# Trabalho Prático



# Inteligência Artificial

Licenciatura Engenharia Informática

Francisco Pinto nº 8170580

Luís Marques nº 8170485

Miguel Carvalho nº 8150146

Junho 2020

Índice

[**1.** **Introdução** 1](#_Toc43542802)

[**2.** **Problema A** 2](#_Toc43542803)

[2.1 Modelação da solução 2](#_Toc43542804)

[2.2 Regras de validação da solução 2](#_Toc43542805)

[2.3 Como é inicializada a primeira geração 3](#_Toc43542806)

[2.4 Configuração do algoritmo por parte do utilizador 4](#_Toc43542807)

[2.5 Implementação dos operadores genéticos 5](#_Toc43542808)

[2.6 Fitness da solução 7](#_Toc43542809)

[2.7 Testes do problema 8](#_Toc43542810)

[2.8 Requisitos implementados 19](#_Toc43542811)

[**3.** **Problema B** 21](#_Toc43542812)

[3.1 Descrição Dataset 21](#_Toc43542813)

[3.2 Descrição dos modelos treinados, configurações, métricas de performance 22](#_Toc43542814)

[3.3 Modelo utilizado pelo robot 29](#_Toc43542815)

[**4.** **Robot Final** 32](#_Toc43542816)

# **Introdução**

Este trabalho tem como objetivo o desenvolvimento de um robot para a *framework Robocode* que se movimente de acordo com um algoritmo genético e dispare de acordo com um modelo de *Machine Learning* treinado previamente. Estes problemas serão tratados de forma separada, o que levará ao desenvolvimento de alguns robots temporários ao longo do desenvolvimento do projeto.

A aprendizagem e aplicação dos conceitos de inteligência artificial será uma mais valia, para possibilitar otimizações nos robots, melhorando a sua movimentação e precisão de disparo.

Um dos subproblemas que será desenvolvido de forma separada é a movimentação, cujo objetivo a utilização de algoritmos genéticos, que nos permite encontrar um caminho válido entre dois pontos no campo de batalha, e contornando os obstáculos do mesmo.

O outro objetivo que também será tratado de forma separada, é o desenvolvimento de um robot *Robocode* que utilize um modelo *Machine Learning*, que visa otimizar a forma como o mesmo dispara contra os seus inimigos, e que registe a taxa de acerto dos mesmos disparos num *dataset*.

# **Problema A**

# Modelação da solução

Para a resolução deste problema decidimos optar por uma abordagem em que cada caminho é composto por um conjunto de pontos pré-determinado pelo utilizador e cada ponto possui coordenadas (x, y) de um ponto no mapa.



Figura 1- Adição de um novo ponto na lista de pontos

É necessária a posterior validação de cada caminho, o seu cruzamento, mutação e seleção inerente de cada uma das fases descritas.

# Regras de validação da solução

Para que cada solução apresentada como um caminho seja válida, a mesma tem de respeitar as seguintes regras:

1. Cada ponto (x, y) de cada caminho tem de ser obrigatoriamente válido, ou seja, não pode exceder os limites do mapa. Esta regra é garantida *a priori* pela programação inicial do problema;
2. A linha entre 2 pontos consecutivos não pode intercetar nenhum dos obstáculos existentes no mapa.

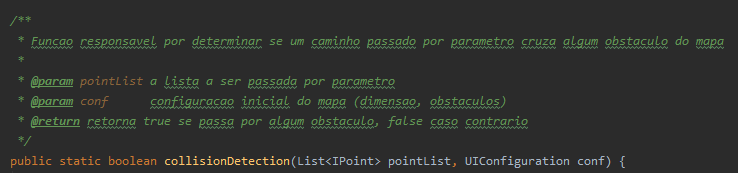


Figura 2- Função de deteção de colisão

Se as regras descritas em cima forem todas validadas (mais especificamente a regra Nº2), então o caminho/solução é dado/a como válido/a.

Caso a regra Nº2 não seja respeitada, o caminho/solução é na mesma adicionada á lista de caminhos gerados pois pode conter certos pontos que, por cruzamento, podem dar origem a um caminho melhor do que o selecionado até ao momento.

# Como é inicializada a primeira geração

A primeira geração é inicializada de forma aleatória.

É gerado um conjunto de pontos cujas coordenadas X e Y são geradas de forma aleatória onde posteriormente irão ser adicionados como pontos de um caminho até o número máximo de pontos permitidos num caminho seja atingido. Esse caminho é depois avaliado de acordo com as suas características (medindo o seu nível de fitness), selecionado como sendo o melhor caminho registado até ao momento ou não, e por fim adicionado a uma lista de caminhos que irão servir para o cruzamento da população mais avante.

