Trabalho Prático Nº2
D31 The Rise of the Ballz

Avram Gîncu - PL2
João André Gomes Marques - PL3
Rui Filipe da Silva Brandão - PL2

Faculdade de Ciências e Tecnologia, Universidade de Coimbra – Polo II, 3030.790
Coimbra, Portugal

uc2017278688@student.uc.pt
uc2017225818@student.uc.pt
uc2017270806@student.uc.pt

Introdução à Inteligência Artificial - 2º Semestre

1. Introdução

Este relatório tem como objetivo descrever os testes efetuados e a analisar os resultados obtidos. Os ambientes utilizados nas experiências foram os fornecidos pelos professores. A implementação dos algoritmos foi baseada no pseudocódigo presente no enunciado e nas recomendações dadas pelos docentes da cadeira. Assim sendo, foram testadas várias combinações de parâmetros e funções fitness. A análise feita irá incidir sobre a forma como o agente evoluiu e a maneira como o seu comportamento foi afetado quando alterados certos parâmetros.

2. Implementação

Pretende-se treinar as capacidades do agente atacar e defender uma bola num jogo de futebol, através da utilização de redes neuronais e algoritmos genéticos. A rede neuronal é constituída por 3 neurónios, recebe 18 parâmetros e devolve 2 (a intensidade da força e o ângulo de aplicação dessa força).

O algoritmo genético é um método de pesquisa heurística inspirado na evolução e seleção natural das espécies.

Começou-se por fazer um conjunto de testes para perceber os potenciais melhores parâmetros de forma individual, experimentando vários valores para o parâmetro em estudo e deixando os outros estáticos e usando sempre a mesma função fitness. Depois de se identificarem esses possíveis melhores valores fizeram-se combinações de parâmetros e compararam-se os resultados. Os melhores resultados foram usados para evoluir o robô nos mapas.

2.1. Tournament Selection

Inicialmente, é gerada uma população formada por um conjunto aleatório de indivíduos que podem ser vistos como possíveis soluções do problema. Durante o processo evolutivo, esta população é

avaliada: para cada indivíduo é dada uma nota, ou índice, refletindo sua habilidade de adaptação a determinado ambiente. O *Tournament Selection* vai determinar o melhor dos escolhidos e esse vai para a próxima geração, passando por um processo de modificação.

2.2. Mutation

O operador de mutação é necessário para a introdução e manutenção da diversidade genética da população, alterando arbitrariamente um ou mais componentes de uma estrutura escolhida, fornecendo assim meios para introdução de novos elementos na população. Desta forma, a mutação assegura que a probabilidade de se chegar a qualquer ponto do espaço de busca nunca será zero.

2.3. Crossover

O *Crossover* é o operador responsável pela recombinação de características dos pais durante a reprodução, permitindo que as próximas gerações herdem essas características. É necessário que o operador preserve as características para que os indivíduos sejam válidos para as próximas gerações. O que acontece é que a função escolhe um ponto no genótipo de um indivíduo e troca com outro indivíduo até esse ponto.

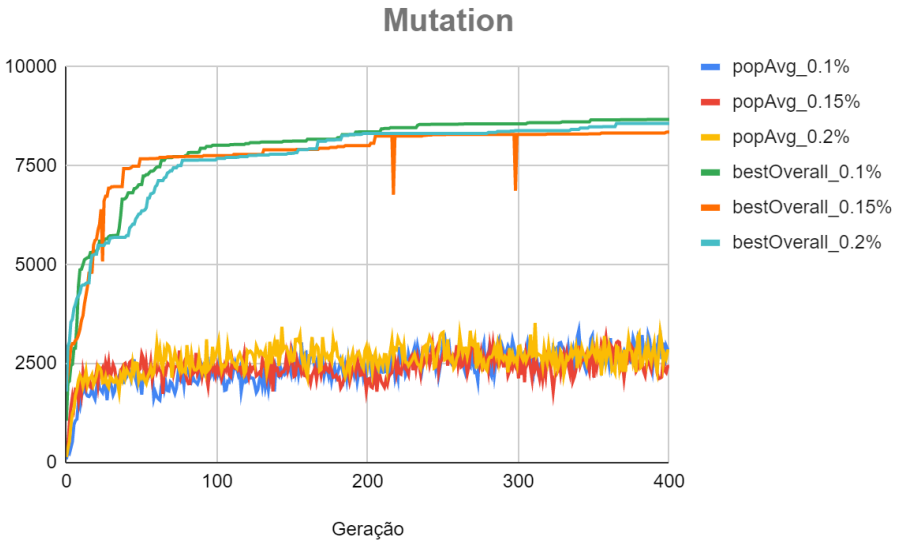
3. Modelação de parâmetros

Após implementar todo o código que faltava para que o algoritmo de evolução funcionasse, foram feitos testes que descrevem quais os melhores parâmetros a usar. O grupo decidiu focar cada parâmetro e testar diferentes valores para cada um de forma a definir quais os melhores números para o algoritmo genético. Para todos estes testes, foi utilizada uma única função de fitness, de forma a poder ser possível sintetizar e analisar todos os resultados corretamente. Também na análise de resultados, foram testados o *PopAverage*, sendo a média de pontuação de cada geração, e o *BestOverall*, sendo o último prioridade na escolha dos melhores resultados em relação aos outros. Todos os testes foram feitos com 400 gerações e realizados 5 vezes, com *RandomSeed's* distintas.

float fitness = distanceTravelled + (distanceToBall.Average() * -1) + (distanceToAdversaryGoal.Average() * -1) + distancefromBallToMyGoal.Average() + (distancefromBallToAdversaryGoal.Average() * -1) + (hitTheBall*10) + (GoalsOnAdversaryGoal*100) + ballSpeed.Average();

3.1. Mutation

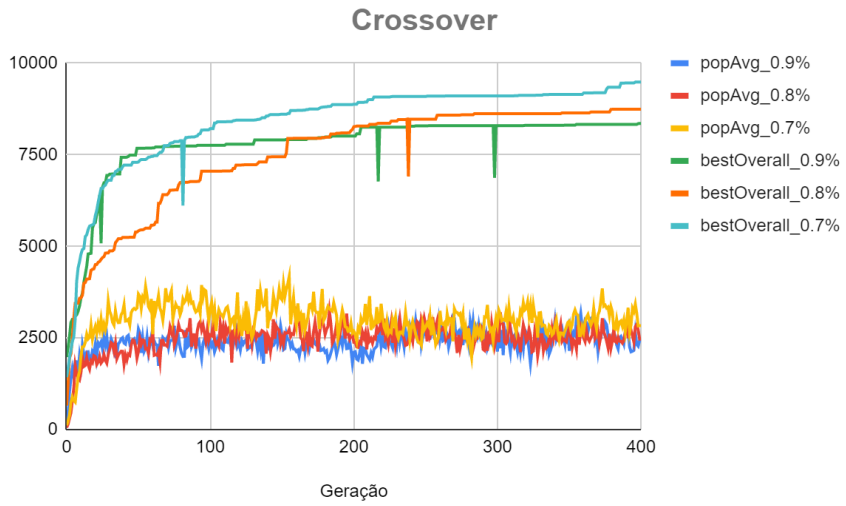
Para os testes de diferentes valores de probabilidade de *mutation*, foram testados percentagens de *0.1%*, *0.15%* e *0.2%*.



Apesar das linhas do *PopAverage* serem semelhantes, a percentagem de ***0.1%*** foi a que mais se destacou, pois conseguiu melhor resultado no *BestOverall*.

3.2. Crossover

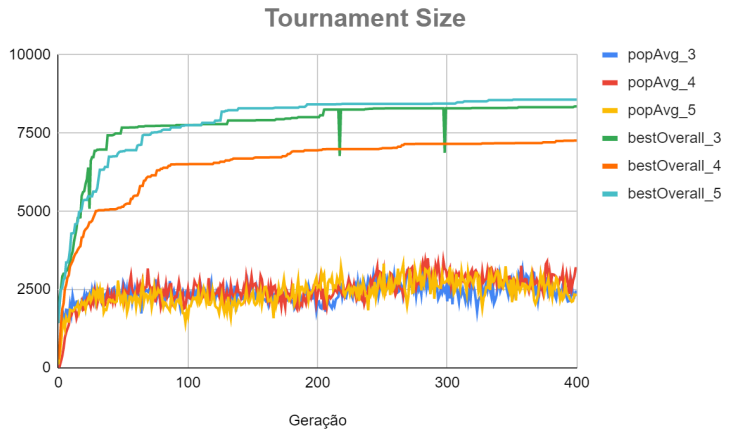
Para os testes de diferentes valores de probabilidade de *crossover*, foram testados percentagens de *0.7%*, *0.8%* e *0.9%*.



