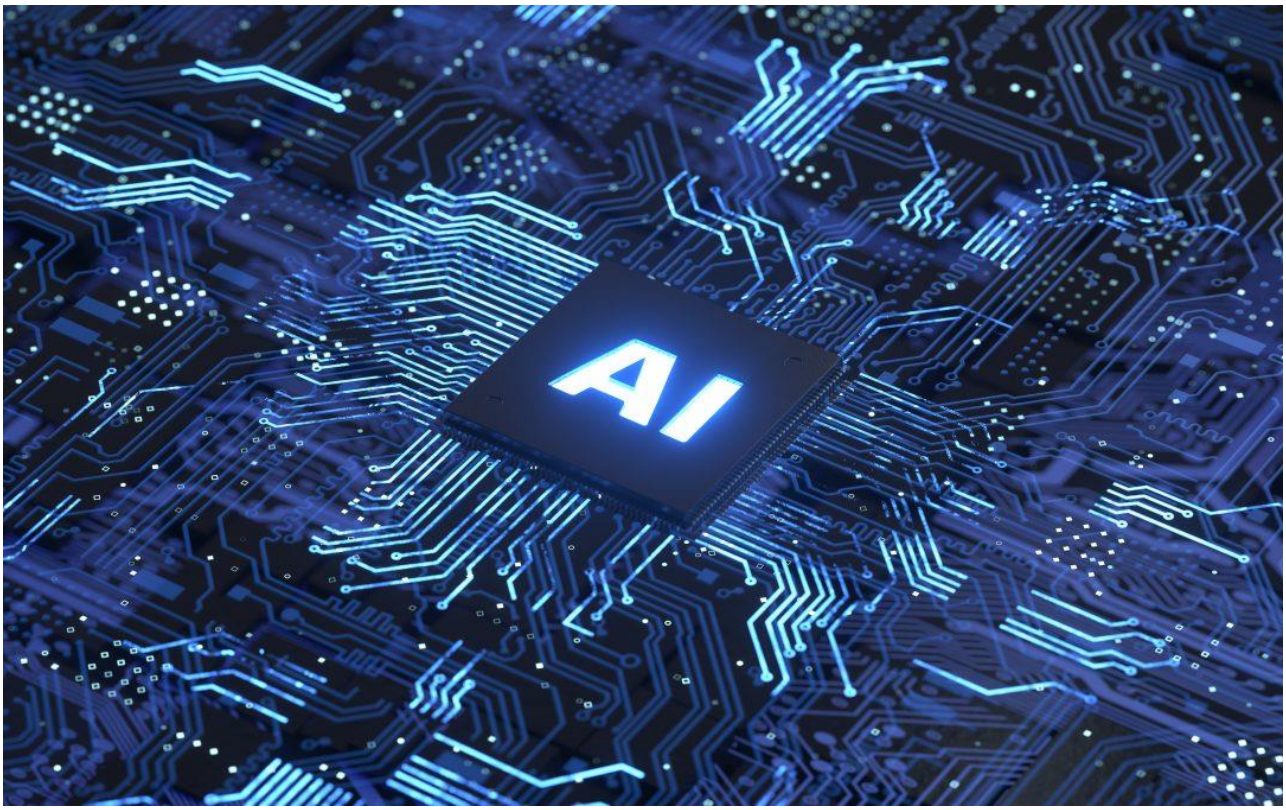
INTRODUCTION TO MACHINE LEARNING - REINFORCEMENT LEARNING

20 DE DEZEMBRO DE 2020



Nome: Miguel Oliveira

Aluno nº: 96200

Email: mdlco@iscte-iul.

Docente: Professor Doutor Luís Nunes

Docente: Professor Doutor Sancho Oliveira

Introdução:

Supondo a existência de um robô, que precise de aprender uma sequência de ações para o levar da posição inicial a uma casa-posição de destino, este operará numa sala, que para estes efeitos, estará dividida em pequenas "células" quadradas, também denominadas por *states*, cada uma representando uma posição nesta sala. Assume-se, também, que o robô apenas se poderá deslocar em quatro direções (esquerda, direita, baixo, cima).

A *maze* contém cem *states* e o robô começar-se-á a deslocar do *state* 1, usando as regras anteriormente definidas. Se porventura o robô atingir o *state* 100, este será recompensado, caso contrário, não. Todos os *states* estão numerados de 1 a 100 e o robô não poderá sair desta sala. Caso tente tomar uma ação que infrinja essa regra, será informado que deverá permanecer onde se encontra.

Foi elaborado um programa em *python* para auxiliar na conceção e análise deste problema e, assim, tentar dar resposta às questões colocadas.

Todos os testes e resultados obtidos foram feitos num computador HP EliteBook 840 G6, com um processador i5-8365U, 16GB de RAM e um sistema operativo Windows de 64 bits.

De modo a que os resultados obtidos possam ser replicados, recorreu-se a um sistema de "*seeds*" que garantem a replicação dos mesmos.

Para se executarem facilmente os resultados de cada exercício, cada um dos exercícios está num ficheiro *.py* respetivo, pois apesar de se ter replicado o código comum aos vários exercícios, é, assim, mais fácil testar e replicar os resultados dos mesmos.

Cada *.py* conta com uma função indicativa com o nome do exercício respetivo e, que funciona como "*main*", onde se podem definir alguns modos de execução do programa, como por exemplo, analisar os tempos de execução ou apenas a média de recompensas por estado.

```
if __name__ == '__main__':
    ex5(evaluate_time=True, learning_mode=False)
```

Já as variáveis do ambiente, como por exemplo, os valores de alfa e discount terão de ser alterados dentro destas funções "*main*".

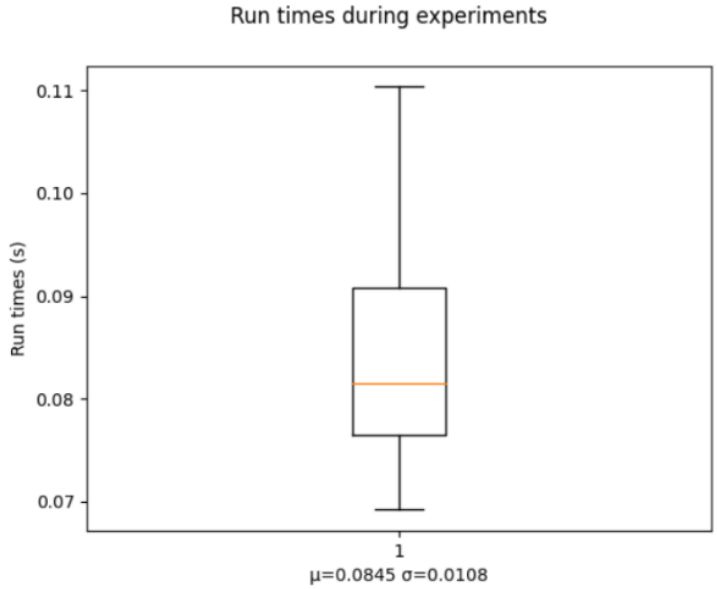
evaluate_time: *True* → tempos de execução | *False* → a média de recompensas por estado.

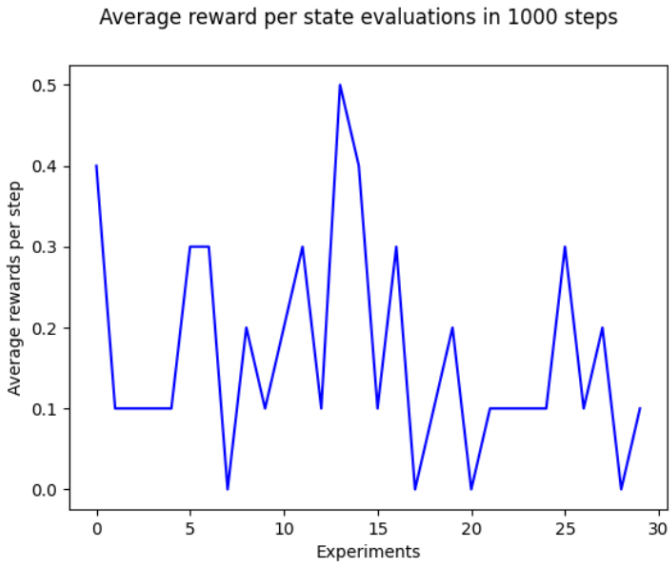
learning_mode: *True* → Escolhe sempre a "melhor ação" | *False* → escolha aleatória

incremental: *True* → Ganância incremental | *False* → Ganância com valor fixo
wall_type: *maze_with_wall* → Formato 1 da *maze* | *maze_with_wall2* → Formato 2 da *maze*

Exercício 3:

Neste exercício o robô movimenta-se aleatoriamente pela *maze*, um passo de cada vez, sempre que atinge o *state* 100 é recompensado.
Foi executado o programa, correu 20.000 vezes, e foram feitas medições no sentido de avaliar a velocidade com que o programa é executado, ou seja, o quão rápido o robô percorre a *maze* em 20.000 passos aleatórios. Este processo foi repetido de forma controlada 30 vezes e, assim, conseguiram-se os seguintes resultados:





Analisando os gráficos acima apresentados, é visível a esperada inconsistência no número de recompensas conseguidas em 1000 passos, uma vez que, são todos aleatórios.

Pelos tempos de execução obtidos, constata-se que o processo no geral é bastante rápido, pois não existe grande complexidade envolvida. Contudo, estes resultados iniciais servirão como resultados base e serão usados para testar se o sistema está a apresentar melhorias em relação às iterações anteriores, que para já se baseiam no conceito da "sorte" para tomar decisões.

Exercício 4:

Nesta etapa é introduzido o mapeamento de valores utilitários aos *states* anteriores, por onde o robô se movimentou dentro da *maze*, numa tentativa de quantificar o valor intrínseco que um dado state pode ter e a sua utilidade na conceção de um caminho para o *state* 100 que o robô poderá usar eficazmente.