É obrigatório a criação de pelo menos um caminho válido na primeira geração. Caso o ciclo de geração de caminhos da 1ª geração chegar ao fim e não contiver na sua lista pelo menos um caminho válido (caminho válido: um caminho que contenha todos os pontos gerados e que nenhuma das suas linhas intercete nenhum dos obstáculos do mapa), então toda a lista será eliminada e o ciclo de criação da primeira geração ocorrerá do inicio até que pelo menos um solução válida seja adicionada com sucesso.

# Configuração do algoritmo por parte do utilizador

O algoritmo genético foi desenvolvido atendendo a pedidos específicos por parte de um utilizador. Este recebe como parâmetro 4 variáveis cuja configuração é possível de ser manipulada em pelo menos 3 delas. As variáveis do algoritmo são: o número de pontos máximo que um caminho deve conter, o número máximo de caminhos que devem ser produzidos em cada geração e por fim a taxa de mutação a ser aplicada ao caminho selecionado como sendo o melhor, resultante da seleção da 1ª e 2ª geração. A 4ª variável é passada por parâmetro, apenas para atender a informações especificas do mapa tais como, as suas dimensões e a criação de um array contendo a posição dos obstáculos existentes. A sua manipulação por parte do utilizador é possível, contudo não é o foco principal pois esta tende a manter-se inalterada permanentemente.

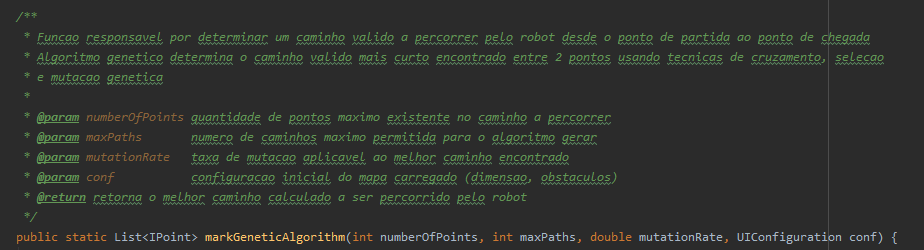


Figura 3- Função principal do Algoritmo Genético

# Implementação dos operadores genéticos

**Seleção:**

A estratégia de seleção adotada é baseada no fitness dos caminhos gerados, onde é selecionado apenas 1 caminho, sendo esse o melhor caminho para passar da fase de cruzamento á fase de mutação.

**Cruzamento:**

O cruzamento é implementado da seguinte forma:

* É selecionado o melhor caminho gerado na fase de iniciação;
* É gerado um caminho temporário com os mesmos pontos do melhor caminho;
* Selecionar 1 ponto aleatório de um caminho;
* Percorrer a lista de caminhos que existe;
* Em cada caminho percorrido, trocar o ponto desse caminho com o ponto do caminho temporário;
* Comparar *fitness* do caminho temporário com o melhor caminho;
* Se o *fitness* for maior, o melhor caminho passa a possuir os pontos do caminho temporário;
* Repetir processo durante *pointsList.size()* (tamanho da lista de pontos passada por parâmetro);

**Mutação:**

A mutação foi implementada da seguinte forma:

* É selecionado o melhor caminho proveniente das operações de inicialização e posteriormente de cruzamento;
* É percorrido os pontos que constituem esse caminho;
* É modificado os valores das suas coordenadas, primeiro no eixo do X e posteriormente do Y e verifica-se se o valor do seu fitness aumenta (o valor máximo de modificação provém da taxa de mutação previamente selecionada pelo utilizador e é dada por: coordenadas X ou Y \* taxaDeMutacao) (é tido em consideração se o valor mutado ultrapassa os limites do mapa ou não);



Figura 4 - Coordenadas mutadas

* Caso o valor do fitness aumente então esse ponto é modificado para ter essas novas coordenadas;
* O eixo dos X ou Y poderão variar em +/- do valor que tomou inicialmente, ou seja, pode ser mutado para a esquerda, direita, cima ou baixo;
* É repetido o processo até que o valor de fitness pare de crescer;

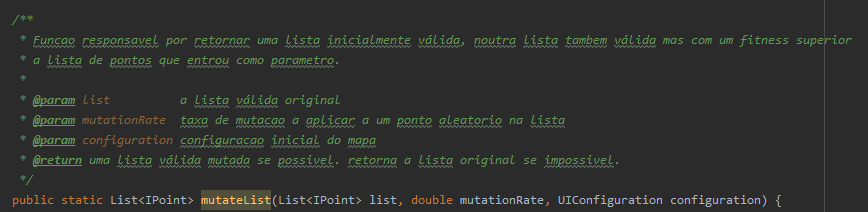


Figura 5 - Função de mutação

# Fitness da solução

O fitness é aplicado em todos os novos caminhos gerados do inicio ao fim do algoritmo genético de modo a determinar se estamos perante uma solução válida ou inválida e se caso estejamos perante uma solução válida, qual é o seu nível comparativamente com os outros caminhos.

