UNIVERSIDADE DE COIMBRA

FIA TP2 PL1 – Relatório Meta 2

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

2019216764 – João Monteiro – uc2019216764@student.uc.pt

2019216747 – João Melo – uc2019216747@student.uc.pt

2019216809 – Miguel Faria – uc2019216809@student.uc.pt

Índice

Introdução

A teoria do evolucionismo de Charles Darwin, na qual é afirmado que a sobrevivência das espécies está relacionada com sua seleção natural, é premissa que serve como fundamento para algoritmos de evolução. Neste projeto, com recurso a ferramenta *Unity*, diversos veículos irão ser instrumento de estudo para a observação da evolução dos mesmos.

Na primeira meta, está presente todo o processo de implementação e uma primeira função *fitness*.

Na segunda meta, encontram-se realizados um conjunto alargado de testes, em ambos os cenários, com o intuito de encontrar os melhores parâmetros de evolução, função de *fitness* e veículos.

Meta 1

Nesta meta foi efetuada toda a implementação necessária ao bom funcionamento do algoritmo, recorrendo ao pseudocódigo fornecido e implementada uma função de fitness adequada, fruto de vários testes feitos no ambiente de simulação do cenário *GapRoad*. Essa função de fitness inicial servirá como ponto de partida para os diversos cenários a realizar.

Funções Implementadas

Algoritmo 1 – Single Point Crossover

Este algoritmo tem por objetivo obter descentes dos veículos com melhor fitness, cruzando porções do genótipo entre eles para os descendentes. A probabilidade de um cruzamento ocorrer provém da variável *crossoverProbability*, presente no ficheiro GeneticAlgorithmConfigurations.cs. Caso o resultado da função aleatória *RandomizationProvider.Current.GetDouble* seja inferior a este valor, o veículo descendente irá ser idêntico ao veículo do qual foi gerado inicialmente. O lugar onde termina e começa as porções do genótipo de cada descendente surge da função *RandomizationProvider.Current.GetInt* , que irá devolver um valor aleatório entre os valores enviados como parâmetro.

Algoritmo 2 – Single Point Mutation

Com a implementação deste algoritmo, foi desenvolvido o operador de mutação *bit-flip* que irá ser aplicado a todos os carros, exceto aos que foram retirados da iteração anterior.

Deste modo, é possível alterar o seu código genético, independentemente do que foi realizado no algoritmo SinglePointCrossover, alterando alguns dos seus valores (passando-os de 0 para 1 ou vice-versa), dependendo da probabilidade *mutationProbability* definida no ficheiro GeneticAlgorithmConfigurations.cs.

Algoritmo 3 – Tournament Selection

Neste algoritmo pretende-se escolher o carro a que foi atribuído o melhor fitness, sendo este, o melhor de um torneio gerado aleatoriamente. O tamanho do torneio, *tournamentSize*, é definido no ficheiro GeneticAlgorithmConfigurations.cs.

Algoritmo 4 – Elitism

O algoritmo inserido tem por fim reinserir na nova população os elementos presentes na população antiga que possuem a melhor fitness. O número de elementos adicionados é limitado pela variável *eliteSize*, estando o valor desta presente no ficheiro GeneticAlgorithmConfigurations.cs.

Função de *Fitness*

A função de *fitness* alcançada é resultado de mais de três testes, usando os parâmetros por defeito presentes no ficheiro GeneticAlgorithmConfigurations.cs. Esta função tem como objetivo obter um veículo capaz de alcançar o fim do percurso no menor espaço de tempo e no menor número de gerações possível.

Ao analisar a expressão fornecida, verifica-se que veículos capazes de terminar a corrida teriam todos o mesmo valor, 1, não havendo distinção entre os mesmos e consequentemente dificultando o processo de evolução. Carros incapazes de terminar seriam atribuídos com valor de 0, não havendo distinção entre carros promissores.

Com o objetivo de resolver este problema foi implementa uma condição inicial que distingue os carros que terminaram o percurso dos que não o fizeram. Assim, estes dois conjuntos irão estar associados a funções de *fitness* diferentes.