Para isto, é criado o vetor V onde são armazenados os valores utilitários dos respetivos *states* da *maze* e usada a seguinte fórmula para calcular a utilidade:

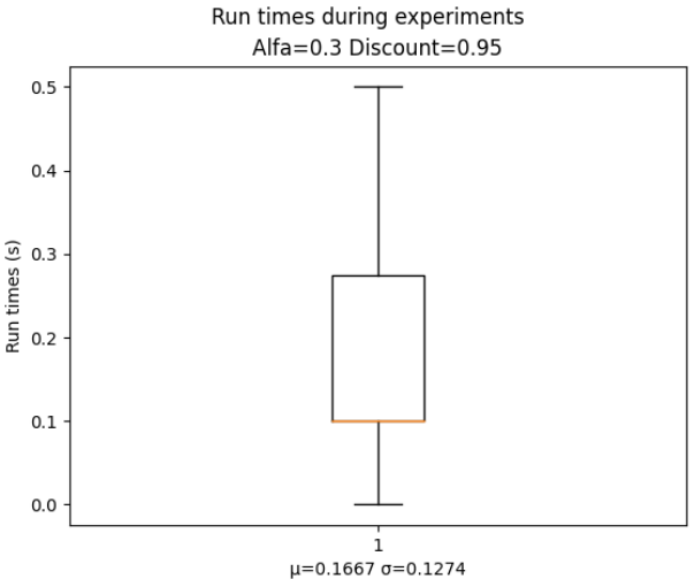
$$V(s) = (1 - \textit{alpha})\ V(s) + \textit{alpha} * (r(s') + \textit{discount} * V(s'))$$

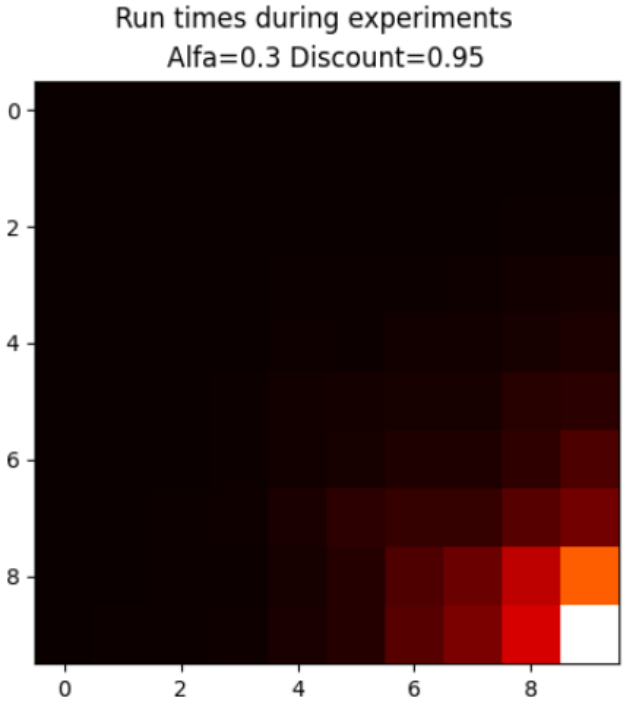
Em que o *alpha* e o *discount* são parâmetros previamente definidos e variam entre 0 e 1. Quando o alfa assume valores mais perto do 0, torna a "aprendizagem" segura, mas lenta, e quando o *discount* assume valores superiores a 0.9, ou seja, mais perto de 1, mais valorizada será a importância de recompensas a longo prazo.

Assim sendo, considerou-se duas situações com valores diferentes para o *alpha* e o *discount* de modo a estudar o comportamento do robô em cenários distintos, um que valorizasse as recompensas a longo prazo e outro a curto prazo.

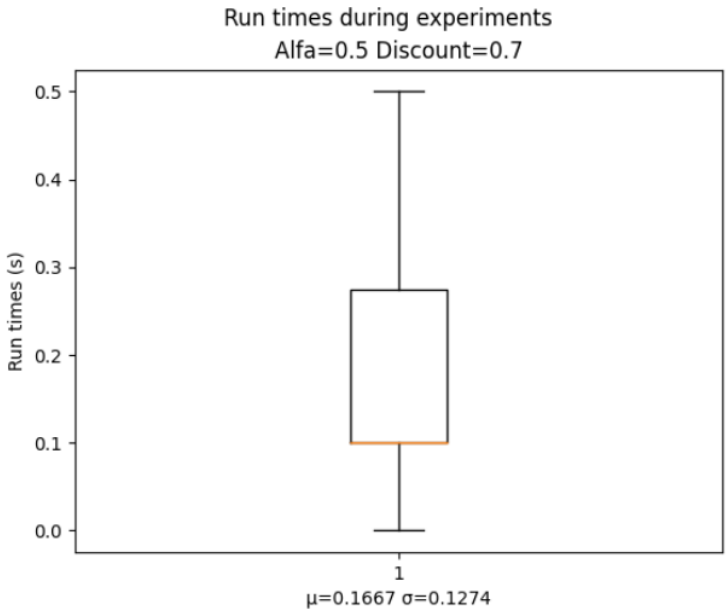
Para o curto prazo os valores usados foram, *alpha* = 0.5 e *discount* = 0.7 e para longo prazo *alpha* = 0.3 e *discount* = 0.95

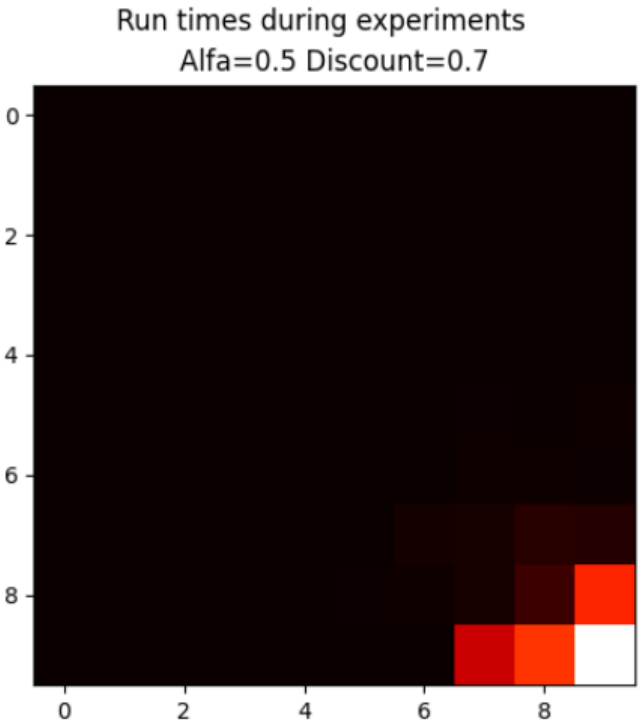
De modo a representar, da melhor maneira, o vetor V recorreu-se a um “Heat Map”.





```
[[  0.11   0.1    0.16   0.29   0.39   0.64   0.62   0.66   0.77   0.94]
 [  0.13   0.15   0.16   0.23   0.5    0.73   0.98   0.92   1.06   1.23]
 [  0.16   0.19   0.19   0.26   0.69   1.     1.55   2.28   1.13   2.15]
 [  0.21   0.28   0.36   0.31   1.07   1.53   2.05   2.64   3.24   5.06]
 [  0.28   0.33   0.4    0.65   2.29   2.59   4.63   4.48   7.5    6.11]
 [  0.36   0.4    0.67   1.2    2.93   4.42   4.6    9.54   12.49  13.55]
 [  0.59   0.62   0.78   1.7    4.26   7.82   17.03  16.21  29.48  29.34]
 [  0.75   0.95   1.36   2.12   8.78   7.75   23.99  41.77  55.64  77.06]
 [  0.77   1.09   4.01   4.25   9.67   14.49  30.03  39.81  73.9   149.33]
 [  0.97   1.51   4.58   6.34   19.42  20.3   22.68  30.92  98.6   332.62]]
```





```
[[  0.     0.     0.     0.     0.     0.     0.     0.     0.     0.  ]
 [  0.     0.     0.     0.     0.     0.     0.     0.     0.     0.  ]
 [  0.     0.     0.     0.     0.     0.     0.     0.01   0.     0.  ]
 [  0.     0.     0.     0.     0.     0.     0.01   0.02   0.01   0.01]
 [  0.     0.     0.     0.     0.     0.01   0.01   0.06   0.06   0.02]
 [  0.     0.     0.     0.     0.     0.01   0.18   0.85   0.22   1.73]
 [  0.     0.     0.     0.     0.01   0.08   0.15   1.37   1.83   1.17]
 [  0.     0.     0.     0.01   0.01   0.15   3.03   3.9    7.84   6.84]
 [  0.     0.     0.     0.01   0.03   1.09   1.54   3.71  12.87  72.97]
 [  0.     0.     0.     0.01   0.26   0.42   0.6   49.6   76.84 173.32]]
```

Observando os resultados, verificou-se que, para ambições a curto prazo não é possível obter um sequência de states pela qual o robô terá de passar até atingir a casa alvo, mas, quando a ambição está orientada ao longo prazo, já é possível traçar um certo caminho desde a casa partida à casa alvo, contudo, este caminho não está bem definido nos primeiros passos após a casa partida, concluindo-se que, com apenas esta informação não é possível afirmar com toda a certeza a existência de uma sequência de ações até à casa alvo, nem que é uma boa solução.

Verifica-se também que os tempos de execução são mais lentos do que os registados na etapa anterior.

Como ponto de melhoria (não foi implementado), poder-se-ia fazer um simples algoritmo, que escolhe sempre o estado seguinte de acordo com qual das suas opções o leva para o que tiver maior valor utilitário.