A função de fitness recebe como parâmetro uma lista de pontos (um caminho), e a configuração do mapa. Após receber esses parâmetros a função irá determinar inicialmente se esse caminho colide com algum dos obstáculos presentes no mapa. Se não colide, é-lhe atribuído 10.000 pontos de fitness, se colide não lhe é atribuído qualquer quantidade de pontos.

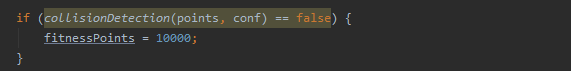


Figura 6 - Atribuição de fitness por não colisão

De seguida, após determinar se o caminho passado por parâmetro colide ou não com algum obstáculo, é determinado a distância total percorrida por esse caminho. Esse valor de distância vai subtraído ao valor que tinha anteriormente sido estabelecido (ou 10.000 ou 0 pontos) e por isso um caminho que não seja válido irá ter sempre um valor de fitness negativo e um caminho que seja válido vai ter um valor positivo de fitness. Cada um desses caminhos irá ser tanto melhor em termos de fitness quanto mais curto for o seu caminho.

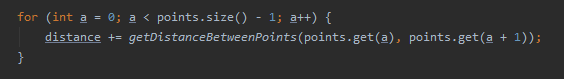


Figura 7 - Cálculo de distância total a retirar ao fitness

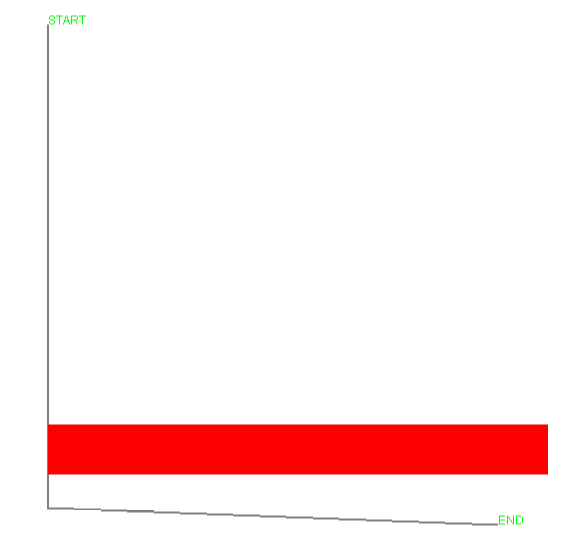
# Testes do problema

A performance e a eficácia do algoritmo foram avaliadas tendo em consideração os 11 mapas de teste fornecidos pelo professor.

Mais testes adicionais foram realizados no Robocode para testar a boa implementação com o software.

**Nota:** Os mapas aqui apresentados poderão não corresponder com os mapas de melhor fitness registado na *Leaderboard* online, são apenas para demonstrar o funcionamento do algoritmo.

