FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Departamento de Engenharia Informática

Fundamentos de Inteligência Artificial
2020/2021 - 2º Semestre

Relatório Trabalho Prático No 2:
D31
The Rise of the Ballz

Realizado por:
Maria Paula Viegas - viegas@student.dei.uc.pt - 2017125592 - PL6
Sofia Silva - sofiasilva@student.dei.uc.pt - 2018293871 - PL5
Sofia Meireles - sofiacosta@student.dei.uc.pt - 2018296218 - PL5

Objetivo

Este trabalho tem como objetivo criar controladores para o D31 de forma a que o mesmo consiga praticar bom futebol no simulador de treino virtual do Unity. O robot será sujeito a diferentes cenários onde terá de aprender tarefas como, controle de bola, defesa e ataque. Para tal, recorremos a uma rede neuronal que será aperfeiçoada pelo algoritmo genético.

Utilizando os conhecimentos do TP1, os conhecimentos das aulas e o package que nos foi fornecido, o nosso principal objectivo é que o controlador seja capaz de conduzir o D31 a ultrapassar um conjunto de tarefas que lhe vão sendo dadas relacionadas com o futebol, de forma a conseguir ganhar ao seu adversário.

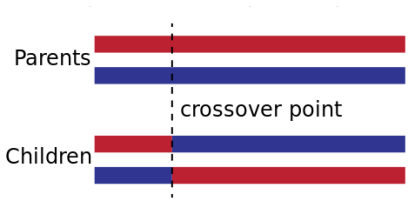
Código

A nível de implementação da meta 1, começámos por transformar os pseudo-códigos fornecidos no enunciado em código funcional. De seguida, implementámos a função de crossover de 1 ponto (*script GeneticIndividual.cs* e *HillClimberIndividual.cs*).

Na função *GaussianMutation*, pretendemos fazer mudanças pequenas e aleatórias em indivíduos da população seguindo a distribuição Gaussiana. Assim, garantimos a diversidade entre gerações. Seguimos o pseudo código e garantimos que as variáveis *mean* e *standard_deviation* sejam ajustáveis no Unity.

Em relação à função *tournamentSelection*, o algoritmo é responsável por selecionar o melhor indivíduo num torneio com uma quantidade de *tournamentSize* indivíduos. O valor de *tournamentSize* é também ajustado no Unity.

Por fim, a função *Crossover* é responsável por combinar a informação genética de um indivíduo com um indivíduo "parceiro" para gerar novos indivíduos. A combinação acontece a partir de um ponto *crossoverPoint* aleatório e tem uma probabilidade definida no Unity.



esquema de funcionamento do Single-Point Crossover

Testes e resultados obtidos

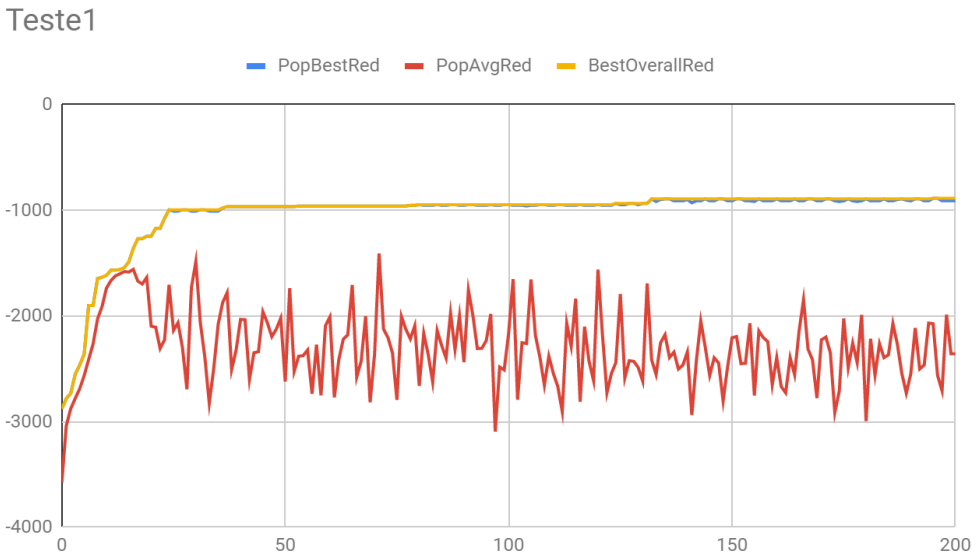
Inicialmente, começámos por fazer testes de parametrização. Para isso, fomos fixando parâmetros e alterando apenas um de cada vez de forma a conseguirmos perceber quais seriam os parâmetros mais adequados. Utilizámos também uma função fitness bastante básica para estes testes:

*fitness = -(distanceToBall.Average() * 15000.0f);*

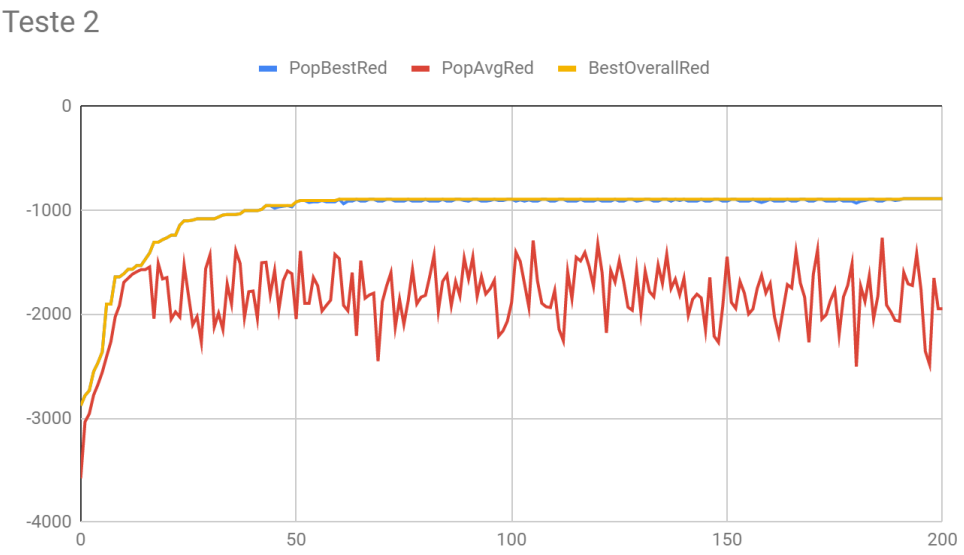
Variáveis como o tamanho da população (20), número de simulações simultâneas (20), número de gerações (200) e o tempo de simulação (5) foram constantes.