Para garantir que o valor de *fitness* de um veículo que termina a pista é sempre superior ao dos que não terminam, é atribuído um valor positivo e negativo, respetivamente. De forma manter a relação nos valores negativos, é calculada o seu inverso, mantendo-se assim a relação de superioridade. Para evitar que a variável *PopAverage* seja igual a menos infinito resultante de uma divisão de 1 por um valor de *MaxDistance* muito perto de 0, foi colocada a condição presente nos carros que não terminem a corrida.

A função implementada capaz de obter veículos que terminam o percurso é a seguinte:

```
if ( isRoadComplete ) then
|       fitness ← (1/MaxDistanceTime) * 0.8 + MaxVelocity * 0.2
else
|       if (MaxDistance <= 1) then
|       |       fitness = -1
|       else
|       |       fitness = -1 / MaxDistance
|       end
end
```

Fig1 – Pseudocódigo da primeira função de *fitness*

Esta função inicial permitiu obter, em diversos testes, vários carros capazes de alcançar o fim do cenário. Tendo como objetivo obter carros capazes de fazer esta tarefa o mais rápido possível [1], para carros que terminaram a corrida, é favorecido o tempo em que o fizeram e também a velocidade máxima que atingiram. Caso o carro não tenha capacidade de alcançar o fim, é favorecido aquele que chegou mais longe, desta forma pretende-se que novas populações sejam capazes de aumentar a distância percorrida relativamente ao que foi anteriormente possível.

---

[1] Verificou-se posteriormente que isto não ia ao encontro da função inicialmente apresentada

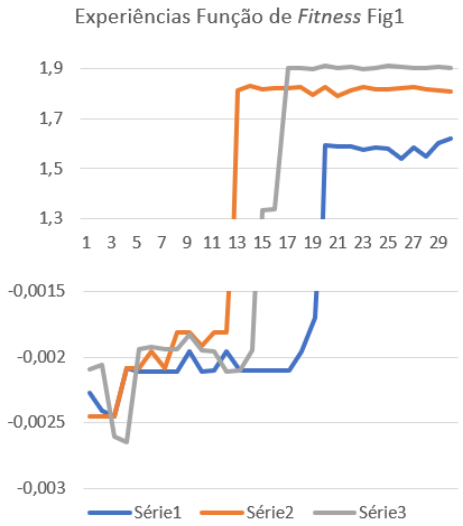


Fig2 – Gráfico das experiências relativas à Fig1

O gráfico da Fig2 apresenta a evolução do valor *fitness* do veículo mais promissor, a cada iteração da experiência.

Observa-se que até encontrar um veículo capaz de terminar a pista existe uma evolução progressiva do valor fitness (excluindo alguns *outliers*); após isso, o seu valor estagna.

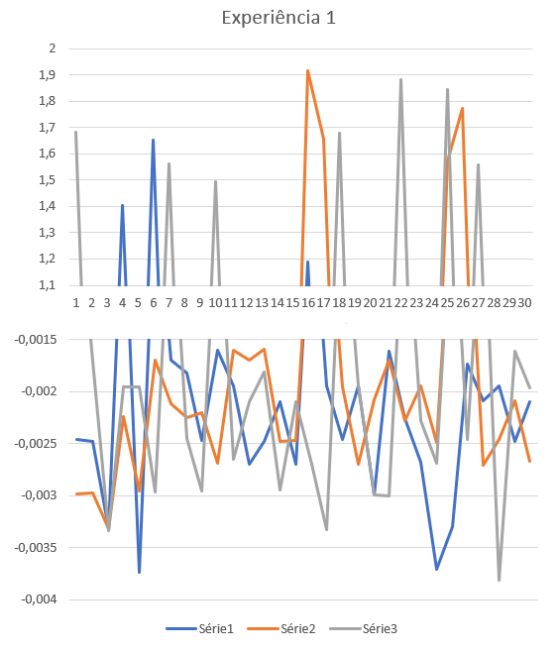
A subida drástica de valores deve-se à natureza da função implementada, na qual é feita a distinção usando valores negativos e positivos, como referido acima.