**Mapa 0**

****

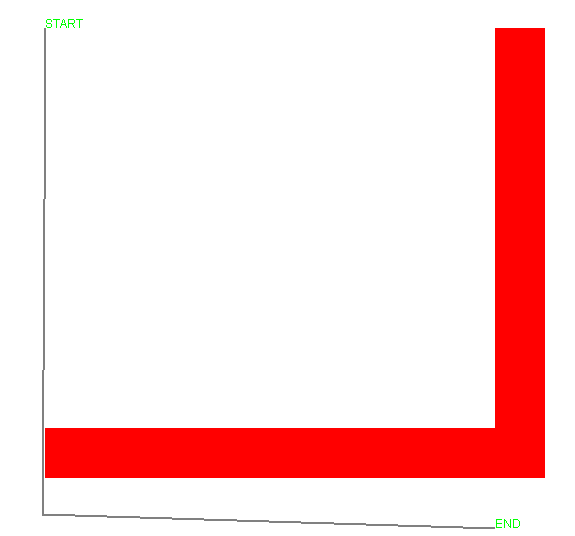
**Fitness:** 984.6798465815165

**Distancia percorrida:** 852.2823368461718

**Interceções:** 0

**Pontos:** 3

**Mapa 1**



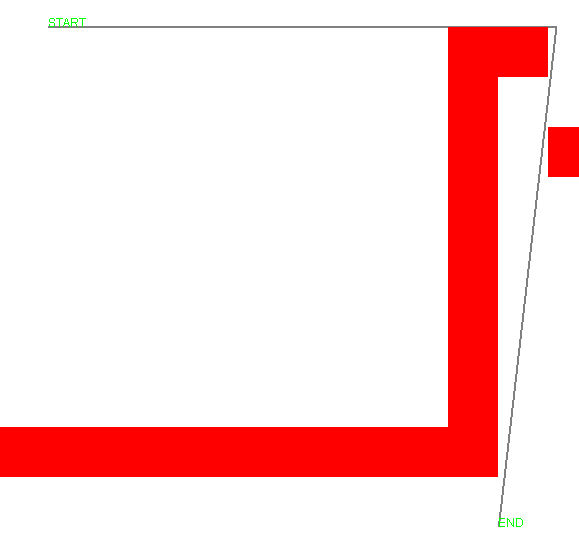
**Fitness:** 1031.1788968919523

**Distancia percorrida:** 907.5459037300968

**Interceções:** 0

**Pontos:** 3

**Mapa 2**



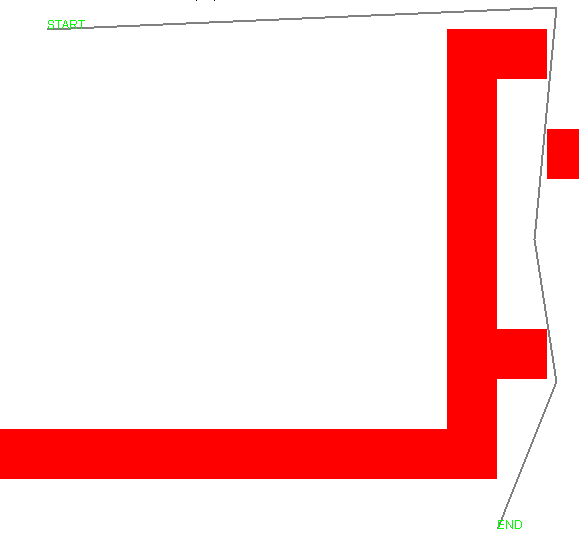
**Fitness:** 1338.3885474517676

**Distancia percorrida:** 1012.3470955674402

**Interceções:** 0

**Pontos:** 3

**Mapa 3**



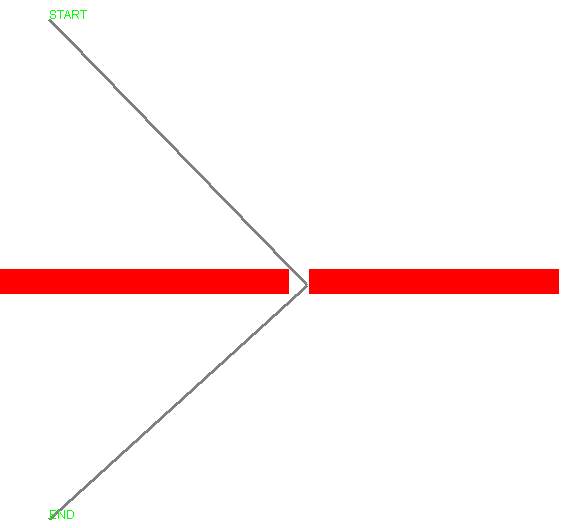
**Fitness:** 1506.5324257824475

**Distancia percorrida:** 1045.6002418957576

**Interceções:** 0

**Pontos:** 5

**Mapa 5**



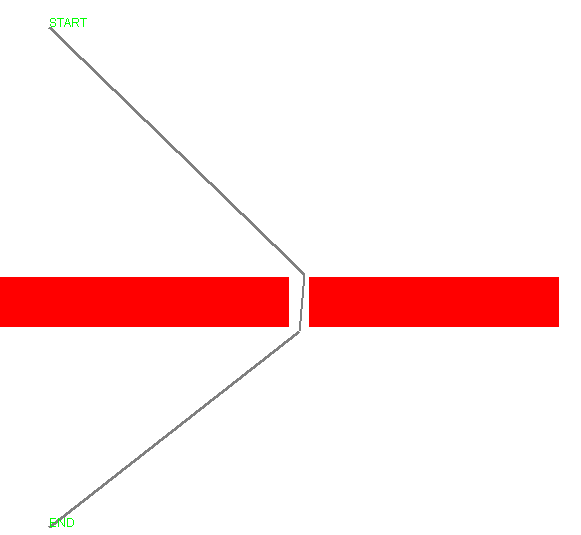
**Fitness:** 851.9828936540308

**Distancia percorrida:** 718.8773394880591

**Interceções:** 0

**Pontos:** 3

**Mapa 6**



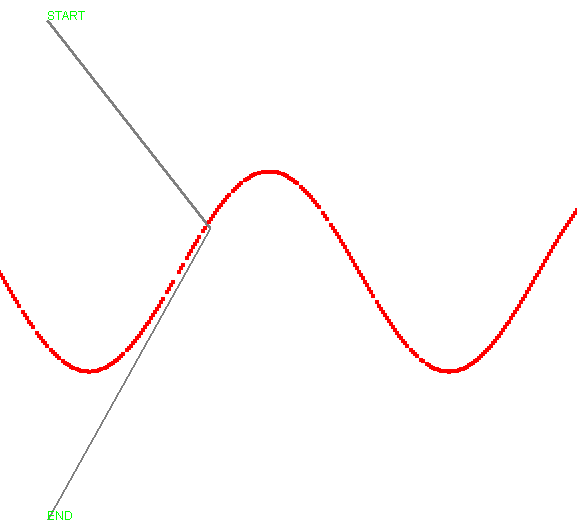
**Fitness:** 788.5934150612361

**Distancia percorrida:** 729.9043414627447

**Interceções:** 0

**Pontos:** 4

**Mapa 7**



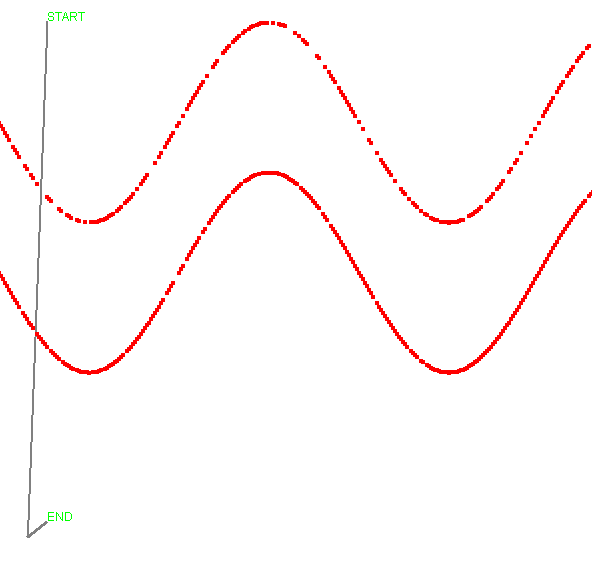
**Fitness:** 638.0769810061138

**Distancia percorrida:** 598.5175580846925

**Interceções:** 0

**Pontos:** 3

**Mapa 8**



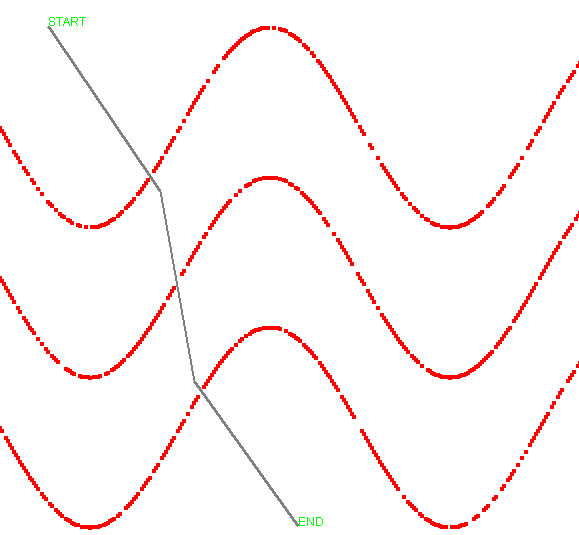
**Fitness:** 778.7873303937281

**Distancia percorrida:** 541.9999483851822

**Interceções:** 0

**Pontos:** 3

**Mapa 10**



**Fitness:** 800.817267880835

**Distancia percorrida:** 570.880263943829

**Interceções:** 0

**Pontos:** 4

**Robocode**

O Robocode é um jogo de programação, onde o objetivo é programar robots que competem uns contra os outros num campo/arena de batalha. O jogador deve treinar e programar o seu robot para que este se movimente de acordo com um algoritmo genético e dispare de acordo com um modelo Machine Learning, de forma a reagir da melhor forma a todas as adversidades que podem aparecer ao longo da batalha.

