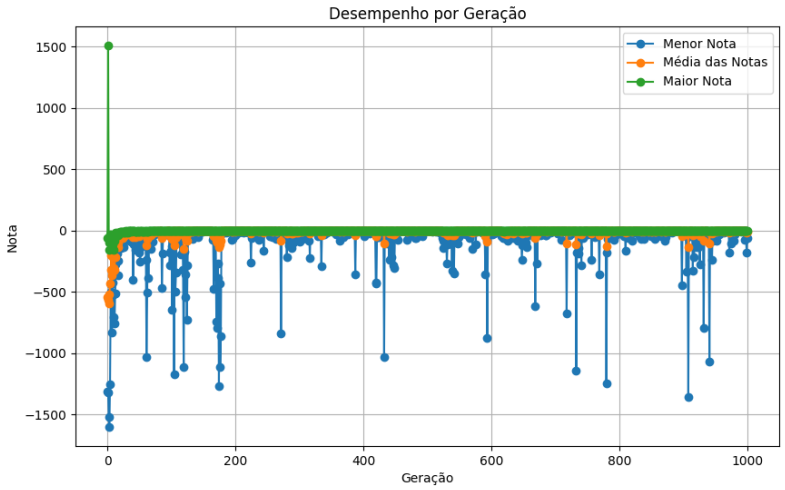


Figura 5.4: Resultado gráfico para 1.000 gerações de uma população com 10 redes neurais.

Este foi o cenário com menor evolução nas notas entre os três analisados. Embora fossem produzidas gerações o bastante para que se obtivessem uma quantidade satisfatória de recombinações, a baixa variabilidade genética fez com que as redes neurais se tornassem muito similares, levando a soluções pouco desenvolvidas, e estacionadas em um ótimo local. Isso pode ser comprovado na Figura 5.4 pela proximidade entre as linhas em laranja do gráfico, que indicam a média das notas da geração com a melhor nota da geração, indicada em verde.

Além disso, mesmo com a existência da mutação, parece que ela não foi o suficiente para que surgisse uma nova rede neural diferente o bastante, e com uma nota mais alta, para alavancar as soluções para diferentes abordagens. Isso pode ser confirmado graficamente pela falta de surgimento de novas soluções com nota máxima acima das notas anteriores, ou seja, a linha verde, das maiores notas, manteve-se próxima da nota 0 durante todo o experimento. No final do experimento, a nota máxima entre os enxames foi de, aproximadamente, apenas 1.500 pontos.

5.2.2 100 gerações de uma população com 100 redes neurais

Para o cenário com 100 gerações de uma população de 100 redes neurais, as notas atingiram os melhores patamares entre os experimentos. O cenário mostrou-se balanceado, com boa variedade genética e alta recombinação, de forma que, ao mesmo tempo, era gerada uma variedade suficiente de indivíduos para atingir novas soluções com notas maiores, e essas soluções eram evoluídas por gerações o bastante para atingir soluções novas de qualidade crescente.

No gráfico da Figura 5.5, é possível verificar que, embora as menores notas, em azul, tenham se mantido baixas, muito provavelmente por mutações que prejudicaram a nota do enxame, as notas médias, em laranja, e as maiores notas, em verde, mantiveram um crescimento constante, com saltos, quando se atingia uma solução nova de qualidade. A nota mais alta entre as simulações foi de mais de 80.000 pontos, uma avaliação que indica que diversos robôs conseguiram cumprir

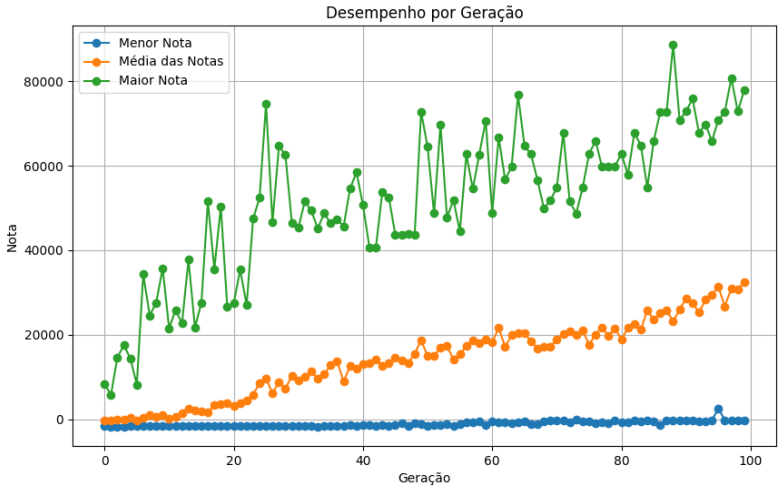


Figura 5.5: Resultado gráfico para 100 gerações de uma população com 100 redes neurais.

o objetivo do forrageamento, diversas vezes, e ainda por cima, permanecendo pouco tempo parados.

5.2.3 10 gerações de uma população com 1.000 redes neurais

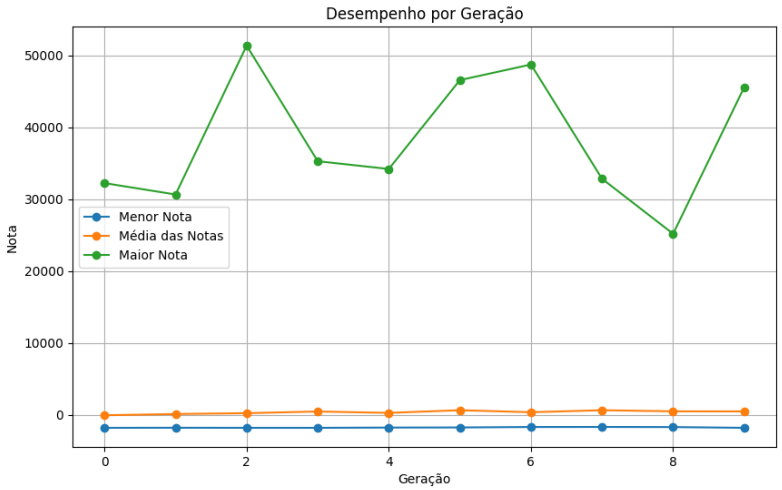


Figura 5.6: Resultado gráfico para 10 gerações de uma população com 1.000 redes neurais.

Para a simulação de 10 gerações de uma população com 1.000 redes neurais, é possível verificar pela Figura 5.6 que desde a primeira geração foram desenvolvidas soluções que já cumpriam o objetivo do forrageamento, porém poucas vezes por simulação, o que permite confirmar que o *script* que gerou as redes neurais iniciais, criou redes neurais com parâmetros adequados para a solução do problema.