Meta 2

Esta meta tem por fim desenvolver a função de *fitness* inicialmente encontrada, realizar diversas experiências de modo a encontrar as configurações ideais do algoritmo e alcançar um veículo capaz de terminar o cenário *HillRoad*.

Função de *Fitness* Inicial

Tendo já realizado as 15 experiências relativas à Tabela 1, embora esta função não corresponda ao comportamento indicado, foram criados os seguintes gráficos relativos aos valores de *fitness*, de maneira a ter uma primeira ideia de quais serão as configurações mais adequadas.



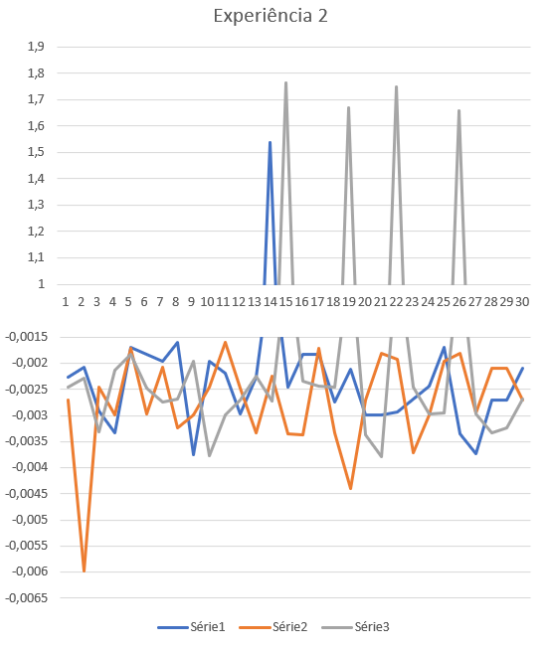


Fig3 – Gráficos das 15 experiências relativas à função *fitness* da Fig1

Não irá ser realizada uma análise elaborada destes gráficos, sendo esta realizada posteriormente nos testes feitos usando a função de *fitness* final. Observa-se claramente que as experiências 1 e 2 possuem uma evolução de *fitness* muito irregular; isto deve-se ao uso do parâmetro *eliteSize* igual a 0, onde a ausência dos veículos com maior *fitness* na população seguinte irá afetar de forma negativa a sua evolução. Nos gráficos restantes, está presente uma evolução mais regular da *fitness*, contudo ainda não é possível tirar conclusões sobre os parâmetros mais propícios de mutação e tamanho do torneio sem a presença de gráficos de evolução da *average fitness* da população.

Função de *Fitness* Atualizada

Após determinar a função de *fitness* na Meta 1, em consequência de uma análise mais profunda e dos testes realizados, foi possível concluir que a função destinada a veículos que terminam o percurso não corresponde às premissas mencionadas nessa Meta. O valor obtido em 1/MaxDistanceTime * 0.8 é muito pequeno (sempre menor que a unidade), enquanto MaxVelocity * 0.2 irá ser consideravelmente maior (superior à unidade na maior parte dos casos). Assim, veículos com *MaxVelocity* apenas um pouco mais elevado irão ter *fitness* superior a um carro que alcançou a meta mais cedo. A seguinte função tem como objetivo corrigir esse erro, mantendo o caráter seletivo entre veículos que terminam ou não:

```
fitness ← MaxDistance + IsRoadComplete * ((1 / MaxDistanceTime) * 1000)
```

Fig4 – Pseudocódigo da função de *fitness* atualizada

Uma vez que 1 / MaxDistanceTime pode resultar numa indeterminação em casos onde o carro falha de imediato (sem percorrer qualquer distância), o valor *fitness* é definido como 0.

Os gráficos seguintes representam a evolução dos valores *fitness* e da massa, usando as configurações base no ficheiro original.