**Demonstração de utilização em Robocode:**

# Requisitos implementados

**R-001**

**Nome:** 1ª população

**Descrição:** O software deverá conseguir produzir a primeira geração de tamanho de população pré-definido pelo utilizador com valores aleatórios.

**Nível:** 5

**Estado:** Finalizado

**R-002  
Nome:** Cruzamento

**Descrição:** Reproduzir população de acordo com o melhor espécimen da geração anterior para possivelmente melhorar os pontos do melhor caminho.

**Nível:** 5

**Estado:** Finalizado

**R-003**

**Nome:** Mutação

**Descrição:** Existência de uma função com a capacidade de mutar o melhor espécimen selecionado das gerações anteriores para aperfeiçoar os pontos de cada caminho.

**Nível**: 5

**Estado:** Finalizado

**R-004**

**Nome:** Fitness

**Descrição:** Elaboração de uma função de fitness interna que seja capaz de determinar o melhor caminho gerado pela função algoritmo genético.

**Nível:** 5

**Estado:** Finalizado

**R-005**

**Nome:** Calcular distancia

**Descrição:** Elaboração de uma função para determinar a distancia existentes entre 2 pontos consecutivos num caminho.

**Nível:** 5

**Estado:** Finalizado

**R-006**

**Nome:** Colisão

**Descrição:** O software deverá possuir a capacidade de determinar se um conjunto de pontos se traduz numa potencial colisão com algum obstáculo existente no mapa carregado, a partir de uma função que determine se existe ou não uma colisão.