Probabilidade de Crossover

• Taxa de mutação: 0.1 • Tamanho de torneio: 5 • Função: Gaussiana • Elitista: sim



Taxa de crossover: 0.7

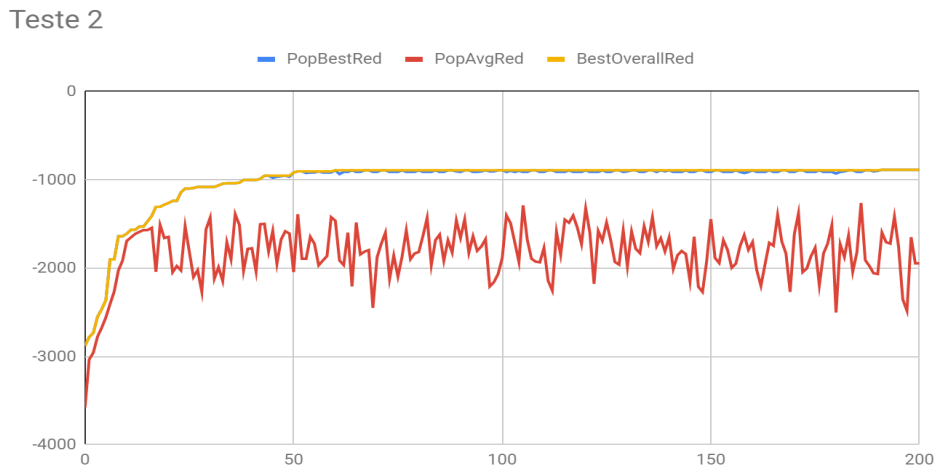


Taxa de crossover: 0.8

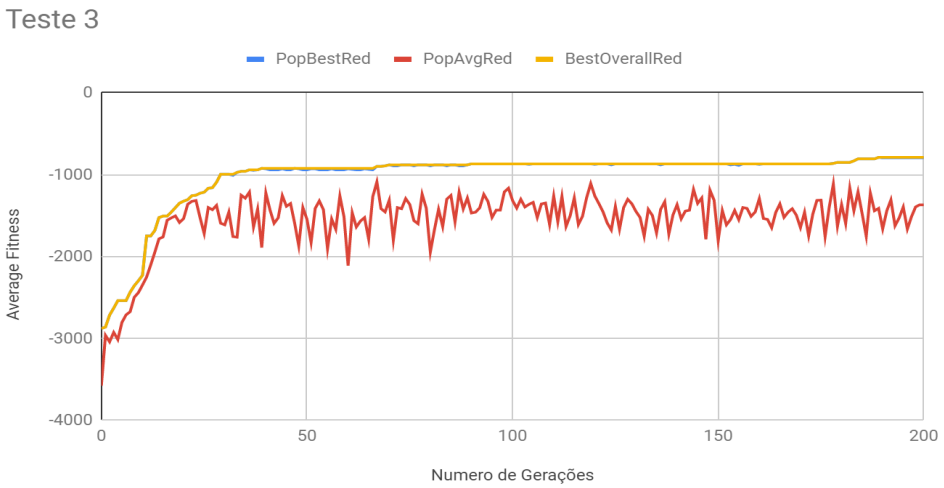
A partir da análise gráfica concluímos que a probabilidade de crossover 0.8 apresenta melhores resultados uma vez que o agente aprende relativamente rápido e a média da população encontra-se próxima do melhor.

Probabilidade de Mutação

• Taxa de crossover: 0.8 • Tamanho de torneio: 5 • Função: Gaussiana • Elitista: sim