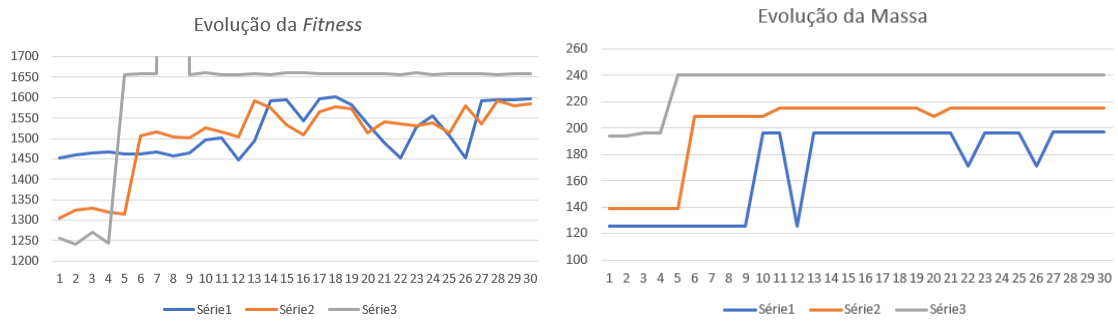


Fig5 – Gráficos da *fitness* e massa das 3 experiências relativas a função da Fig4

O pico presente na iteração 8 do gráfico da *fitness* corresponde a um *outlier*, devido a um carro anormal que realizava o percurso de forma ilegal ("voando" e atravessando partes do percurso), tendo sido por isso ajustado o eixo vertical para uma melhor visualização dos restantes dados.

Como se consegue visualizar, após um carro adquirir uma *fitness* elevada, a diferença de valores existente no gráfico relativo à *fitness* deve-se ao facto de o carro não conseguir terminar a pista ou ter uma pior prestação que o normal. É possível determinar que se trata do mesmo veículo verificando que o valor da massa não se altera.

Consta-se que, de uma forma geral, carros com maior *fitness* possuem uma maior massa; isto vai de encontro ao esperado, pois veículos mais compridos (e consequentemente com maior massa devido ao tamanho dos vetores que o constituem) pressupõem uma maior aptidão para atravessar os intervalos. Apesar de o aumento da massa aparentar ser favorável, a sua utilização na função de *fitness* pode não corresponder ao esperado, pois a forma do carro não é garantida, não tendo sido por isso utilizada.

Função de *Fitness* menos seletiva

As funções anteriores eram caracterizadas pela sua natureza seletiva, impedindo que carros promissores, mas que não terminam o percurso, possuam fitness superior a carros que a terminam, mas de forma bastante lenta. Isto pode levar a um *bottleneck* no algoritmo, diminuindo a sua eficiência evolutiva. Assim, o uso da variável *MaxVelocity* tem como meio permitir que isto não aconteça, passando a função a ser:

```
fitness ← MaxDistance + (MaxVelocity * 100) +
          + IsRoadComplete * ((1 / MaxDistanceTime) * 1000)
```

Fig6 – Pseudocódigo da função de *fitness* atualizada, com uso de *MaxVelocity*

Um exemplo concreto onde esta função prevalece relativamente às funções encontra-se ilustrado a seguir.

Carro 1:
Termina a pista, com MaxDistanceTime = 74 e MaxVelocity = 8.5
fitness = 656 + (8.5*100) + 1*((1/74)*1000) = 1519.5

Carro 2:
Não termina a pista, com MaxDistance = 580 e MaxVelocity = 9.5
fitness = 580 + (9.5*100) = 1530

Fig7 – Exemplo da nova função de *fitness*

Foram realizadas três experiências com o objetivo de analisar se os resultados da nova função não prejudicavam a evolução em relação à função anterior. De seguida encontra-se o gráfico relativo aos valores *fitness* para esta nova função:

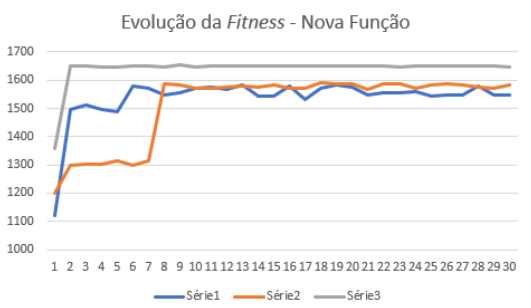


Fig8 – Gráfico da evolução da *fitness*

Relativamente ao gráfico de *fitness* anterior, observa-se que os seus valores permanecem adequados, não apresentando nenhum entrave ao problema. Assim, procedeu-se à análise dos gráficos seguintes relativos ao tempo necessário para terminar o percurso para as duas funções.