**Nível:** 5

**Estado:** Finalizado

**R-007**

**Nome:** Algoritmo genético

**Descrição:** O software deverá ser capaz de reunir todas as ações necessárias a tomar durante a atividade de um algoritmo genético e que retorne no fim da sua atividade, um caminho com uma lista de pontos que represente um caminho válido a percorrer e o caminho mais curto gerado durante a sua atividade.

**Nível:** 5

**Estado:** Finalizado

**R-008**

**Nome:** Anticolisão

**Descrição:** O robot deverá ser capaz de, no caso de haver uma colisão iminente com o robot e um inimigo, modificar em tempo real o seu caminho e assim evitar uma colisão. No caso de não haver nenhum caminho válido a tomar, o robot deverá terminar a sua deslocação.

**Nível:** 3

**Estado:** Finalizado

# **Problema B**

# Descrição Dataset

Mediante o problema proposto, decidimos exportar um dataset com algumas variáveis que consideramos importantes para a construção do modelo no H2O. Estas foram:

* *“targetName”*: nome do robot alvo;
* *“targetPosX”*: coordenada X do robot alvo;
* *“targetPosY”*: coordenada Y do robot alvo;
* *“targetHeading”*: direção para a qual o robot alvo está a apontar;
* *“targetVelocity”*: velocidade do robot alvo;
* *“power”*: potência da bala que disparamos;
* *“distance”*: distância a que se encontra o robot alvo;
* *“hit*”*:* valor inteiro com intenção booleana para definir se a bala disparada para o robot que queríamos atingiu com sucesso o alvo.

# Descrição dos modelos treinados, configurações, métricas de performance

Com a utilização da ferramenta H2O conseguimos gerar vários modelos com várias configurações, dos quais selecionamos apenas seis. Destes últimos, um destacou-se por entre os demais, sendo aquele que registou um MSE (*Mean Squared Error*) e um RMSE (*Root Mean Squared Error*) menores e com valores aceitáveis, isto é, não apresentavam características de *overfitting* nem de *underfitting*.

Modelos gerados:

* *deeplearning-400-300-200-100\_battle\_results\_error*

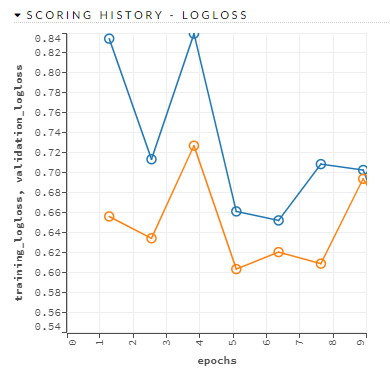


Figura 8 - Gráfico do modelo deeplearning-400-300-200-100\_battle\_results

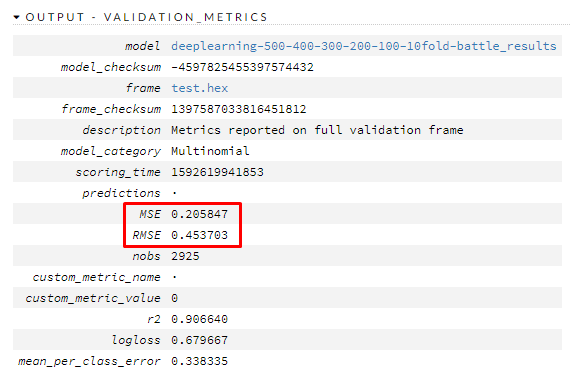


Figura 9 - Tabela de testes do modelo deeplearning-400-300-200-100\_battle\_results

* *deeplearning\_500\_400\_300\_200\_100\_10fold\_battle\_results*

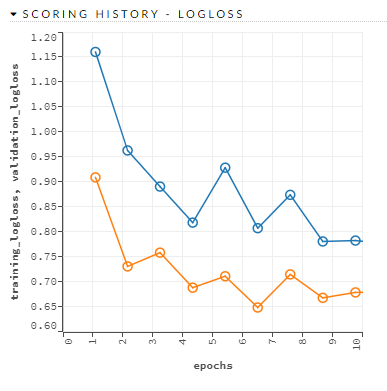


Figura 10 - Gráfico do modelo deeplearning\_500\_400\_300\_200\_100\_10fold\_battle\_results

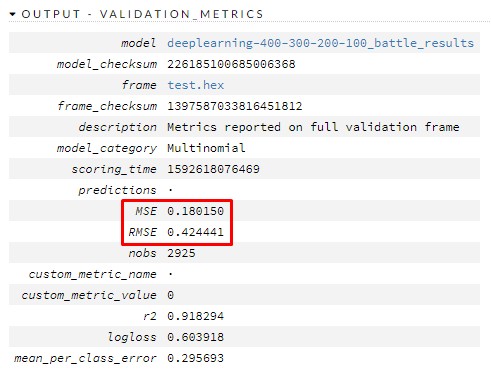


Figura 11 - Tabela de testes do modelo deeplearning\_500\_400\_300\_200\_100\_10fold\_battle\_results