Taxa de mutação: 0.05

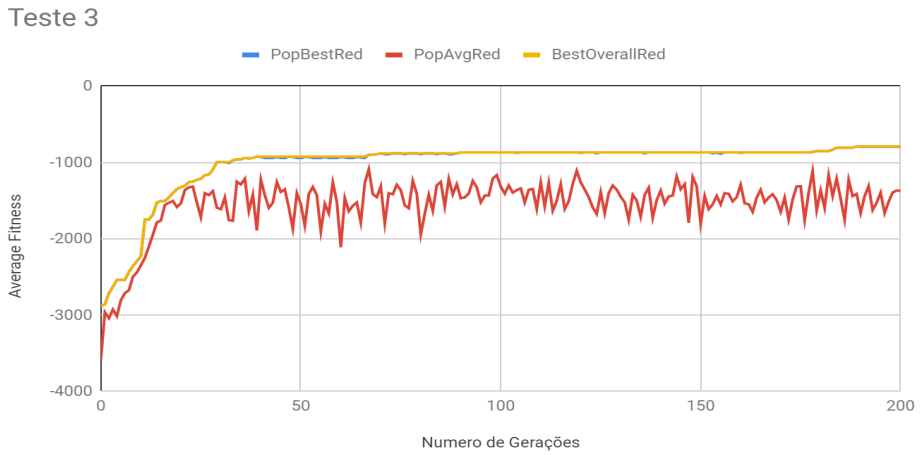


Taxa de mutação: 0.1

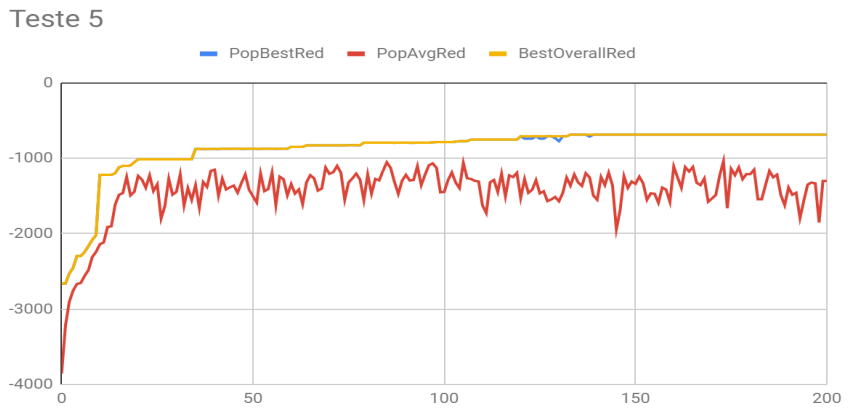
Uma vez que com taxa de mutação 0.1 o agente no final ainda consegue aprender mais um pouco e a média da população se encontra mais próxima do melhor, concluímos que 0.1 de taxa de mutação seria melhor. Os valores da taxa de mutação não podem ser altos porque deixam a simulação demasiado aleatória para chegar a qualquer conclusão. Valores baixos previnem a estagnação.

Tamanho do torneio

• Taxa de crossover: 0.8 • Função: Gaussiana • Elitista: sim • Taxa de mutação: 0.05



Tamanho do torneio 5

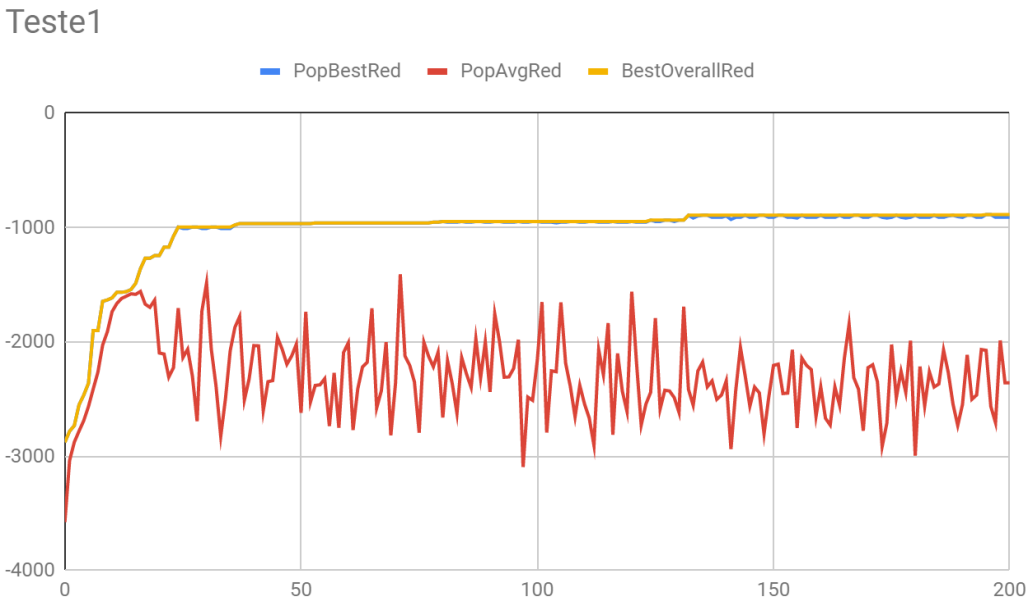


Tamanho do torneio 2

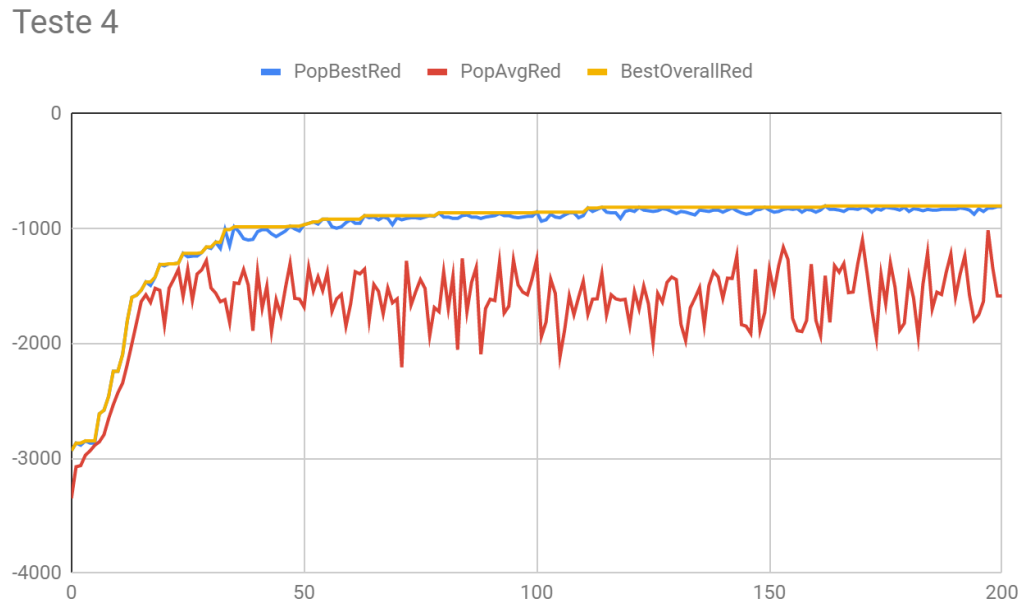
Apesar de ambos os gráficos não apresentarem grandes diferenças entre si, optámos por seguir os nossos testes com torneios de tamanho 5 pois considerámos que possuia um comportamento mais estável. Com o tamanho do torneio a 2, o melhor tem mudanças abruptas ao início e ao longo das gerações, o melhor e a média começam a divergir mais acentuadamente.