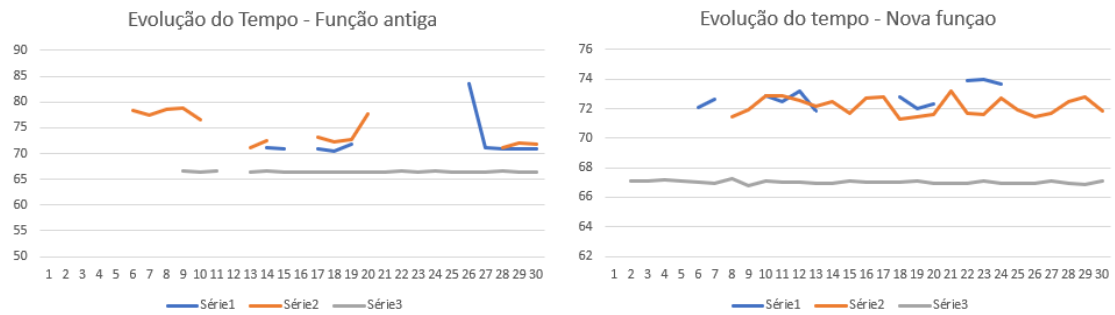


Fig9 – Gráficos da evolução do tempo das funções

É de notar que veículos que terminam o percurso podem não conseguir terminar nas iterações seguintes; desta forma, apesar da variável IsRoadComplete indicar o valor *true*, isto pode não ser verdadeiro nas outras iterações, tendo por isso sido colocado nos gráficos apenas os carros que percorreram os 656 metros. Observa-se que na nova função foram encontrados carros capazes de terminar o percurso com um menor número de gerações, necessitando, em média, de 5, enquanto na função anterior, a média é de aproximadamente 10. Além do número de gerações necessárias para encontrar um veículo capaz de terminar a pista ser menor, a média do tempo da última iteração manteve-se em ambas as funções muito semelhante. No entanto, constata-se que, usando a nova função, os veículos obtidos são mais aptos, acabando o percurso de forma mais consistente nas iterações seguintes ao primeiro sucesso.

Análise dos melhores parâmetros do algoritmo

Dada a natureza estocástica das abordagens evolucionárias, foram realizadas 15 experiências relativas à Tabela 1 do enunciado, de modo a determinar os parâmetros mais propícios.

