Uma imagem com texto

Descrição gerada automaticamente

Reinforcement learning – a1rl

André Filipe Frade Guerra

Outubro/2022

Conteúdo

[Introdução 3](#_Toc115904842)

[Tabuleiro de jogo 3](#_Toc115904843)

[Classes e métodos 3](#_Toc115904844)

[Exercício 1 6](#_Toc115904845)

[Resultados exercício 1 7](#_Toc115904846)

[Exercício 2 8](#_Toc115904847)

[Resultados 2a 9](#_Toc115904848)

[Resultados 2b 10](#_Toc115904849)

[Exercício 3 13](#_Toc115904850)

[Resultados exercício 3 13](#_Toc115904851)

[Exercício 4 14](#_Toc115904852)

[Resultados exercício 4 14](#_Toc115904853)

## Introdução

Neste relatório irá ser explicado num resumo o funcionamento do tabuleiro de jogo, as classes criadas, funcionalidade dos métodos criados e por fim a execução e discussão dos resultados dos exercícios.

## Tabuleiro de jogo

O tabuleiro de jogo é constituído por uma matriz 12x12 que contem em cada posição o objeto “Position” com os seguintes atributos:

Uma imagem com texto, dispositivo, contador

Descrição gerada automaticamentestate é o número identificador da posição.

parede é um booleano que indica se a posição é uma parede.

reward é o valor da recompensa dessa posição.

O tabuleiro contém nas laterais e rodeado, posições com a característica parede = TRUE, fazendo com que seja impossível ir para essa posição.

## Classes e métodos

Para desenvolver os exercícios da primeira prática da disciplina de Introdução à Aprendizagem Automática foi desenvolvido um código em python usando o vsCode para realizar a mesma.



As seguintes classes foram criadas, sendo as 3 primeiras auxiliares para as seguintes, que são os exercícios resolvidos.

Começando pela classe “Funcs.py”, contem métodos que são comuns nos exercícios:

Uma imagem com texto

Descrição gerada automaticamenteMétodos usados para calcular o desvio padrão.

Uma imagem com texto

Descrição gerada automaticamenteMétodo que cria uma matriz 12x12 em que as linhas e colunas das extremidades têm a propriedade parede = True.

Para cada posição é definido um número.

Por fim é atribuído à posição de destino o valor da recompensa.

Uma imagem com texto, monitor, ecrã

Descrição gerada automaticamenteMétodo que defina aleatoriamente a próxima ação.

Uma imagem com texto, ecrã

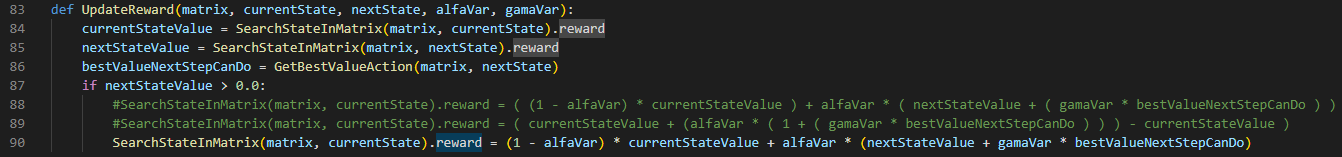
Descrição gerada automaticamenteMétodo que retorna a nova posição dependendo da ação realizada.

nota: tendo em conta que é uma matriz 12x12 as ações subir e descer subtraem e adicionam 12 à posição atual.

Uma imagem com texto

Descrição gerada automaticamenteMétodo IsParede valida se a posição é uma parede.

Método SearchStateInMatrix retorna a posição através da pesquisa pelo numero do estado.



Método UpdateReward atualiza a recompensa da posição atual tendo em conta a posição para onde vai a seguir.

Uma imagem com texto

Descrição gerada automaticamente

Método GetBestValueAction retorna a recompensa mais alta tendo em conta a atual posição.

Método GetBestStepAction retorna o estado para o qual a atual posição deve ir tendo em conta o valor da recompensa mais alta. Caso as recompensas em volta sejam todas iguais, é retornado um estado aleatório.