Elitismo

• Taxa de crossover: 0.8 • Função: Gaussiana • Tamanho do torneio: 5 • Taxa de mutação: 0.05



Elitismo: sim



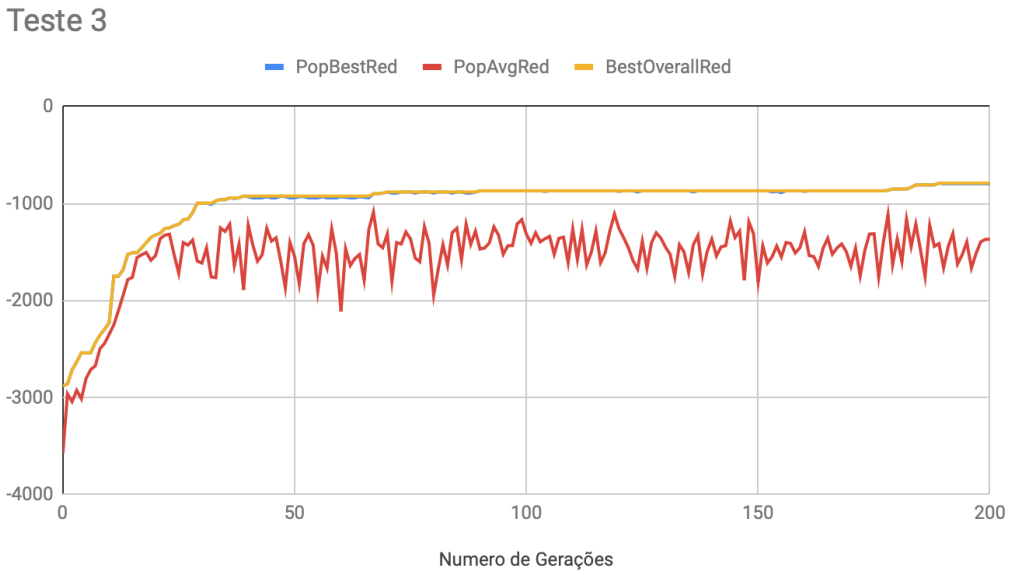
Elitismo: nao

Através da análise gráfica, é possível observar que com elitismo o robot começa a aprender mais rápido, sendo portanto melhor que sem elitismo. Vemos também que o sem elitismo continua com um pequeno crescimento nas gerações finais porque o elitismo traz um risco de estagnação num máximo local.

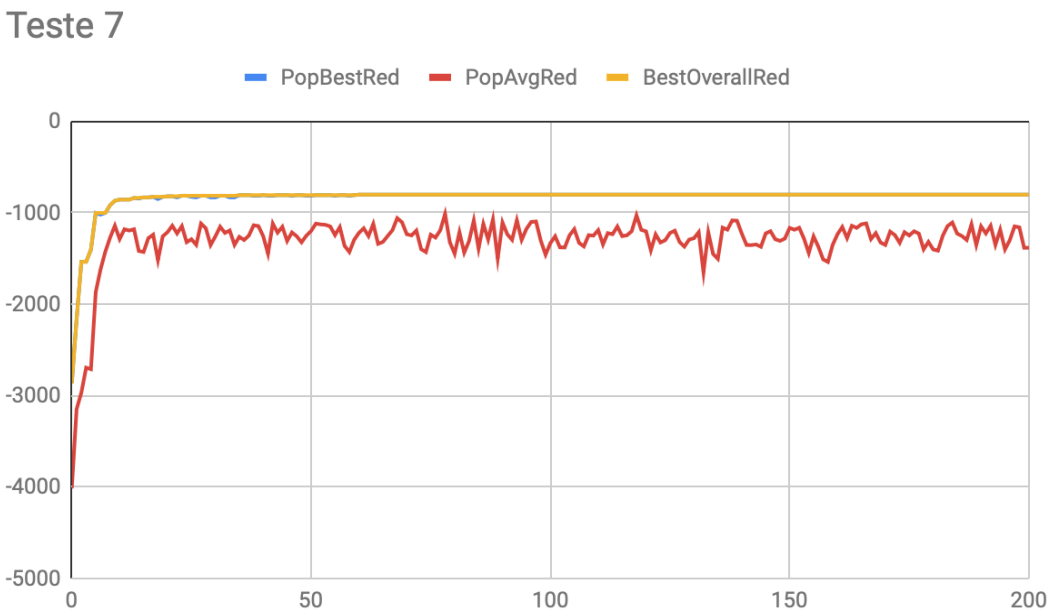
Este comportamento está dentro do esperado porque o elitismo tem como objetivo evitar que os indivíduos melhores sejam perdidos e a geração continue sempre a melhorar, sem piorar. O elitismo consegue alcançar isso uma vez que guarda um determinado número de indivíduos com os melhores resultados para a geração seguinte.

Função Gaussiana - Mean

• Taxa de crossover: 0.8 • Função: Gaussiana • Tamanho do torneio: 5 • Taxa de mutação: 0.05 • Elitista: sim