Exercício 5:

Nesta etapa, partindo de alguns princípios introduzidos na anterior, o robô ao percorrer a maze, atualiza, recorrendo a uma nova fórmula, os valores utilitários de cada célula da maze, contudo, enquanto que na etapa anterior, cada célula apenas tinha um valor utilitário correspondente, nesta, existe em cada célula um valor utilitário para cada ação possível numa determinada célula, ou seja, ao invés de termos um vetor V, temos uma matriz Q de tamanho 100 x 4, uma vez que, existem 100 células e 4 possíveis ações em cada uma, mesmo que algumas dessas ações levem o robô a estados inválidos.

De modo a atualizar os valores utilitários da matriz Q recorre-se à seguinte fórmula:

$$\mathrm{Q(s,a)} = (1 - \mathrm{alfa})\,\mathrm{Q(s,a)} + \mathrm{alfa} * (\,\mathrm{r(s)} + \mathrm{discount} * (max_{a'}\mathrm{Q(s',a')}))$$

Nesta etapa, a matriz Q é "treinada" de duas formas, uma em que o robô percorre aleatoriamente a maze, atualizando em cada passo a matriz Q e outra em que o robô escolhe o movimento seguinte segundo os valores utilitários da matriz Q, ou seja, quando o robô se encontra no state X, ele escolhe como movimento seguinte a ação que tiver o maior valor utilitário desse *state*.

Para os casos em que a ação escolhida leva o robô a um movimento inválido, por exemplo, que implique sair fora da maze, uma iteração é consumida, o robô permanece no mesmo *state* e o valor dessa ação escolhida é atualizado, em situações dessas, por norma, o valor dessa ação desce, permitindo que a certa altura essa mesma ação deixe de ser a melhor e uma nova possa ser escolhida.

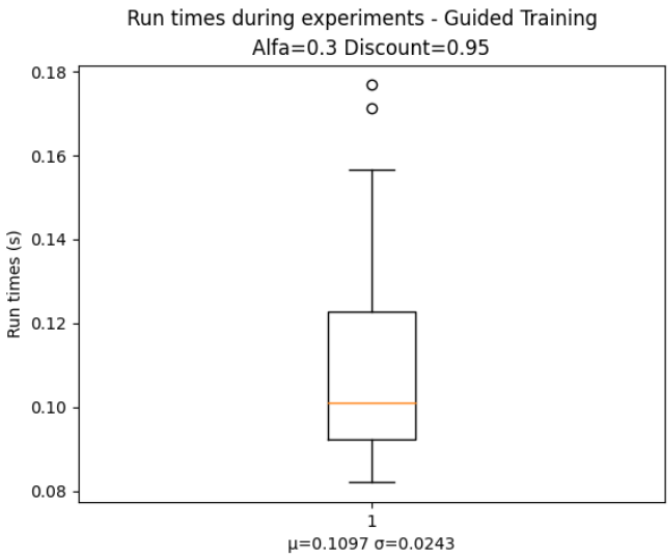
De notar que nem sempre essa diminuição de utilidade é rápida, podendo segundo esta lógica se escolher sempre a ação com maior valor utilitário levar o robô a ficar preso em ciclos consideráveis e, por conseguinte, reduzir o número médio de recompensas por estado.

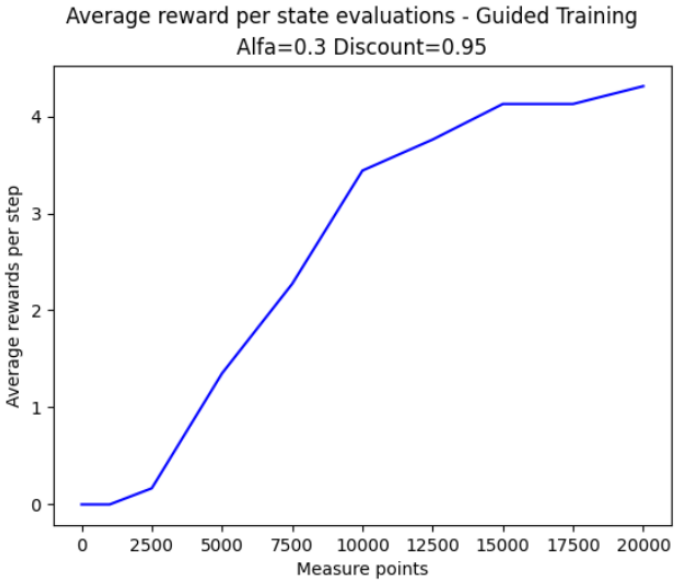
Assim, foram feitas as seguintes avaliações ao desempenho do robô com este algoritmo: mediu-se o número médio de recompensas por estado para quando o treino da matriz Q era feito unicamente com movimentos aleatórios e, depois escolhendo sempre o movimento seguinte, que tiver a ação com maior valor utilitário.

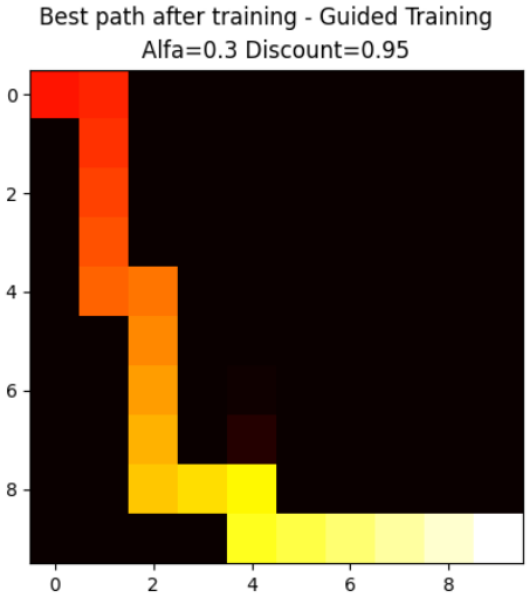
Para visualizar melhor o resultado da matriz Q após o treino "inteligente" foi elaborado uma "Heat Map"

Tal como na etapa anterior, os valores do alfa e do discount influenciam o resultado final, contudo, para estas avaliações foram usados os seguintes valores: alfa=0.3 e discount=0.95

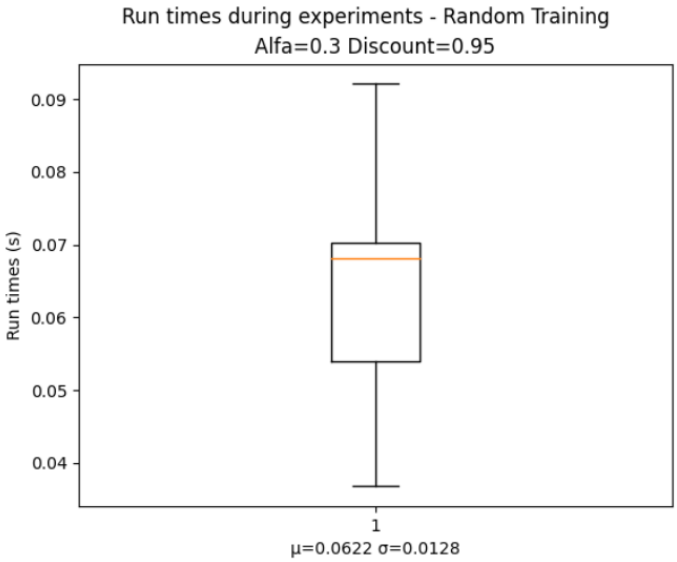
"**Testes inteligentes**":

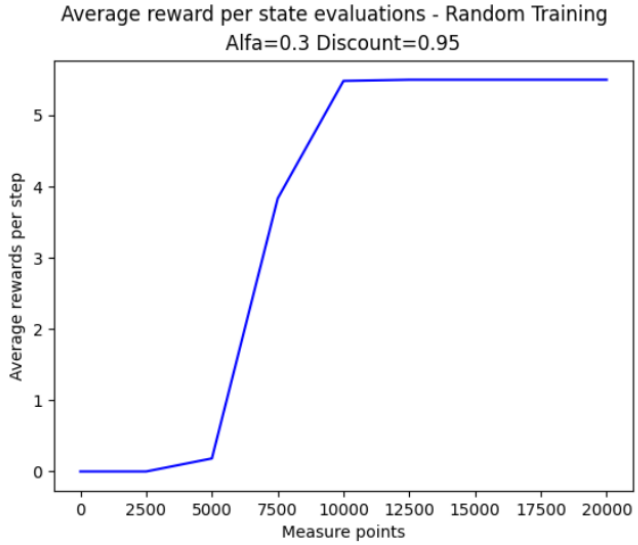


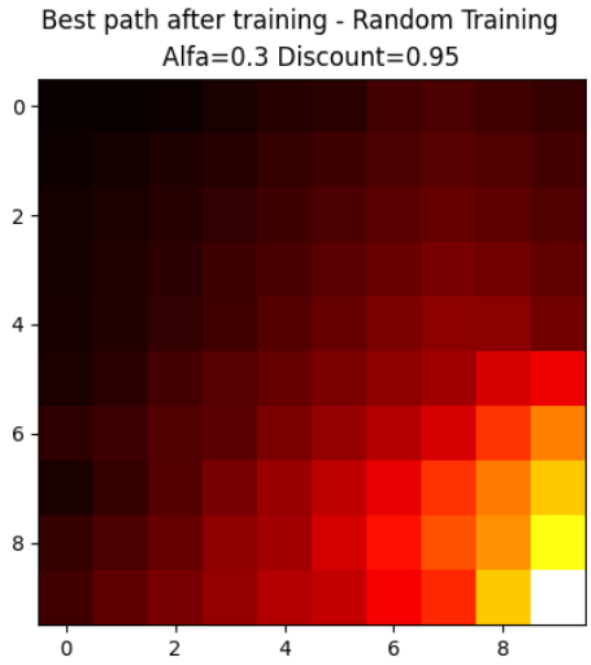




Testes aleatórios:


Run times during experiments - Random Training
Alfa=0.3 Discount=0.95
Run times (s)
0.09
0.08
0.07
0.06
0.05
0.04
1
μ=0.0622 σ=0.0128


Average reward per state evaluations - Random Training
Alfa=0.3 Discount=0.95
Average rewards per step
5
4
3
2
1
0
0
2500
5000
7500
10000
12500
15000
17500
20000
Measure points



Analisando os dados obtidos para os casos em que o robô treinou a matriz Q com movimentos aleatórios, é claro que não é possível determinar um "melhor" caminho desde o *state* inicial ao *state* 100, tal como se visualiza pelo "Heat Map" correspondente, contudo, verificam-se melhores resultados no número médio de recompensas por estado comparando aos resultados obtidos quando o robô se guia sempre pela "melhor ação". Isto deve-se ao facto de, quando a matriz Q é aleatoriamente treinada, não deixando o robô ficar preso em soluções "sub ótimas" que é o que acontece quando o treino é pela "melhor ação". Essa solução "sub ótima", em que o robô fica preso, é possível ser visualizada no "Heat Map" da matriz Q após o treino pela "melhor ação".