Com estes resultados, é visível que a probabilidade de *crossover* a ***0.7%*** é melhor que as outras duas, alcançando melhores resultados e mais rapidamente.

3.3. Tournament Size

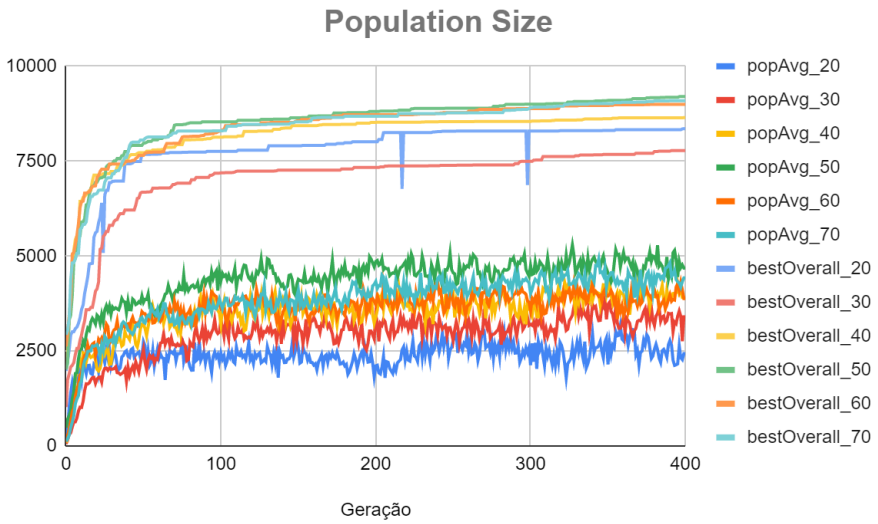
Na seleção de indivíduos de uma antiga geração para uma nova, foram definidos valores como *3*, *4* e *5*. Estes valores não podem ser muito altos, pois existe o problema de maximização do algoritmo evolucionário.



No *tournament size*, os melhores resultados foram obtidos com um valor de ***5***.

3.4. Population Size

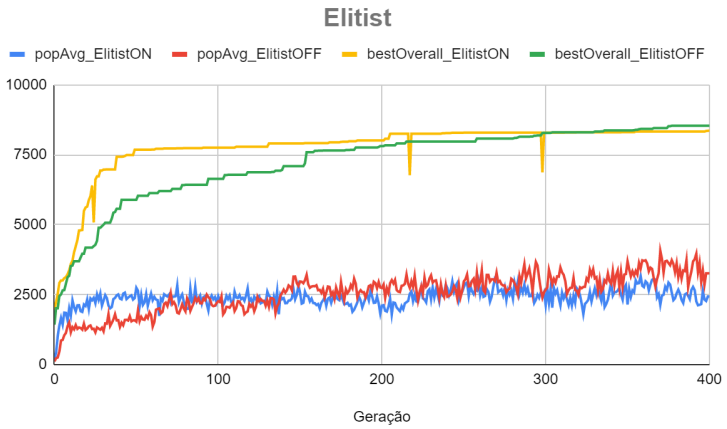
Na definição do número de indivíduos que cada geração tinha, foram feitos testes com populações de *20*, *30* e *40*. Mais tarde, foi delineado testar com valores mais altos, pois com uma população maior, existe uma maior probabilidade de conseguir o comportamento desejado da função fitness. Foram então com *50*, *60* e *70*.



Apesar de à partida se esperar que a população com *70* fosse a melhor, a população de valor ***50*** destacou-se mais, sendo então a escolhida para a próxima fase.

3.5. Elitist

Para os testes de elitista, foram só feitos dois, um em que o melhor indivíduo da geração é automaticamente escolhido para a próxima geração, e outro com esse parâmetro desligado.



Apesar de, com o Elitista ativado, se chegar a melhores resultados mais cedo, com ele **desativado** consegue-se um melhor resultado, tendo também uma curva de evolução mais gradual.

3.6. Resultado final dos primeiros testes

Com os primeiros testes finalizados, foram definidos os melhores valores para cada parâmetro.

Mutation = ***0.1%***
Crossover = ***0.7%***
Tournament Size = ***5***
Population Size = ***50***
Elitist = ***OFF***

4. Combinação de parâmetros

Após se testar e analisar os resultados obtidos anteriormente, o grupo pensou que já se podia avançar para a próxima fase de evolução dos controladores, de forma a resolver todos os mapas. Devido ao feedback dos docentes da disciplina percebeu-se que era ainda necessário fazer várias combinações de parâmetros e perceber qual era a melhor. Foram então feitos mais testes, desta vez para conseguir determinar qual a melhor combinação de parâmetros para o algoritmo genético.

Como o tempo já era escasso para poder realizar todas as combinações pedidas, e como já tinham sido realizados testes, estes foram aproveitados para fundamentar os próximos. Consequentemente, foram criadas várias situações de combinação para o *Mutation*, *Crossover*, *Tournament Size* e não para o *Population Size* e o *Elitist*, pois esses já teriam sido testados nos outros parâmetros, não sendo necessário repetir esses testes.

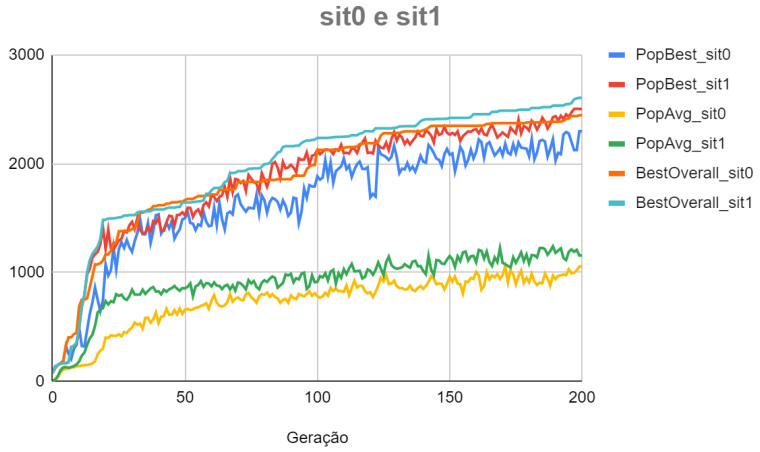
As gerações foram reduzidas de 400 para 200 e o tempo de cada jogo de 25 para 10 segundos de forma a que os testes fossem mais rápidos, tendo simplesmente o objetivo de comparar médias entre diferentes valores.

Nestas novas iterações, foram avaliados os dados de *PopBest*, *PopAverage* e *BestOverall*, tornando assim, uma análise mais completa das tendências de cada combinação. Foi, também, utilizada a mesma função de fitness, que foi utilizada para os testes anteriores.

4.1. Mutation

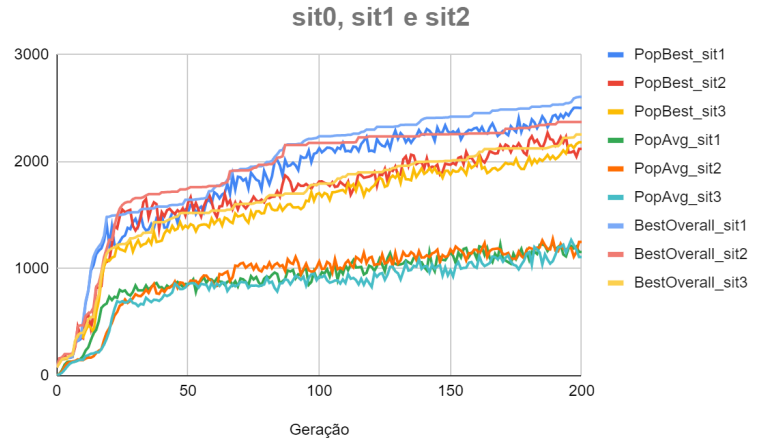
O grupo percebeu que os valores testados anteriormente, no caso de *Mutation* e *Crossover*, eram muito altos. Portanto, nestes novos testes, os melhores resultados foram comparados com valores mais baixos. Na probabilidade de *Mutation*, *0.1%* foi mantido no decorrer de todas as situações, mudando apenas os outros parâmetros. Após mudar os valores de *Crossover*, os dados foram comparados e definindo o melhor, modificando depois os valores de *Tournament Size*, *Population Size* e *Elitist*.