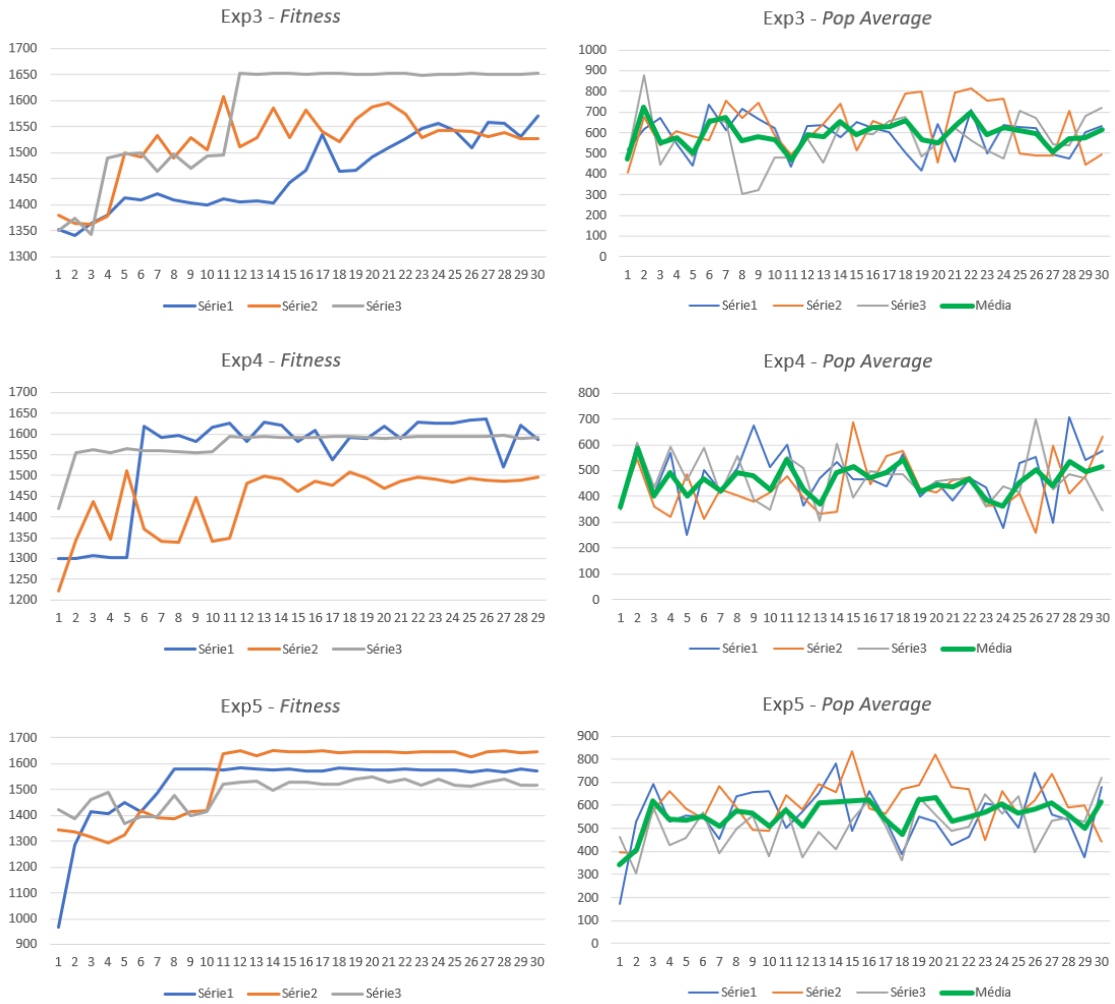


Fig10 – Gráficos da evolução da *fitness* e do *pop average*

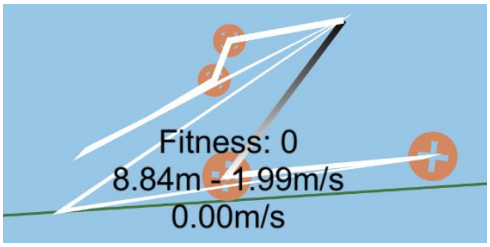
Para determinar quais os valores para a mutação e tamanho do torneio mais eficazes foram criados os gráficos da average fitness da populaçao para além da fitness do melhor veículo.

Como observado nas 15 experiências para a primeira função de *fitness*, verifica-se que na 1 e 2, onde o *eliteSize* possui valor de 0, a evolução da *fitness* é muito irregular, não havendo uma evolução ascendente deste valor. As experiências 3, 4 e 5 apresentam as evoluções mais promissoras. A experência 4 é aquela que das 3 apresenta os valores mais baixos de *average fitness*; isto deve-se ao facto de um valor de mutação de 0.2 gerar veículos com características muito drásticas afetando de forma prejudicial a sua eficácia. Assim, é possivel concluir que os melhores valores para o elitismo é 2 e mutação é 0.05; contudo a determinação do tamanho do torneio não é tão trivial. Na experiência 3 estão presentes os melhores valores de *average fitness* da população, porém a experiência 5 apresenta uma evolução de *fitness* mais adequada, sendo os seus veículos muito mais fiáveis e constantes no seu desempenho. Deste modo, para determinar o melhor valor para o tamanho do torneio, foi analizado o código fonte, onde se conclui que, se o tamanho do torneio for maior, os indivíduos mais fracos têm uma menor chance de serem selecionados, pois se um indivíduo fraco for selecionado para participar num torneio, há uma maior probabilidade de um indivíduo mais forte estar presente.

Em suma, os parâmetros mais adequados são os da experiência 3, passando a ser utilizados nas experiências seguintes.