Quanto aos tempos de execução, quando o treino era aleatório obtiveram-se melhores resultados, contudo, comparando com etapas anteriores, vê-se que quando o treino é feito pela "melhor ação" os tempos de execução tornam-se mais demorados.

Em suma, escolher sempre a melhor ação, apesar de guiar o robô para uma boa solução, também leva a que o mesmo fique preso a essa mesma solução, nunca sendo capaz de tentar explorar uma melhor rota, e assim, impedindo o robô de conseguir alcançar a melhor rota possível. Para além disso, também se encontraram casos em que o robô ficava preso em ciclos durante muito tempo, o que dificulta a sua missão, chegando até a haver casos em que durante o treino nunca conseguiu atingir o *state* 100.

Exercício 6:

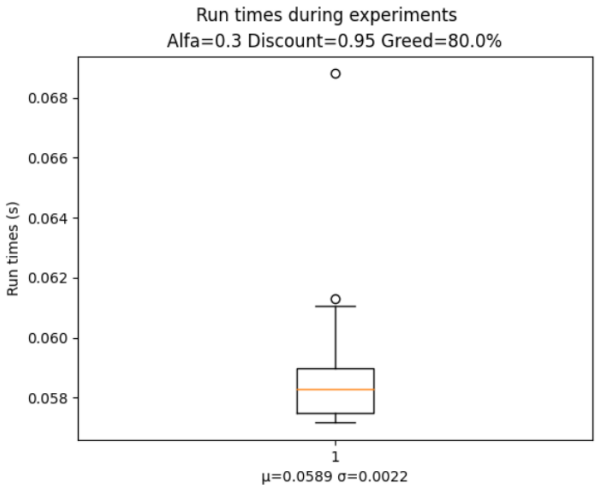
Tal como analisado na etapa anterior, o robô está a ficar preso a uma dada solução, impedindo-o de explorar melhores soluções que possam existir, assim, é agora dada a possibilidade de, para além de tomar decisões baseadas no conhecimento adquirido

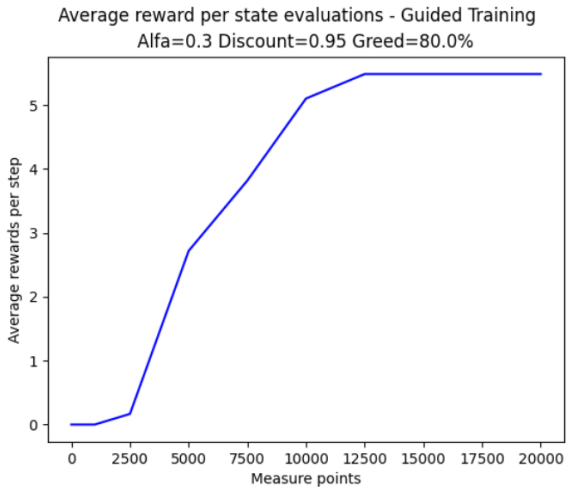
enquanto progride pela *maze*, poder ter uma certa probabilidade de fugir do seu rumo padrão - escolher sempre a ação com maior utilidade – e, assim evitar ficar preso por muito tempo em ciclos repetitivos. Para isso, foi introduzido o conceito de "ganância", que por outras palavras dá ao robô a possibilidade de, segundo o seu nível de ganância, decidir entre escolher o movimento seguinte pela "melhor ação" ou fazer um movimento aleatório.

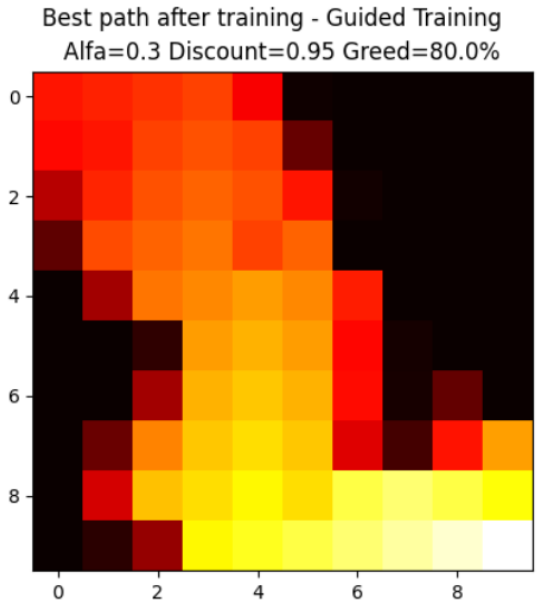
Para ilustrar isso, foram feitas experiências com valores de ganância nos 60% e 80%.

Foi também feita a experiência, variando os valores da ganância, de uma forma incremental à medida que o treino do robô ia progredindo, ou seja, dos 20.000 passos do treino, a cada 10% dos passos completos, isto é, a cada 2000 passos, a ganância do robô aumenta 7%. Para efeitos de testes, os níveis iniciais de ganância escolhidos foram 30%.

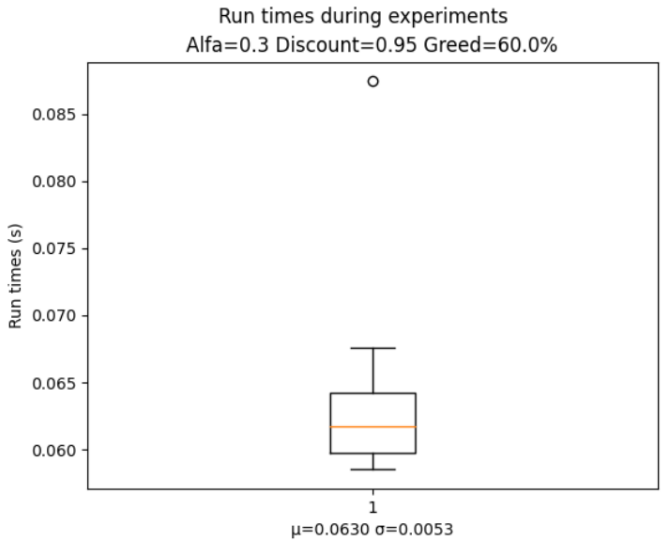
"Testes inteligentes" com níveis de ganância fixos em 80%:

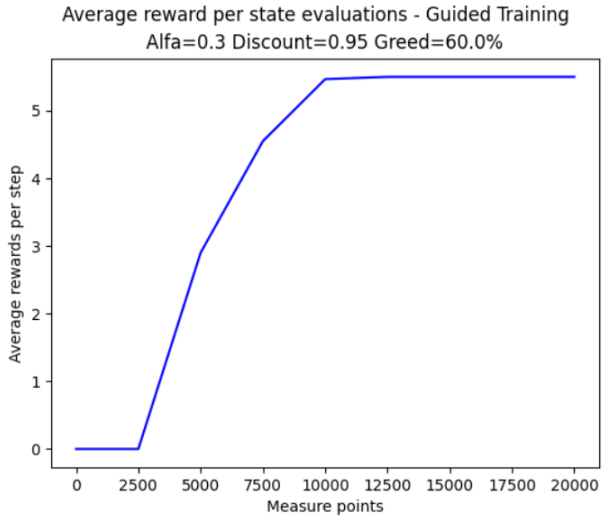


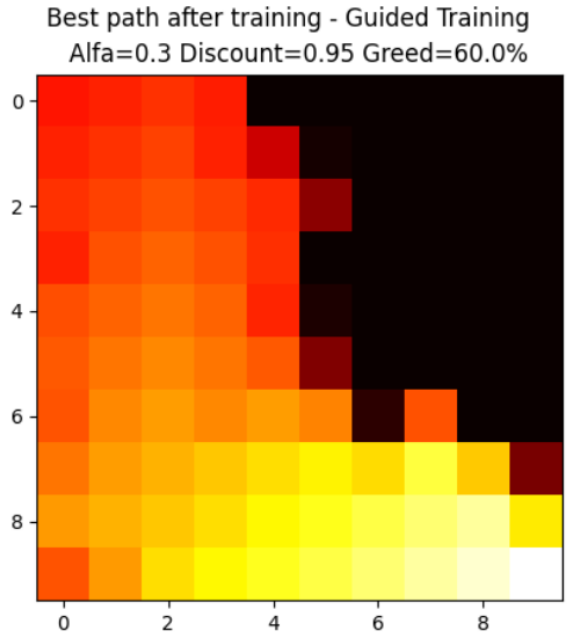




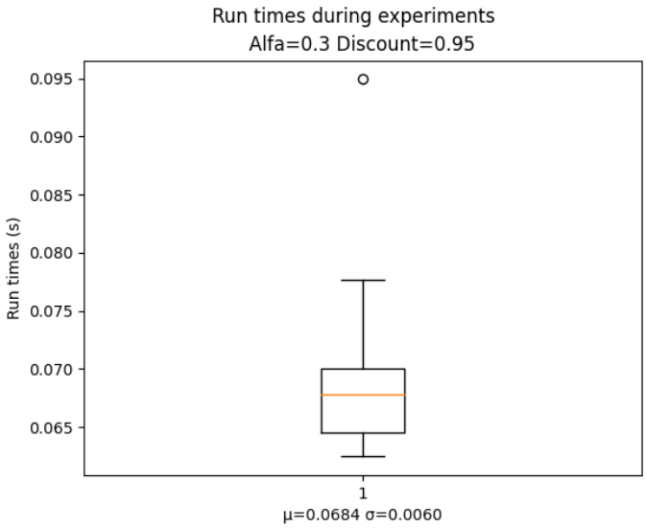
“Testes inteligentes” com níveis de ganância fixos em 60%:

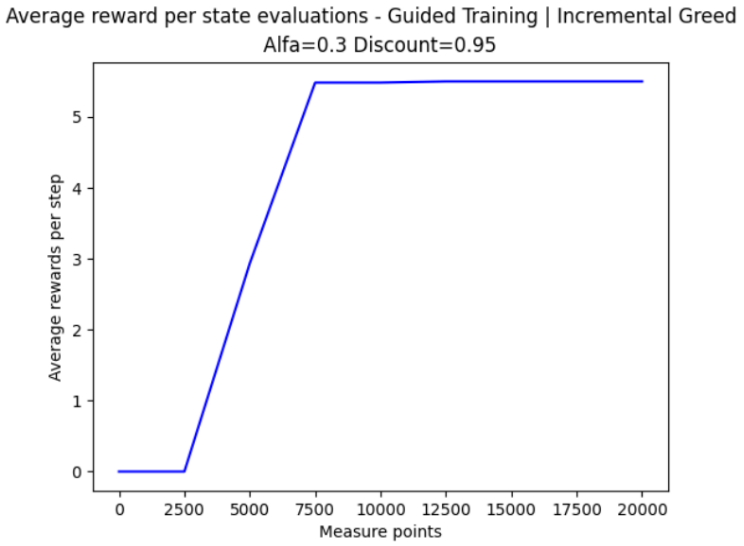


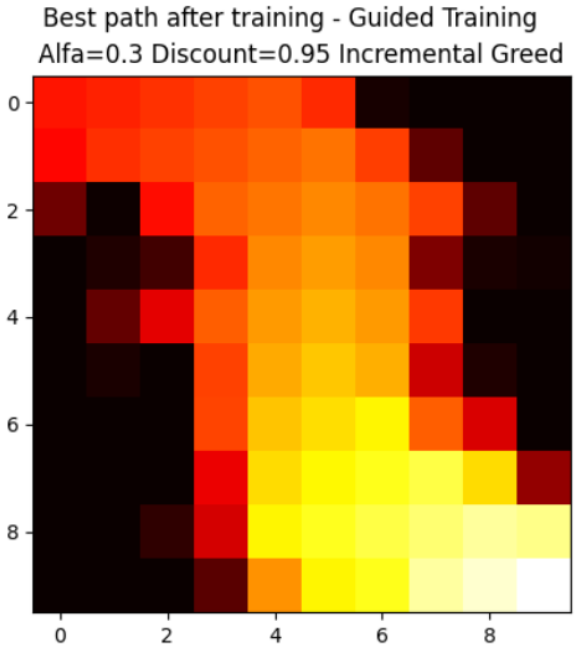




“Testes inteligentes” com níveis de ganância incrementais:







Analisando os efeitos, verifica-se que a introdução do conceito de ganância melhorou bastante os resultados dos treinos quando é escolhida sempre a "melhor ação", pois assim, impede que o robô fique preso em ciclos repetitivos durante tanto tempo. O robô consegue agora, em metade do tempo necessário de treino, alcançar resultados mais elevados do que na etapa anterior.

Pelo "Heat Map" é também possível constatar que o robô já não fica preso numa única solução no fim do seu treino, pois tem agora possibilidade de explorar caminhos alternativos, ou seja, já não existe apenas uma solução ótima.

Comparando também os resultados com diferentes níveis de ganância verifica-se que, quanto mais elevado este for, menos o robô é capaz de explorar a *maze* em procura de rotas novas e melhores do que as anteriormente encontradas, pois um nível de ganância

igual a 100% seria o mesmo que se tinha na etapa anterior, ou seja, o robô iria ficar preso ao conceito de escolher sempre a “melhor ação”.

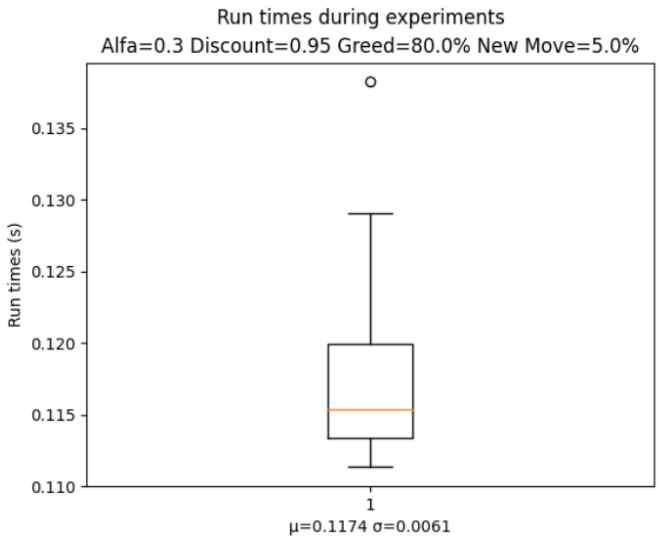
Já no caso em que o nível de ganância era incremental, conseguiram-se os melhores resultados, até agora, uma vez que, este conceito tira partido do melhor dos “dois mundos”, pois começa inicialmente o treino com valores de ganância baixos, é permitido ao robô explorar durante os primeiros 10.000 passos, com níveis inferiores a 65%, e já perto do fim do treino, valores de ganância mais elevados, garantindo que quase todos os movimentos seriam “os melhores”.

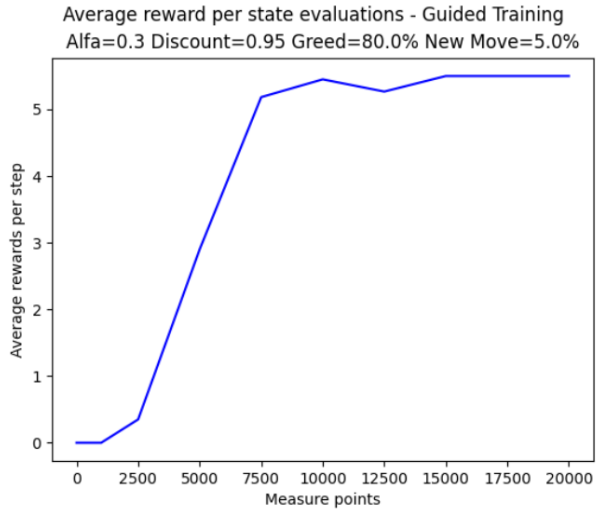
Ou seja, isto permite ao robô ganhar um melhor conhecimento da *maze* na primeira metade do treino, e depois na segunda metade, em que é assumido que o robô já tem um bom conhecimento da *maze*, é tentado tirar partido desse reconhecimento anterior, deixando o robô mais confiante com as suas escolhas, pois à partida, seria de esperar menos situações em que pudesse ficar preso em ciclos repetitivos.

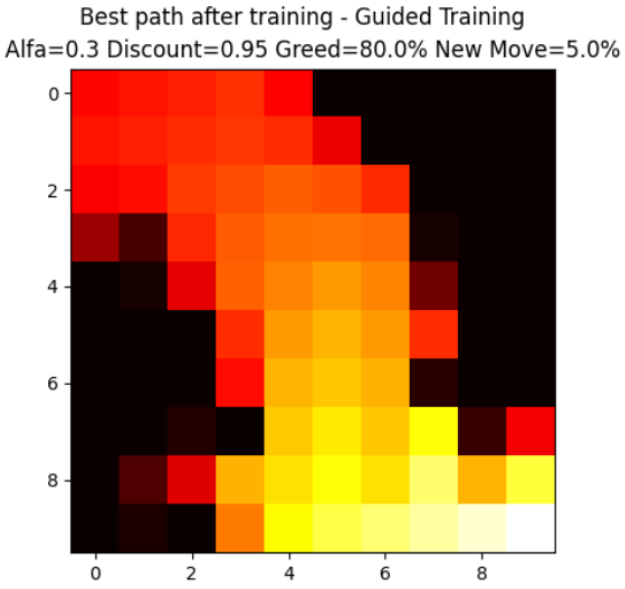
Exercício 7:

Adicionando nesta etapa mais uma nova forma de alterar a decisão de movimento do robô desta vez podendo, já depois de estar decidido o movimento seguinte, alterar-se a escolha para uma outra. Isto vem tornar o cenário atual da experiência num ambiente não-determinístico, ou seja, existe agora uma incerteza sobre o estado futuro do ambiente, mesmo tendo informações atuais.