Além disso, nota-se que o fator mais proeminente nas avaliações foi a ampla variedade genética por geração, uma vez que, embora as maiores notas (em verde) sejam relativamente altas, as notas médias (em laranja) e mais baixas (em azul), se mantiveram muito próximas dos valores iniciais. Isso mostra que não houve um grande desenvolvimento de todas as redes neurais,

somente de algumas, resultantes de combinações genéticas vantajosas. Além disso, percebe-se que, não houve um aumento constante por geração, devido a baixa quantidade delas. A maior nota de todos os enxames foi de mais de 50.000 pontos, um valor que indica que os indivíduos conseguiram realizar o forrageamento de forma satisfatória, além disso, as notas médias e mais baixas cresceram de forma pouco expressiva, ficando próximas dos 0 pontos, indicando enxames pouco desenvolvidos.

5.2.4 Comparação entre experimentos

Por meio das simulações foi possível identificar uma intrínseca relação entre uma variedade genética adequada e a qualidade das soluções, comparando-se o gráfico da Figura 5.4 com as demais, pois, embora o algoritmo genético possua mecanismos para aumentar a diversidade genética, como a recombinação e a mutação, esses operadores genéticos não são efetivos o bastante para compensar a variação genética insuficiente.

Além disso, foi identificado, pela comparação entre os gráficos das Figuras 5.5 e 5.6 que, embora em menor grau quando comparado a variação genética, uma quantidade de gerações alta também é essencial para garantir o crescimento das notas e amadurecimento das soluções da população entre as gerações.

Por isso, para obter uma boa evolução e uma solução final com as melhores notas, é necessário um balanço entre uma boa variabilidade genética, ou seja, um grande número de soluções por geração, e uma grande quantidade de gerações, para garantir uma grande quantidade de soluções, que possam ser recombinadas e melhoradas por uma quantidade suficiente de gerações para atingir uma solução eficiente para o problema.

5.3 IMPACTO DO ELITISMO

Uma abordagem aplicada por diversos algoritmos genéticos, a fim de acelerar o aumento da aptidão das soluções e evitar retrocessos nesses valores, é a implementação do elitismo.

O elitismo consiste em garantir que os melhores indivíduos de uma geração, aqueles com maior valor de aptidão, sejam preservados na próxima geração, sem sofrerem os efeito da seleção, cruzamento ou mutação. Dessa forma, evita-se a perda de soluções promissoras devido ao caráter aleatório do processo evolutivo.

Essa técnica tem impacto direto na convergência do algoritmo: ao manter as melhores soluções, o elitismo tende a aumentar a aptidão média da população ao longo do tempo. No entanto, se usado de forma excessiva, pode reduzir a diversidade genética da população, levando a uma estagnação prematura em ótimos locais.

Para verificar o impacto do elitismo no algoritmo genético desenvolvido, foi implementado um elitismo simples, em que a solução com melhor nota da população da geração é automaticamente copiada para a próxima. Para mensurar o impacto, foram executadas duas instâncias do algoritmo genético com os mesmos parâmetros: 100 gerações de uma população com 100 indivíduos, mutação gaussiana reversa e taxa de mutação de 10%, sendo que uma utilizou elitismo e a outra não. Os resultados estão representados nas Figuras 5.7 e 5.8, respectivamente.

É possível verificar que, ao contrário do esperado, a aplicação do elitismo não levou a um maior desempenho global das soluções. Na verdade, o elitismo limitou o crescimento da nota máxima em aproximadamente 50% quando comparado à versão sem ele.

Uma possível explicação para esse comportamento é que o elitismo favoreceu a convergência para um ótimo local. Nas primeiras 20 gerações, a versão com elitismo já atingia cerca de 70.000 pontos, enquanto a sem elitismo ainda explorava soluções variadas com notas

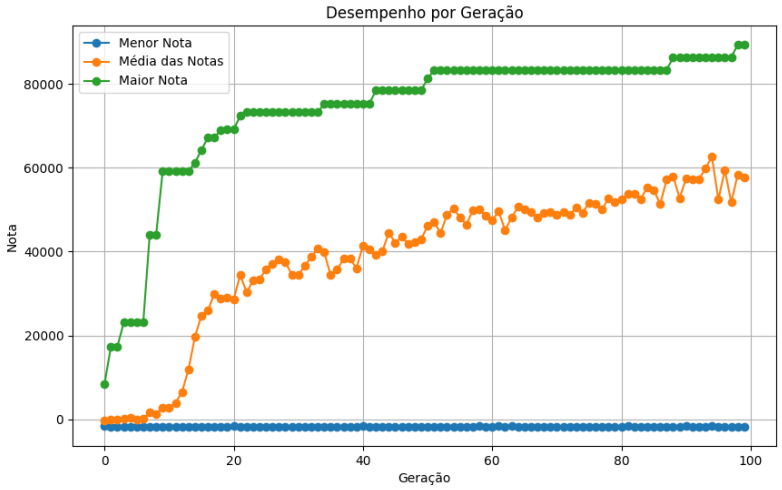


Figura 5.7: Resultado gráfico para 100 gerações de uma população de 100 redes neurais com aplicação do elitismo.

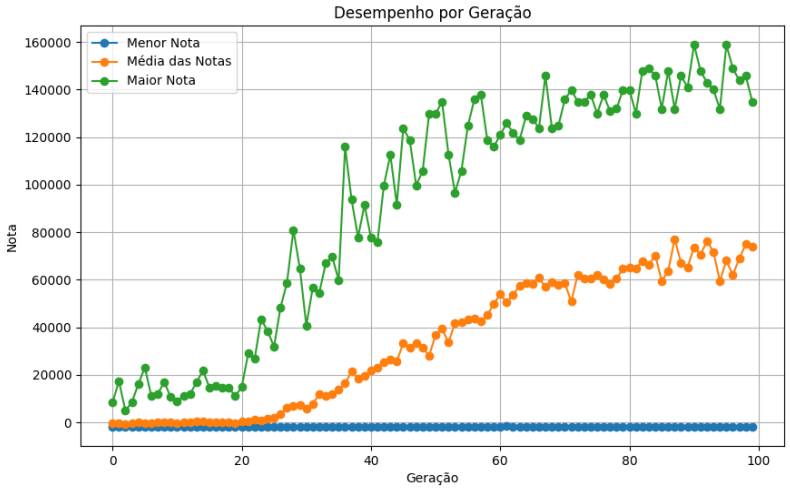


Figura 5.8: Resultado gráfico para 100 gerações de uma população com 100 redes neurais sem aplicação do elitismo.

mais baixas. No entanto, após esse ponto, a versão com elitismo apresentou longos períodos de estagnação, ao mesmo tempo que a versão sem elitismo continuou melhorando gradualmente, atingindo picos próximos a 160.000 pontos.

