UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Marco Aurélio Martins Mijam

Sistema para desenvolvimento de inteligência artificial para futebol de robôs utilizando algoritmo evolutivo

São Carlos

2019

Marco Aurélio Martins Mijam

Sistema para desenvolvimento de inteligência artificial para futebol de robôs utilizando algoritmo evolutivo

Monografia apresentada ao Curso de Engenharia Elétrica com Ênfase em Sistemas de Energia e Automação, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Eduardo do Valle Simões

São Carlos

2019

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Mijam, Marco Aurélio Martins
M618s Sistema para desenvolvimento de inteligência artificial para futebol de robôs utilizando algoritmo evolutivo / Marco Aurélio Martins Mijam; orientador Eduardo do Valle Simões. São Carlos, 2019.

Monografia (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2019.

1. Algoritmo evolutivo. 2. Algoritmo genético. 3. Rede neural artificial. 4. Futebol de robôs. 5. Simulador físico. 6. Inteligência artificial. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Marco Aurélio Martins Mijam

Título: "Sistema para desenvolvimento de inteligência artificial para futebol de robôs utilizando algoritmo evolutivo"

Trabalho de Conclusão de Curso defendido e aprovado em 29 / 11 / 19 ,

com NOTA 7,5 (sete , cinco), pela Comissão Julgadora:

Prof. Dr. Eduardo do Valle Simões - Orientador - SSC/ICMC/USP

Prof. Dr. Valdir Grassi Júnior - SEL/EESC/USP

Prof. (Dr.) Associado Emerson Carlos Pedrino - UFSCar

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

AGRADECIMENTOS

Agradeço aos meus pais Geraldo e Edivânia Mijam, por em todas as fases da minha vida sempre me apoiarem e estarem ao meu lado, por todos os conselhos que me deram, por me servirem de inspiração muitas vezes e me motivarem a ser melhor a cada dia. Ao meu irmão e melhor amigo João Victor Mijam, por todas as conversas que já tivemos e estar comigo sempre.

Aos meus amigos Matheus Zanetti, Daniel Kondo e Gustavo Labegalini que me acompanharam durante toda a graduação, ajudando a tirar dúvidas, dando suporte na graduação e ajudando a deixar a vida na universidade mais leve com sua amizade.

Ao meu amigo e irmão da vida Rodrigo Araujo, que morou comigo durante a maior parte da graduação, dividimos problemas, grandes reflexões sobre a vida e madrugadas estudando juntos.

Ao pessoal do laboratório LAPRAS por todo o conhecimento e experiências compartilhadas, em especial a Bruna Dell'Avanzi, Camila Stenico, Guilherme Rocha e João Marcos Guido pelo apoio e suporte na fase final do projeto.

Ao meu orientador Eduardo Simões, por todo o apoio na escrita do trabalho, me orientando e inspirando com suas ideias e animação.

À Clariane Molina, por todo o apoio na etapa final do projeto e ter ajudado em vários momentos do trabalho.

À secretária do departamento de engenharia elétrica, Jussara Ramos, por sempre estar disposta a ajudar os alunos, com toda a sua gentileza, suavizando a vida na graduação.

"[...] aprendeu rápido porque seu primeiro ensinamento foi sobre como aprender. E a sua primeira lição foi a confiança básica de que ele pode aprender."

(Frank Herbert)

RESUMO

MIJAM, M. A. M. **Sistema para desenvolvimento de inteligência artificial para futebol de robôs utilizando algoritmo evolutivo**. 2019. 52p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019.

O projeto tem como objetivo desenvolver um simulador de robôs da categoria de futebol de robôs *Very Small Size* e o integrar a uma rede neural artificial que pode os controlar. Foi desenvolvido um algoritmo genético capaz de treinar a rede neural artificial utilizando o simulador. O sistema todo foi desenvolvido de forma modular para ser facilmente modificado por terceiros e utilizado com outros algoritmos de inteligência artificial. O treinamento de uma rede neural artificial utilizando o algoritmo genético resultou em uma inteligência artificial capaz de mover um robô de um ponto a outro do campo, provando-se assim o conceito de utilizar o simulador como ferramenta para desenvolvimento de inteligência artificial com algoritmo evolutivo.

Palavras-chave: Algoritmo evolutivo. Algoritmo genético. Rede neural artificial. Futebol de Robôs. Simulador físico. Inteligência artificial.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AE	Algoritmo Evolutivo
AG	Algoritmo Genético
RNA	Rede Neural Artificial
PMC	Perceptron Multicamadas
VSS	IEEE Very Small Size
VSSS	IEEE Very Small Size Soccer
IA	Inteligência Artificial
RoboCup	*Robot World Cup Initiative*

SUMÁRIO

1 INTRODUÇÃO

Com o crescimento do uso da robótica no meio industrial e com o advento da quarta revolução industrial há uma crescente demanda por capacitação de pessoas e pesquisa nas áreas de robótica, computação na nuvem e inteligência artificial (IA). O uso dessas técnicas no meio industrial faz com que novos produtos e serviços sejam criados, custos sejam reduzidos e que haja um aumento na produtividade (Petrasch; Hentschke, 2016).

Muitos dos desenvolvimentos tecnológicos mais recentes estudados hoje em dia tem sua inspiração vinda da biologia, como por exemplo as redes neurais artificiais (RNA) e os algoritmos genéticos (AG), que ambos são tipos de IA. A RNA tem sua inspiração no funcionamento do neurônio e em suas capacidades de armazenar informações e encontrar padrões (Uhrig, 1995). Os AGs tem sua origem no funcionamento do código genético humano da teoria da evolução de Darwin (DARWIN, 1859).

Como uma tentativa de motivar o desenvolvimento tecnológico surgiu a *Robot World Cup Initiative* (RoboCup). A RoboCup provê um problema onde há uma grande variedade de tecnologias que podem ser integradas e examinadas (KITANO et al., 1997a). Incentivados pela RoboCup, diversos grupos vem pesquisando temas relacionados com robótica, IA e simulação. Alguns dos projetos motivados pela RoboCup é (MERIÇLI; MERIÇLI; AKıN, 2010) e (FERNÁNDEZ; COTTA; CEBALLOS, 2008), onde foi utilizado um algoritmo genético (AG) para desenvolver uma IA que faz a tomada de decisões de alto nível de um time de futebol de robôs, como por exemplo decidir se o robô chuta a bola ou se passa ela para outro jogador. Outro projeto que foi motivado pela RoboCup é (ALBAB; WIBOWO; BASUKI, 2017) onde foi desenvolvido um algoritmo de planejamento de rotas para robôs móveis utilizando algoritmos genéticos. Foi desenvolvido também um simulador de robôs para facilitar o desenvolvimento de estratégias de futebol, que a equipe utilizou para participar da RoboCup (SILVA et al., 2010).

Uma forma diferente da que foi mostrada anteriormente de se utilizar o AG para o treinamento de uma IA para jogar futebol de robôs, é de em vez de se utilizar o AG para o treinamento da tomada de decisões de alto nível, utilizá-lo também para a tomada de decisões no baixo nível, como por exemplo a movimentação do goleiro, o movimento que o robô faz para chutar no gol ou o jeito que ele intercepta a bola. Uma problemática que surge para a utilização de um AG para esse tipo de problema é que ele envolve uma complexidade maior da IA gerada pelo AG, o que faz com que o seu treinamento demore mais e isso pode ser inviável de ser feito utilizando um simulador como o desenvolvido por (SILVA et al., 2010), é necessário um simulador capaz de processar a partida de forma mais rápida mesmo que menos precisa.

1.1 Objetivo

O objetivo desse projeto é a construção de uma plataforma para desenvolvimento de softwares de inteligência artificial para controle de futebol de robôs na categoria VSS (*IEEE Very Small Size*), utilizando algoritmos evolutivos para otimização das estratégias. O sistema possui integrado um simulador de robôs que pode operar sem exibição gráfica para ganhar velocidade. A plataforma é projetada em módulos, de forma a tornar possível futuras modificações de suas partes: simulador, parte gráfica, mecanismo da IA e algoritmo de otimização. Com esta plataforma serão realizados testes para se estudar a aplicabilidade de AGs e redes neurais artificiais (RNA) na construção de uma IA para o controle de um time de futebol de robôs. Dessa maneira, esse trabalho pretende avaliar se é possível desenvolver um IA para futebol de robôs utilizando uma RNA treinada por um AG.

2 FUNDAMENTOS TEÓRICOS

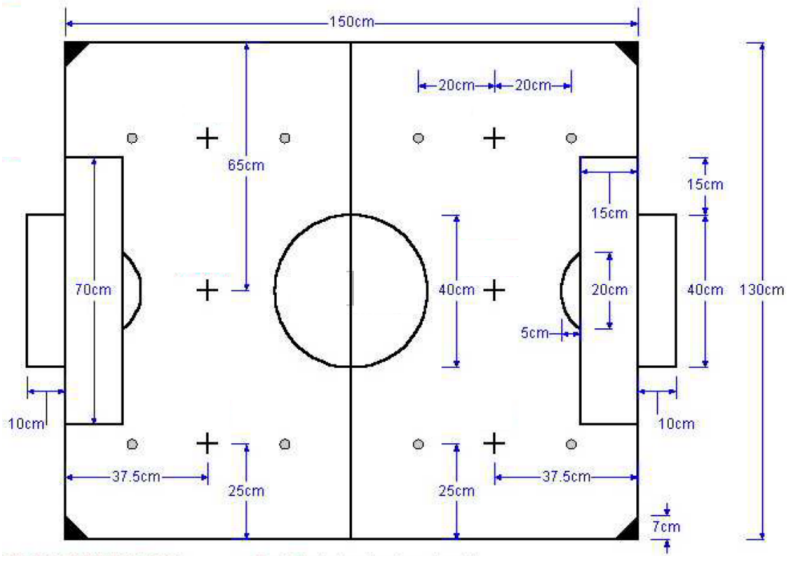
2.1 Futebol de Robôs

A RoboCup é uma competição de robótica anual internacional. Ela foi fundada em 1997 e tem como objetivo, até a metade do século XXI, desenvolver uma equipe de robôs humanoides totalmente autônomos capazes de derrotar a equipe campeã mundial de futebol humana (KITANO et al., 1997b). Diversas categorias de futebol de robôs fazem parte da RoboCup, porém neste trabalho será comentado apenas da categoria VSS, que é a que o projeto foi focado. As regras do jogo em detalhes estão disponíveis em (IEEE, 2008). A seguir será abordado algumas características básicas do jogo que podem ser relevantes para o projeto.