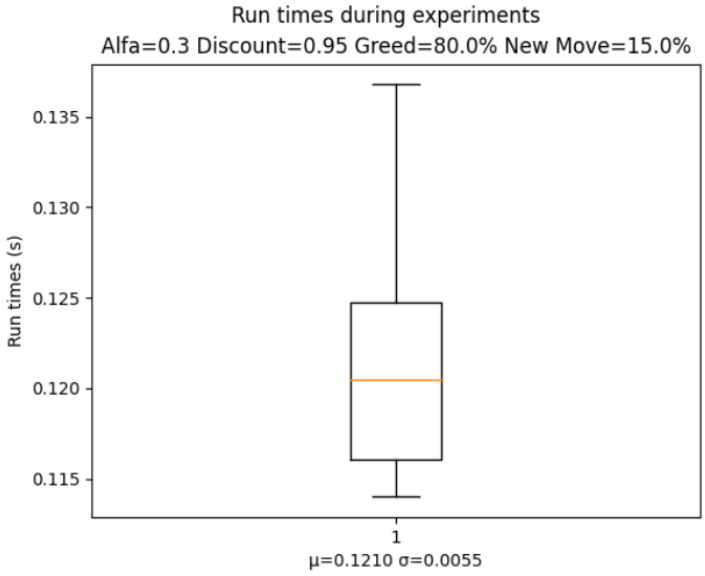
Testes inteligentes” com níveis de ganância fixos em 80% e possibilidade de nova ação de 5%:

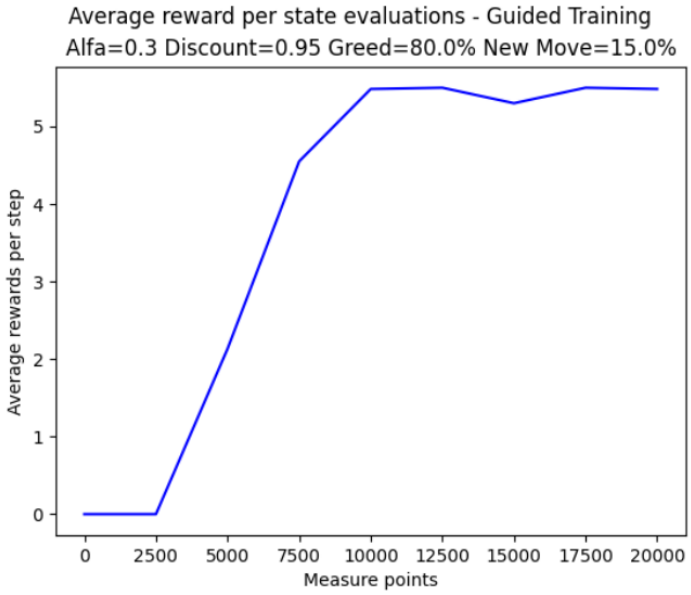


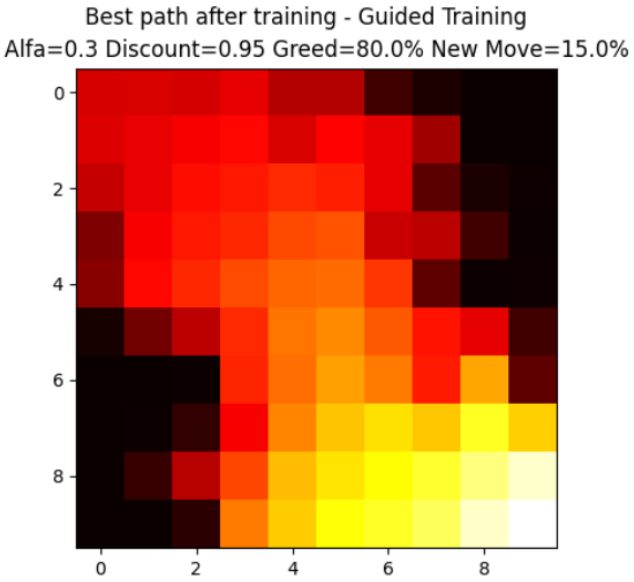




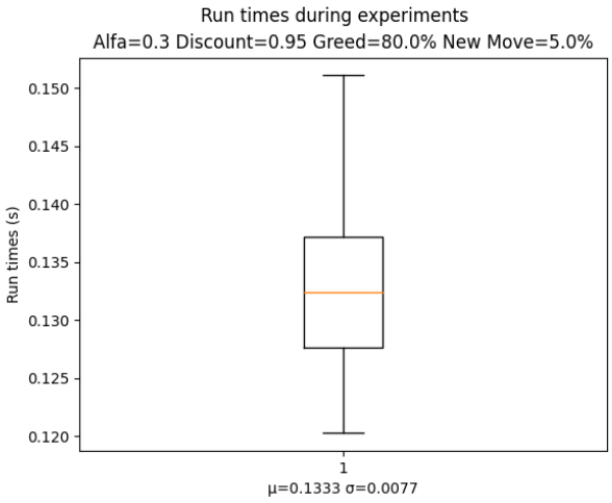
Testes inteligentes" com níveis de ganância fixos em 80% e possibilidade de nova ação de 15%:

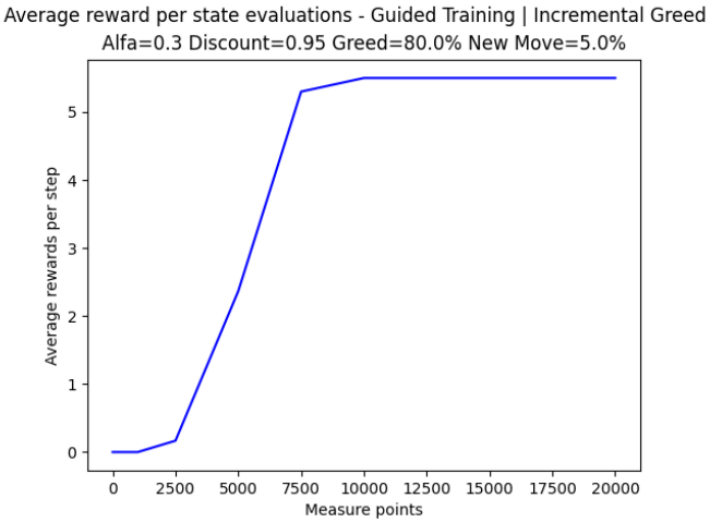


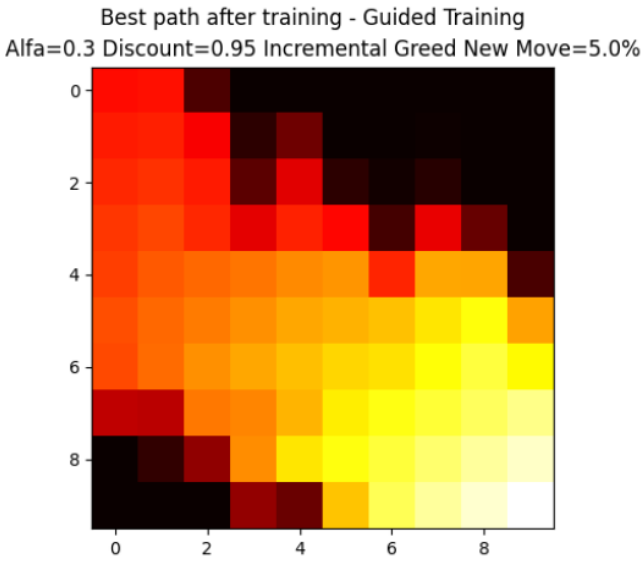




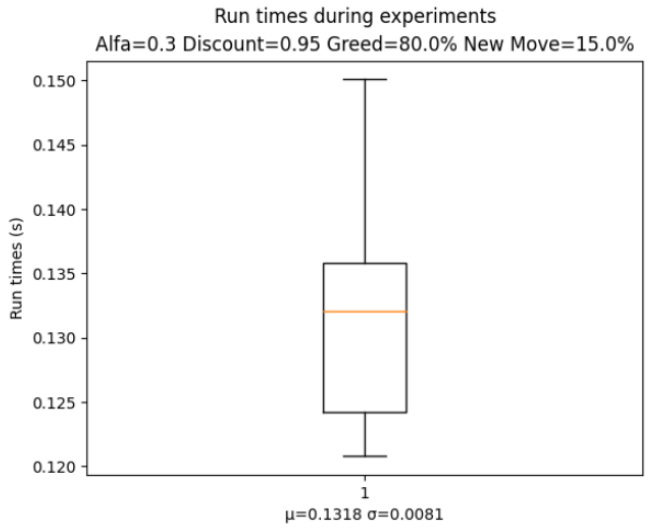
Testes inteligentes" com níveis de ganância incrementais e possibilidade de nova ação de 5%:

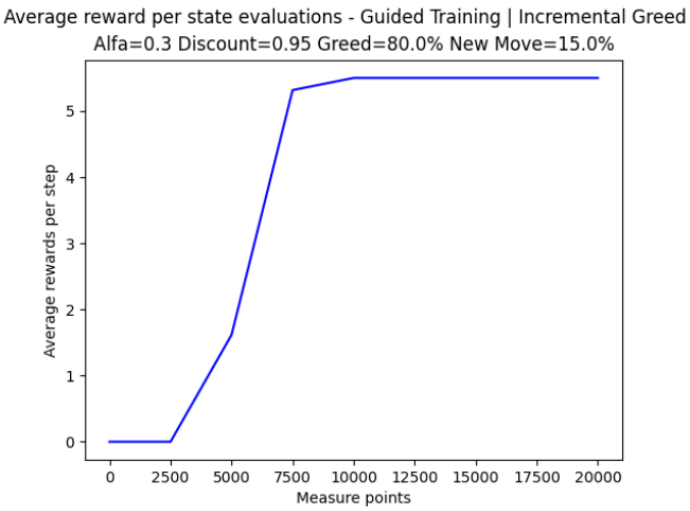


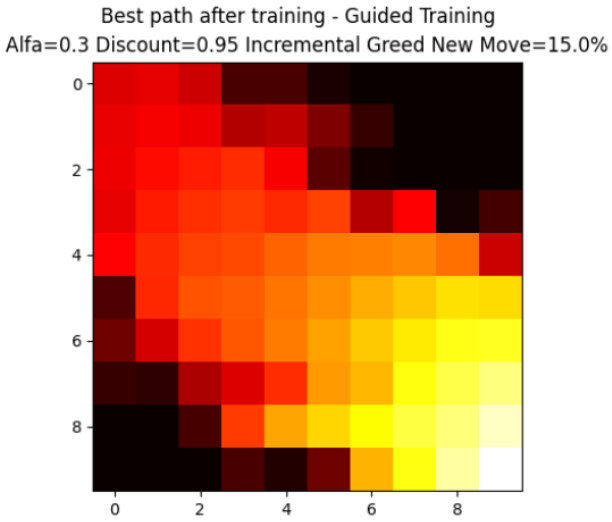




Testes inteligentes" com níveis de ganância incrementais e possibilidade de nova ação de 15%:







Analisando os resultados obtidos, verifica-se que a introdução de um sistema não-determinístico conseguiu resultados ligeiramente melhores em relação à etapa anterior nos casos em que a ganância assume valores fixos.