A hipótese para esse comportamento está no impacto que o elitismo exerce sobre a diversidade genética. Sem elitismo, várias soluções de desempenho semelhante, mas com genótipos diferentes, são combinadas, permitindo a geração de soluções novas e potencialmente melhores. Já com o elitismo, uma solução muito superior tende a dominar a seleção (especialmente no caso da seleção via roleta, que foi a implementada no trabalho), reduzindo drasticamente a chance de outras abordagens prosperarem. Isso dificulta a saída das soluções de ótimos locais, pois há menos variabilidade genética e menor probabilidade de mutações benéficas.

Assim, embora o elitismo ajude a manter boas soluções, e evitar quedas no crescimento das melhores notas, seu uso precisa ser equilibrado com estratégias que garantam que a diversidade não será perdida, como uso de mais de um algoritmo de seleção para recombinação.

5.4 IMPACTO DA MUDANÇA DE ARENAS

Com o intuito de verificar mudanças de comportamento e na velocidade do crescimento das notas de aptidão dos enxames em diferentes cenários, foram desenvolvidas duas arenas adicionais do mesmo tamanho da original utilizada nos outros experimentos, porém com a adição de obstáculos. Uma das arenas possui obstáculos sendo duas paredes, com o objetivo de verificar o comportamento dos enxames no caso de um mapa que exigisse um desvio para transitar entre a fonte de alimento e o ninho. A outra arena possui como obstáculo uma parede entre o alimento e o resto da arena, para verificar o comportamento do enxame quando é necessário encontrar um caminho não convencional para acessar a fonte de alimento.

Para comparação gráfica dos mapas com o caso base, foram utilizados os mesmos parâmetros do experimento do elitismo no caso sem o uso desse operador (100 gerações de uma população com 100 indivíduos, mutação gaussiana reversa, taxa de mutação de 10%, e sem elitismo), no mapa sem obstáculos, obtendo-se o resultado gráfico apresentado na Figura 5.9 [1].

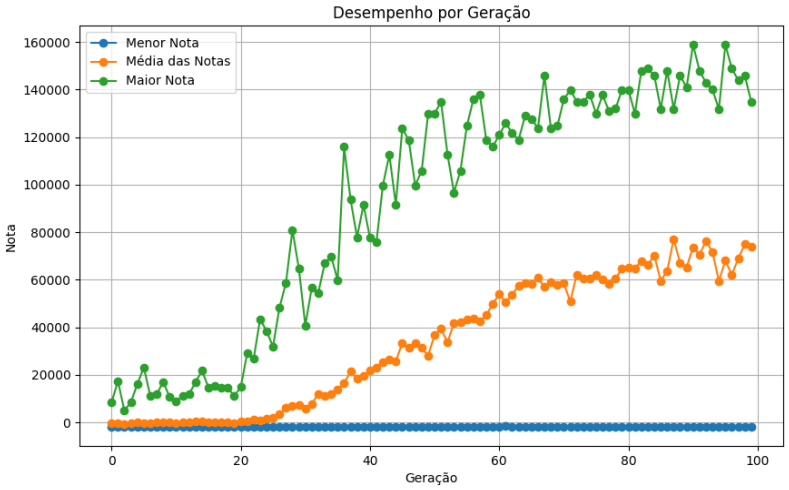


Figura 5.9: Resultado gráfico da execução de 100 gerações de uma população com 100 redes neurais no mapa sem obstáculos.

5.4.1 Arena com afunilamento

A primeira das arenas alternativas é um mapa com duas paredes no meio, além das paredes limitadoras da área, ambas partindo dos cantos, de forma que os robôs precisem ir para o centro do mapa para transitar entre a região do ninho e a região com a fonte de alimento. A Figura 5.10 apresenta a arena, com alguns robôs transitando por ela.

A execução desse experimento se deu com os seguintes parâmetros: 100 gerações de uma população com 100 enxames, sem elitismo, mutação com função gaussiana inversa e taxa de mutação de 10%. Os resultados apresentados na Figura 5.11 mostraram-se bastante similares ao caso base, da Figura 3.2, porém, de forma mais suave no melhor enxame e nas médias das aptidões dos enxames, não chegando a atingir os mesmos patamares na nota máxima. A maior

[1] A execução do melhor enxame no mapa sem obstáculos pode ser visualizada pelo endereço `https://youtu.be/rVTgszNxqjU`

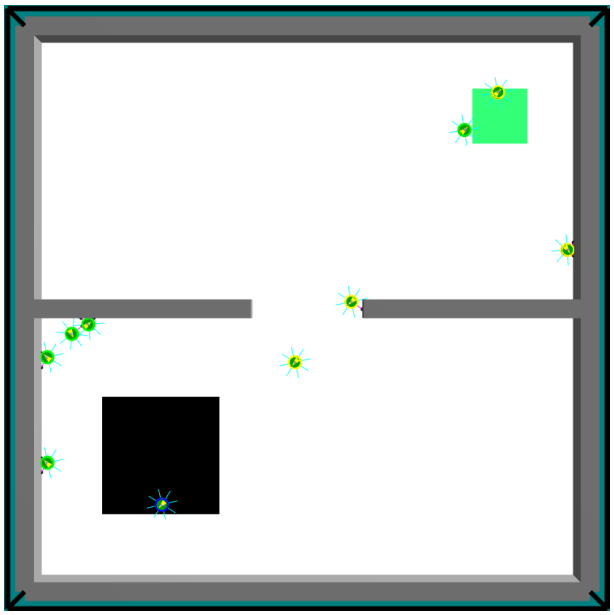


Figura 5.10: Mapa com afunilamento, com um enxame transitando por ele.

nota no mapa com afunilamento foi de menos de 120.000 pontos, relativamente menor que a pontuação de 160.000 atingida na execução sem obstáculos [2].

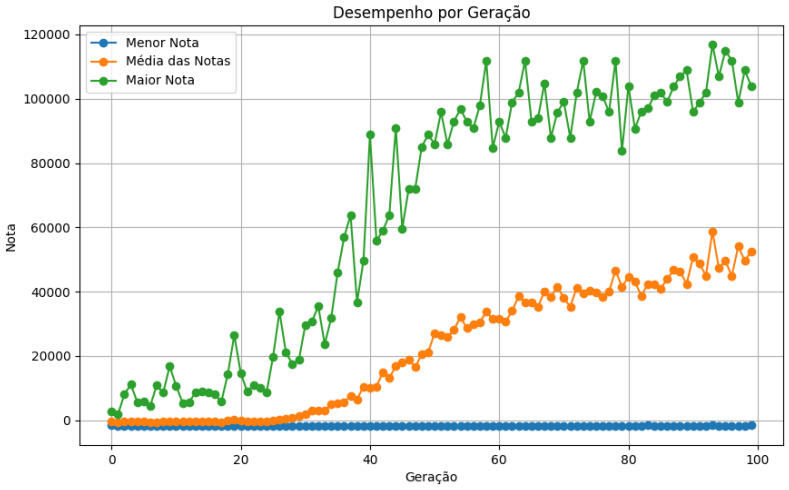


Figura 5.11: Resultado gráfico da execução de 100 gerações de uma população com 100 redes neurais no mapa com afunilamento.