O jogo é composto por dois times de robôs de três robôs, uma bola de golf e um campo de futebol. Os robôs são controlados remotamente por um algoritmo de IA, sem nenhuma intervenção humana e tem o objetivo de ganhar o jogo. O jogo é dividido em duas partes de 5 minutos cada e ganha o jogo o time que tiver marcado o maior número de gols no adversário.

O tamanho de cada robô é limitado a 7,5cm x 7,5cm x 7,5cm e deve ser totalmente independente. O campo deve ser da cor preta, de madeira e retangular com as medidas de 150cm x 130cm, e com paredes de 5cm de altura e 2,5cm de espessura. Na figura Figura 1 estão todas as dimensões do campo.

Figura 1: Dimensões do campo de futebol de robôs da categoria VSS



Adaptado de: (IEEE, 2008)

2.2 Algoritmos Evolutivos

O algoritmo evolutivo (AE) é uma técnica de otimização importante que vem crescido na ultima década. Devido a sua natureza flexível e comportamento robusto herdado da computação evolutiva, ele se torna um método eficiente para solução de problemas de otimização global e é utilizado principalmente quando se tem uma capacidade de computação limitada e informação insuficiente ou imperfeita.

Existem diversas subáreas dentro dos AEs, dentre elas estão a programação evolutiva, estratégia evolutiva, programação genética e o algoritmo genético (AG), que é a mais popular dentre elas e foi inspirado nos mecanismos da evolução e genética natural (SRINIVAS; PATNAIK, 1994). Todas as subáreas dos AEs trabalham com o princípio comum da evolução simulada de indivíduos utilizando o processo de seleção, mutação e reprodução. Pode-se diferenciar as diferentes subáreas dos algoritmos evolutivos com base na sua implementação e no modo com que elas são aplicadas a um problema em particular (VIKHAR, 2017).

Na implementação de um algoritmo genético existem algumas fases que devem ser elaboradas para a iteração do algoritmo, elas serão descritas em detalhes a seguir.

2.2.1 População Inicial

O processo de se aplicar um algoritmo genético para a resolução de um problema, se inicia com a criação de uma população inicial. A população inicial é composta por indivíduos, sendo cada um deles, uma solução para o problema gerado geralmente aleatoriamente.

Os indivíduos são representados por um conjunto de parâmetros, sendo cada um desses parâmetros chamado de gene. Os parâmetros do indivíduo são representados como uma sequencia de números.

Uma outra nomenclatura utilizada para o conjunto dos genes do indivíduo é cromossomo.

2.2.2 Função de Aptidão

A função de aptidão desempenha um papel importante no sucesso do algoritmo genético em encontrar a melhor solução para um problema, pois é ela que classifica o cromossomo com relação à sua performance. Isso é uma ligação importante entre o AG e o sistema. (Man; Tang; Kwong, 1996)

2.2.3 Reprodução

Durante a fase de reprodução do algoritmo genético, indivíduos são selecionados aleatoriamente, utilizando uma estratégia que favorite os melhores classificados pela função de aptidão, para gerar descendentes que irão compor a próxima geração.

Após a escolha dos indivíduos, eles passarão por dois processos. O primeiro é o cruzamento, que a partir do cromossomo de dois ou mais indivíduos é gerado o cromossomo de um terceiro indivíduo. Esse novo cromossomo é gerado a partir da combinação dos genes dos indivíduos anteriormente selecionados. Esse processo geralmente não é utilizado em todos os indivíduos, alguns costumam ser escolhidos para passarem seus genes para a próxima geração sem a interferência do cruzamento.

O segundo processo é chamado de mutação, ele é aplicado a cada cromossomo após o processo de cruzamento. Esse processo altera aleatoriamente cada gene em um valor bem pequeno. A mutação é uma etapa muito importante para a busca da melhor solução em todo o espaço de busca, pois ela proporciona uma pequena aleatoriedade na busca, o que ajuda a garantir que nenhum ponto no espaço de busca tem probabilidade zero de ser examinado. (BEASLEY; BULL; MARTIN, 1993)

2.2.4 Predação

O processo de predação consiste na remoção de determinados indivíduos, não passando eles para a próxima geração e nem permitindo que eles se reproduzam. Geralmente se escolhe os piores indivíduos de acordo com a função de aptidão para serem removidos ou terem maiores chances de serem removidos, a fim de se diminuir as chances de uma característica indesejada seja passada para a próxima geração, porém pode-se também optar por remover um indivíduo aleatório desde que esse não seja o melhor indivíduo da geração.

2.3 Rede Neural Artificial

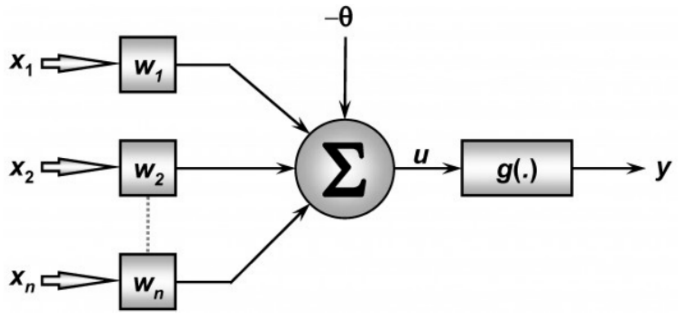
Os neurônios biológicos são capazes de transmitir impulsos elétricos a outros neurônios, esses impulsos elétricos são chamados sinapses. No cérebro os neurônios estão ligados uns aos outros como uma rede (LODISH et al., 2000). O neurônio artificial foi criado com inspiração no funcionamento do neurônio biológico.

Um dos neurônios artificiais mais simples é o perceptron, que pode ser observado na Figura 2.

Para se calcular a saída do neurônio, primeiro multiplica-se cada entrada (x_1, x_2, ..., x_n) por seu respectivo peso sináptico (w_1, w_2, ...,w_n). Esses valores multiplicados são então todos somados junto com o limiar de ativação e por fim, é aplicado uma função de aplicação a esse valor. A função de aplicação deve ser limitada de preferência superior e inferiormente, um exemplo de função de aplicação é a tangente hiperbólica. Matematicamente define-se a saída do neurônio como a seguir:

$$y = g(\sum_{i=1}^{n} w_i \cdot x_i - \theta) \tag{2.1}$$

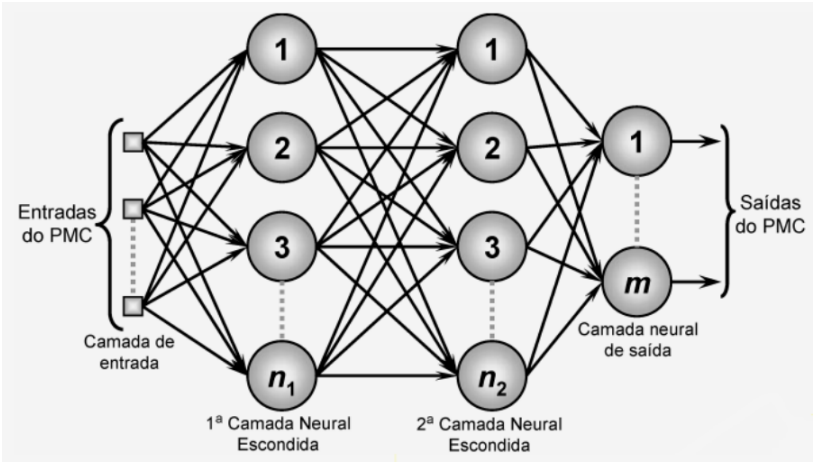
Figura 2: Neurônio artificial perceptron. Na figura (x) são entradas do neurônio, (w) são os pesos sinápticos, (θ) é o limiar de ativação, (u) é o potencial de ativação, (g) é a função de ativação e (y) é a saída do neurônio



Fonte: (SILVA; SPATTI; FLAUZINO, 2010)

O perceptron multi camadas (PMC) é uma rede de neurônios do tipo perceptron. A rede é dividida em camadas, onde as entradas dos neurônios que estão na primeira são as entradas da rede, e as saídas dos neurônios que estão na última camada são as saídas da rede. A conexão entre os neurônios é feita de forma que as saídas dos neurônios de uma camada são as entradas dos neurônios da próxima camada. A Figura 3 mostra essa conexão.

Figura 3: Rede do tipo perceptron multi camadas

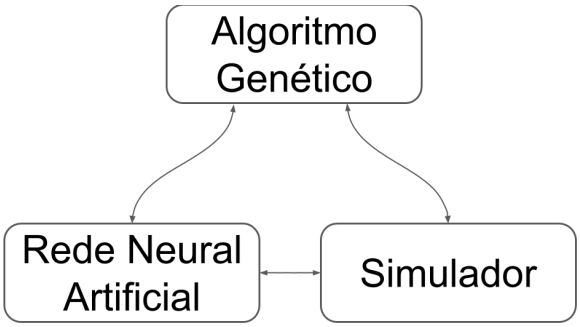


Fonte: (SILVA; SPATTI; FLAUZINO, 2010)

3 DESENVOLVIMENTO

O projeto foi desenvolvido na linguagem de programação C++ utilizando o *framework* Qt para a parte gráfica e algumas de suas classes de geometria. A sua arquitetura foi feita de forma modular, composto por vários módulos que se comunicam entre si. Ele foi feito dessa forma para que caso futuramente surja a necessidade de se modificar alguma parte dele, como por exemplo trocar o motor gráfico ou utilizar um outro tipo de rede neural, isso possa ocorrer sem grandes dificuldades. Os três módulos principais que compõem o sistema são: algoritmo genético, simulador e rede neural artificial (RNA). O código do sistema pode ser encontrado em: https://github.com/marco7m/futbot

Figura 4: Representação de alto nível do sistema



Esses três módulos que podem ser observados na Figura 4 trocam informações entre si e agem um sobre o outro. O simulador representa um campo de futebol com robôs, a RNA controla os robôs do simulador e o AG realiza o treinamento da RNA otimizando os pesos de seus neurônios. O AG utiliza o simulador durante a etapa de treinamento da RNA.

3.1 Simulador

Uma das primeiras etapas do desenvolvimento do projeto foi a pesquisa de simuladores de robôs da categoria *IEEE Very Small Size* que poderiam ser utilizado com um AG para o desenvolvimento de IA para o futebol de robôs. Como não foi encontrado nenhum simulador capaz de simular em modo acelerado, ou seja, várias vezes mais rápido que o sistema real, o que é importante para o desenvolvimento do AG, foi desenvolvido um simulador com essa capacidade.