Testes no cenário *HillRoad*

O seguinte carro obteve a maior *fitness* na realização de todas as experiências no cenário *GapRoad*.



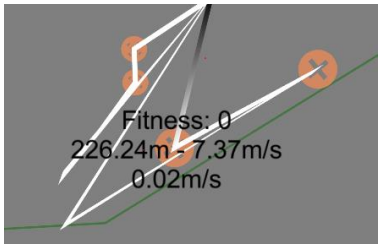


Fig11 – Melhor carro obtido no cenário *GapRoad*, implementado no HillRoad

No cenário *GapRoad*, este possui as características:

- Fitness – 1653,38
- Tempo de execução – 66,78
- Número de rodas – 8
- Massa – 198

Ao executar um teste com este carro no cenário *EvaluationHillRoad*, a sua prestação é bastante má, chegando apenas aos 226.24 de 360 metros, devendo-se ao facto de não ser estritamente necessário que um carro desenvolvido num cenário tenha a capacidade de terminar com sucesso qualquer ambiente onde seja inserido.

Após esta experiência, passou-se à realização de testes com a função de *fitness* no *HillRoad*, concluindo-se rapidamente qual o problema da função para este cenário. Uma vez que a função favorece carros mais rápidos (que no cenário *GapRoad* mostra ser bastante vantajoso), devido às caracteristicas do cenário *HillRoad*, havendo subidas e descidas, a velocidade dos carros vai ser bastante influenciada pelo ambiente, não indicando com tanta certeza se um carro é realmente mais rápido do que outro. Verificou-se também que alguns carros nas "lombas" do topo de uma colina tornavam-se instáveis, podendo rodar sobre si mesmos, ficando assim com uma *MaxVelocity* bastante mais elevada do que qualquer outro carro. Devido às caracteristicas da função de *fitness* e do cenário, a distância tem um impacto muito menos significativo do que a velocidade máxima, dado que caso um carro rode sobre si próprio na fase inicial e atinja uma velocidade de 9.9, este fica de imediato com uma *fitness* de pelo menos 990. Isto provoca uma dificuldade extrema em encontrar uma carro com *fitness* superior que seja mais apto para a conclusão deste cenário.

De maneira a atingir o objetivo de terminar o ambiente *HillRoad*, a função de *fitness* teve de ser alterada novamente, sendo a expressão a seguinte:

```
fitness ← MaxDistance + (MaxDistance / MaxDistanceTime) +
          + IsRoadComplete * ((1 / MaxDistanceTime) * 1000)
```

Fig12 – Pseudocódigo da função de fitness para *HillRoad*

Foi substituído o uso da *MaxVelocity* pela velocidade média, uma vez que permite superar os problemas anteriormente mencionados. A remoção da multiplicação da velocidade por 100 deve-se ao facto de já não ser necessário uma ênfase tão grande para superar o valor da expressão restante.

Com o objetivo de encontrar um veículo capaz de terminar o cenário *HillRoad*, foi utilizada a função *fitness* mencionada acima. A experiências realizadas possuem 100 gerações, onde foi possível encontrar em duas delas um carro capaz de terminar o percurso. Os gráficos abaixo representam a evolução da *fitness* em 5 das experiências.

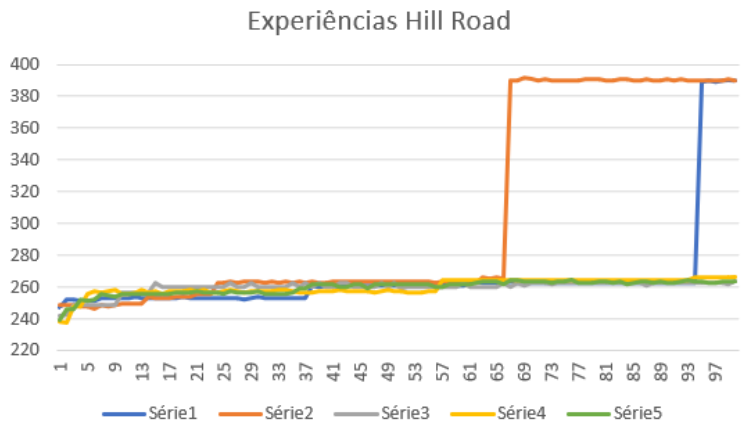


Fig13 – 5 experiências de 100 gerações no cenário *HillRoad*

Em busca de conseguir encontrar um carro que terminasse este cenário nas 30 gerações, foram realizadas enúmeras execuções com os mesmos parâmetros, tendo sido possível encontrar 2 veículos em execuções diferentes. Os gráficos seguintes representam a evolução da *fitness* para as duas experiências.