Quanto aos tempos de execução, nota-se que a cada etapa os tempos têm vindo a aumentar, comparando com os primeiros registos no caso do exercício 3, em que o sistema era todo aleatório, contudo, os resultados obtidos são cada vez melhores.

A introdução do sistema não determinístico também trouxe alterações aos caminhos tomados pelo robô, na sua missão de encontrar o *state* 100, uma vez que são notados caminhos mais dispersos e bastante longes daquilo que era a solução única obtida no exercício 5. Sendo que quanto maior o valor da probabilidade de uma alteração "à última da hora" do movimento escolhido, mais dispersas são as opções de rotas escolhidas pelo robô até ao objetivo.

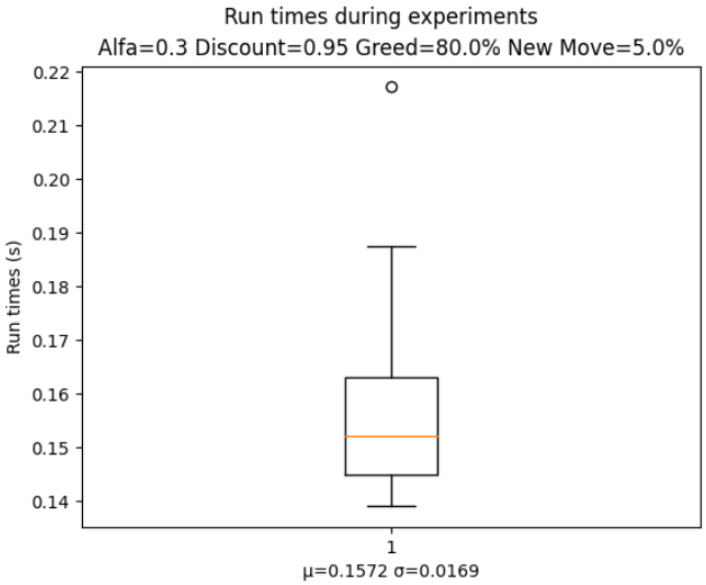
Exercício 8:

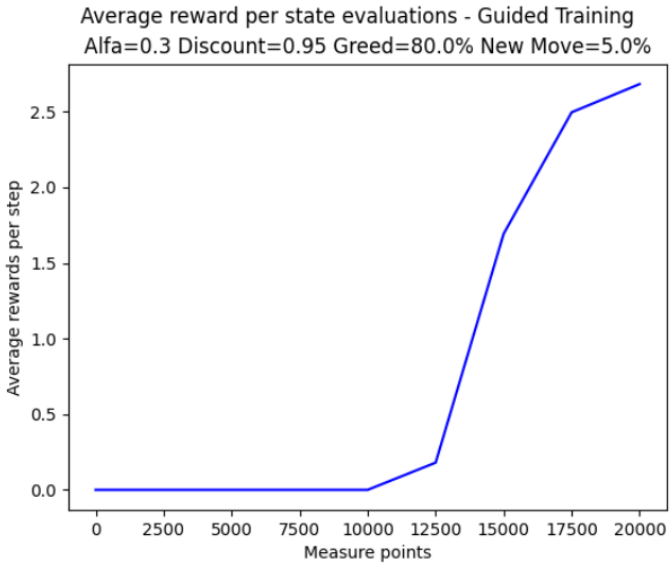
Nesta etapa, é alterado o aspeto da *maze* por onde o robô se desloca, substituindo um "campo aberto", por um com obstáculos, impedindo a existência de um caminho direto da casa partida ao objetivo.

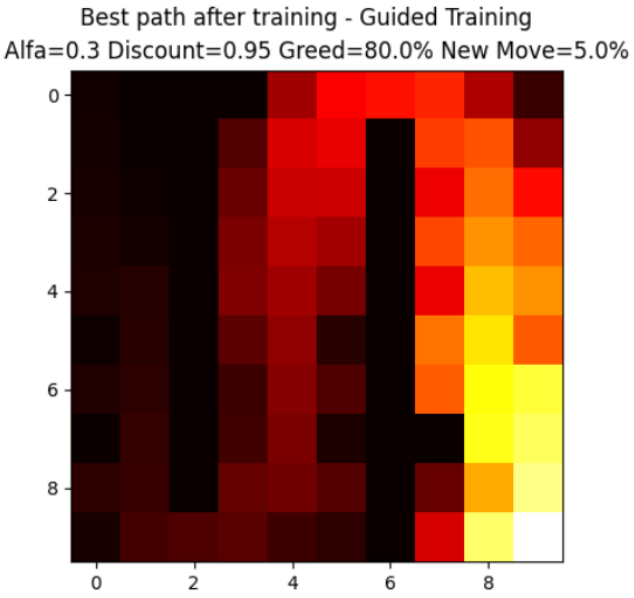
Assim, foram introduzidos dois novos formatos de *maze*, que incluem duas disposições diferentes de obstáculos (paredes), mas ambas possibilitando apenas um número limitando de rotas viáveis diferentes.

Sabendo que durante o seu treino o robô pode optar por escolher ações que levam a movimentos inválidos, anteriormente sendo estes, tentar sair fora da *maze*, com a existência de obstáculos a possibilidade de a ação escolhida levar a um *state* bloqueado levou a introdução de penalizações na recompensas sempre que o robô escolher uma ação que colida com uma parede, ou seja, nas etapas anteriores o robô ou obtinha 0 como recompensa ou 100 quando encontrava o objetivo, agora pode obter recompensas negativas. O valor escolhido para as recompensas negativas foi -0.5.

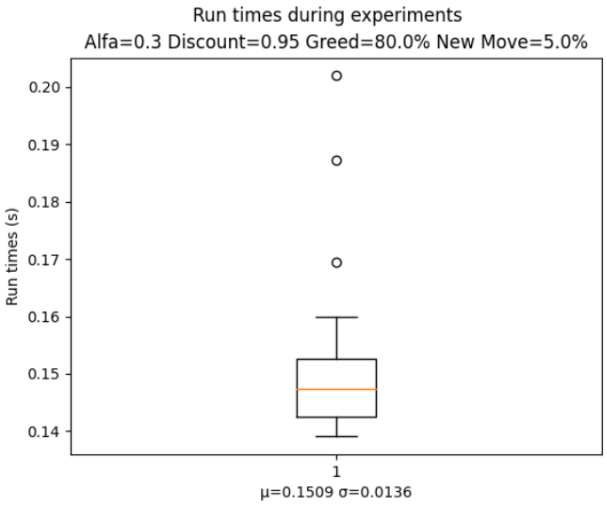
Formato 1, greed=80% new_move=5% alfa=0.3:

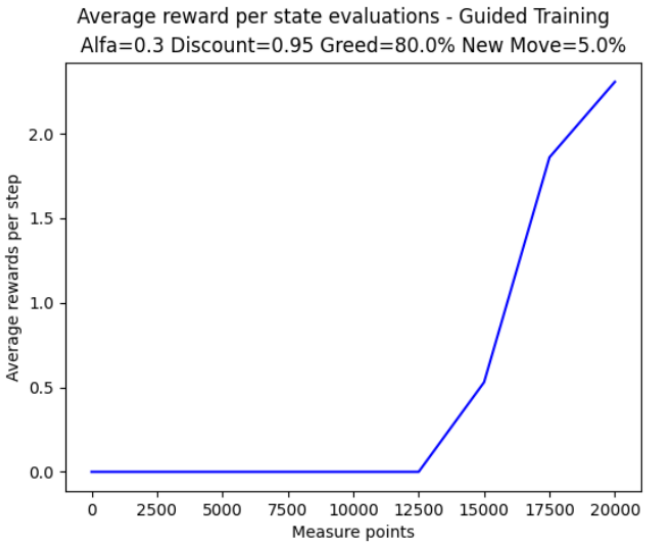


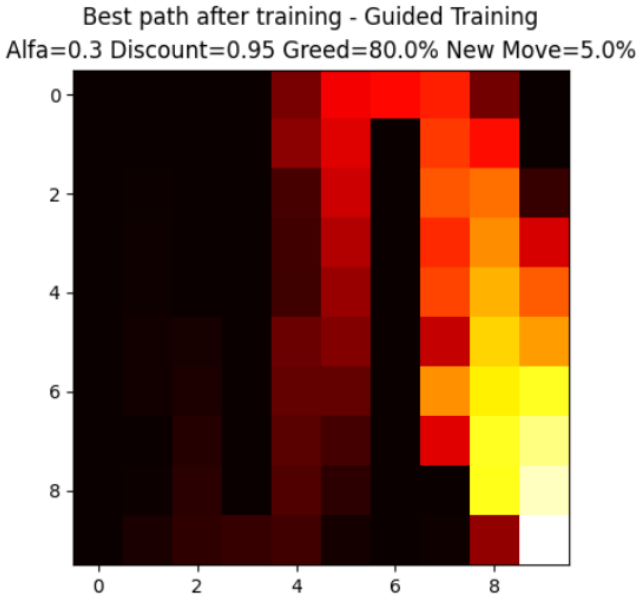




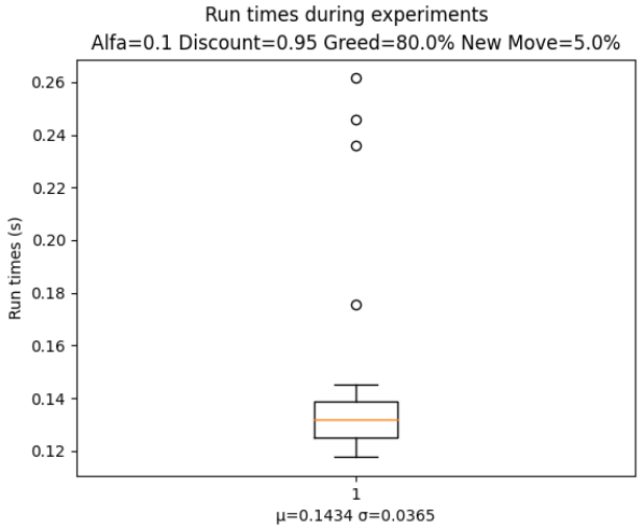
Formato 2, greed=80% new_move=5% alfa=0.3:

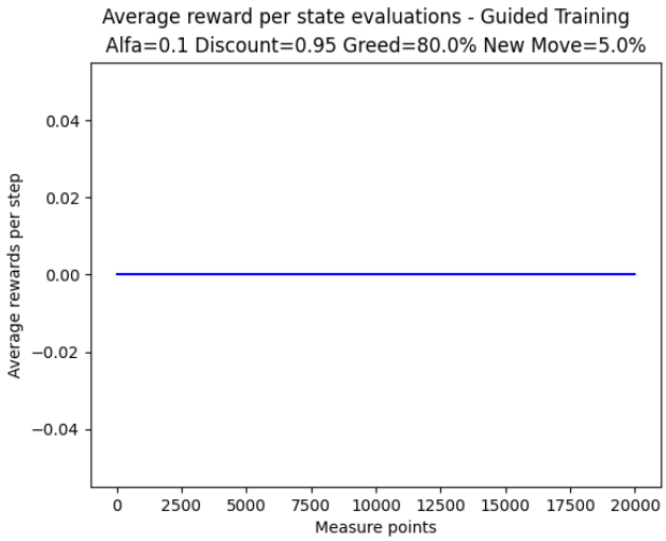


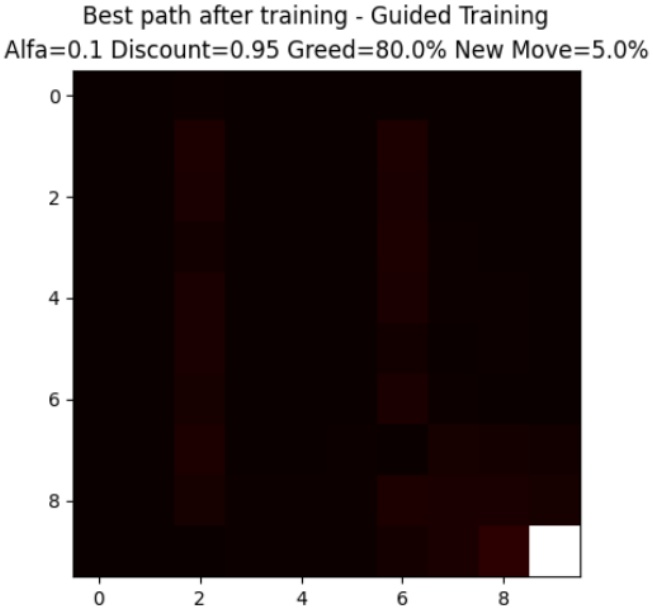




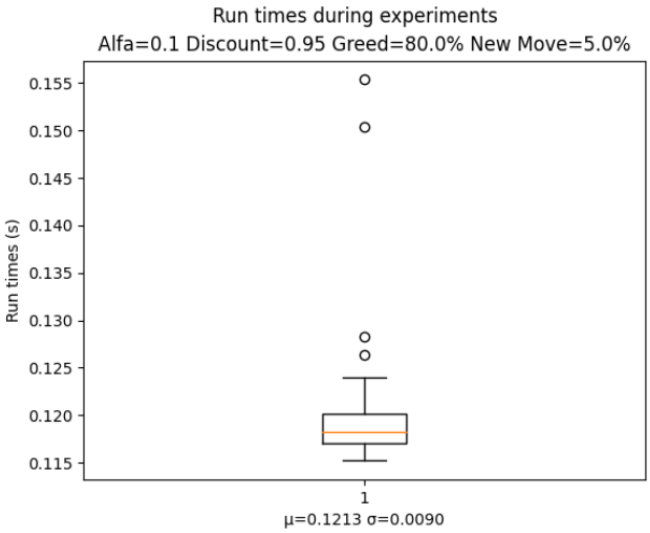
Formato 1, greed=80% new_move=5% alfa=0.1:

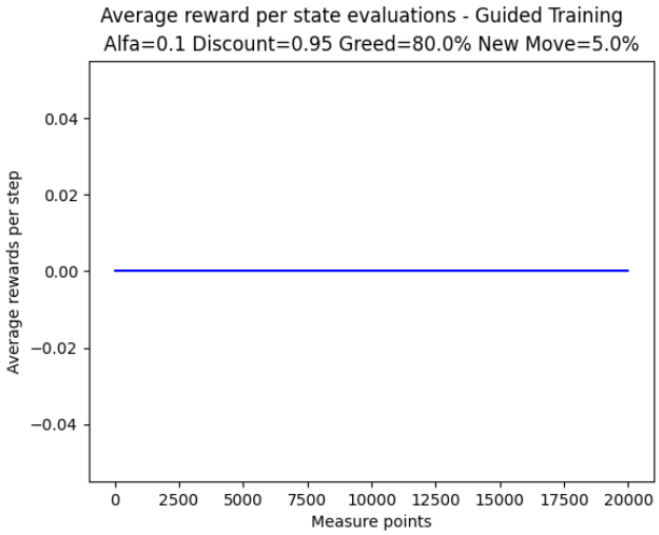


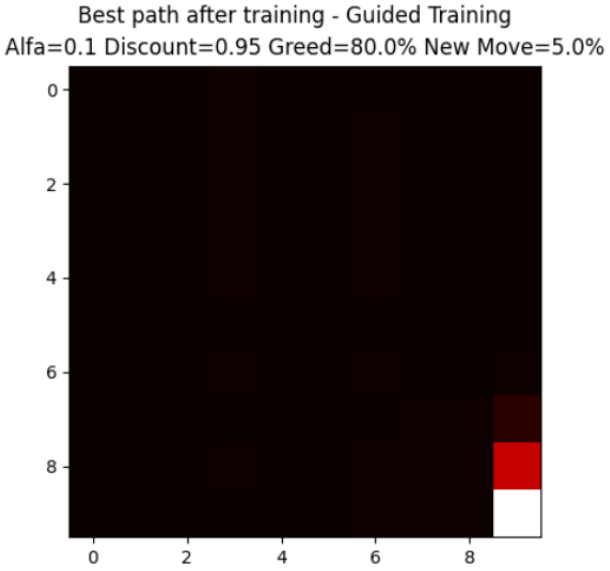




Formato 2, greed=80% new_move=5% alfa=0.1:







Com a adição de obstáculos pela *maze* e a consecutiva penalização por escolher uma ação que levasse o robô a colidir com uma parede, constatando-se que, agora, o robô despende inúmeros movimentos colidindo com paredes, sempre que os valores de utilidade gerados aleatoriamente no inicio da execução do programa que as mesmas têm, fizerem-nas parecer, na perspetiva do robô, a melhor ação, assim, até que esse valor seja suficientemente baixo ou pela decisão de movimento ter sido alterada por chance, o robô colidirá com a parede.

Isto levou a uma grande diminuição no número de vezes que o robô atinge o *state* 100 comparando com as mesmas características de testes que as etapas anteriores. Contudo, testes onde se varia o alfa para valores mais baixos, (0.3 → 0.1) leva a uma queda ainda mais drástica nos resultados obtidos, fazendo com que o robô, praticamente, nunca conseguisse atingir o objetivo. Uma vez que um alfa, cada vez mais perto de 0,

leva a uma aprendizagem mais lenta e consolidada, o que se torna num fator impeditivo, uma vez que o robô despende agora muito tempo em colisões com paredes.

Quanto aos tempos de execução estes por consequência também aumentaram novamente, registando os valores mais elevados de todas as etapas até agora.

Exercício 9:

Devido à maior exigência e dificuldade no nível de programação desta etapa, optou-se por não a implementar, mas apenas analisar e propor uma possível implementação.

Considerando este novo caso em que se tenta adotar um ambiente mais "realista", dois fatores têm de se ter em conta, a valor utilitário de cada ação, mas, também agora, a distância a que essa ação levaria o robô a ficar de uma parede.

Foi analisado nas etapas anteriores que nem sempre o valor utilitário de uma ação era indicador de um bom movimento, levando o robô a ficar preso em ciclos repetitivos e dispendiosos, especialmente na etapa anterior em que se introduziu o conceito de obstáculos.

Contudo, tentar levar o robô sempre para o mais longe de uma parede de modo a evitar que este colidisse com uma e obtivesse uma recompensa negativa, também não seria uma solução, pois chegar ao objetivo implica fazer uma trajetória perto das mesmas.

Assim, um equilíbrio tem de ser encontrado e uma possível abordagem seria definir uma regra que terá em conta a distância e o valor utilitário da ação e assim obter-se-á uma nova classificação para a ação. Para isso poder-se-ia definir que quando a distância a uma parede fosse pouca, mas o valor utilitário daquela ação era no mínimo 10% mais elevado que todas as outras, esta seria a escolhida, caso contrário, optar-se-ia por escolher a ação que de todas, tivesse o valor utilitário mais elevado, mas cuja distância fosse igual ao superior à média de todas as distâncias de todas as ações possíveis para um dado *state.*

Para obter-se a distância do robô a uma dada parede seria calculada a distância euclidiana da posição atual com a parede mais próxima que estivesse na direção de uma dada ação.

A nível de resultados, para este formato de ambiente seria esperado resultados superiores ao da etapa anterior, que apenas contava com o fator do valor utilitária de cada ação para definir a rota do robô e que se viu ser insuficiente.