Os robôs do simulador estão em um campo com uma bola e todas as medidas de tamanho utilizadas estão de acordo com o manual de regras da competição *IEEE Very Small Size Soccer* (VSSS) (IEEE, 2008). O propósito principal do simulador é de ser

utilizado no treinamento de um algoritmo genético, na função de aptidão durante a etapa de classificação das diferentes IAs. Portanto foi feito um esforço para o código ser eficiente em termos de processamento e memória, e de ser fácil de se interfacear com outras partes do sistema.

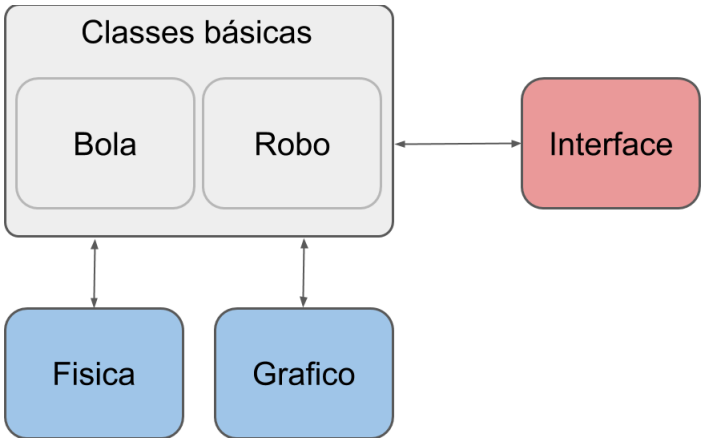
O simulador possui duas classes internas que são seus componentes mais básicos e importantes: *Robo* e *Bola*. Essas classes possuem funções e variáveis que controlam os parâmetros internos, como por exemplo posição e velocidade, assim como identificam o id de cada robô e para qual time ele está jogando. Ponteiros para objetos dessas classes são compartilhados entre as classes *Fisica*, *Grafico* e *Interface*, de forma que todos eles apontem para a mesma variável, permitindo assim que essas três classes atuem e leiam as informações do robô e da bola sem a necessidade de consumir memória e processamento passando valores de uma classe para outra.

O simulador foi desenvolvido em dois módulos principais: física e gráfico. O módulo física é responsável pelos cálculos de colisões e movimentação dos robôs e da bola. O módulo gráfico é responsável pela exibição gráfica para o usuário do resultado da simulação.

A comunicação entre os módulos física e gráfico se dá por ponteiros de objetos da classe *Bola* e *Robo* que são compartilhados entre eles. O intuito dessa subdivisão em dois módulos é para que, caso no futuro se deseje utilizar outro *framework* para a parte gráfica do simulador, ou se deseje utilizar alguma engine física no simulador, essa mudança não seja muito trabalhosa e possa ser feita só alterando as funções dentro do respectivo módulo.

Outra motivação para o desenvolvimento modular do simulador, o dividindo entre simulação física e exibição gráfica, é para aumentar a sua performance durante o treinamento do algoritmo evolutivo. Esses dois módulos são completamente separados e funcionam independente um do outro, deste modo se torna possível fazer a simulação dos robôs e se obter os resultados da partida ou mesmo gravar ela em um arquivo, sem consumir recursos de processamento para a exibir graficamente. Fazendo a simulação desse modo, se consegue focar todo o processamento do computador na simulação em si e ela fica muito mais rápida.

Figura 5: Representação em blocos do simulador com as suas principais classes



3.1.1 Exibição Gráfica

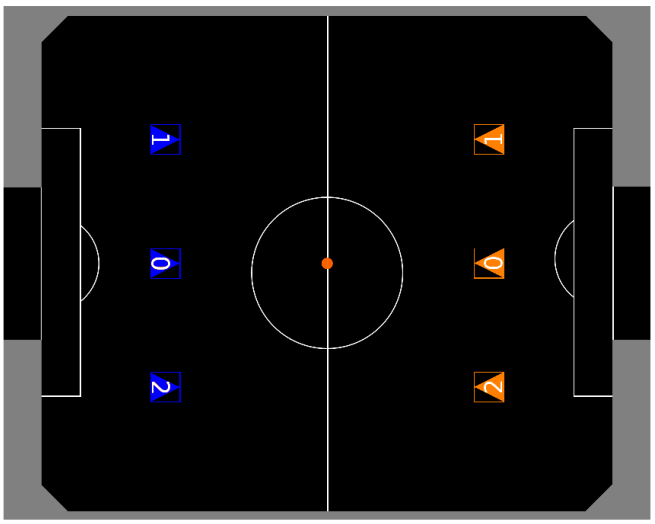
O módulo responsável pela exibição gráfica do que está acontecendo no simulador, para que se possa acompanhar a movimentação dos robôs e da bola e fazer uma análise visual do comportamento da IA está implementado dentro da classe *Grafico*. Esse módulo não é necessário para nada além da exibição gráfica, não alterando o valor de nenhuma variável importante do sistema.

A classe *Grafico* possui internamente ponteiros para objetos da classe *Robo* e *Bola* e possui uma função de atualização importante que deve ser implementada obrigatoriamente, assim como a classe *Fisica*. Essa função de atualização, sempre que chamada, atualiza a interface gráfica com as ultimas posições de cada robô e da bola.

A interface gráfica foi desenvolvida utilizando o *framework* Qt , mas se por algum motivo for desejado alterar a ferramenta utilizada para fazer a interface gráfica, é necessário fazer duas modificações dentro da classe *Grafico*. A primeira é modificar a função de atualização da classe *Grafico*, essa função sempre que chamada deve atualizar a posição dos robôs e da bola na tela a partir dos valores de posição nas variáveis internas dos ponteiros para objetos da classe *Robo* e *Bola* que ela possui. A segunda modificação é alterar o construtor da classe gráfico, para receber os ponteiros para objetos da classe *Robo* e *Bola*, que são compartilhados com a classe *Fisica*, e para inicializar a interface gráfica.

Na Figura 6 está uma imagem gerada pela interface gráfica do simulador. Nela constam seis robôs, a cor deles representa o seu time, e cada robô possui uma numeração de identificação utilizada para diferenciar cada robô em relação aos outros de seu time. O ponto laranja no centro do campo é a bola e a linha branca desenhada dividindo o campo em dois, o circulo no meio e as formas próximas aos gols são apenas para fins de demarcação visual das partes do campo.

Figura 6: Interface gráfica do simulador com dois times de robôs e uma bola laranja no meio do campo



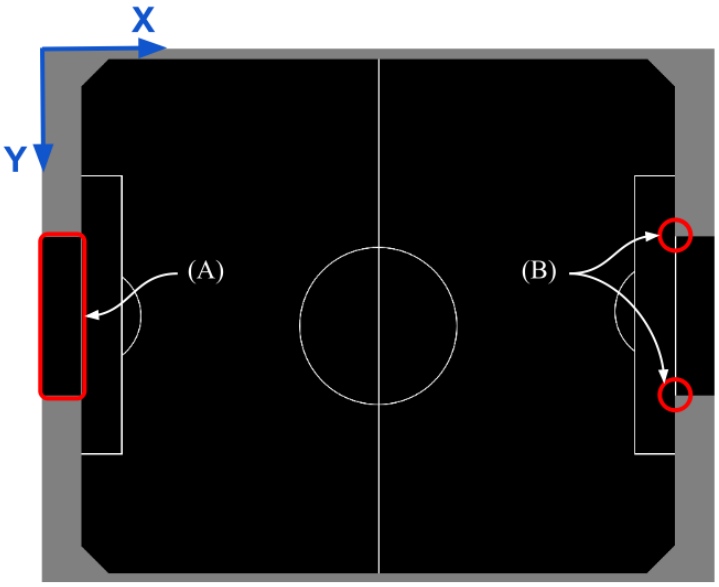
3.1.2 Simulação Física

O módulo responsável pelos cálculos de colisão entre os robôs, a bola e o campo, assim como o cálculo da trajetória e movimentação de cada robô e da bola está implementado dentro da classe *Fisica*. Essa classe possui internamente ponteiros para objetos da classe *Robo* e *Bola* e possui uma função de atualização importante que deve ser implementada obrigatoriamente. Essa função, sempre que chamada, atualiza a posição e velocidades de cada robô e da bola realizando todos os cálculos de colisão e movimentação necessários para isso.

Como o objetivo primário do simulador é conseguir simular o mais rápido possível uma partida, a precisão da simulação foi comprometida e recursos como a derrapagem dos robôs e algumas outras situações específicas, não funcionam exatamente como no caso real. Isso foi feito de forma proposital e a proposta é utilizar esse simulador como uma ferramenta para o pré treinamento das IAs, que seria a fase do treinamento que exigiria mais recursos computacionais e iria desde uma IA sem treinamento algum, até alguma IA que já consegue desenvolver a sua função com esse simulador. O próximo passo seria desenvolver um simulador mais preciso, mesmo que ele exija mais recursos computacionais, e utilizá-lo para treinar uma IA pré treinada, como um ajuste fino antes de utilizar a IA com robôs reais.

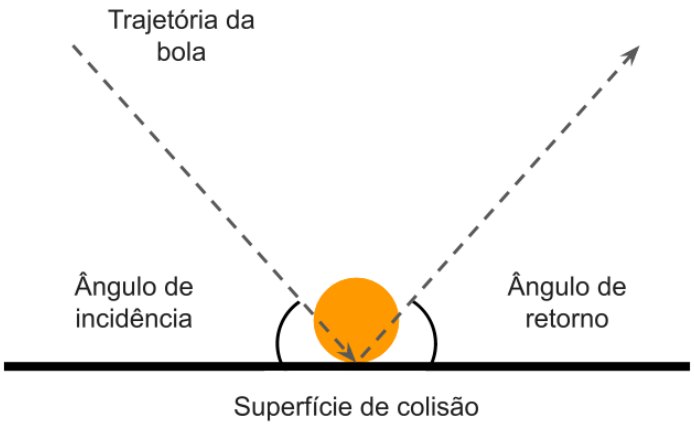
Para o desenvolvimento de um simulador mais preciso, ou mais adequado para qualquer outra aplicação futura em que ele venha a ser utilizado, basta alterar a classe *Fisica* implementando dentro da função de atualização as chamadas para outras funções ou realização de cálculos que atualizem os valores das variáveis de posição e velocidade dos robôs e da bola da forma que for desejado.

Figura 7: Indicação da posição dos gols e traves. A indicação (A) representa a área do gol esquerdo e (B) representa as traves do gol direito



O campo em que o robô está no simulador, representado na Figura 7 possui uma superfície em preto pela qual os robôs e a bola podem se mover livremente, e em volta dessa superfície há uma parede representada pela cor cinza. A colisão do robô com a parede ou laterais de outros robôs faz com que ele cesse o seu movimento impedindo que ele se mova sobre essas superfícies. A interação da bola com a lateral dos robôs ou a parede faz com que ela, ao se chocar com a superfície, se mova da forma indicada na Figura 8, de forma que o ângulo de incidência e o ângulo de retorno são iguais.