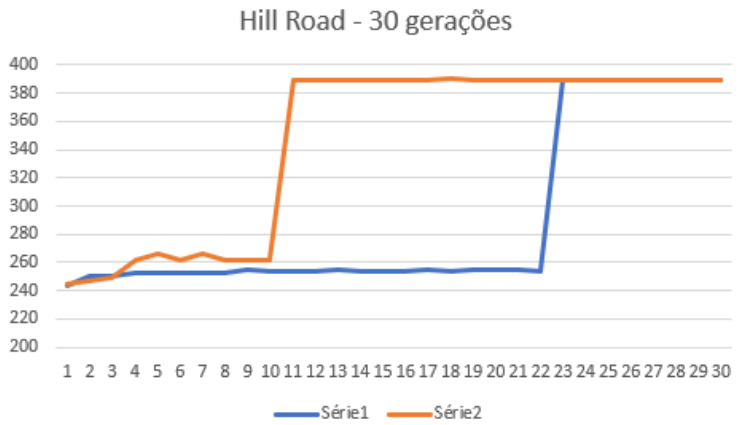


Fig14 – 2 experiências bem-sucedidas de 30 gerações no cenário *HillRoad*

A seguinte figura apresenta o carro com a melhor fitness obtido:

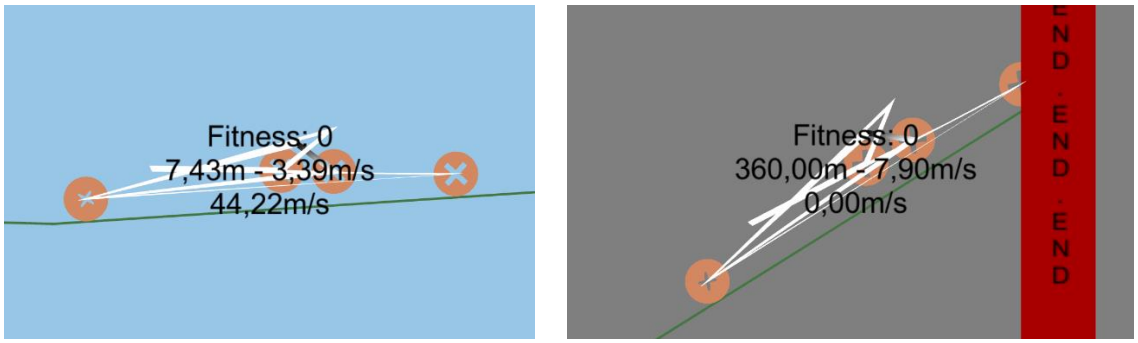


Fig15 – Melhor carro obtido no cenário *HillRoad*

No cenário *HillRoad*, este possui as características:

- Fitness – 391,36
- Tempo de execução – 43,63
- Número de rodas – 8
- Massa – 171.5

Conclusão

Este projeto permitiu desenvolver aptidões e conhecimentos relativos a ferramentas de *Unity*, sobre algoritmos evolucionários e as suas limitações, bem como verificar que os ambientes moldam os indivíduos que neles habitam de forma diferente. Constatou-se que um algoritmo evolucionário, dependendo da sua função de fitness, pode não ter os resultados esperados em ambientes diferentes. Observou-se que o número de gerações de cada experiência possui um papel importante na evolução de cada veículo, no entanto, o tempo necessário para a sua execução constituiu um entrave para uma análise mais aprofundada. Os parâmetros influenciam também a evolução de forma significativa, sendo a sua escolha relevante para o sucesso das experiências.