Uma imagem com texto

Descrição gerada automaticamenteMétodo que retorna uma matriz apenas com os valores das recompensas para construir o heatmap.

Uma imagem com texto

Descrição gerada automaticamente

Método que testa a alínea 2a.

## Exercício 1

Presente no ficheiro exer1.py, este ficheiro começa por declarar vários valores iniciais que definem o jogo, como cálculos, runs, steps e o tamanho da matriz.

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamenteEste pedaço de código faz correr a simulação do exercício 1 tendo em conta os runs e os steps por run.

Confirma se a próxima ação é uma parede, caso seja, volta a tentar outra ação aleatória.

Caso chega ao máximo de steps é declarada uma FAIL RUN.

Uma imagem com texto

Descrição gerada automaticamenteCódigo auxiliar para depois calcular as métricas pretendidas.

Uma imagem com texto

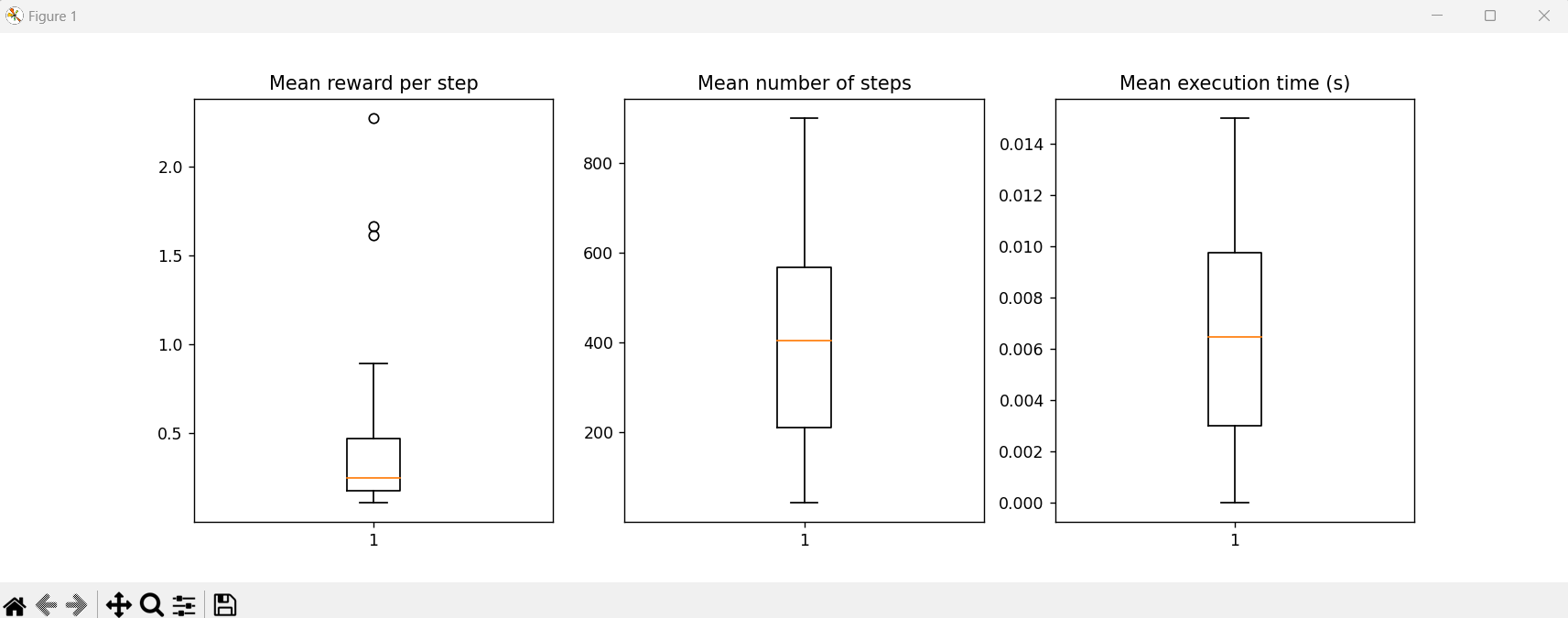
Descrição gerada automaticamente

Código que mostra os cálculos pretendidos no exercício, como também no final as boxplots pretendidas.