Uma hipótese para essa menor nota é o maior tempo gasto pelas redes neurais em tentar superar o obstáculo do afunilamento, e em seguida, ajustar a rota para atingir a área com alimento.

5.4.2 Arena com obstáculo ao lado da fonte de alimento

A segunda arena alternativa é um mapa de mesmo tamanho e posição da fonte de alimento e do ninho que o mapa sem obstáculos, porém com uma parede entre a fonte de alimento

[2] A execução do melhor enxame no mapa com afunilamento pode ser visualizada pelo endereço `https://youtu.be/gQT7AWmji5o`

e o restante da arena, exigindo que os robôs façam um caminho fora do convencional para chegar ao alimento, desviando da parede e indo até o final do caminho para atingir a fonte. A Figura 5.12 apresenta a arena, com um enxame transitando por ela.

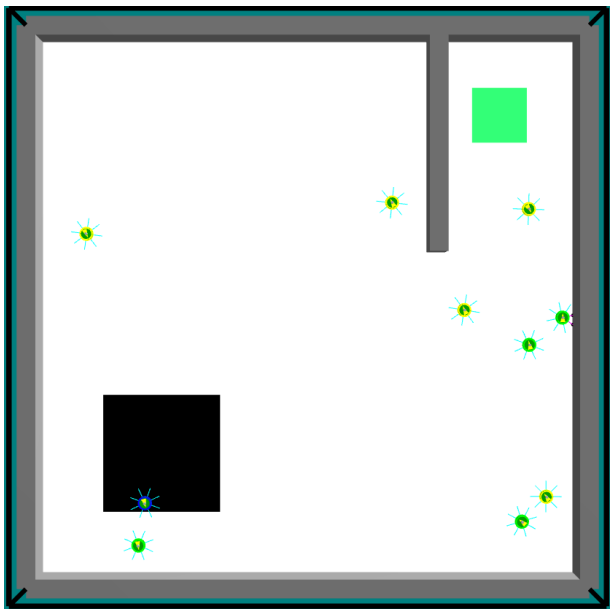


Figura 5.12: Mapa com obstáculo ao lado da fonte de alimento, com um enxame transitando por ele.

A execução desse experimento se deu com os seguintes parâmetros: 100 gerações de uma população com 100 enxames, sem elitismo, mutação com função gaussiana inversa e taxa de mutação de 10%. Os resultados apresentados na Figura 5.13 mostraram uma queda maior ainda que a verificada no primeiro mapa alternativa. O resultado máximo atingido na execução foi de pouco mais de 70.000 pontos, enquanto no mapa sem obstáculos foi de 160.000, mais que a metade da pontuação do enxame desenvolvido no mapa sem obstáculos.

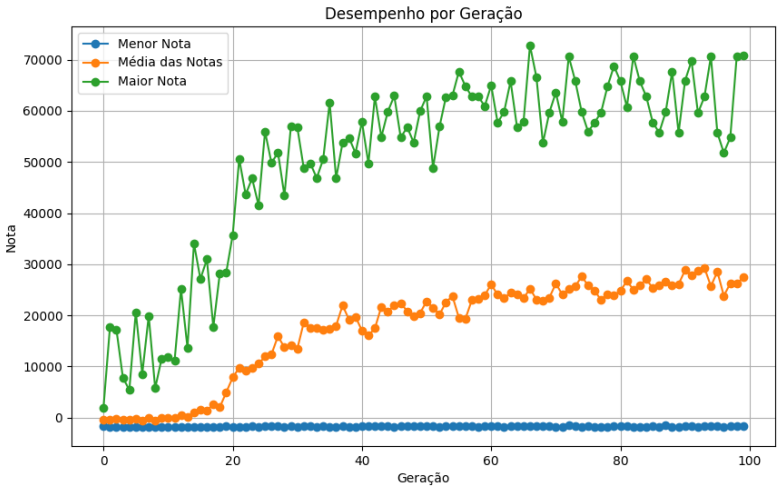


Figura 5.13: Resultado gráfico da execução de 100 gerações de uma população com 100 redes neurais no mapa com obstáculo ao lado da fonte de alimento.

Essa diferença de pontos de aptidão, indica que a quantidade de vezes que os robôs conseguiram completar o forrageamento no mapa com o obstáculo ao lado da fonte de alimento

foi menos da metade quando comparado ao mapa sem obstáculos. Essa mudança se explica pelo tamanho maior do caminho mínimo para atingir a fonte de alimento, além do maior tempo inicial gasto pelos enxames para encontrar um caminho viável para a fonte de alimento. Da mesma forma que ocorreu no mapa com afunilamento, os gráficos de notas médias e nota máxima foi mais suave, porém com uma suavização mais evidente, essa característica provavelmente se dá pela maior complexidade do mapa com o obstáculo lateral [3].

5.5 TESTES FINAIS

Por fim, foram realizados testes nos três mapas apresentados anteriormente a fim de maximizar a pontuação do melhor enxame, com todas as condições favoráveis para esse objetivo, entre elas, uma grande quantidade de enxames por geração (100), sendo evoluídos por 1.000 geração, utilizando o estilo de mutação que mais favoreceu o aumento da pontuação, ou seja, a função gaussiana inversa, além de não utilizar o elitismo.

O motivo para o aumento da quantidade de gerações foi garantir que as soluções encontradas nas arenas com obstáculos conseguissem evoluir para soluções mais otimizadas, além de permitir verificar se o mapa sem obstáculos atingiria um ótimo ideal para o problema de forrageamento nesse mapa.