Situação 0: *Mutation = 0.1* ***Crossover = 0.7*** *TS = 5 Population = 50 Elitist = OFF*
Situação 1: *Mutation = 0.1* ***Crossover = 0*** *TS = 5 Population = 50 Elitist = OFF*



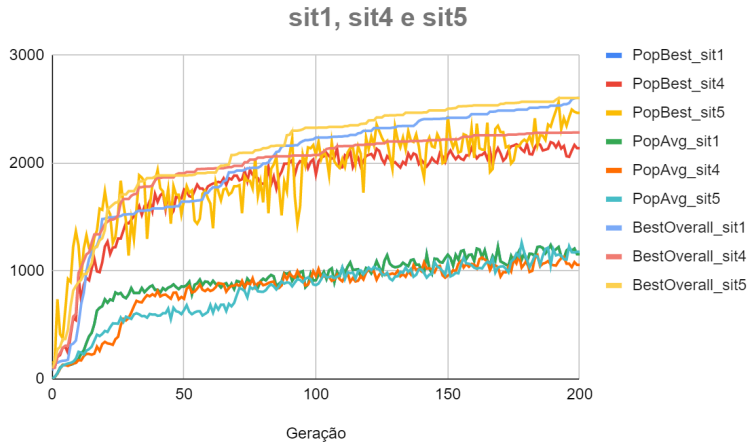
Com o gráfico é possível afirmar que a probabilidade de *Crossover* a **0** é melhor.

Situação 1: *Mutation = 0.1 Crossover = 0* ***TS = 5*** *Population = 50 Elitist = OFF*
Situação 2: *Mutation = 0.1 Crossover = 0* ***TS = 4*** *Population = 50 Elitist = OFF*
Situação 3: *Mutation = 0.1 Crossover = 0* ***TS = 3*** *Population = 50 Elitist = OFF*



Nas situações 1, 2 e 3, a primeira foi quem se destacou mais, definindo que o melhor valor para o *Tournament Size* seja **5**.

Situação 1: *Mutation = 0.1 Crossover = 0 TS = 5* ***Population = 50*** *Elitist = OFF*
Situação 4: *Mutation = 0.1 Crossover = 0 TS = 5* ***Population = 60*** *Elitist = OFF*
Situação 5: *Mutation = 0.1 Crossover = 0 TS = 5* ***Population = 70*** *Elitist = OFF*



O teste com mais população, neste caso de **70**, conseguiu melhores resultados.

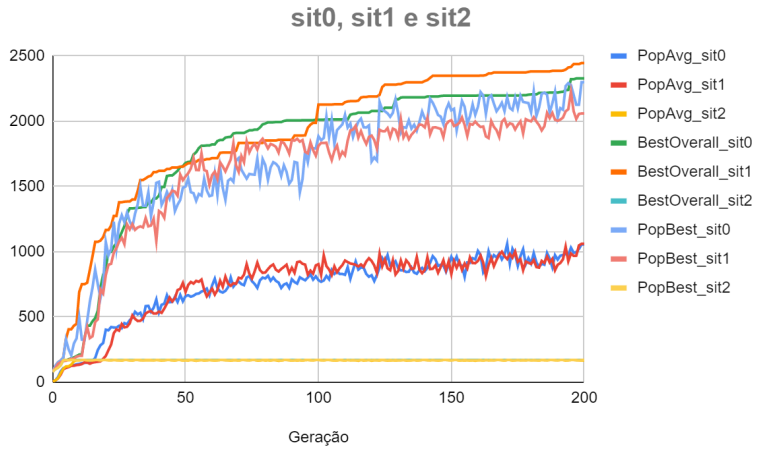
Situação 5: *Mutation = 0.1 Crossover = 0 TS = 5 Population = 70* ***Elitist = OFF***
Situação 6: *Mutation = 0.1 Crossover = 0 TS = 5 Population = 70* ***Elitist = ON***

Os melhores resultados aparecem quando o *Elitist* está **ativado**, tendo tendências mais estáveis e graduais.

Melhor Situação: *Mutation = 0.1 Crossover = 0 TS = 5 Population = 70 Elitist = ON*

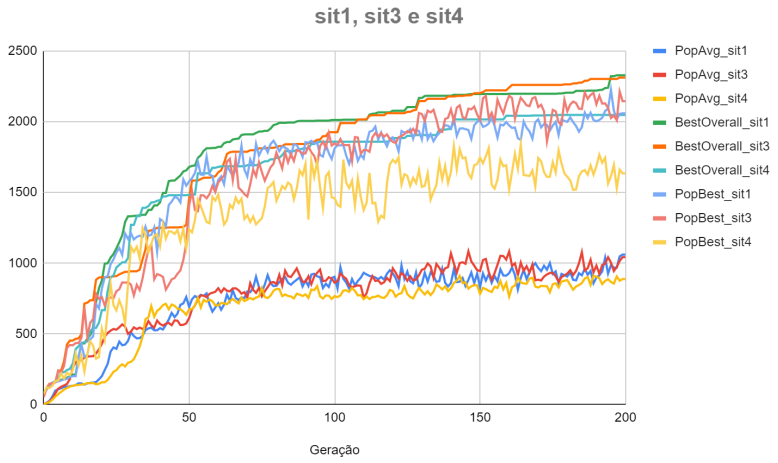
4.2. Crossover

Situação 0: Mutation = 0.1 *Crossover = 0.7 TS = 5 Population = 50 Elitist = OFF*
Situação 1: Mutation = 0.05 *Crossover = 0.7 TS = 5 Population = 50 Elitist = OFF*
Situação 2: Mutation = 0 *Crossover = 0.7 TS = 5 Population = 50 Elitist = OFF*



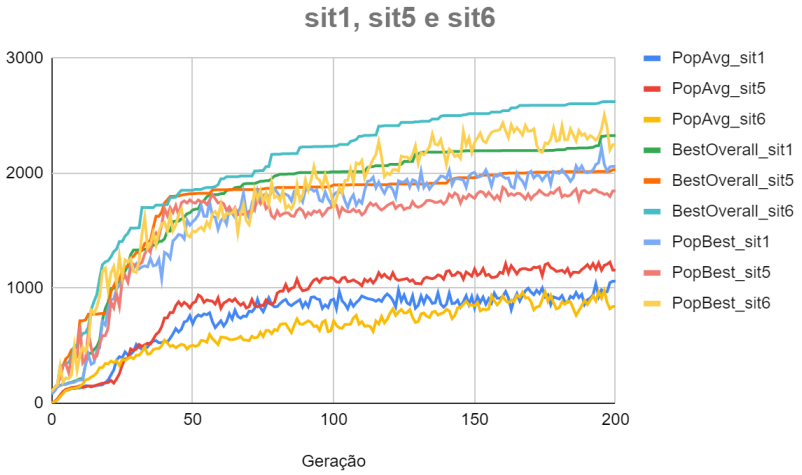
O gráfico mostra que a probabilidade de *Mutation* a **0.05%** é a melhor. Também é notável que o *PopAverage* da *Mutation* a 0% estagnou desde muito cedo.

Situação 1: *Mutation = 0.05 Crossover = 0.7* ***TS = 5*** *Population = 50 Elitist = OFF*
Situação 3: *Mutation = 0.05 Crossover = 0.7* ***TS = 4*** *Population = 50 Elitist = OFF*
Situação 4: *Mutation = 0.05 Crossover = 0.7* ***TS = 3*** *Population = 50 Elitist = OFF*



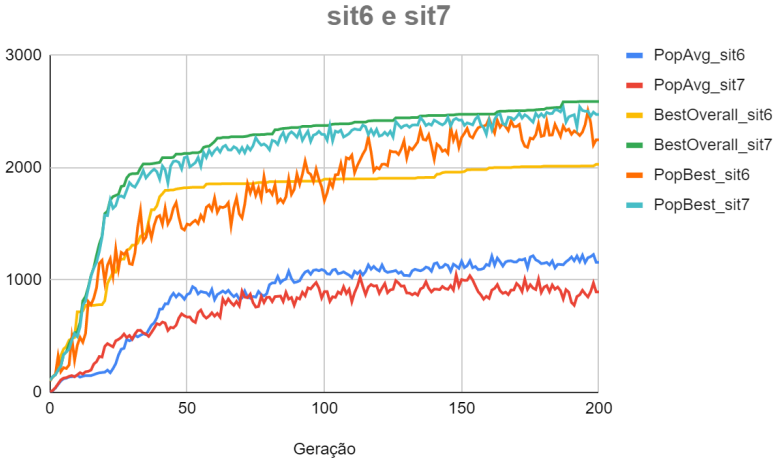
O *Tournament Size* a **5** mostra ser melhor, apesar de conseguir o melhor resultado de *BestOverall* só nas gerações mais finais, tendo um pico muito mais cedo que os outros.