Figura 8: Comportamento da bola ao colidir com uma superfície

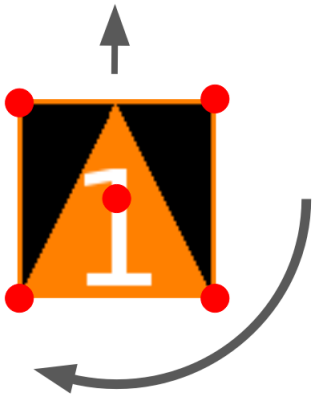


Todos as posições de objetos representadas no simulador possuem dois valores, um é a componente de posição no eixo X e o outro é a componente de posição no eixo

Y. O eixo X e Y está representado na Figura 7 no seu canto superior esquerdo. Outras informações como velocidade por exemplo, também são calculadas considerando essa base.

O robô é definido por 5 pontos, um no seu centro, e outros quatro, um em cada vértice de um quadrado de lados iguais, além disso ele possui uma face que é considerada a frente do robô. Os robôs possuem dois graus de liberdade para movimentação: movimentação linear e movimentação angular. A movimentação linear permite que o robô se mova linearmente para frente e para trás (considerando a face da frente do robô). A movimentação angular permite que o robô rotacione com eixo no seu ponto central. A Figura 9 mostra os pontos que definem o robô e seus dois graus de liberdade.

Figura 9: Representação de como o robô é visto pela simulação física. Os pontos vermelhos são os pontos que representam o robô e as setas são os seus graus de liberdade. A face superior do robô nessa imagem é a sua face frontal



3.1.2.1 Cálculo de Colisão

Primeiramente é necessário se definir o que é considerado uma colisão. Como o algoritmo atua a intervalos de tempo discretos, muitas vezes a posição é calculada após a colisão já ter de fato ocorrido, por conta disso, é considerado a colisão de um ponto com uma reta o primeiro momento calculado pelo algoritmo em que o ponto atravessa a reta.

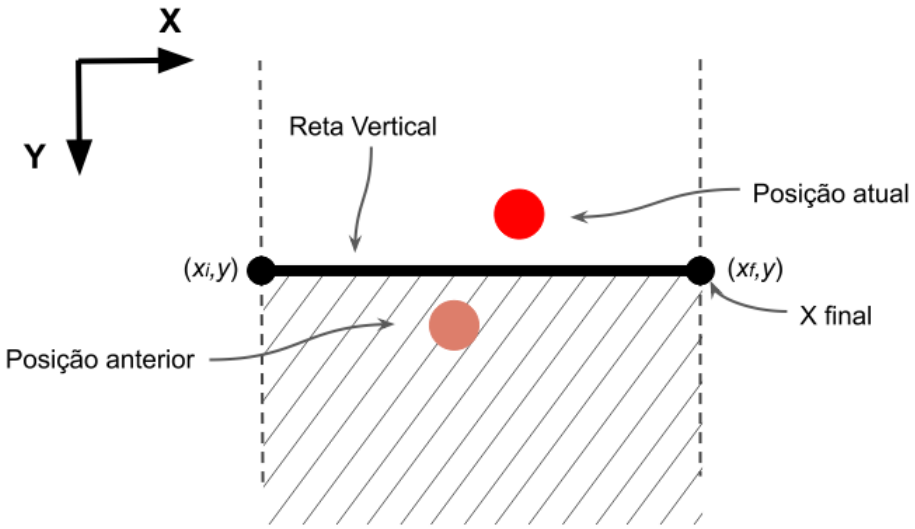
Para se detectar as colisões que podem ocorrer na simulação física, foram utilizadas três técnicas diferentes: comparar os valores de coordenas puros dos objetos, determinar de qual lado um ponto está em relação a uma reta, e determinar se um ponto se encontra dentro de um quadrado. Essas técnicas serão melhor explicadas a seguir.

a) Determinar colisão por comparação de posição.

Este é o método mais simples de se detectar a colisão, porém no simulador ele só pode ser aplicado para a colisão de um ponto com uma reta horizontal ou vertical. Para o caso de uma reta horizontal, é necessário saber a posição em X que ela começa e a que ela termina, e também é necessário saber a sua posição em Y. Se a posição em X do ponto

for maior que a posição em X inicial da reta e menor que a posição em X final da reta, significa que o ponto está em condições de colidir com a reta. Se cumprido o requisito de estar em condições de colidir com a reta, o próximo passo é comparar a posição em Y do ponto com a posição em Y da reta. Se essa posição do ponto era menor que a da reta no momento anterior, e no próximo momento ela passou a ser maior que a da reta, então houve uma colisão. O mesmo ocorre para caso a posição fosse maior no momento anterior e no próximo passou a ser menor. Para o caso da reta ser vertical, o cálculo de colisão é feito da mesma maneira, porém invertendo os eixos X e Y.

Figura 10: Exemplo de colisão por comparação de posição de um ponto com uma reta horizontal. A posição X inicial é representada por x_i, a posição X final é representada por x_f e a posição em Y da reta é representada por y_i



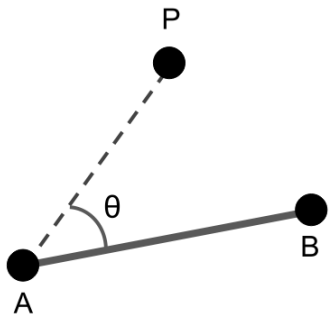
b) Determinar de qual lado um ponto está em relação a uma reta.

Esta técnica, em relação à técnica anterior, requer mais tempo de processamento, porém ela permite que o algoritmo determine de qual lado um ponto está em relação a uma reta, e por consequência descobrir se o ponto mudou de um lado para o outro da reta utilizando esta técnica em dois momentos diferentes, para uma reta com qualquer inclinação, e não apenas para uma reta vertical ou horizontal.

Na Figura 11 observa-se uma reta $\overline{AB}$ definida pelos pontoa A e B, e um ponto P. Para se determinar de que lado da reta o ponto está, primeiro deve-se definir uma orientação para a reta, se a orientação da reta for definida no sentido de A para B, pode-se representar esta reta pelo vetor $\overrightarrow{AB}$ e o menor ângulo entre $\overrightarrow{AB}$ e $\overrightarrow{AP}$ é θ. É calculado então o produto vetorial do vetor $\overrightarrow{AB}$ com o vetor $\overrightarrow{AP}$, dado pela Equação 3.1, no qual $\hat{k}$ é um vetor unitário perpendicular ao plano em que estão os ponto A, B e P.

$$\overrightarrow{AB} \times \overrightarrow{AP} = \hat{k} \left\| \overrightarrow{AB} \right\| \left\| \overrightarrow{AP} \right\| \ sen(\theta) \tag{3.1}$$

Figura 11: Reta e ponto genéricos para cálculo de colisão

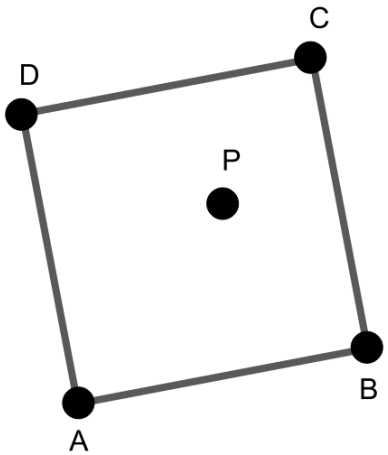


Se o módulo do produto vetorial calculado $\left\|\overrightarrow{AB} \times \overrightarrow{AP}\right\|$ for maior do que 0, o ponto P está acima da reta $\overline{AB}$ (tomando-se por referência o exemplo da Figura 11), se ele for menor do que 0, ele está abaixo da reta, e se ele for igual a 0, ele está na reta. Isso ocorre pois como: $\left\|\overrightarrow{AB}\right\| >= 0$ e $\left\|\overrightarrow{AP}\right\| >= 0$, quem vai determinar o sinal do produto é o $sen(\theta)$, que é sempre maior do que zero para $\theta > 0^o$ (ponto P do lado de cima da reta) e é sempre menor do que zero para $180^o < \theta < 360^o$ (ponto P do lado de baixo da reta).

c) Determinar se um ponto se encontra dentro de um quadrado.

Utilizando-se a técnica de se determinar de qual lado um ponto está em relação a uma reta é possível de se determinar se um ponto se encontra dentro ou fora de um quadrado, basta verificar de que lado o ponto está em relação a cada lateral do quadrado. Foi encontrado outra forma de se calcular se um ponto está dentro ou fora de um quadrado que computacionalmente aparenta ser tão eficiente quanto a técnica anterior aplicada a cada lado de um quadrado, foi implementada esta técnica também que será explicada abaixo para fins comparativos em relação à performance computacional.

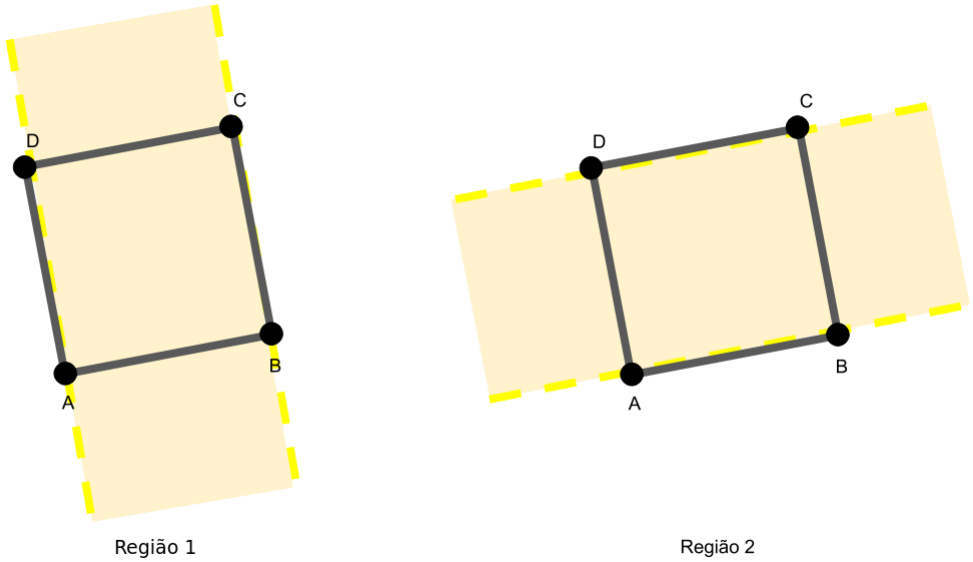
Figura 12: Quadrado representando o robô dentro do simulador, com um ponto P que no qual se está estudando para saber se ele está colidindo com o robô ou não



Dado um ponto P e um quadrado definido pelos pontos A, B, C e D como o

mostrado na figura Figura 12, para se determinar se o ponto está dentro do quadrado, a estratégia utilizada passa por dois passos, o primeiro é verificar se o ponto está na região entre as retas $\overline{AB}$ e $\overline{DC}$, o segundo é verificar se o ponto está na região entre as retas $\overline{AD}$ e $\overline{BC}$. Se o ponto estiver nas duas regiões ao mesmo tempo, ele está portanto dentro do quadrado.