Mean: 0

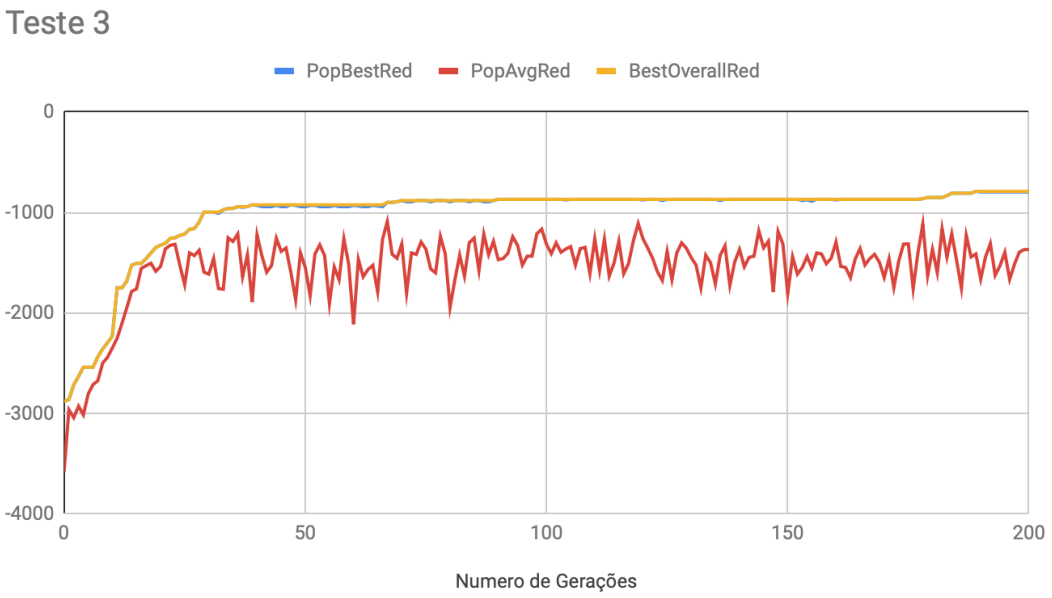


Mean: 0.5

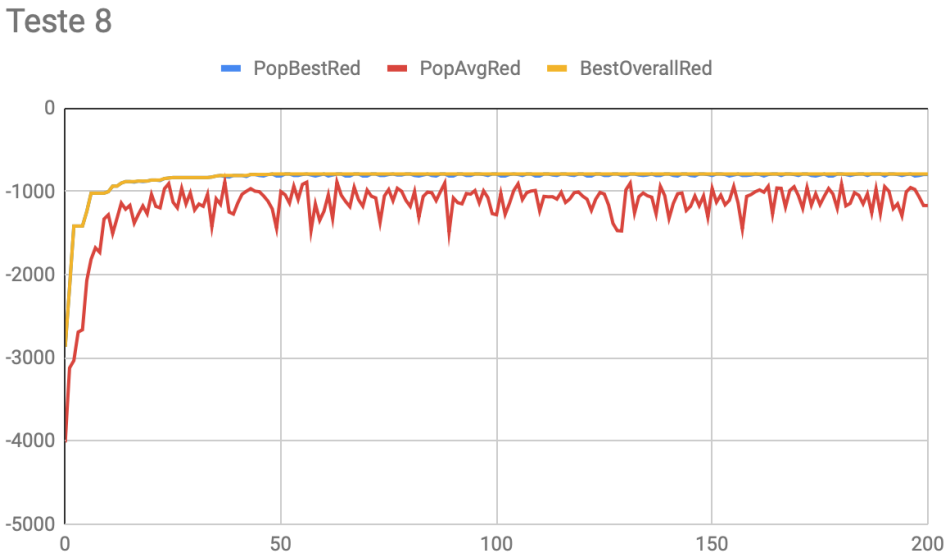
Decidimos prosseguir com o valor de mean 0 visto que é o padrão da função gaussiana e que ela aparenta ter um crescimento ligeiramente maior.

Função Gaussiana - Standard Deviation

• Taxa de crossover: 0.8 • Função: Gaussiana • Tamanho do torneio: 5 • Taxa de mutação: 0.05 • Elitista: sim



Standard Deviation: 0.5

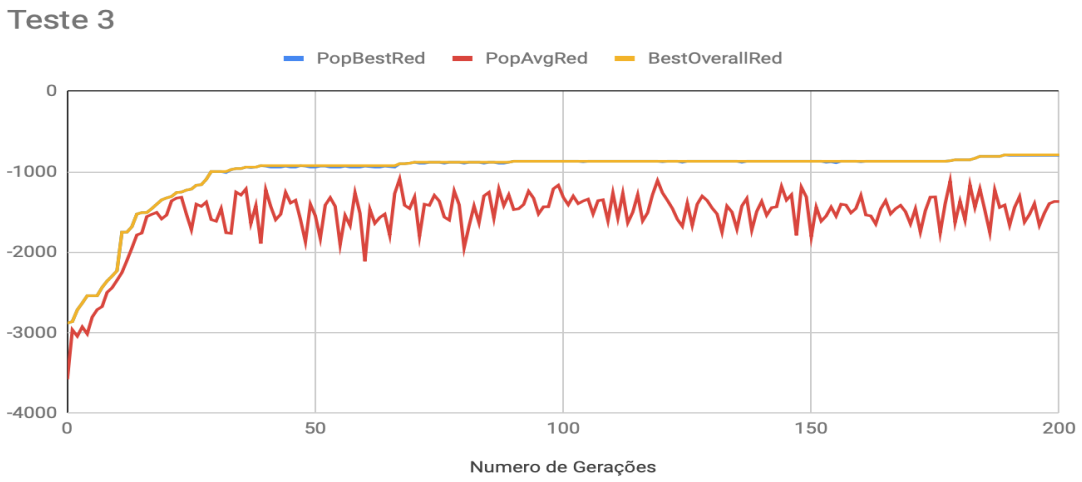


Standard Deviation: 0.8

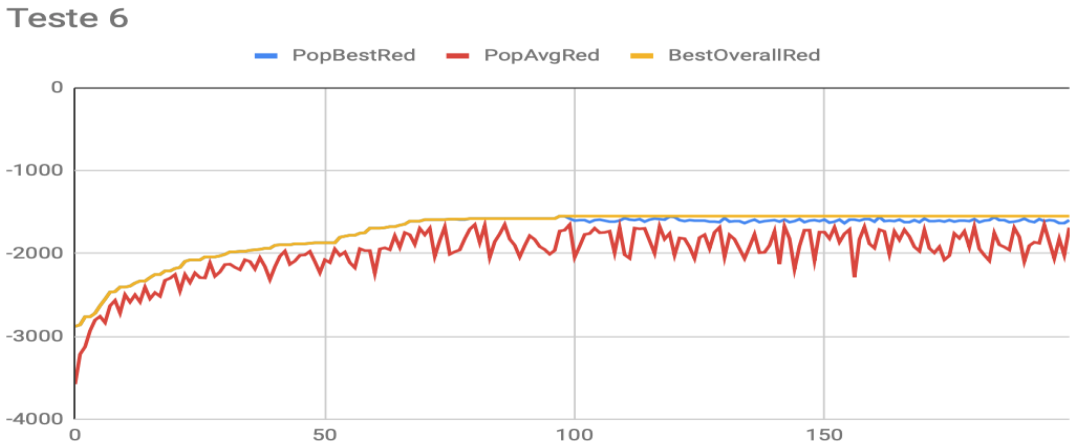
Também decidimos prosseguir com o valor de standard deviation: 0.5 visto que é o padrão da função gaussiana e que ela aparenta ter um crescimento ligeiramente maior.