Situação 1: *Mutation = 0.05 Crossover = 0.7 TS = 5* ***Population = 50*** *Elitist = OFF*
Situação 5: *Mutation = 0.05 Crossover = 0.7 TS = 5* ***Population = 60*** *Elitist = OFF*
Situação 6: *Mutation = 0.05 Crossover = 0.7 TS = 5* ***Population = 70*** *Elitist = OFF*



Como foi concluído nos testes de *Mutation*, o mesmo se passa nesta iteração, sendo a população com **70** a que tem melhores resultados.

Situação 6: *Mutation = 0.05 Crossover = 0.7 TS = 5 Population = 70* ***Elitist = OFF***
Situação 7: *Mutation = 0.05 Crossover = 0.7 TS = 5 Population = 70* ***Elitist = ON***

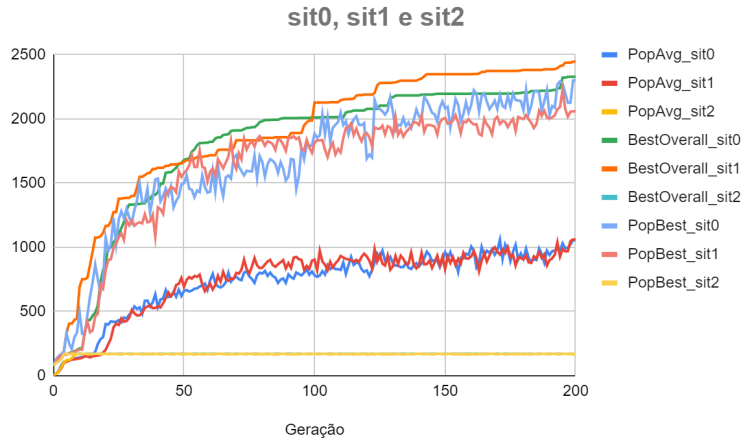


Com este gráfico é possível determinar que com o *Elitist* **ativado** aparece melhores resultados.

Melhor situação: *Mutation = 0.05 Crossover = 0.7 TS = 5 Population = 70 Elitist = ON*

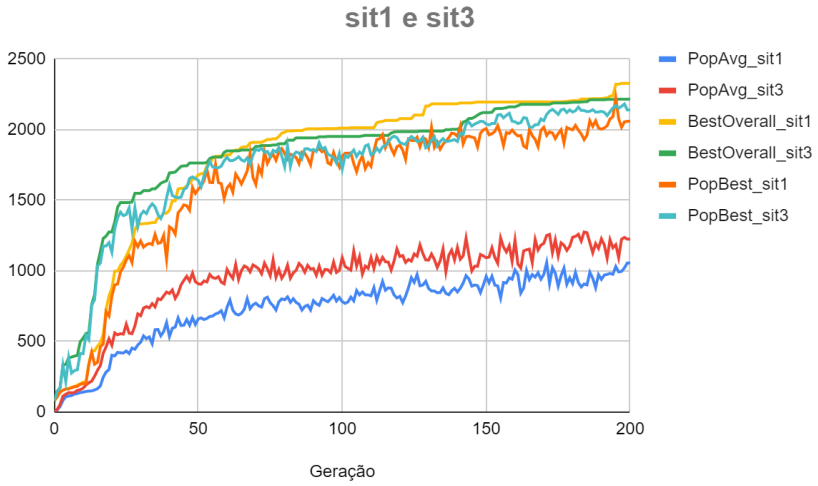
4.3. Tournament Size

Situação 0: Mutation = 0.1 *Crossover = 0.7 TS = 5 Population = 50 Elitist = OFF*
Situação 1: Mutation = 0.05 *Crossover = 0.7 TS = 5 Population = 50 Elitist = OFF*
Situação 2: Mutation = 0 *Crossover = 0.7 TS = 5 Population = 50 Elitist = OFF*



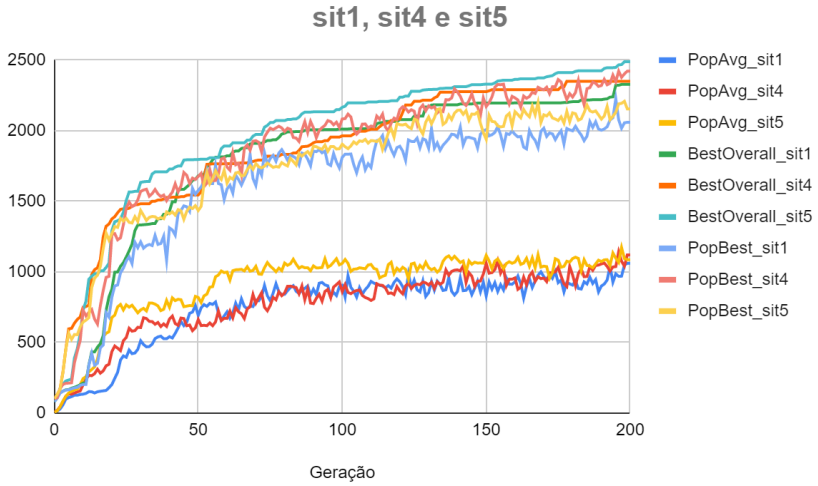
O gráfico mostra que a probabilidade de *Mutation* a **0.05%** é a melhor.

Situação 1: *Mutation = 0.05* ***Crossover = 0.7*** *TS = 5 Population = 50 Elitist = OFF*
Situação 2: *Mutation = 0.05* ***Crossover = 0*** *TS = 5 Population = 50 Elitist = OFF*



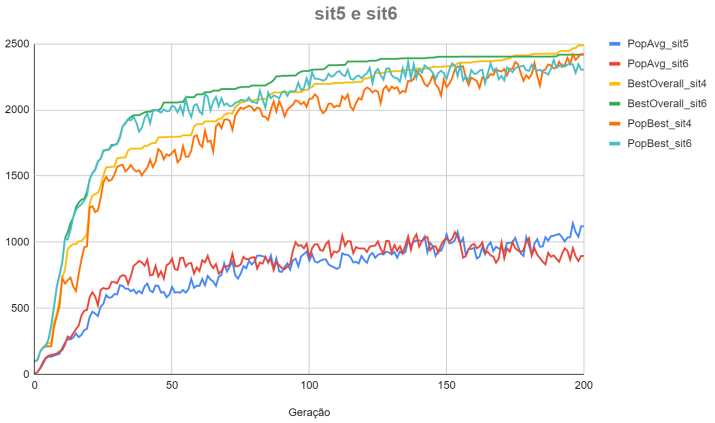
Com estes dados é possível determinar que a probabilidade de *Crossover* a **0.7%** é melhor.

Situação 1: *Mutation = 0.05 Crossover = 0.7 TS = 5* ***Population = 50*** *Elitist = OFF*
Situação 4: *Mutation = 0.05 Crossover = 0.7 TS = 5* ***Population = 60*** *Elitist = OFF*
Situação 5: *Mutation = 0.05 Crossover = 0.7 TS = 5* ***Population = 70*** *Elitist = OFF*



Após a análise dos resultados obtidos, a população com **70** indivíduos é a combinação que se destacou mais.

Situação 5: *Mutation = 0.05 Crossover = 0.7 TS = 5 Population = 70* ***Elitist = OFF***
Situação 6: *Mutation = 0.05 Crossover = 0.7 TS = 5 Population = 70* ***Elitist = ON***



Apesar de, tendo o *Elitist* desativado, se chegar a um resultado melhor, com ele **ativado** atinge melhores resultados mais cedo, sendo essa a prioridade.