Figura 13: A figura representa duas regiões em amarelo de um mesmo quadrado, caso um ponto estiver ao mesmo tempo nas duas regiões, significa que ele está dentro do quadrado



Para se determinar se o ponto está dentro da região 1, de acordo com a Figura 13, primeiro define-se dois vetores unitários: $\hat{j}$ no sentido do vetor $\overrightarrow{AD}$ e $\hat{k}$ no sentido do vetor $\overrightarrow{AB}$. Então se define dois outros vetores: $\overrightarrow{P_j}$ que é a projeção do vetor $\overrightarrow{AP}$ no vetor $\hat{j}$, e o vetor $\overrightarrow{P_k}$ que é a projeção do vetor $\overrightarrow{AP}$ no vetor $\hat{k}$, como pode-se ver na Figura 14.

Como:

$$\overrightarrow{AB} = \left\|\overrightarrow{AB}\right\| \cdot \hat{k} + 0 \cdot \hat{j} \tag{3.2}$$

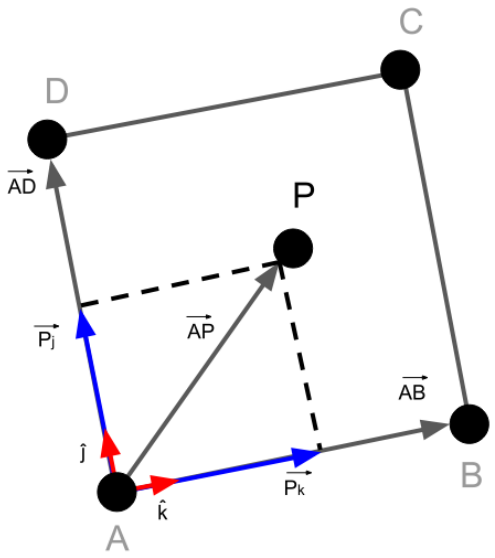
$$\overrightarrow{AP} = \left\|\overrightarrow{P_k}\right\| \cdot \hat{k} + \left\|\overrightarrow{P_j}\right\| \cdot \hat{j} \tag{3.3}$$

Portanto:

$$\overrightarrow{AB} \cdot \overrightarrow{AP} = \left\|\overrightarrow{AB}\right\| \cdot \left\|\overrightarrow{P_k}\right\| + 0 \cdot \left\|\overrightarrow{P_j}\right\| \tag{3.4}$$

$$\overrightarrow{AB} \cdot \overrightarrow{AP} = \left\|\overrightarrow{AB}\right\| \cdot \left\|\overrightarrow{P_k}\right\| \tag{3.5}$$

Figura 14: Representação dos vetores utilizados para determinar se um ponto encontra-se dentro de um quadrado



E pode-se afirmar que:

$$0 < \left\|\overrightarrow{AB}\right\| \cdot \left\|\overrightarrow{P_k}\right\| < \left\|\overrightarrow{AB}\right\| \cdot \left\|\overrightarrow{AB}\right\| \tag{3.6}$$

Se:

$$0 < \left\|\overrightarrow{P_k}\right\| < \left\|\overrightarrow{AB}\right\| \quad ; \quad \left\|\overrightarrow{AB}\right\| > 0 \tag{3.7}$$

Quando as condições na Equação 3.7 são cumpridas, o ponto P está dentro da região 1, ilustrada na Figura 13. Se substitui então a Equação 3.5 na Equação 3.6, para se obter:

$$0 < \overrightarrow{AB} \cdot \overrightarrow{AP} < \left\|\overrightarrow{AB}\right\| \cdot \left\|\overrightarrow{AB}\right\| \tag{3.8}$$

E como:

$$\left\|\overrightarrow{AB}\right\| \cdot \left\|\overrightarrow{AB}\right\| = \overrightarrow{AB} \cdot \overrightarrow{AB} \tag{3.9}$$

Se substitui a Equação 3.9 na Equação 3.8 para se obter:

$$0 < \overrightarrow{AB} \cdot \overrightarrow{AP} < \overrightarrow{AB} \cdot \overrightarrow{AB} \tag{3.10}$$

A Equação 3.10 é a equação utilizada para se determinar se um ponto se encontra dentro da região 1, basta que a equação seja satisfeita para que isso ocorra. De modo

análogo, pode-se fazer esse mesmo desenvolvimento matemático para chegar à Equação 3.11, que determina se um ponto se encontra dentro da região 2.

$$0 < \overrightarrow{AD} \cdot \overrightarrow{AP} < \overrightarrow{AD} \cdot \overrightarrow{AD} \tag{3.11}$$

Portanto, para se determinar se o ponto está dentro do quadrado, basta verificar se as condições na Equação 3.10 e Equação 3.11 são satisfeitas.

3.1.2.2 Algoritmo do Simulador

Toda vez que a função de atualização da classe *Fisica* é chamada, o simulador calcula a próxima posição em que cada robô e a bola irão estar (considerando o intervalo de tempo desde a última chamada dessa função), então é verificado se houve alguma colisão e se faz o tratamento adequado dela. Após isso, é atualizada as posições dos robôs e da bola. A seguir se encontra um passo a passo mais detalhado do algoritmo implementado dentro da função de atualização da classe *Fisica*:

a) É feito o cálculo da próxima posição dos robôs.

Para se calcular o ângulo em que o robô estará com a sua frente direcionada θ_f (0^o o robô está direcionado para cima no simulador e 180^o o robô está direcionado para baixo no simulador) é utilizada a Equação 3.12, na qual θ_i é o ângulo atual e ω é a velocidade angular do robô.

$$\theta_f = \theta_i \cdot \omega \cdot \Delta T \tag{3.12}$$

Na Equação 3.12, ΔT representa o intervalo de tempo desde a última vez que a função de atualização foi chamada. O valor utilizado de ΔT no simulador foi definido empiricamente através de vários testes, um valor de ΔT muito pequeno faz com que o simulador fique lento por aumentar o número de vezes que os cálculos físicos são realizados, e um valor de ΔT muito alto introduz erros na simulação.

As equações Equação 3.13 e Equação 3.14 calculam a próxima posição em X, representado por P_{xf}, e em Y, representado por P_{yf}, dos robôs. A variável v representa a velocidade linear dos robôs e as posições atuais dos robôs em X e Y são representadas por P_{xi} e P_{yi} respectivamente. Os eixos X e Y utilizados como referencial estão indicados na Figura 7.

$$P_{xf} = P_{xi} - sen\left(\theta_i \cdot \frac{\pi}{180}\right) \cdot v \cdot \Delta T \tag{3.13}$$

$$P_{yf} = P_{yi} + cos\left(\theta_i \cdot \frac{\pi}{180}\right) \cdot v \cdot \Delta T \tag{3.14}$$

b) Checa todos os vértices do robô (vértices do quadrado que representa o robô dentro do simulador), se algum deles colidiu com a parte externa do campo.

Essa etapa checa primeiramente a colisão com todas as laterais do campo que estão na posição vertical ou horizontal, para isso se utiliza a técnica (a) da subseção 3.1.2.1. A seguir se checa se houve colisão com as partes do campo que não estão na posição vertical ou horizontal. Para isso, se utiliza a técnica (b) abordada na subseção 3.1.2.1.

c) É verificado se alguma trave do gol se encontra dentro do quadrado que representa o robô utilizando-se a técnica de encontrar um ponto dentro de um quadrado descrita na subseção 3.1.2.1.

d) Checa se algum robô colidiu com outro robô.

Para essa parte do algoritmo, para fins de performance, primeiramente se checa se algum robô se encontra na região de colisão de outro robô. A região de colisão é quando pelo menos um dos vértices do robô, se encontra dentro de um círculo centrado no ponto central do robô e com raio igual à distância entre o ponto central do robô e algum de seus vértices. A Figura 15 mostra a região de colisão de um robô, se o vértice de algum outro robô se encontra a região de colisão dele, se considera que esse outro robô essa na região de colisão.

Figura 15: Região de colisão de um robô representada por um círculo vermelho

Quando um robô se encontra na região de colisão de outro robô, é então verificado se algum dos vértices desses robôs está colidindo com o outro. Isso é feito utilizando a técnica mostrada na subseção 3.1.2.1 para se determinar se um ponto se encontra dentro de um quadrado.

e) Atualiza a posição de todos os robôs.

Atualiza a posição de todos os robôs que não colidiram para a posição calculada no item (a). Os robôs que sofreram algum tipo de colisão não se movem.

f) Colisão da bola com o robô.

Primeiramente se verifica se a bola se encontra na região de colisão de algum robô. Se a bola está na região de colisão de algum robô, então é checado se a bola colidiu com alguma das arestas do robô. Se a bola colidiu com o robô, então é encontrado o ponto de colisão da bola com o robô, a velocidade relativa do robô naquele ponto de colisão e é somada essa velocidade relativa do ponto de colisão com a velocidade da bola, levando-se em consideração o efeito observado na Figura 8 para se calcular a velocidade resultante da bola.

g) Colisão da bola com o campo.

Verifica-se se a bola colidiu com o campo, se sim, a nova velocidade da bola é calculada, considerando-se o efeito representado na Figura 8.

h) Atualiza a posição da bola.

Após esse último item, o cálculo de colisões e trajetória do simulador é finalizado até a próxima chamada de sua função de atualização, quando esse processo se repete novamente.

3.1.3 Interface

Existe uma classe no sistema chamada *Interface* que pode ser vista na Figura 5, ela tem o proposito de servir como o único meio de comunicação entre o simulador e o resto do sistema, passando informações sobre os robôs e a bola na simulação para o AG e para a RNA quando solicitado e recebendo dados dos mesmos também.