Função Gaussiana - Método de Mutação

• Taxa de crossover: 0.8 • Tamanho do torneio: 5 • Taxa de mutação: 0.05 • Elitista: sim



Função: Gaussiana



Função: Random

Pela análise dos gráficos, concluímos que a função de mutação gaussiana é mais vantajosa, uma vez que a população aprende mais em geral e possui o comportamento desejado do *boost* de aprendizagem inicial evoluindo para um patamar de estabilidade.

Cenários:

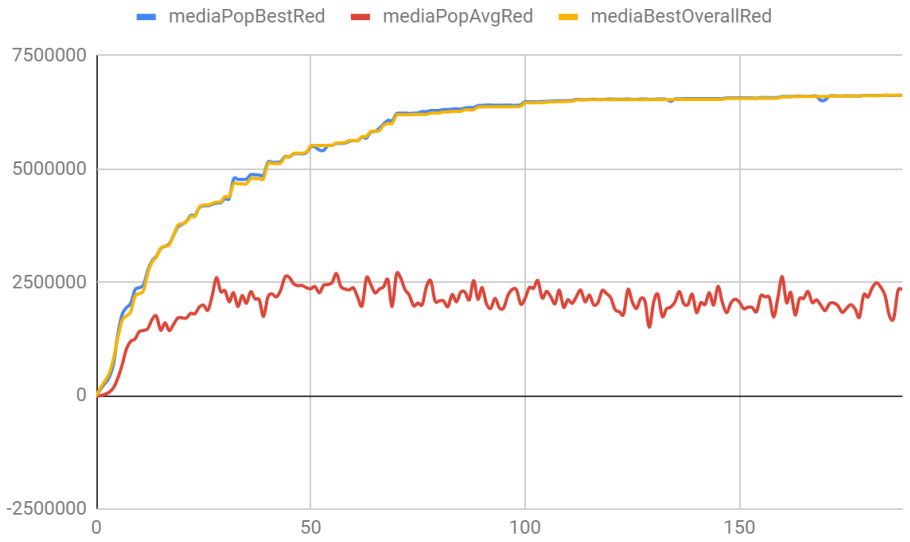
Para todos os cenários utilizámos os parâmetros acima indicados, com base nos melhores resultados obtidos nos testes de parametrização.

Evolving - ControlTheBallToAdversaryGoal

Após a realização de inúmeros testes, com diversas funções fitness diferentes, a função que mais se destacou foi:

---

*fitness = hitTheBall * 15000f - distanceToBall.Average() * 15000f - distanceToAdversaryGoal.Average() * 2000f - distancefromBallToAdversaryGoal.Average() * 12000f;*



média dos resultados obtidos nos testes para a função fitness indicada

Através da análise ao gráfico, verificamos que o melhor valor obtido foi 6624659.

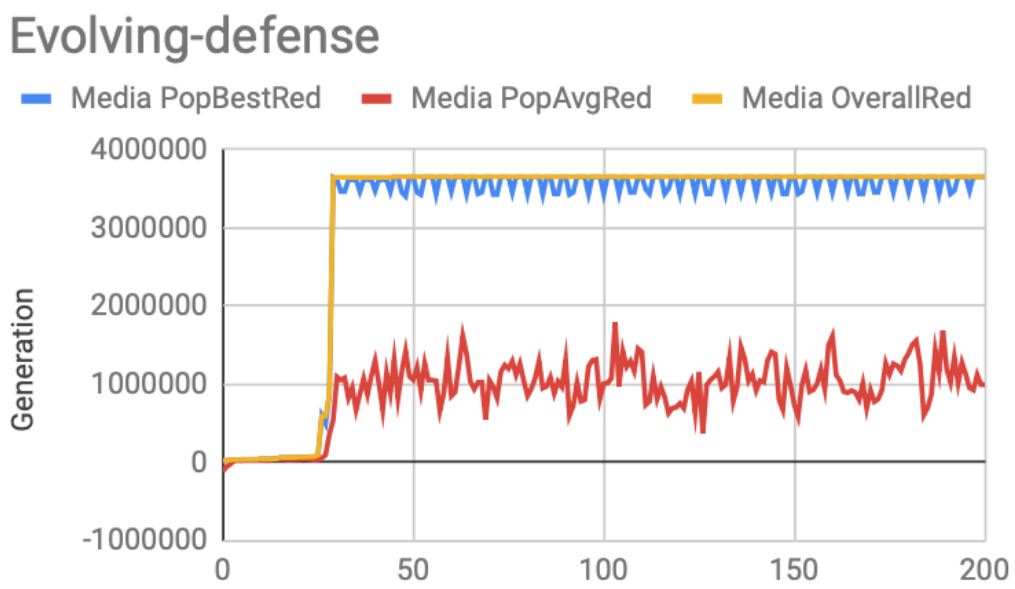
Este resultado foi obtido com tanto o agente como a bola a iniciarem em posições fixas.