* *drf\_50\_20\_10fold*

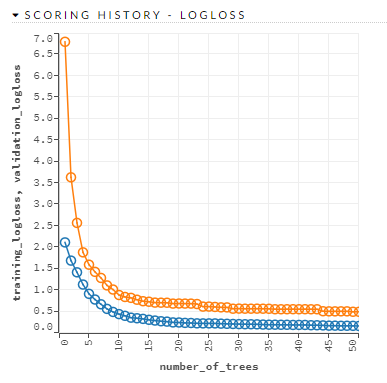


Figura 12 - Gráfico do modelo drf\_50\_20\_10fold

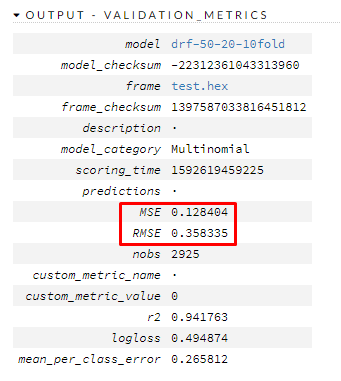


Figura 13 - Tabela de testes do modelo drf\_50\_20\_10fold

* *drf\_50\_20\_battle\_results*

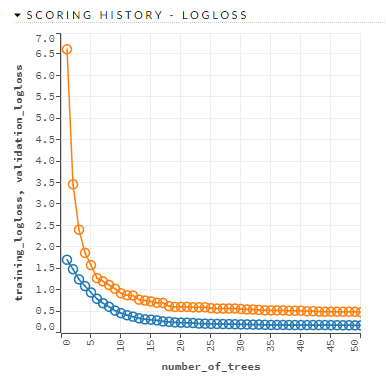


Figura 14 - Gráfico do modelo drf\_50\_20\_battle\_results

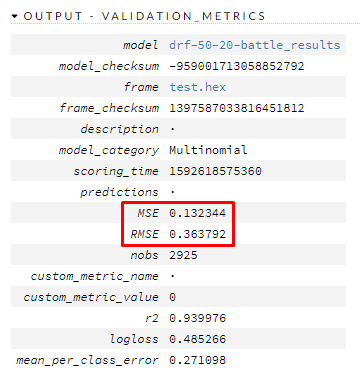


Figura 15 - Tabela de testes do modelo drf\_50\_20\_battle\_results

* *drf\_100\_50\_10fold\_battle\_results*

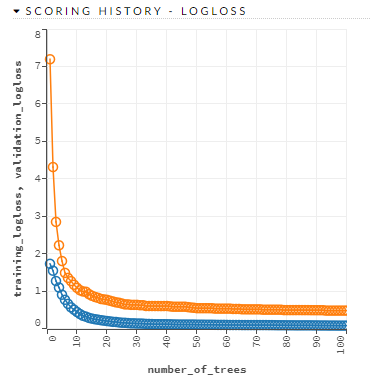


Figura 16 - Gráfico do modelo drf\_100\_50\_10fold\_battle\_results

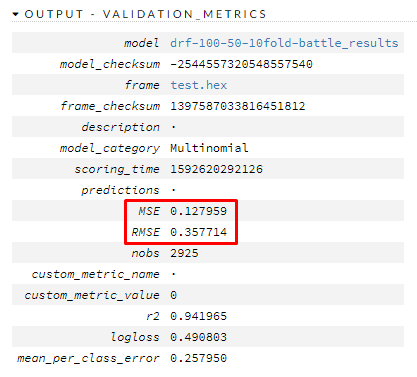


Figura 17 - Tabela de testes do modelo drf\_100\_50\_10fold\_battle\_results

* *drf\_100\_50\_battle\_results*

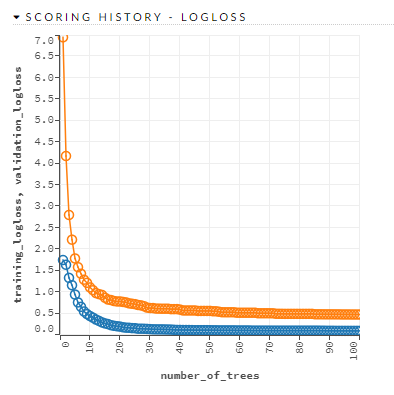


Figura 18 - Gráfico do modelo drf\_100\_50\_battle\_results

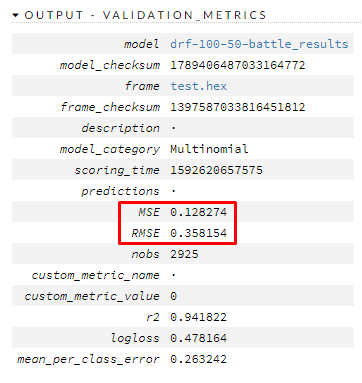


Figura 19 - Tabela de testes do modelo drf\_100\_50\_battle\_results

# Modelo utilizado pelo robot

Após a nossa rigorosa análise de todos os seis modelos gerados, selecionamos o modelo que apresentou valores de MSE e RMSE mais reduzidos, sendo este o modelo *drf\_100\_50\_10fold\_battle\_results*, que obteve os seguintes resultados:

Gráfico:

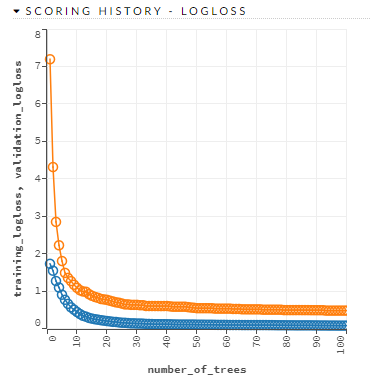


Figura 20 - Gráfico do modelo drf\_100\_50\_10fold\_battle\_results

Tabela de testes:

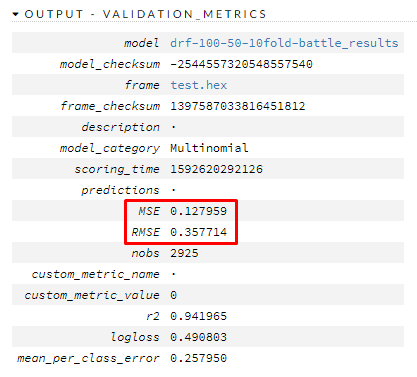


Figura 21 - Tabela de testes do modelo drf\_100\_50\_10fold\_battle\_results

Tabela de previsões:

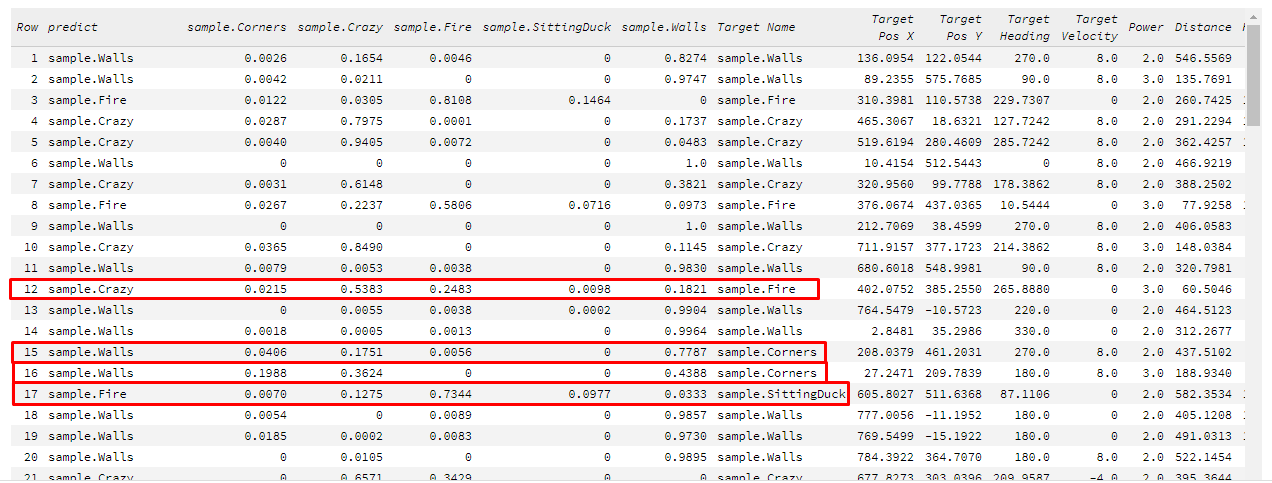


Figura 22 - Tabela de previsões do modelo drf\_100\_50\_10fold\_battle\_results

Como podemos verificar, sinalizado a vermelho temos as previsões que não se encontram de acordo com os valores reais, o que significa que o modelo que selecionamos também tem as suas falhas, mas de todos os que treinamos e geramos foi o melhor.

# **Robot Final**

Com a finalização da elaboração do projeto, foi levado a cabo a construção do último robot que possuirá todas as características de todos os robots construídos até ao momento.

O último robot (*MARK3*) irá possuir as características de movimentação de acordo com o algoritmo genético desenvolvido no robot *MARK1* e as características de disparo de acordo com a técnica de *Machine Learning* do robot *MARK2*.

Terminado o MARK3, ficamos a possuir um robot com todas as capacidades propostas no enunciado e com capacidade de aplicar essas mesmas características em ambiente real (*Robocode*) e com uma capacidade de resposta aceitável.