### Resultados exercício 1

Uma imagem com texto

Descrição gerada automaticamente



## Exercício 2

O exercício dois é uma cópia do primeiro exercício, com algumas alterações no pedaço de código que corre a simulação e os resultados. Os valores iniciados na classe também mudaram de maneira que agora os steps têm um valor de 20.000.

No código que corre a simulação foi adicionado o método UpdateReward que atualiza as recompensas a cada ação feita, e o Test1000 que corre a simulação a cada X steps, neste caso nos steps 100, 200, 500, 600, 700, 800, 900, 1000, 2500, 5000, 7500, 10000, 12500, 15000, 17500 e 20000.

Uma imagem com texto

Descrição gerada automaticamente

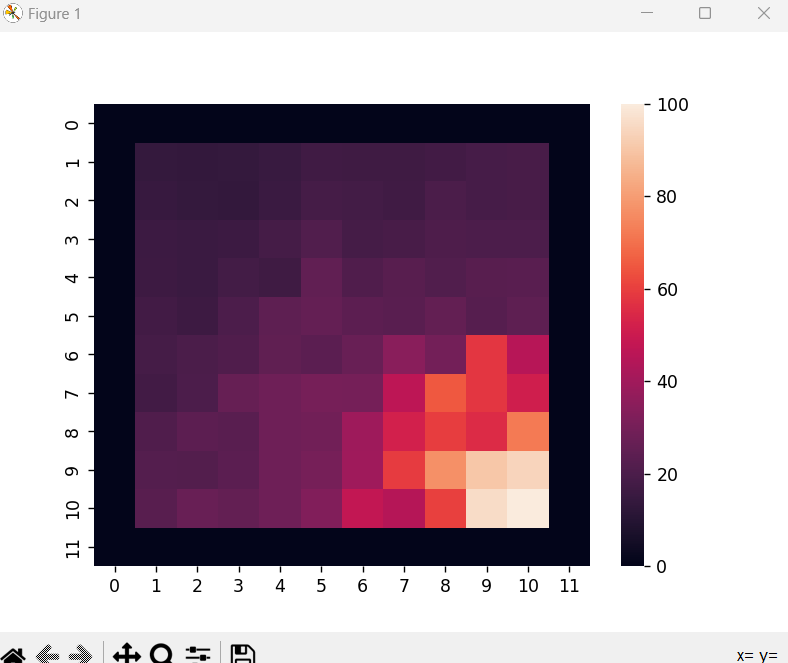
Uma imagem com texto

Descrição gerada automaticamente

Código auxiliar para calcular os valores pedidos e depois o gráfico do heatmap.