Evolving - Defense

Após a realização de inúmeros testes, com diversas funções fitness diferentes, a função que se destacou foi:

```
distance = Math.Exp(distanceTravelled));
distanceGoals=Math.Pow(2,distancefromBallToAdversaryGoal.Average())*2000-Math.Pow(1/2, distanceToMyGoal.Average())*1000;
if (GoalsOnMyGoal > 0)
    fitness = distance * 10000 - GoalsOnMyGoal * 30000 + hitTheBall * 40000;
else
```

*fitness = distance * 10000 + hitTheBall * 400000 + distanceGoals - hitTheWall * 2000;*



média dos resultados obtidos nos testes para a função fitness indicada

Após a realização de vários testes com o intuito de aperfeiçoar a nossa função fitness para retirarmos o melhor partido da mesma e consequente análise dos gráficos, concluímos que a função ainda poderia ser melhorada, principalmente para o cenário *Evolving - DefenseRandom*. Vemos que apesar dos melhores resultados terem o valor alto, a média da performance é baixa e muito longe do melhor possível.

Evolving - OnevsOne

Relativamente às funções fitness, começámos por escolher as duas anteriores que apresentavam melhores resultados. No entanto, depois de testarmos estas duas funções, chegámos à conclusão que a função fitness de defesa não estava a fazer o que era esperado. Deste modo, resolvemos utilizar uma função fitness diferente. Assim, definimos a função para o ataque:

---

*fitness1 = hitTheBall * 15000f - distanceToBall.Average() * 15000f - distanceToAdversaryGoal.Average() * 2000f - distancefromBallToAdversaryGoal.Average() * 12000f;*

e para a defesa:

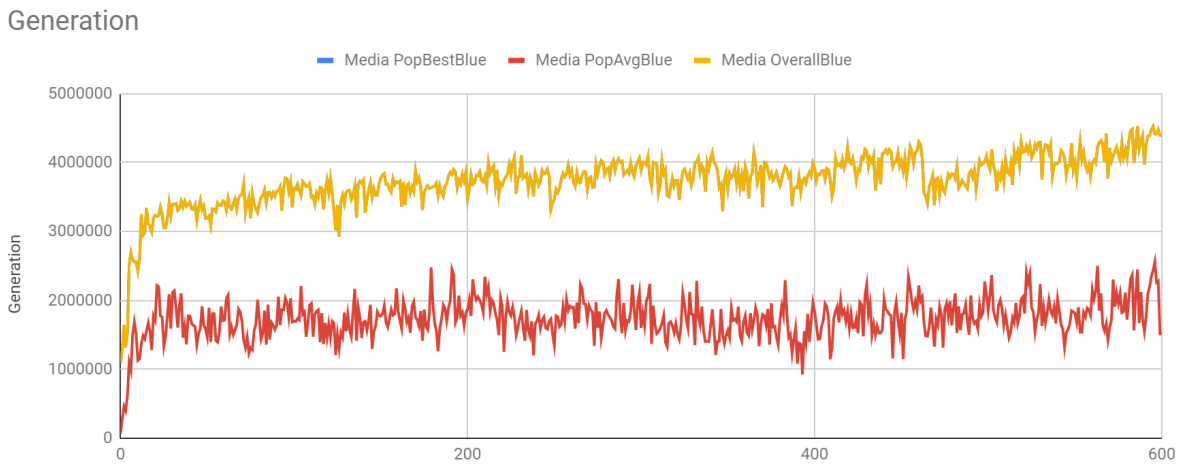
*fitness2 = hitTheBall * 10000.0f + distanceGoals - hitTheWall * 500.0f - GoalsOnMyGoal * 500f;*

sendo que,

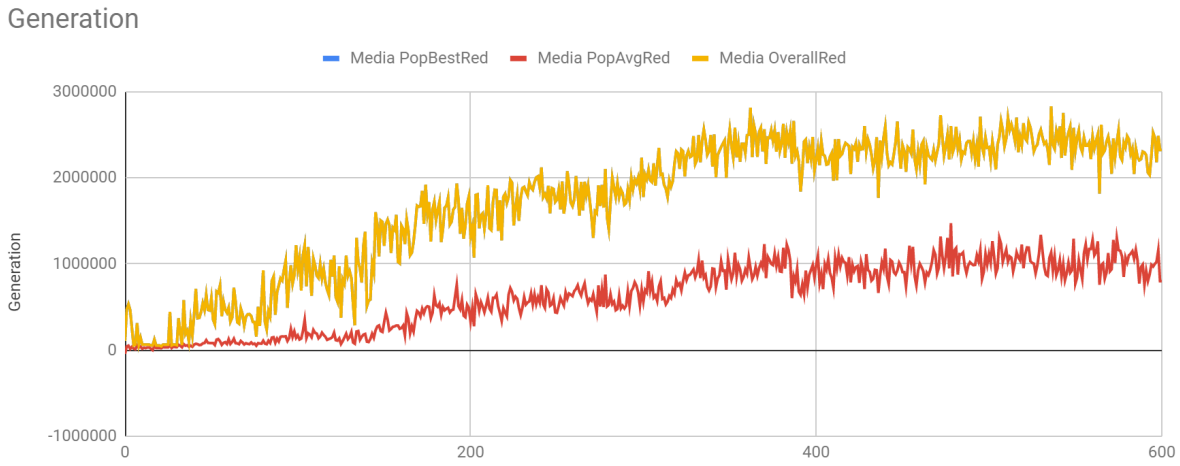
*distanceGoals = (float)(Math.Pow(2, distancefromBallToAdversaryGoal.Average()) * 10000.0f - Math.Pow(1 / 2, distanceToMyGoal.Average()) * 1000.0f);*

O maior valor para a função *fitness1* foi 4524566.333, com a seed 500, usando uma função gaussiana com um tamanho de torneio de 5, uma taxa de mutação de 0.1, uma taxa de crossover de 0.8, com o agente e a bola sempre a iniciar na mesma posição.

O maior valor para a função *fitness2* foi 2830184.667, com a seed 500, usando uma função gaussiana com um tamanho de torneio de 5, uma taxa de mutação de 0.1, uma taxa de crossover de 0.8, com o agente e a bola sempre a iniciar na mesma posição.