Melhor situação: *Mutation = 0.05 Crossover = 0.7 TS = 5 Population = 70 Elitist = ON*

4.4. Melhores Resultados

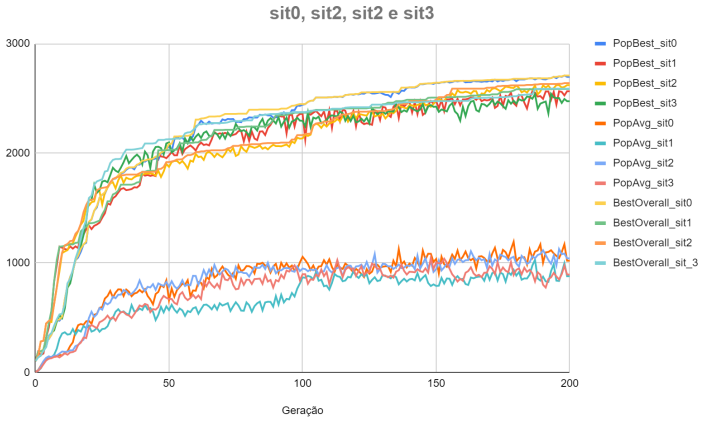
Tendo os melhores resultados de cada um dos parâmetros, vai-se agora comparar os 3, havendo também alguns testes onde se variou os resultados que deram diferentes.

Melhor de Mutation:
Mutation = 0.1 Crossover = 0 TS = 5 Population = 70 Elitist = ON
Melhor de Crossover:
Mutation = 0.05 Crossover = 0.7 TS = 5 Population = 70 Elitist = ON
Melhor de Tournament:
Mutation = 0.05 Crossover = 0.7 TS = 5 Population = 70 Elitist = ON

Primeira variação:
Mutation = 0.05 Crossover = 0 TS = 5 Population = 70 Elitist = ON
Segunda variação:
Mutation = 0.1 Crossover = 0.7 TS = 5 Population = 70 Elitist = ON

O objetivo é comparar as 4 situações, sendo que a situação de *Tournament Size* é igual à de *Crossover*.

Primeira variação *= situação 0*
Segunda variação *= situação 1*
Melhor de Mutation *= situação 2*
Melhor de Crossover e Tournament Size *= situação 3*



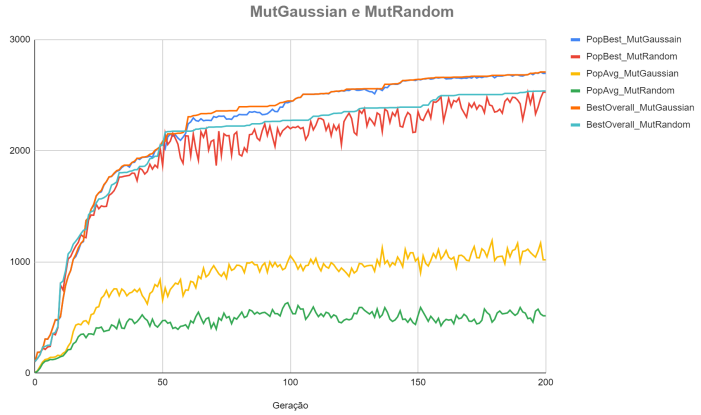
Com o gráfico é possível determinar que a melhor situação é a **0**. Apesar de possuir uma probabilidade de *Crossover* a 0, esta combinação consegue ser melhor que as outras.

Melhor situação: *Mutation = 0.05 Crossover = 0 TS = 5 Population = 70 Elitist = ON*

Estes parâmetros vão ser utilizados para a evolução dos agentes.

4.5. Mutation Random

Com a melhor combinação de parâmetros, foi testado novamente só que com a *Mutation* a *Random*, e não *Gaussian*.



Podemos assim determinar que a *Mutation* **Gaussian** tem melhores resultados.

5. Resolução dos mapas

5.1. Evolving - ControlTheBallToAdversaryGoal

Para a resolução deste primeiro ambiente evolucionário, o grupo decidiu criar 3 funções de fitness diferentes, testá-las no HillClimber com 200 gerações para perceber que comportamentos o agente iria ter e compará-las posteriormente.

Primeira função:

float fitness = float fitness = distanceTravelled + (distanceToBall.Average() * -1) + (distanceToAdversaryGoal.Average() * -1) + distancefromBallToMyGoal.Average() + (distancefromBallToAdversaryGoal.Average() * -1) + (hitTheBall*10) + (GoalsOnAdversaryGoal*100) + ballSpeed.Average() + ((GoalsOnMyGoal*-1)*200) + ((hitTheWall*-1)*10);

Segunda função:

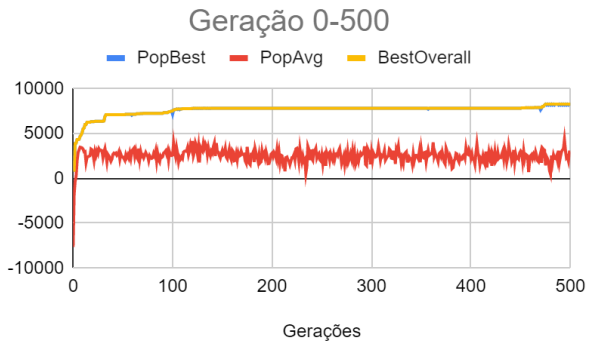
float fitness = 1000*GoalsOnAdversaryGoal + distancefromBallToMyGoal.Average() - distanceToBall.Average() + 100*hitTheBall;

Terceira função:

float fitness = distanceTravelled + (distanceToBall.Average() * -1 * 100) + (distanceToAdversaryGoal.Average() * -1 * 500) + distancefromBallToMyGoal.Average() + (distancefromBallToAdversaryGoal.Average() * -1 * 100) + (hitTheBall * 100) + (GoalsOnAdversaryGoal * 1000) + ((GoalsOnMyGoal * -1) * 500) + (hitTheWall * -1 * 100) + ballSpeed.Average();

Após analisar as 3 funções no primeiro cenário, foi escolhida a **função 1**.

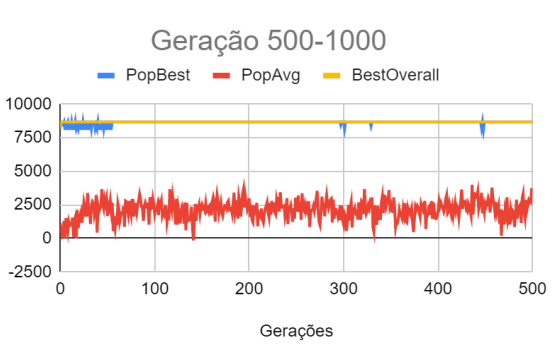
Durante os testes foi-se alterando certos parâmetros da equação, que após algumas modificações acabou por ficar assim:

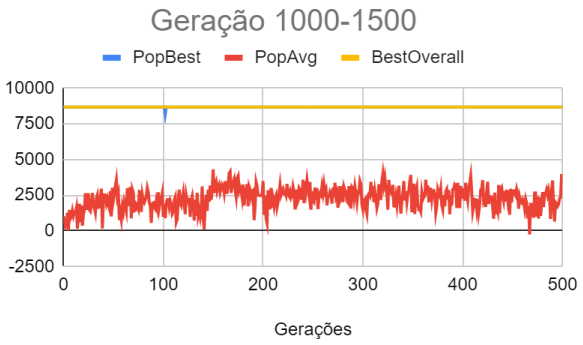


Observações: O agente começa a marcar golos com frequência, mas há certas instâncias em que leva a bola até a baliza inimiga e para, ou a deixa ali e segue sozinho. Optou-s por alterar o peso de alguns parâmetros na equação.

Função Final:

*float fitness = distanceTravelled + (distanceToBall.Average() * -1) + (distanceToAdversaryGoal.Average() * -1) + distancefromBallToMyGoal.Average() + ((distancefromBallToAdversaryGoal.Average() * -1) *10)+ (hitTheBall*10) + (GoalsOnAdversaryGoal*500) + ((GoalsOnMyGoal*-1)*200) + ((hitTheWall*-1)*10);*





Observações: Com esta função o agente já marcava golos na baliza inimiga com frequência por volta das 1000 gerações. A partir da geração 1000, não há mudança no comportamento do agente, por isso optou-se por não evoluir mais pois já marcava golos.

5.2. Evolving - ControlTheBallToAdversaryGoalRandom

Para resolver este mapa optou-se por usar a mesma função de ataque anterior, e depois adaptá-la consoante os resultados dos testes.