Os resultados obtidos nesse experimento foram bastante promissores nos três casos e estão detalhados a seguir:

5.5.1 Mapa sem obstáculos

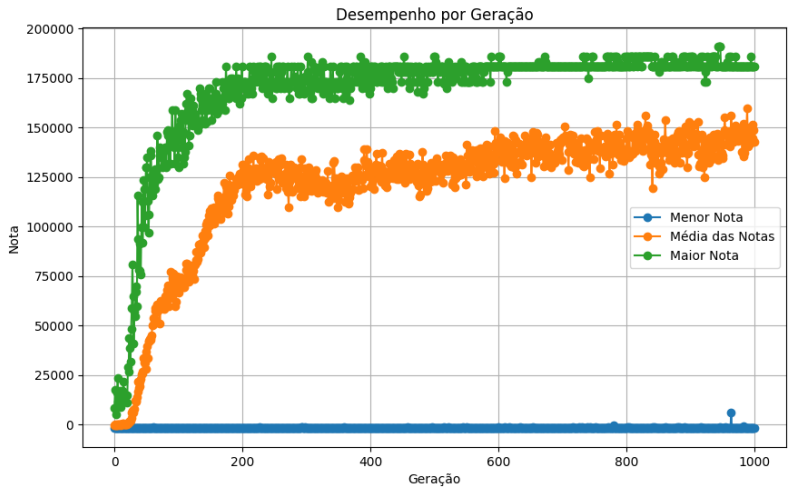


Figura 5.14: Resultado gráfico da execução de 1.000 gerações de uma população com 100 redes neurais no mapa sem obstáculos.

Para o mapa sem obstáculos, ao executar o algoritmo por 1.000 gerações, foram obtidos os dados exibidos na Figura 5.14, em que foi possível verificar que após a geração 200, o crescimento da melhor nota foi lento, e após a geração 600 convergiu para um ótimo local, exibindo estagnação em dezenas de gerações, indicando uma possível convergência de soluções

[3] A execução do melhor enxame no mapa com obstáculo lateral pode ser visualizada pelo endereço `https://youtu.be/qAl59SMHtoM`

em um ótimo local, o que prejudica bastante a variedade genética, porém, pode também indiciar que as soluções estão se aproximando de um ótimo global para a solução do problema.

A maior nota de aptidão das gerações foi de mais de 180.000 pontos, um aumento de mais de 20.000 pontos quando comparado com a execução de 100 gerações, porém, uma variação pequena levando-se em conta o poder computacional necessário 10 vezes maior. Pelo gráfico é possível também definir que o período de maior impacto nas notas máximas foi entre a geração 0 e a geração 200, aproximadamente, em que as soluções máximas passaram a atingir valores próximos aos 175.000 pontos, somente 5.000 pontos abaixo das soluções encontradas pelo algoritmo 800 gerações depois.

Analisando a simulação do melhor enxame de todos as 1.000 gerações, que atingiu uma nota de aptidão de 190.932 pontos, é possível notar que o comportamento das redes neurais possui duas fases:

Inicialmente, enquanto nenhum robô atingiu a fonte de alimento, todos os robôs partem para direções diferentes, a fim de maximizar a probabilidade de encontrar uma fonte de recurso. A Figura 5.15 mostra uma captura de tela da simulação com os robôs exprimindo esse comportamento.

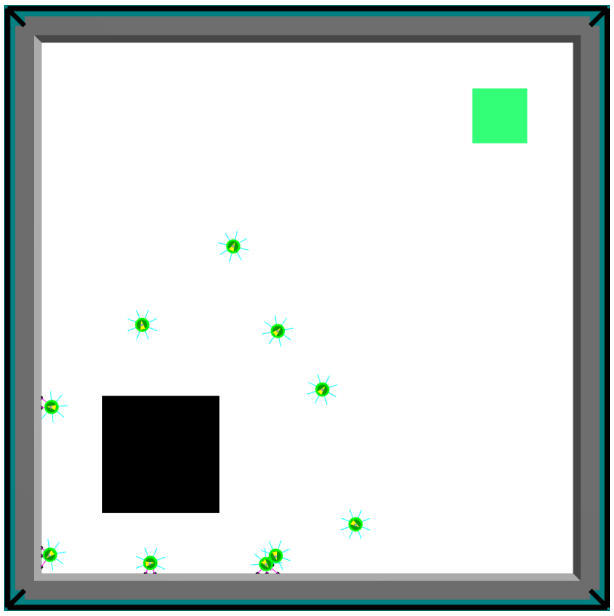


Figura 5.15: Captura de tela da execução da melhor rede neural, com todos os robôs indo em diferentes direções.

Então, assim que um dos robôs atinge a fonte de alimento, os robôs passam a receber pelo sensor de *range and bearing* a posição em que o primeiro robô encontrou alimento e todos passam a seguir um caminho circular, de forma que eles se choquem o mínimo possível e consigam fazer o caminho entre o ninho e a fonte de alimento. Esse comportamento pode ser verificado na Figura 5.16.

Por meio da análise da simulação, também foi possível verificar que o enxame conseguiu atingir a fonte de alimento 27 vezes, e voltar para o ninho com uma unidade de alimento 22 vezes. Uma média de 2,7 objetivos parciais , e 2,2 objetivos totais do forrageamento alcançados por robô em apenas 2.000 ciclos da simulação.

5.5.2 Mapa com afunilamento

No mapa com o obstáculo de afunilamento, foram obtidos os dados presentes na Figura 5.17. Por meio da análise gráfica, é possível verificar que a execução do algoritmo também

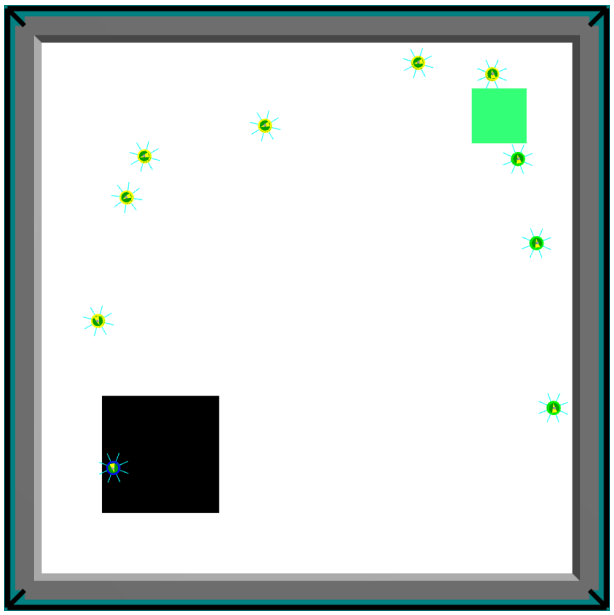


Figura 5.16: Captura de tela da execução da melhor rede neural, com todos os robôs seguindo um caminho circular para realizar o forrageamento.

esbarrou em um ótimo local, uma vez que, a partir da geração 900, a variação da população decaiu bastante, já que a média das notas se aproximou bastante da melhor nota, e a melhor nota ficou constante por dezenas de gerações.