Além de servir como meio de comunicação, essa classe também faz a conversão dos valores de forma conveniente para o simulador e o outros módulos que o utilizarão. Para o treinamento de uma rede neural, é vantajoso que os valores estejam normalizados, isso acarreta em um menor erro de estimação e menor tempo de treinamento necessário (Sola; Sevilla, 1997). A normalização adotada foi fazer todos os valores variarem entre 0 e 1, portanto, para a utilização do simulador, as variáveis lidas nele, desde as de posição quanto as de velocidade, estão nesse intervalo. A classe interface traduz esses dados normalizados para dados nas unidades que o simulador trabalha facilitando a comunicação entre os módulos.

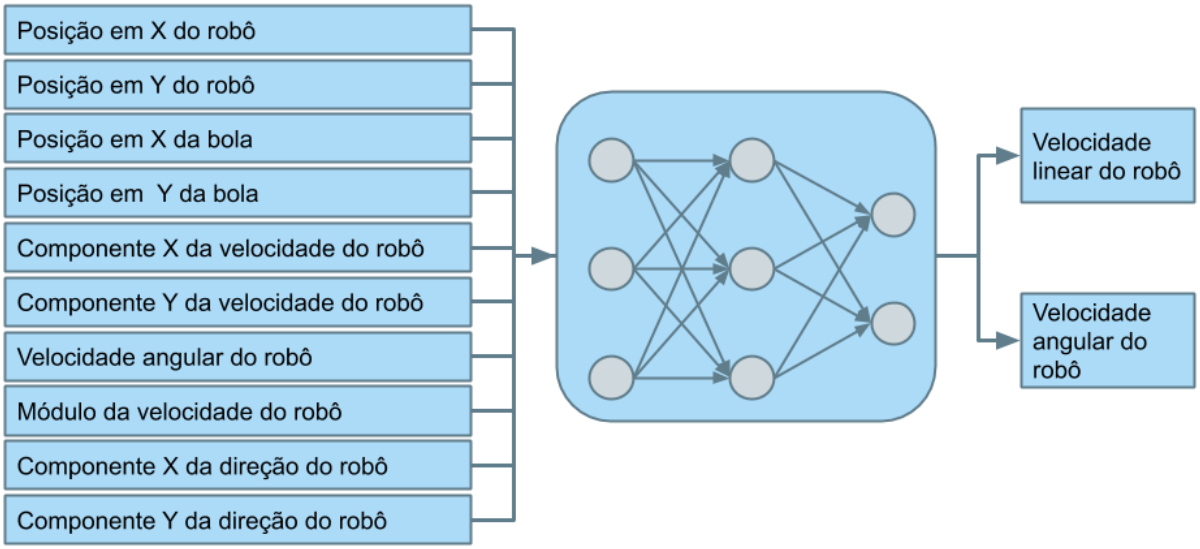
3.2 Rede Neural Artificial

Foi implementada uma RNA do tipo PMC que consegue se comunicar com o simulador utilizando a classe *Interface*. A RNA atua como a IA de um time de robôs, porém ela pode controlar quantos robôs forem desejados, não se limitando a um time de 3 robôs.

É possível inserir nas entradas da RNA quaisquer variáveis disponíveis do simulador, utilizando a classe *Interface* e as saídas da RNA são a velocidade angular e velocidade linear de cada robô que ela controla. Na rede neural proposta neste projeto, as informações inseridas na entrada e saída da rede neural estão definidas a seguir:

- Entradas da RNA
  a) Posição em X do robô
  b) Posição em Y do robô
  c) Posição em X da bola
  d) Posição em Y da bola
  e) Componente X da velocidade do robô
  f) Componente Y da velocidade do robô
  g) Velocidade angular do robô
  h) Módulo da velocidade do robô
  i) Componente X da direção do robô
  j) Componente Y da direção do robô
- Saídas da RNA
  a) Velocidade linear do robô
  b) Velocidade angular do robô

Figura 16: Diagrama exibindo as variáveis de entrada de saída da RNA utilizada



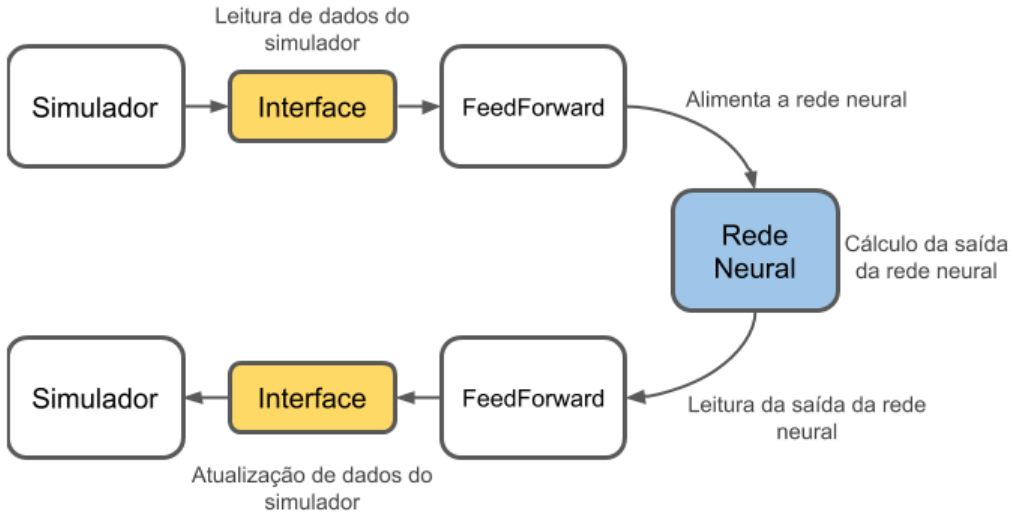
Ao se criar uma nova RNA, deve-se inserir como parâmetro de entrada como vai ser a sua topologia, ou seja, quantas entradas a RNA vai ter, quantas saídas (duas para

cada robô que ela controlar), quantas camadas internas de neurônios ela vai ter e quantos neurônios em cada camada.

As RNAs podem ser salvas em um arquivo após seu treinamento para posterior utilização ou aplicação de uma nova etapa de treinamento. Foi também desenvolvida uma função para ler um arquivo previamente salvo e gerar uma RNA a partir dele.

A RNA sozinha é capaz de, dado os valores que estão na sua entrada calcular os valores da saída, porém é necessário atualizar as entradas com os últimos valores calculados pelo simulador, e é necessário inserir os valores calculados pela RNA de volta ao simulador, para realizar essas ações foi criada a classe *FeedForward*. A classe *FeedForward* atualiza as entradas da RNA com os valores do simulador, faz a chamada da função que calcula a saída dela e atualiza o simulador com os valores calculados. A Figura 17 apresenta o fluxo de dados, desde a sua leitura do simulador utilizando a *Interface* pela classe *FeedForward*, até o cálculo das próximas ações de cada robô e a inserção dessas informações no simulador.

Figura 17: Diagrama com o fluxo de dados de cada iteração do *FeedForward*

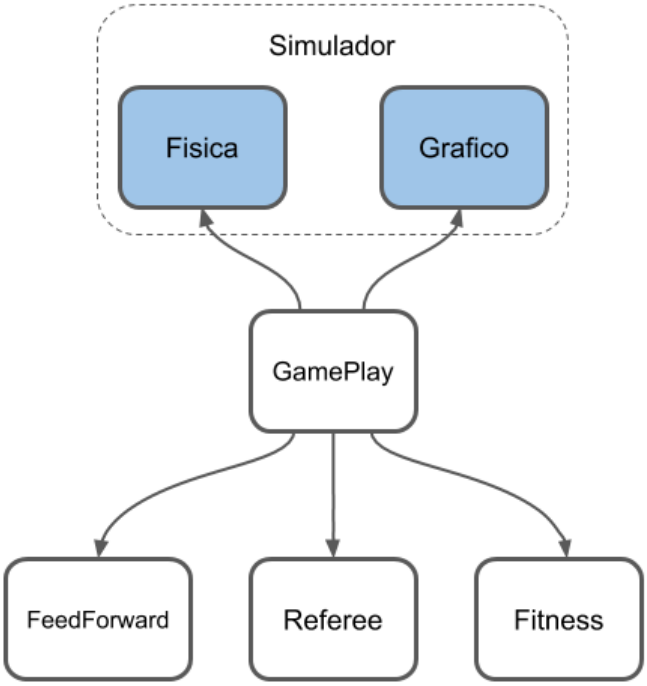


3.3 Classe *GamePlay*

Uma das partes centrais do sistema desenvolvido é a classe *GamePlay*, ela é responsável por inicializar e sincronizar os módulos que são necessários para simular uma partida de futebol de robôs, sendo que a partida é definida nesse contexto como o período em que a simulação física dos robôs está ocorrendo, junto com os diversos módulos que interagem com a simulação. Os módulos que interagem com *GamePlay* são as classes: *Fisica*, *Grafico*, *Referee*, *Fitness* e *FeedForward*.

A classe *Referee* é responsável por atuar sobre a simulação como se fosse o arbitro de um jogo de futebol, tomando ações que influenciam no jogo quando determinados

Figura 18: Diagrama mostrando as classes que a *GamePlay* controla



critérios são atingidos. Exemplos de ações que a classe *Referee* pode tomar são: finalizar uma partida quando o tempo limite de duração se esgota, reiniciar os robôs e a bola para as posições iniciais quando um gol é feito e fazer a contagem de gols e finalizar a partida quando um time atinge determinado número de gols. A classe *Fitness* faz parte do algoritmo evolutivo, ela será descrita com mais detalhes no capítulo 3.4.

Tendo acesso e controle de todas as classes ilustradas na Figura 18 a classe *GamePlay* é utilizada sempre que é necessário simular algo. Ela já possui implementado dentro dela funções para simular uma partida sem a parte gráfica, com a parte gráfica, com apenas um robô, com dois times de três robôs, etc. Dessa forma toda a simulação da partida, e a interação do simulador com as classes da inteligência artificial fica modularizada dentro dela.