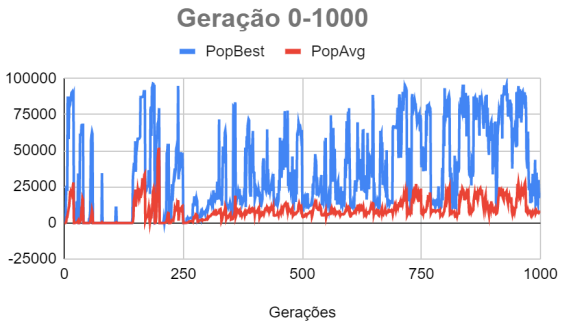
Função para o mapa random:

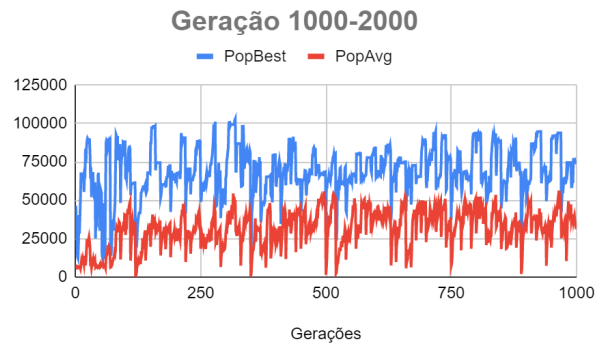
float fitness = float fitness = distanceTravelled + (distanceToBall.Average() * -1) + (distanceToAdversaryGoal.Average() * -1) + distancefromBallToMyGoal.Average() + (distancefromBallToAdversaryGoal.Average() * -1) + (hitTheBall*10) + (GoalsOnAdversaryGoal*100) + ballSpeed.Average() + ((GoalsOnMyGoal*-1)*200) + ((hitTheWall*-1)*10);

Observações: Após 1000 gerações da primeira equação, o agente marca de vez em quando autogolo, e leva a bola muito perto da baliza dele. Ele dribla a bola muito bem pelo campo, mas poucas são as situações em que marca golo.

Função Final:

float fitness = ((distanceToBall.Average() * -1) * 200) + (distancefromBallToMyGoal.Average() * 100) + ((distancefromBallToAdversaryGoal.Average() * -1) * 500) + (hitTheBall * 100) + (GoalsOnAdversaryGoal * 1000) + ((GoalsOnMyGoal * -1) * 500);





Observações: Após 1000 gerações o agente marca golo algumas vezes, outras ele vai contra a bola mas não chega a levá-la para a baliza, e certas vezes marca auto golo.

5.3. Evolving - Defense

Para a resolução do terceiro ambiente evolucionário, o grupo decidiu criar 3 funções de fitness diferentes, testá-las no HillClimber com 200 gerações para perceber que comportamentos o agente iria ter e compará-las posteriormente.

Primeira função:

float fitness = ((distanceToBall.Average() * -1)*50) + (distancefromBallToMyGoal.Average() * 100) + (hitTheBall * 100) + ((GoalsOnMyGoal * -1) * 500) + agentSpeed.Average()*10;

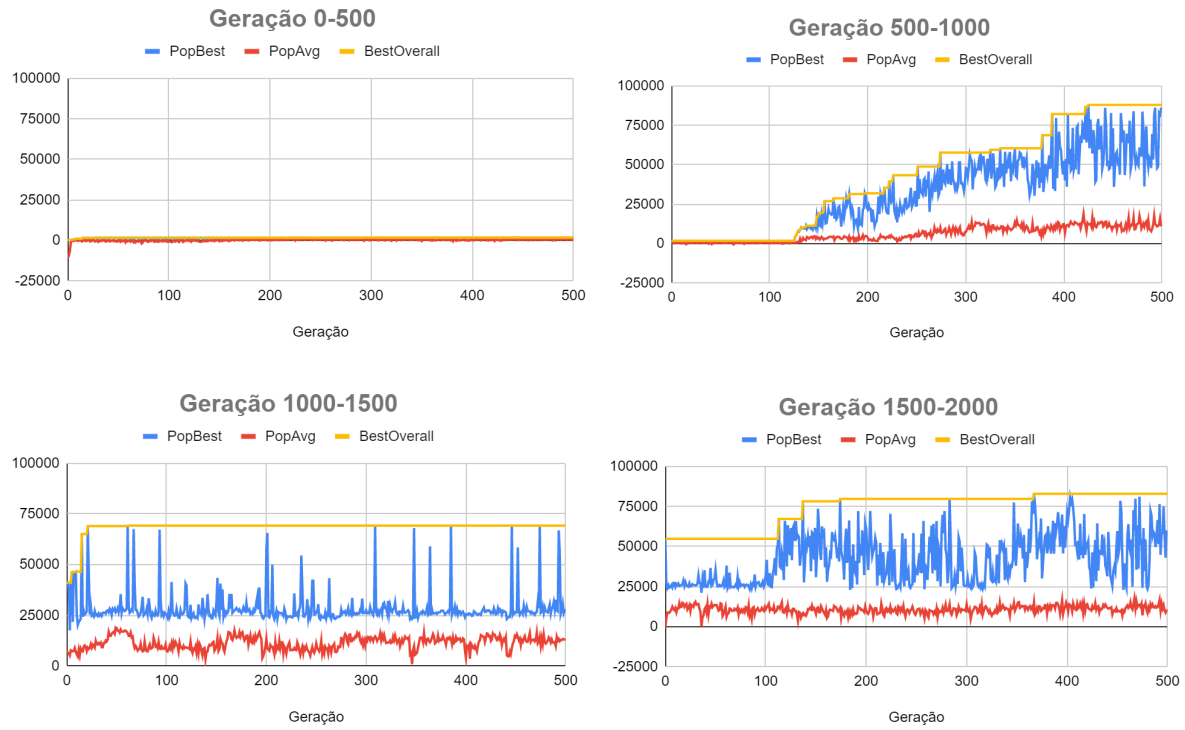
Segunda função:

float fitness = -1000*GoalsOnMyGoal + 100*distancefromBallToMyGoal.Average() – 10*distanceToBall.Average() + distanceTravelled + 100*hitTheBall;

Terceira função:

float fitness = distanceTravelled + (distanceToBall.Average() * -1 * 100) + (distanceToAdversaryGoal.Average() * -1 * 500) + distancefromBallToMyGoal.Average() + (distancefromBallToAdversaryGoal.Average() * -1 * 100) + (hitTheBall * 100) + (GoalsOnAdversaryGoal * 1000) + ((GoalsOnMyGoal * -1) * 500) + (hitTheWall * -1 * 100) + ballSpeed.Average();

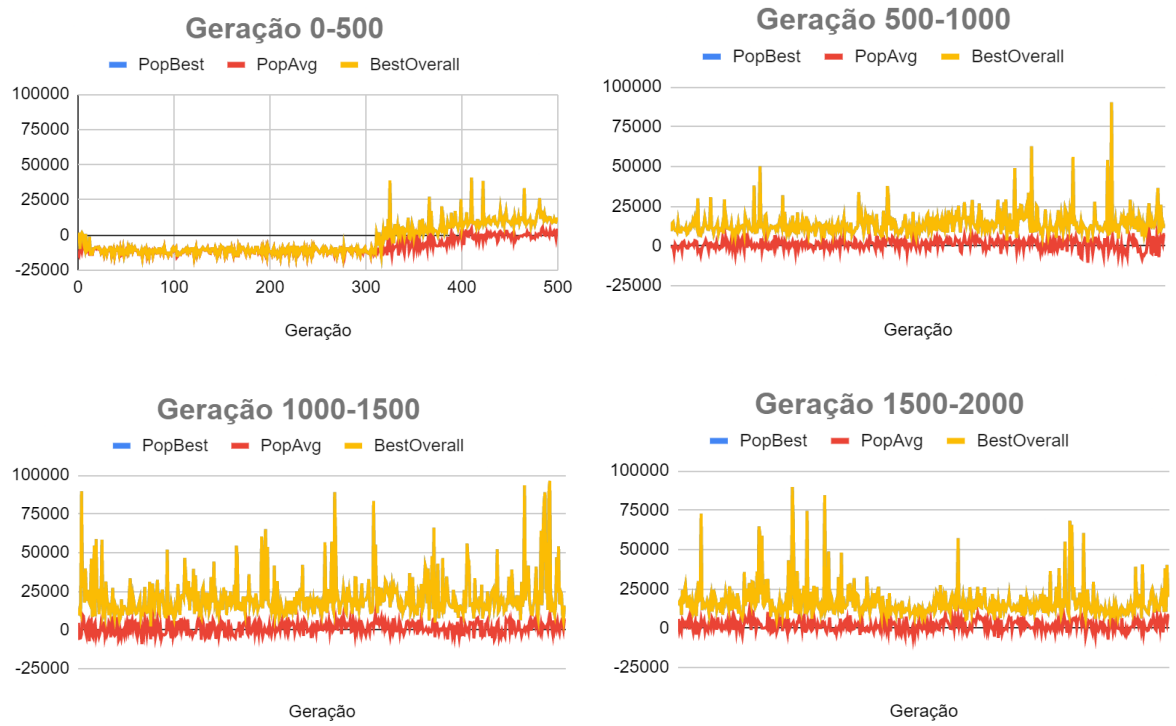
Após analisar as 3 funções no primeiro cenário, foi escolhida a **função 2**.