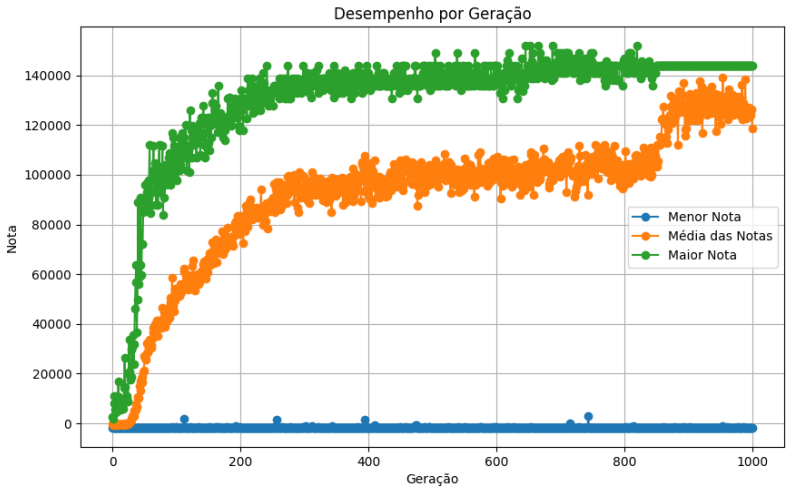


Figura 5.17: Resultado gráfico da execução de 1.000 gerações de uma população com 100 redes neurais no mapa com afunilamento.

Da mesma forma que ocorreu no mapa sem obstáculos, as gerações inicias foram as mais expressivas em termos de crescimento da pontuação do melhor enxame, e a partir de um certo ponto, a melhoria na nota máxima foi bem menos expressiva. Porém, esse comportamento ao invés de ter cessado em torno da geração 200, como ocorreu no mapa sem obstáculos, ocorreu por volta da geração 300, em que as notas máximas orbitavam em torno da pontuação 140.000, sendo aproximadamente 10.000 pontos abaixo das notas máximas dos enxames.

Essa variação de aproximadamente 100 gerações entre a convergência de valores para um ótimo local entre os mapas confirma a hipótese apresentada na Seção 5.4, que pela dificuldade

maior em ultrapassar o obstáculo, as redes neurais precisavam de mais tempo de treinamento para atingir notas superiores quando comparadas com o mapa sem obstáculos.

O melhor enxame atingiu uma pontuação de 151.940 pontos, e a abordagem desse enxame segue uma solução parecida à encontrada no mapa sem obstáculos, em que existe uma fase de exploração e em seguida uma fase em que o enxame passa a seguir um caminho englobando a fonte de alimento e o ninho.

A fase de exploração é apresentada na Figura 5.18, e se mostra bastante similar à realizada pelo melhor enxame no mapa sem obstáculos.

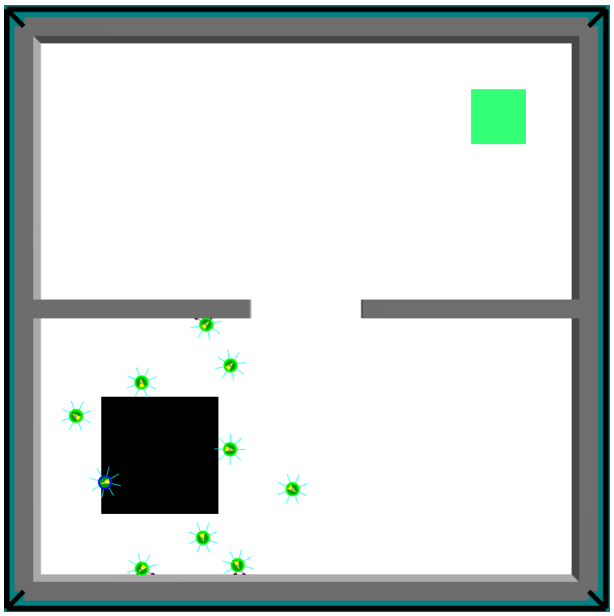


Figura 5.18: Captura de tela da execução da melhor rede neural no mapa com afunilamento, com todos os robôs indo em diferentes direções.

Já a fase do forrageamento em si, apresentada na Figura 5.19 se mostrou um pouco diferente pela particularidade do mapa em que a simulação foi realizada. Nela, o enxame desenvolveu uma solução que envolve se guiar pelas paredes da arena, para evitar que os robôs indo para a fonte de alimento esbarrassem nos robôs que estão voltando, essa abordagem é bastante interessante por ser personalizada para o ambiente em que os robôs se encontram, em que os indivíduos precisam passar por uma região estreita, compartilhada por vários robôs indo em diferentes direções.

O melhor enxame conseguiu obter uma unidade de alimento 21 vezes, conseguindo realizar o forrageamento completo 18 vezes. A quantidade de obtenção de alimento foi aproximadamente 22% menor que o do mapa sem obstáculos, e a quantidade de finalizações do forrageamento foi parecida, com 18% menos finalizações.

5.5.3 Mapa com obstáculo ao lado da fonte de alimento

Para o mapa com um obstáculo entre a fonte de alimento e o restante da arena, o comportamento gráfico de atingir um ótimo local, como ocorreu nos últimos dois mapas, não chegou a ocorrer. Embora tenha ocorrido um salto nas notas dos melhores enxames entre a geração 0 e 200, aproximadamente, logo em seguida, ocorreu um aumento lento, porém consistente, nas melhores notas. Isso pode indicar que, se o experimento se estendesse por mais gerações, a nota dos melhores enxames continuaria crescendo, até que atingisse um ótimo local, como ocorreu nos outros casos.

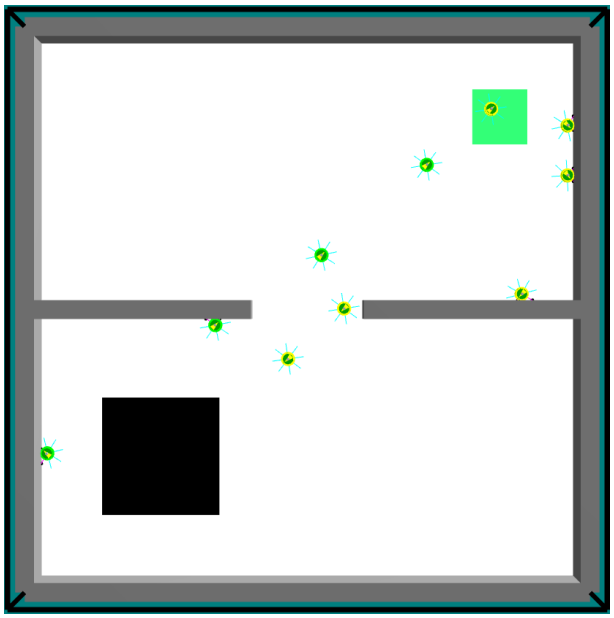


Figura 5.19: Captura de tela da execução da melhor rede neural, com os robôs seguindo um caminho rente às paredes para realizar o forrageamento.