3.4 Algoritmo Genético

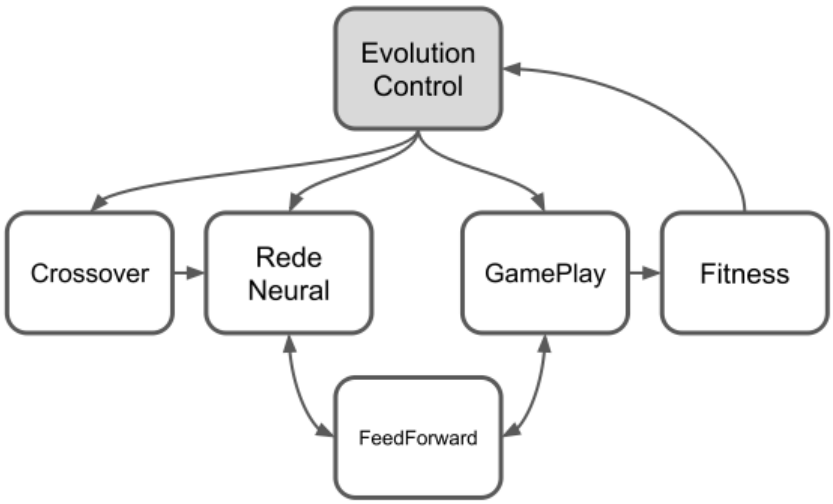
Foi implementado um algoritmo genético capaz de ensinar uma RNA, por meio do ajuste dos pesos de seu neurônio, a realizar determinada ação com os robôs. Para validação do sistema e testes iniciais foi desenvolvida uma IA que faz um robô ir de um ponto a outro do campo.

O cromossomo do AG que é utilizado para representar o indivíduo é formado pelos pesos de uma RNA. Para a estruturação do cromossomo como sendo uma lista de números, os primeiros genes do cromossomo são os pesos dos neurônios da primeira camada da

RNA de cima para baixo, em seguida é feito o mesmo processo para a segunda camada e a terceira, até a última camada da RNA.

Existem quatro classes que são importantes para o AG que foi implementado: *EvolutionControl*, *Crossover* e *Fitness*. A classe *Fitness* acompanha cada iteração da simulação, ela é uma das partes mais importantes na definição do desempenho da rede neural, pois ela da pontos de aptidão para a rede neural a cada iteração da simulação de acordo com o modo que ela foi programada, e esses pontos influenciam diretamente em quais características serão passadas para a próxima geração do AG e quais serão excluídas. A classe *Crossover* contém funções responsáveis pela reprodução e mutação dos indivíduos do AG. A classe *EvolutionControl* controla todo o AG. Um fluxograma mostrando as relações entre as principais classes referentes ao AG encontra-se na Figura 19.

Figura 19: Diagrama mostrando as interações entre as principais classes relacionadas ao AG



Foram feitos vários testes diferentes com o AG a fim de se conseguir treinar uma RNA para controlar um robô e fazer ele se mover de um ponto a outro do mapa mantendo-se as mesmas configurações do AG, alterando-se apenas o número de neurônios e de camadas das RNAs. A classe *EvolutionControl* é responsável por criar a população inicial do AG, inicializando 100 RNAs com os pesos de seus neurônios contendo valores aleatórios entre -1 e 1. O número de indivíduos da população se mantém constante durante todo o treinamento. Após isso, a classe *GamePlay* é utilizada para simular um jogo com cada indivíduo e a classe *Fitness* calcula um ponto de aptidão para cada indivíduo. Os 10 melhores indivíduos, segundo o seu ponto de aptidão, permanecem para a próxima geração, enquanto os 90 piores são substituídos por uma mutação dos 10 melhores. Essa substituição é feita de forma aleatória, ou seja, um dos 10 melhores é escolhido aleatoriamente, é feita uma cópia dele, a classe *Crossover* altera de forma aleatória seus genes pela soma ou subtração de um valor númerico bem pequeno de cada peso dos neurônios da RNA, e então

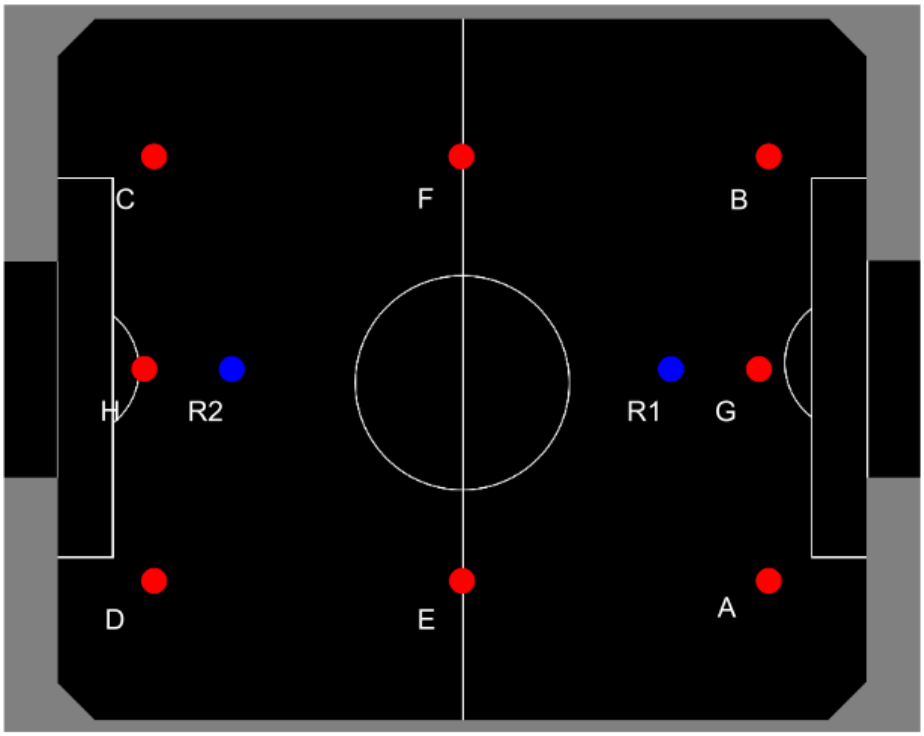
essa cópia substitui um dos 90 piores indivíduos da população, esse processo é repetido para todos os piores indivíduos. Não foi feito o cruzamento do indivíduos, por fins de simplicidade foi preferido fazer apenas a mutação para este trabalho.

Para o treinamento realizado, foi feita uma função na classe *GamePlay* que deixa apenas um robô e uma bola em campo, e faz a simulação durar por 20 segundos. Foram feitas 3 funções na classe *Fitness*: uma que atribui pontos de aptidão relacionados com a distância que o robô está da bola favorecendo que ele se aproxime dela, uma função que atribui pontos negativos sempre que o robô gire sobre o próprio eixo, para que ele evite ficar girando muito e outra função que atribui pontos sempre que o robô encosta na bola. Foi feito também uma função na classe *Referee* que faz com que sempre que o robô encosta na bola, o robô volte para uma posição determinada, e a bola vá para outra posição no campo, com o propósito de se treinar uma IA capaz de mover o robô para vários pontos dispersos no campo, partindo de locais diferentes. Os pontos podem ser vistos na Figura 20 e a sequência de combinações de inicialização do robô e da bola estão indicados na Tabela 1. Quando a sequência da tabela termina, se o robô encostar na bola novamente, ela reinicia.

Tabela 1: Sequência de posições iniciais do robô e da bola

Ordem	Robô	Bola
1	R1	A
2	R1	B
3	R1	C
4	R1	D
5	R1	E
6	R1	F
7	R1	G
8	R1	H
9	R1	A
10	R1	C
11	R2	A
12	R2	B
13	R2	C
14	R2	D
15	R2	E
16	R2	F
17	R2	G
18	R2	H
19	R2	A
20	R2	C

Figura 20: Indicações no campo das posições iniciais do robô e bola para o treinamento



4 RESULTADOS

4.1 Simulador

Foi feito um teste comparativo da velocidade de processamento das duas técnicas de detecção de colisão mencionadas na subseção 3.1.2.1 para a determinação de se um ponto se encontra dentro ou fora de um quadrado. A primeira das técnicas é a aplicação da técnica do item (b) da subseção 3.1.2.1 em todas as laterais do quadrado, caso o ponto estiver do lado interno do quadrado considerando-se os quatro lados, isso significa que o ponto está dentro do quadrado. Utilizando essa técnica com um computador com as especificações descritas na Tabela 2, o seu tempo de processamento medido em um teste feito foi de 23^{-9}s na média. A segunda técnica, descrita pelo item (c), quando utilizada no teste pelo mesmo computador, processa em 80^{-9}s na média. Portanto a primeira técnica é 3,5 vezes mais eficiente do que a segunda técnica em termos de tempo de processamento.

Tabela 2: Especificações técnicas do computador utilizado para a realização dos testes

Modelo	Inspiron 15 Série 3000
Fabricante	DELL
Processador	Intel(R) Core^{TM} i3-6006U CPU
Frequência do processador	2.00 GHz
Memória RAM	4Gb
Nível de otimização do compilador	-O2

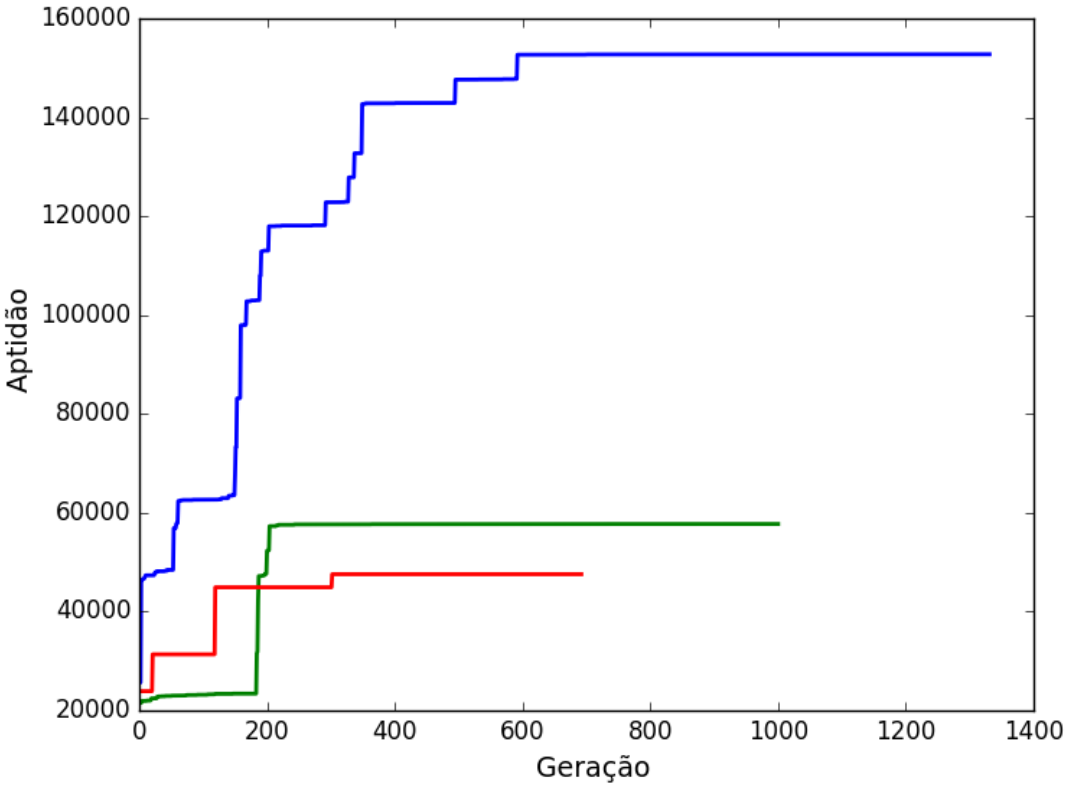
Foi feito um teste de performance no simulador utilizando o mesmo computador da Tabela 2, para saber o quão rápido ele consegue simular uma partida com dois times de 3 robôs controlados por IA. Para cada IA possuindo duas camadas internas com 100 neurônios cada, simulando uma partida de 200 segundos, foi necessário 1,85 segundos, ou seja, a simulação foi 108 vezes mais rápida que o real. Foi feito outro teste com cada IA possuindo duas camadas internas com 200 neurônios cada, nesse caso ao se simular uma partida de 200 segundos, foi necessário 5,25 segundos, ou seja, a simulação foi 38 vezes mais rápida.