Observações: Nas primeiras 500 gerações, os agentes ainda sofrem golos com bastante frequência. Da geração 500 à 1000, os agentes já começam a aprender melhor como defender a e há uma evolução notória no seu comportamento de defesa. Na geração 1000 a 1500 o comportamento dos agentes não é muito distinto e não há uma evolução muito significativa, de notar que os agentes começam a levar a bola para um dos cantos e mantêm-se lá. Na geração 1500 a 2000 o comportamento de defesa da e de levar a bola para um dos cantos é mais notório a o agente evolui mais nesse sentido.

5.4. Evolving - DefenseRandom

A função usada para este cenário foi a melhor função encontrada no cenário anterior (função 2).



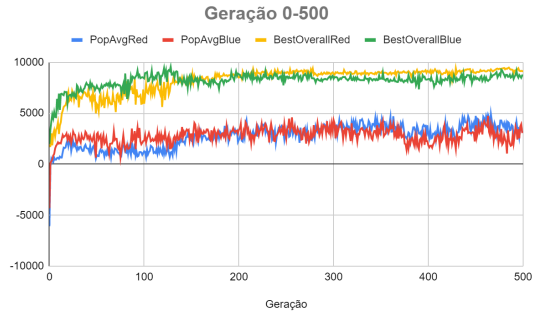
Observações: Os agentes tiveram uma evolução visível a partir da geração 300 e o seu comportamento não alterou muito daí em diante. Isto pode-se explicar devido à natureza do mapa e da posição da bola e também devido à probabilidade usada no crossover. Para contornar este problema foi mudada a taxa do crossover para 0.7 e foram feitas mais 1000 gerações.

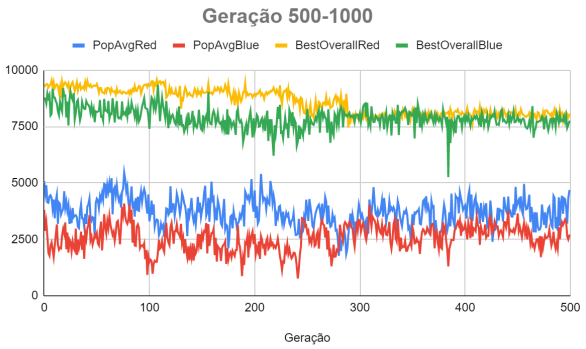
5.5. Evolving - OnevsOne

Para a evolução do cenário co-evolucionário, o grupo decidiu experimentar 3 situações distintas, em que a primeira utilizaria a função de ataque que serviu de solução para o primeiro mapa. A segunda situação seria utilizar a função de defesa contra a mesma. Numa terceira situação iria estar a função de defesa contra a de ataque de forma a perceber qual seria melhor. Mais tarde o grupo decidiu experimentar do 0 a função de defesa contra de ataque, só que desta vez os agentes iriam ter posições random e a bola também.

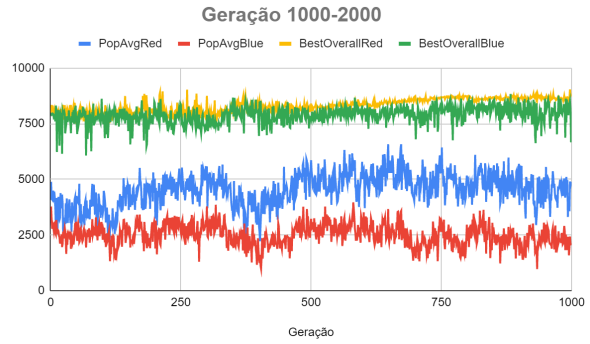
Primeira Situação: Ataque vs Ataque

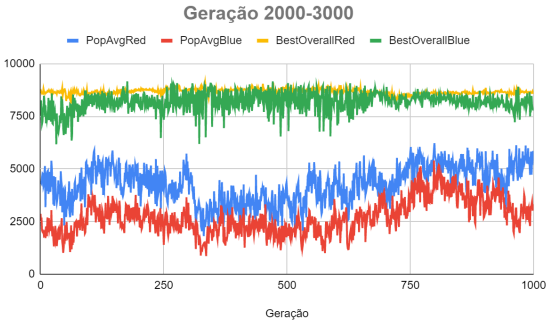
float fitness = distanceTravelled + (distanceToBall.Average() * -1) + (distanceToAdversaryGoal.Average() * -1) + distancefromBallToMyGoal.Average() + (distancefromBallToAdversaryGoal.Average() * -1) + (hitTheBall*10) + (GoalsOnAdversaryGoal*100) + ballSpeed.Average() + ((GoalsOnMyGoal*-1)*200) + ((hitTheWall*-1)*10);





Observações: Os agentes vão um contra o outro e normalmente quem consegue ficar atrás da bola consegue marcar golo. Existem algumas iterações em que os indivíduos se esquecem da bola e estão os dois perdidos no mapa. A partir da geração 100 nota-se que ambos os agentes se estabilizaram, mas a partir da geração 700 existe uma alternância entre os dois, sendo que no mapa os dois disputam a bola muito arduamente.

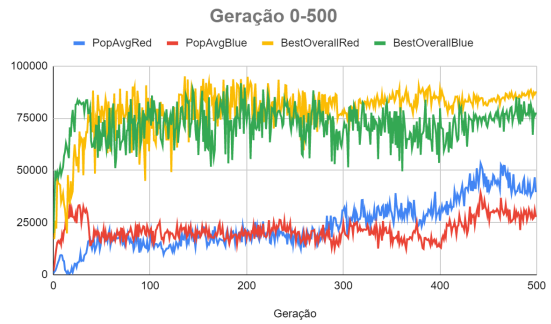


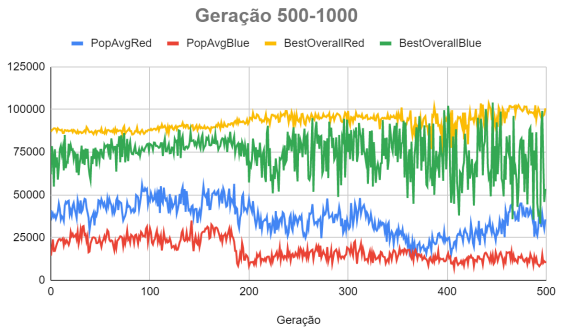


Observações: Ao fim de 3000 gerações chega-se à conclusão que tendo a mesma função de fitness para ambos os indivíduos (o azul e o vermelho) não conseguem criar uma história para um jogo, isto é, vão os dois à bola e estão parados a fazer força um contra o outro, fazendo com que a bola nem saia do meio campo.

Segunda Situação: Defesa vs Defesa

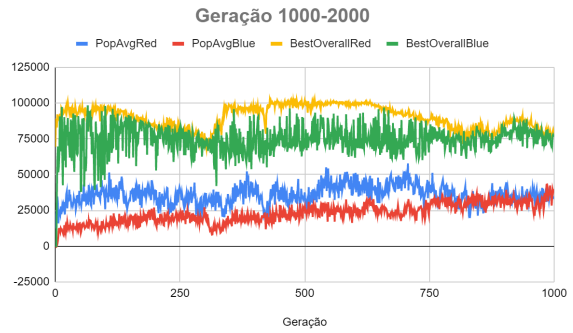
float fitness = -1000*GoalsOnMyGoal + 100*distancefromBallToMyGoal.Average() – 10*distanceToBall.Average() + distanceTravelled + 100*hitTheBall;

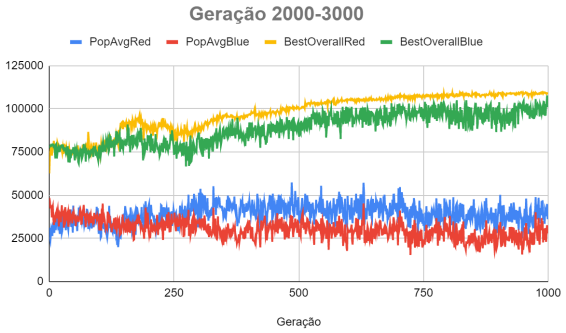




Observações: Os dois indivíduos vão um contra o outro, mas o azul normalmente tem um comportamento mais agressivo, o vermelho consegue ter mais pontos, pois primeiro defende e

começa a atacar. Estes comportamentos vão mudando pois o vermelho consegue também ser mais agressivo e chega primeiro à bola.