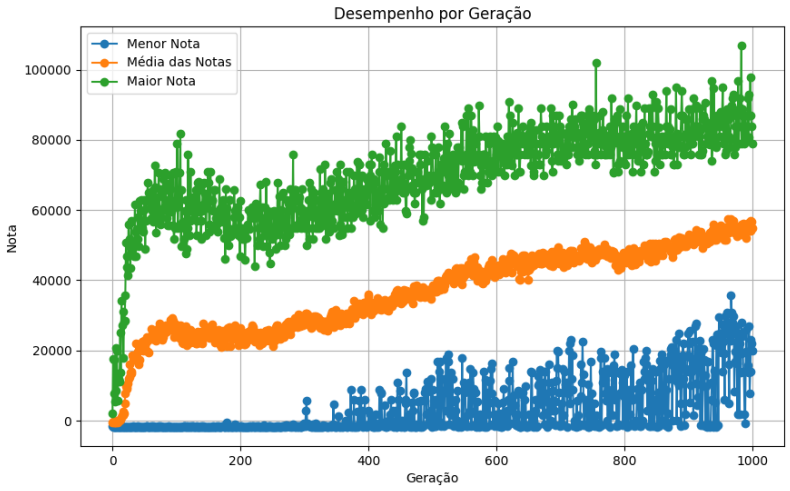


Figura 5.20: Resultado gráfico da execução de 1.000 gerações de uma população com 100 redes neurais no mapa com obstáculo ao lado da fonte de alimento.

Um comportamento que também foi diferentes do apresentado nos últimos dois experimentos, foi que as menores notas não se mantiveram próximas a zero o experimento todo. A partir da geração 300, os piores enxames começaram a seguir o aumento das notas dos enxames médios e melhor enxame, indicando que todos os enxames passaram a atingir, mesmo que parcialmente os objetivos do forrageamento. O gráfico pode ser verificado na Figura 5.20.

A maior nota dos enxames, atingiu valores acima dos 100.000 pontos, porém, ainda foi bem menor que os valores das outras simulações: 106.870 pontos de aptidão, indicando que esse foi o problema mais difícil para o algoritmo genético solucionar.

A estratégia de realizar uma busca em todas as direções e após um dos robôs atingir a fonte de alimento, desenvolver um caminho também ocorreu, porém o processo de buscar a fonte de alimento demorou mais, impactando na pontuação final do enxame. Além disso, o caminho

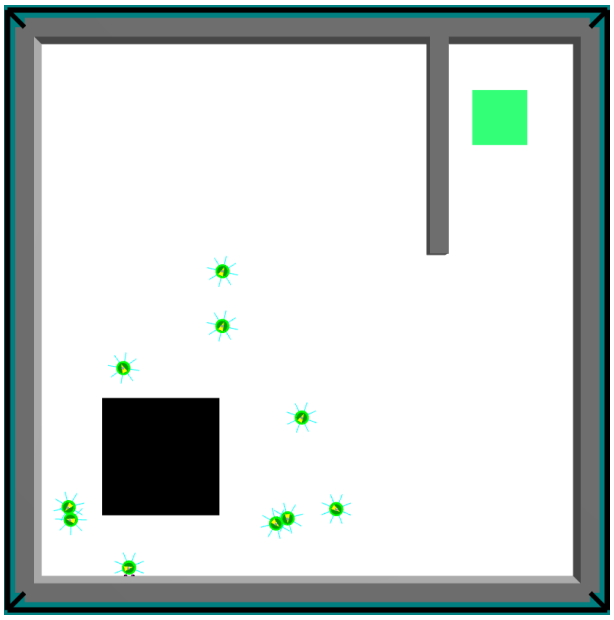


Figura 5.21: Captura de tela da execução da melhor rede neural no mapa com obstáculo entre a fonte de alimento e o restante da arena, com todos os robôs indo em diferentes direções.

desenvolvido pelo enxame não foi padronizado, causando diversos choques entre robôs, além de robôs se perdendo e não conseguindo atingir seus objetivos. A fase de busca é ilustrada pela Figura 5.21, e a fase de forrageamento pela Figura 5.22.

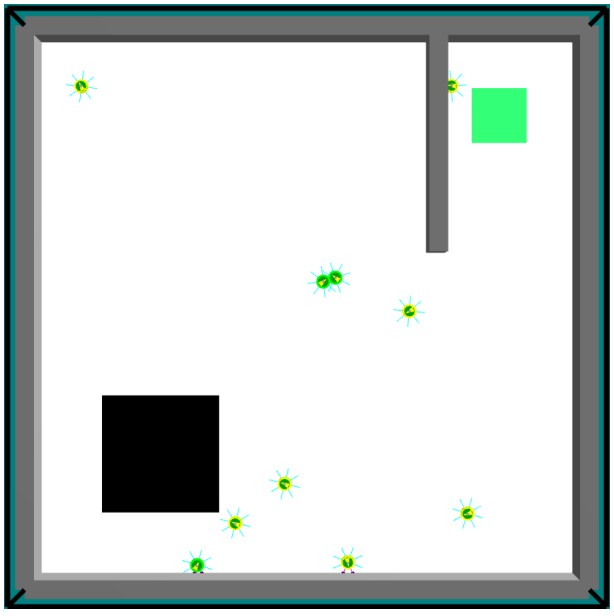


Figura 5.22: Captura de tela da execução da melhor rede neural, com os robôs realizando o forrageamento sem um caminho padronizado.

O melhor enxame, embora sem um caminho padrão e com diversos choques entre indivíduos, conseguiu com que os robôs chegassem a fonte de alimento 14 vezes, e retornassem para o ninho com uma unidade de alimento 13 vezes. Embora esse enxame tenha conquistado as menores quantidades de finalizações dos objetivos, curiosamente, teve um aproveitamento entre a obtenção do alimento, e retorno dele para o ninho de aproximadamente 93%.

5.6 CONSIDERAÇÕES FINAIS

Os experimentos conduzidos neste capítulo permitiram uma análise aprofundada do comportamento de um algoritmo genético aplicado à evolução de redes neurais em um cenário de robótica de enxame com tarefas de forrageamento. Diversas configurações foram testadas, envolvendo diferentes estratégias de mutação, tamanhos e quantidades de gerações, aplicação de elitismo e modificações no ambiente físico da simulação, permitindo observar o impacto direto desses fatores sobre a qualidade das soluções geradas.