Existem alguns problemas conhecidos do simulador, que são aceitáveis porém. Quando a bola colide exatamente com o vértice do robô, ela entra dentro dele as vezes. Quando a bola é prensada pelo robô na parede do campo ou por outro robô, ela acaba entrando dentro deles e por quando haver uma colisão, os robôs simplesmente pararem de se mover, em vez de agirem de forma mais real. Considerando que o foco é obter um simulador que simule minimamente bem e seja rápido, ele cumpre o seu papel.

4.2 Algoritmo Genético

A fim de se provar que é possível utilizar uma RNA treinada por um AG para o controle de um time de futebol de robôs, utilizou-se como primeiro passo desenvolver uma IA que é capaz de se mover no campo de um ponto qualquer a outro ponto qualquer. Foi utilizado três topologias diferentes de RNA para esse processo. A primeira topologia possui apenas uma camada interna com 60 neurônios. A segunda topologia possui duas camadas internas, contendo 200 neurônios cada camada. A terceira topologia possui duas camadas internas com 300 neurônios cada uma delas. Na Figura 21 pode ver os resultados do treinamento das três topologias, onde o tempo de duração dos treinamentos foram entre 8h e 12h.

Figura 21: Gráfico comparando as três topologias de RNA treinadas pelo AG. A linha verde representa a primeira topologia treinada de 60 neurônios na camada interna. A linha azul representa a segunda topologia treinada de duas camadas internas com 200 neurônios cada uma. A linha verde representa a terceira topologia treinada, com duas camadas internas de neurônios contendo 300 neurônios cada uma



A topologia com 2 camadas internas e 200 neurônios em cada uma foi a que obteve os melhores resultados, ela conseguiu passar por 28 pontos da Tabela 1 em 20 segundos, enquanto as outras conseguiram passar por no máximo 10 pontos e 8 pontos. Pode-se ver

que ela já se inicializa melhor que todas as outras, provavelmente no espaço de soluções, na inicialização dos pesos dos neurônios ela já começou mais próxima do máximo global, em relação às outras, e isso fez com que ela se desenvolvesse mais facilmente em direção ao máximo global, enquanto as outras provavelmente necessitariam de mais gerações para conseguirem sair da região de máximo local em que se encontram. A topologia com a menor quantidade de neurônios, representada pela linha verde na Figura 21, teve um grande crescimento perto da geração 200, provavelmente foi quando a IA evoluiu o suficiente para alcançar pela primeira vez a bola, e depois se manteve estável sem crescer mais, isso leva a acreditar que a RNA, devido ao baixo número de neurônios e a alta complexidade de realizar uma trajetória com um robô com dois graus de liberdade, tenha chegado ao seu limite, não sendo mais capaz de encontrar uma melhor solução para o problema.

5 CONCLUSÃO

Devido à necessidade de um simulador rápido o suficiente para o treinamento de uma IA utilizando AG, e ao fato de não ter sido encontrado um software com essa característica, foi desenvolvido um simulador físico em duas dimensões de robôs da categoria VSS de futebol de robôs em que foi priorizado a eficiência de processamento computacional sobre a precisão da física da simulação. A importância de se ter um simulador que seja rápido é que o AG necessita de muito tempo para encontrar uma solução (os testes realizados tiveram duração entre 8h e 12h), se o simulador não for rápido o suficiente, esse tempo pode se prolongar para dias ou semanas. O simulador consegue simular uma partida de futebol de robôs dezenas de vezes mais rápido que o tempo real (38 vezes mais rápido utilizando o computador especificado na Tabela 2), permitindo que seja feito um número grande de simulações em pouco tempo. A arquitetura do simulador, assim como a de todo o sistema desenvolvido, foi feita em módulos, o que ajudou em diversas partes do desenvolvimento do projeto, seja para isolar problemas de código e facilitar a sua resolução, ou para modificar partes do software para que ele se adéque melhor ao seu objetivo. A modularização do sistema facilita o trabalho de projetos futuros, pois permite que sejam adaptadas ou modificadas partes do código com facilidade, sem haver a preocupação com estabilidade de outras partes do código que não foram modificadas.

Utilizando uma RNA treinada por um AG foi possível controlar a movimentação de um robô e se pode concluir que o sistema pode ser utilizado como ferramenta para o treinamento de IA com AG. O robô treinado pelo AG é capaz de se mover em direção a bola no campo, foi testado e ele cumpriu seu objetivo com sucesso para a bola estando em 8 posições diferentes e o robô partindo de duas posições iniciais diferentes. O AG apresenta grande potencial para a resolução de problemas complexos em conjunto com a RNA, porém os resultados obtidos não são suficientes para provar que esses algoritmos fornecem uma boa solução para implementar uma IA capaz de controlar um time inteiro de robôs, que consiga realizar jogadas complexas no nível das IAs que jogam na Robocup. Estudos posteriores são necessários para se concluir sobre os limites da sua aplicação no futebol de robôs.

As disciplinas de física e geometria analítica cursadas pelo aluno serviram de base para o desenvolvimento do simulador, principalmente no cálculo de colisão e da dinâmica do movimento dos robôs e da bola, assim como as disciplinas de programação oferecidas no curso que serviram como conhecimento base para o desenvolvimento do código do projeto. O código foi feito em C++ e se aprendeu sobre a importância da aplicação de técnicas de engenharia de software, pois em alguns momentos grandes partes do código tiveram de ser modificadas ou replanejadas, e um planejamento anterior mais detalhado da sua estrutura

teria evitado esses problemas.

REFERÊNCIAS

ALBAB, R. T. U.; WIBOWO, I. K.; BASUKI, D. K. Path planning for mobile robot soccer using genetic algorithm. **Proceedings IES-ETA 2017 - International Electronics Symposium on Engineering Technology and Applications**, v. 2017-Decem, p. 276–280, 2017.

BEASLEY, D.; BULL, D. R.; MARTIN, R. An Overview of Genetic Algorithms : Part 1 , Fundamentals. **University Computing**, p. 1–16, 1993.

DARWIN, C. **On the origin of species by means of natural selection, or, The preservation of favoured races in the struggle for life /**. London :John Murray„ 1859. -1859. 564 p. Https://www.biodiversitylibrary.org/bibliography/68064 — Freeman, R.B. Charles Darwin (2nd ed.), | 373 — Garrison-Morton: | 220. Disponível em: <https://www.biodiversitylibrary.org/item/135954>.

FERNÁNDEZ, A. J.; COTTA, C.; CEBALLOS, R. C. Generating emergent team strategies in football simulation videogames via genetic algorithms. **9th International Conference on Intelligent Games and Simulation, GAME-ON 2008**, n. January 2014, p. 120–125, 2008.

IEEE. Rules for the IEE Very Small Competition. v. 2008, n. d, p. 1–14, 2008. Disponível em: <http://www.cbrobotica.org/wp-content/uploads/2014/03/VerySmall2008{_}en.>

KITANO, H. et al. RoboCup: The Robot World Cup Initiative. **Proceedings of the International Conference on Autonomous Agents**, p. 340–347, 1997.

_____. RoboCup: The Robot World Cup Initiative. **Proceedings of the International Conference on Autonomous Agents**, p. 340–347, 1997.

LODISH, H. et al. Molecular cell biology 4th edition. **National Center for Biotechnology Information, Bookshelf**, 2000.

Man, K. F.; Tang, K. S.; Kwong, S. Genetic algorithms: concepts and applications [in engineering design]. **IEEE Transactions on Industrial Electronics**, v. 43, n. 5, p. 519–534, Oct 1996.

MERIÇLI, Ç.; MERIÇLI, T.; AKıN, H. L. A Reward Function Generation Method Using Genetic Algorithms: A Robot Soccer Case Study. **9th International Conference on Autonomous Agents and Multiagent Systems AAMAS 2010**, n. January, 2010.

Petrasch, R.; Hentschke, R. Process modeling for industry 4.0 applications: Towards an industry 4.0 process modeling language and method. In: **2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)**. [S.l.: s.n.], 2016. p. 1–5.

SILVA, I. D.; SPATTI, D.; FLAUZINO, R. **REDES NEURAIS ARTIFICIAIS PARA ENGENHARIA E: CIENCIAS APLICADAS - CURSO PRATICO**. ARTLIBER, 2010. ISBN 9788588098534. Disponível em: <https://books.google.com.br/books?id=w2VHbwAACAAJ>.

SILVA, W. C. et al. USPDroidsSS robot soccer simulator for Very Small Size Category. **Proceedings - 2010 Latin American Robotics Symposium and Intelligent Robotics Meeting, LARS 2010**, p. 138–143, 2010.

Sola, J.; Sevilla, J. Importance of input data normalization for the application of neural networks to complex industrial problems. **IEEE Transactions on Nuclear Science**, v. 44, n. 3, p. 1464–1468, June 1997.

SRINIVAS, M.; PATNAIK, L. M. Genetic Algorithms: A Survey. **Computer**, v. 27, n. 6, p. 17–26, 1994. ISSN 00189162.

Uhrig, R. E. Introduction to artificial neural networks. In: **Proceedings of IECON '95 - 21st Annual Conference on IEEE Industrial Electronics**. [S.l.: s.n.], 1995. v. 1, p. 33–37 vol.1.

VIKHAR, P. A. Evolutionary algorithms: A critical review and its future prospects. **Proceedings - International Conference on Global Trends in Signal Processing, Information Computing and Communication, ICGTSPICC 2016**, p. 261–265, 2017.