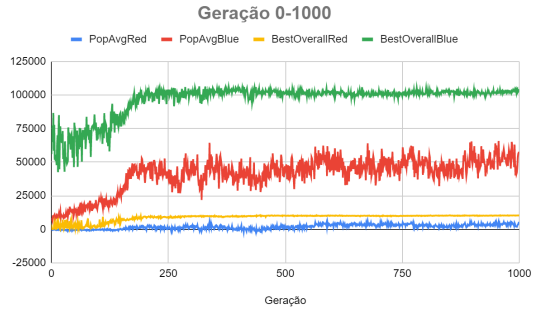


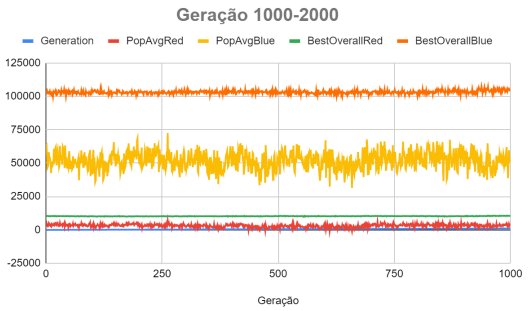


Observações: Também aqui, ao fim de 3000 gerações, chega-se à conclusão que tendo a mesma função de fitness para ambos os indivíduos (o azul e o vermelho) eles vão os dois à bola e estão parados a fazer força um contra o outro.

Terceira Situação: Ataque vs Defesa

Para esta situação, foram selecionados os melhores das situações acima descritas, e então evoluí-los a partir das 3000 gerações. O azul utilizou a função de defesa e o vermelho a função de ataque.

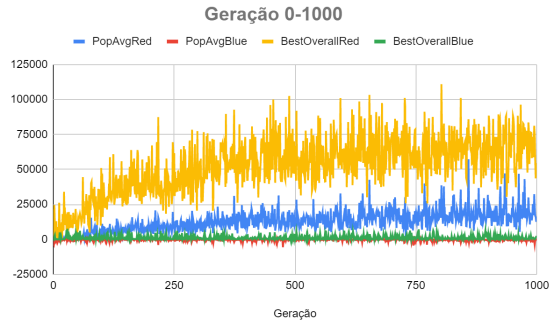


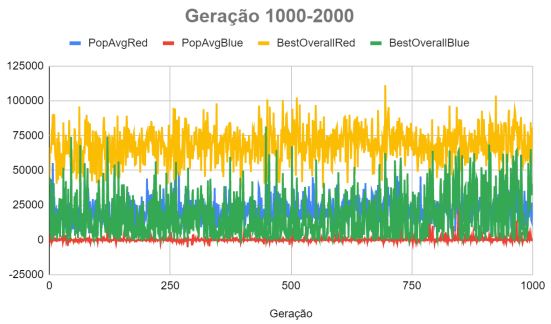


Observações: A evolução desta situação foi um bocado precária, pois os agentes já iam contra a bola e ficavam um contra o outro nas outras situações. Nesta situação, o comportamento deles foi parecido mais para o final, levando a bola desta vez para um canto. No início, o agente com a função de defesa conseguiu marcar mais golos, sendo determinado que essa função era melhor.

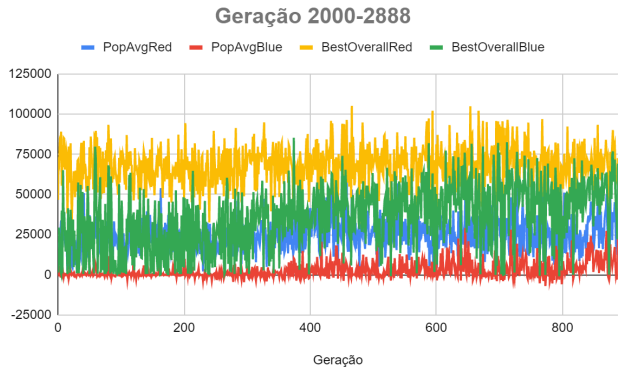
Quarta Situação: Ataque vs Defesa // Bola Random // Posição Random

Nesta situação foi decidido começar do zero, com as mesmas funções de ataque e defesa, mas desta vez atribuiu-se posições random aos indivíduos e à bola. Com isto, o grupo procurou explorar mais os potenciais de cada função.

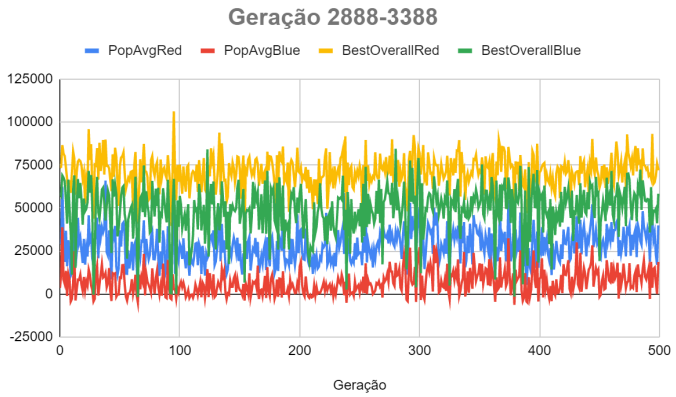




Observações: O peso do *distanceToBall* foi aumentado para que os indivíduos procurem a bola, para não andarem perdidos no mapa. A partir da geração 100, nota-se que o vermelho, com a função de defesa, têm mais facilidades em ir para a bola, mas marca muitos golos na própria baliza. O agente com a função de ataque continua muito perdido mesmo após 500 gerações.



Observações: No geral havia boas iterações. O vermelho percorria muito o mapa enquanto o azul era mais controlado e perspicaz. Muitas vezes o indivíduo vermelho marcava golos na própria, mas depois marcava golo na baliza adversária, outras vezes marcava dois golos na baliza do azul. Porém, o azul quase nunca marcava golos.
Para um último teste, o grupo decidiu aplicar o Crossover a 0.7%. Foi sentido que a falta de Crossover leva a que um agente com várias gerações não consiga aplicar no seu genótipo as várias situações que podem acontecer no cenário. Então a partir da geração 2888 foi introduzido o Crossover para ver se o agente ficava mais responsivo.



Observações: Neste teste é notável que o agente conseguiu ficar mais responsivo, mesmo apenas tendo feito 500 gerações, mas fica a sensação que para ficar mesmo bem otimizado teria que começar do 0, não havendo histórico de iterações até então.

6. Conclusão

Graças aos testes preliminares, conseguiu-se decidir qual os melhores parâmetros para evoluir o agente para cada mapa. Os testes foram inúmeros, pois se teve que testar cada parâmetro e combiná-los uns com os outros.

Durante os testes de evolução, descobriu-se que *distanceToBall, GoalsOnAdversaryGoal, distancefromBallToAdversaryGoal, hitTheBall e GoalsOnMyGoal,* foram parâmetros chave para construir uma boa base para ter uma equação sólida, motivo pelo qual se encontram em todas as equações feitas, e com pesos mais elevados.

Após os testes preliminares, o crossover 0 provou ser o melhor, mas ao evoluir nos mapas random (defesa/ataque), o agente não chegava a evoluir de forma a resolver o mapa como esperado, por isso optou-se por meter o crossover a 0.7, o que deu resultados mais positivos e dinâmicos.

7. Webgrafia

https://en.wikipedia.org/wiki/Genetic_algorithm
https://stackoverflow.com/questions/31933784/tournament-selection-in-genetic-algorithm
https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)
https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm)
https://en.wikipedia.org/wiki/Chromosome_(genetic_algorithm)