Em relação às estratégias de mutação, foi possível constatar que a escolha da função de variação exerce influência decisiva sobre a capacidade do algoritmo de escapar de ótimos locais e explorar de forma mais eficaz o espaço de busca. A mutação gaussiana tradicional mostrou-se conservadora demais, resultando em pouca variação genética e convergência precoce. Já a mutação linear apresentou desempenho intermediário, com maior diversidade, mas crescimento mais lento. A mutação gaussiana reversa destacou-se por sua agressividade controlada, promovendo saltos evolutivos mais significativos, o que a tornou a abordagem mais eficaz no contexto apresentado.

Os experimentos relacionados ao tamanho da população e quantidade de gerações evidenciaram a importância de se manter um equilíbrio entre diversidade genética e tempo de evolução. Gerações pequenas com muitas iterações não foram suficientes para promover avanços relevantes, devido à baixa variabilidade da população. Por outro lado, grandes populações com poucas iterações também limitaram o refinamento das soluções. O melhor desempenho foi obtido com um número balanceado entre quantidade de redes neurais por geração e número de gerações, reforçando a importância do planejamento cuidadoso da estrutura do algoritmo genético.

A análise do elitismo revelou que sua aplicação, embora traga estabilidade à evolução, pode, em certos contextos, comprometer o desempenho ao reduzir drasticamente a variabilidade genética. No experimento conduzido, a ausência de elitismo permitiu que o algoritmo continuasse explorando novas soluções por mais tempo, atingindo melhores notas finais. Isso indica que, especialmente em cenários onde a diversidade genética é crucial, o elitismo deve ser utilizado com cautela ou complementado com estratégias que preservem a variabilidade da população.

Em relação ao ambiente, os testes com diferentes arenas demonstraram que a complexidade do mapa influencia diretamente a velocidade de convergência e a pontuação final das soluções. Arenas com obstáculos mais desafiadores exigiram mais tempo de evolução e apresentaram pontuações máximas mais baixas, indicando que a dificuldade do ambiente pode limitar o desempenho do enxame se não houver tempo e diversidade suficientes para adaptação. Ainda assim, as redes neurais foram capazes de desenvolver comportamentos adaptativos específicos para cada arena, como o uso das paredes como guia, o que evidencia a capacidade de generalização e especialização do modelo evolutivo.

Por fim, os testes finais reforçaram a importância de uma configuração adequada dos parâmetros evolutivos. A combinação de uma função de mutação eficaz, ausência de elitismo, grande população e número elevado de gerações mostrou-se capaz de gerar comportamentos complexos e eficientes, com enxames colaborando de forma coordenada e atingindo pontuações elevadas mesmo em ambientes desafiadores.

De forma geral, os resultados obtidos validam a abordagem proposta, demonstrando que é possível evoluir redes neurais eficazes para tarefas de forrageamento em robótica de enxame por meio de técnicas evolutivas. Além disso, os experimentos revelaram nuances importantes sobre a dinâmica da evolução artificial e destacaram pontos críticos para a configuração de algoritmos genéticos eficientes nesse contexto.

6 CONCLUSÃO

Este trabalho demonstrou que é possível resolver problemas de coletas de recursos por meio de um algoritmo genético que manipula redes neurais, utilizando um simulador de robótica realista para isso. Através da evolução das redes neurais, durante gerações, os enxames desenvolveram comportamentos inteligentes e coletivos, conseguindo resolver o problema do forrageamento de forma eficiente e cooperativa em diferentes cenários.

A escolha de um controlador neural unificado, replicado em todos os indivíduos do enxame, mostrou-se coerente com os princípios da inteligência de enxame, permitindo a adaptação coletiva do grupo sem a necessidade de um controle centralizado desenvolvido programaticamente.

A utilização do simulador *ARGoS3*, com modelos realistas de sensores e atuadores, garantiu um ambiente veloz, reprodutível, e viabilizou a evolução de redes neurais por milhares de gerações em poucas horas de simulação, algo que seria inviável se feito utilizando robôs físicos, devido ao tempo necessário para executar todas as instâncias dos experimentos até que houvesse a emersão de um comportamento coletivo.

Além disso, foi possível verificar por meio deste trabalho, o impacto de diferentes operadores, abordagens e cenários no desenvolvimento dos comportamentos coletivos pelos enxames.

Como continuidade deste trabalho, sugere-se uma implementação das redes neurais obtidas nesse projeto em controladores físico de um enxame de *e-pucks* reais, a fim de verificar se os comportamentos demonstrados via simulador poderão ser observados em um contexto real também. Ademais, sugere-se aplicar o *novelty search* no algoritmo para evitar que as soluções fiquem presas em ótimos locais, assim como expandir a evolução das redes neurais para que, além de modificar parâmetros, permita evoluir a própria topologia da rede, como ocorre, por exemplo, no algoritmo do NEAT e HyperNEAT.

REFERÊNCIAS

Bartz-Beielstein, T., Branke, J., Mehnen, J. e Mersmann, O. (2014). Evolutionary algorithms. *WIREs Data Mining and Knowledge Discovery*, 4(3):178–195.

Beni, G. e Wang, J. (1993). Swarm intelligence in cellular robotic systems. Em *Robots and Biological Systems: Towards a New Bionics?*, página 703–712, Riverside, California - USA.

Bredèche, N., Montanier, J., Weel, B. e Haasdijk, E. (2013). Roborobo! a fast robot simulator for swarm and collective robotics. *CoRR*, abs/1304.2888.

Cheraghi, A. R., Shahzad, S. e Graffi, K. (2021). Past, present, and future of swarm robotics.

Haykin, S. (2009). *Neural Networks and Learning Machines*. Número v. 10 em Neural networks and learning machines. Prentice Hall.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

King, D. E. (2009). Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758.

Magnenat, S., Waibel, M. e Beyeler, A. (2011). Enki: The fast 2d robot simulator. *URL http://home. gna. org/enki*.

Michel, O. (2004). Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42.

Millard, A. G., Joyce, R., Hilder, J. A., Fleşeriu, C., Newbrook, L., Li, W., McDaid, L. J. e Halliday, D. M. (2017). The pi-puck extension board: A raspberry pi interface for the e-puck robot platform. Em *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, páginas 741–748.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D. e Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. Em *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, volume 1, páginas 59–65.

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L. M. e Dorigo, M. (2012). ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295.

Rohmer, E., Singh, S. P. N. e Freese, M. (2013). Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. Em *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. www.coppeliarobotics.com.

Trianni, V., IJsselmuiden, J. e Haken, R. (2016). The saga concept: Swarm robotics for agricultural applications.