Agente azul



Agente vermelho

Análise de resultados

Evolving - ControlTheBallToAdversaryGoal

O objetivo deste mapa consistia em fazer com que o agente marcasse o maior número de golos que conseguisse. Para isso, foi necessário tirar partido dos inputs fornecidos no enunciado. Embora tenham sido disponibilizados 18 inputs, apenas recorremos à utilização de 4 para obter este resultado.

Uma vez que o objetivo deste cenário passa por tentar que o agente marque o máximo número de golos na baliza adversária, recorremos ao *distanceToBall* (sendo que para a sua utilização foi necessário recorrer ao método *Average()* uma vez que é uma lista de distâncias) pois necessitamos que o agente se aproxime da bola. Como o objetivo passa também pela necessidade que a bola e o agente se aproximem da baliza de modo a que este consiga marcar recorremos também ao *distancefromBallToAdversaryGoal* e ao *distanceToAdversaryGoal.*

De modo a que o agente não se aproxime apenas da baliza sem levar a bola com ele, recompensámos o agente pelo número de toques na bola *hitTheBall.*

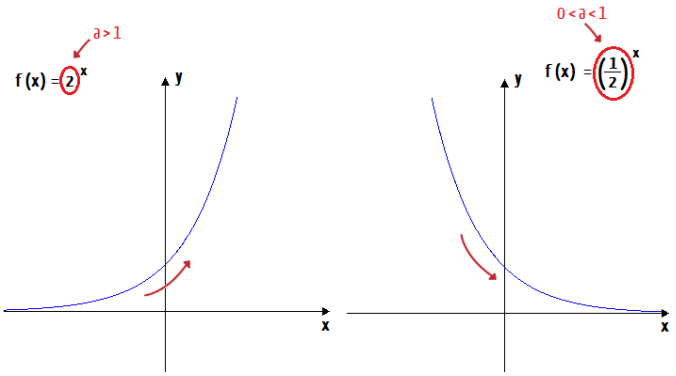
O facto da média da população se encontrar consideravelmente abaixo, algo que não esperávamos uma vez que com o avançar das gerações, esperava-se que o agente começasse a marcar em todas as gerações. Uma possível justificação para este acontecimento dever-se-á ao facto dos agentes sofrerem mutações o que, consequentemente, poderá levar um bom agente a perder as suas habilidades tornando-se semelhante a um dos agentes primários.

Evolving - Defense

O objetivo destes mapas é ensinar o robot D31 a impedir que a bola entre na sua própria baliza, a baliza de cor vermelha. Em *Evolving-Defense* temos a bola a partir sempre do centro e em *Evolving-DefenseRandom* temos a bola a partir de diferentes posições.

Para que o objetivo fosse alcançado, optámos por valorizar o input *hitTheBall* de forma a influenciar o robot a ir contra a bola, valorizámos também a *distanceTravelled* de forma a que o robot não fique parado. Desmotivamos cenários que haja autogolos (*GoalsOnMyGoal*) e que o robot fique sempre a ir contra parede (*hitTheWall*). Estimulámos, por fim, que uma vez que ele impeça a bola (GoalsOnMyGoal = 0) o robot leve a bola para a baliza adversária e que se distancie da própria baliza (*distanceGoals*).

Optámos por aplicar uma função exponencial para as distâncias das balizas de forma a experimentar funções fitness mais complexas que fugissem ao modelo linear. Além disso, uma vez que consigamos que o robot aprenda a defender, queremos que ele aprenda rápido a mandar a bola para longe da sua baliza.



modelos de funções exponenciais crescente e decrescente

Utilizámos a mesma função fitness em ambos os mapas de defesa, *Evolving-Defense* e *Evolving-DefenseRandom*, a fim de testar a qualidade da nossa função fitness em cenários não estáticos.

Evolving - OnevsOne

O objetivo deste mapa era simular um jogo de futebol entre os dois agentes. Para tal, definimos que um deles era o atacante, o agente azul, e outro, o vermelho, o defesa.
Com a mudança da função fitness de defesa referida em cima, verificámos que os agentes iam de encontro à bola, sendo que o azul, o atacante, chegava primeiro à bola e o vermelho, o defesa, defendia, como era esperado.
A função *fitness1* foi a que obteve melhores resultados no Evolving - ControlTheBallToAdversaryGoal, sem qualquer alteração, uma vez que o agente fazia o que era suposto.
A função *fitness2* é uma função simplificada da que obteve melhores resultados no Evolving - Defense, uma vez que com a introdução de um novo agente esta função fazia com que o agente saísse da baliza para um dos lados do campo, não fazendo o que era esperado. Esta simplificação fez com que o agente vermelho começasse a defender as bolas. Assim, adicionámos peso ao *hitTheBall* e distanceGoals e retirámos a condição if.

Os valores máximos das funções *fitness* não variam muito a partir de um certo ponto dado que de geração em geração é sempre mantido o melhor valor (elitista).

Conclusões

A realização deste trabalho tinha como objetivo treinar o nosso D31 para que o mesmo conseguisse controlar a bola, defender e atacar. Uma vez que é um projeto extenso, que leva a uma infinidade de combinações possíveis, sabemos que com a realização de ainda mais testes teríamos conseguido alcançar melhores resultados, especialmente para cenários random, uma vez que os mesmo precisam de >1000 gerações.

Com os testes supra mencionados, concluímos que obtivemos resultados aceitáveis para certos cenários, embora reconheçamos que o facto de terem sido realizados em máquinas diferentes tenham tido um pouco de influência na obtenção dos mesmos. No entanto, para que os resultados fossem melhores, com mais tempo, poderíamos ter testado com diferentes redes neuronais(adicionando mais camadas escondidas, etc), testado para mais funções fitness com diferentes pesos e ter feito mais testes para a bola e o agente a começar em posições aleatórias.

Anexos

Segue excel com os restantes testes:
https://docs.google.com/spreadsheets/d/1NPGWFgFZkuUHpa6sgvKUVYVmQ6e-Ze0txoHWyKY67JI/edit?usp=sharing