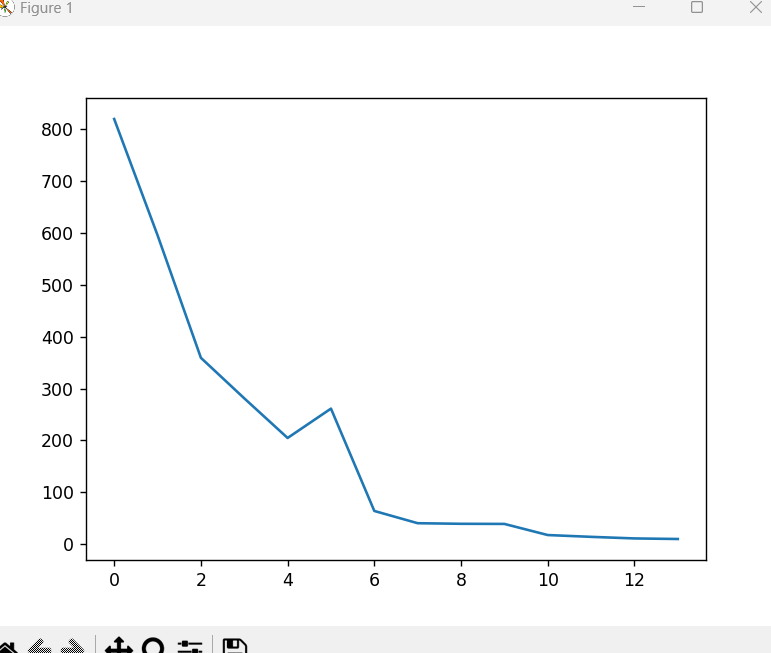
### Resultados 2a

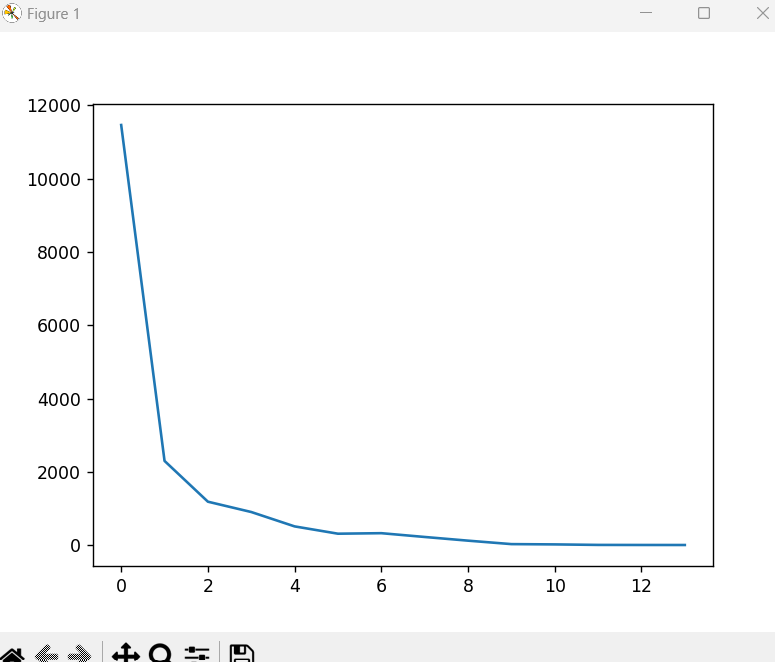




Resultado do heatmap com os valores das recompensas de cada posição.

Podemos observar este mapa a ser construído a cada run descometendo as linhas 57, 58 e 59.

Também podemos observar um gráfico para cada teste nos steps pretendidos descomentando as linhas 54 e 55, onde inicialmente vemos um gráfico que vai alterando ao longo de cada run:



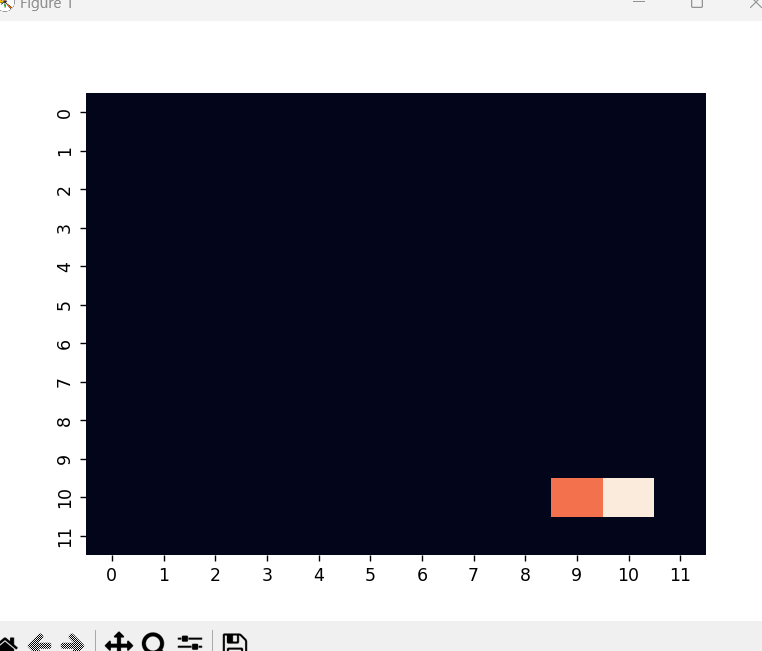
### Resultados 2b

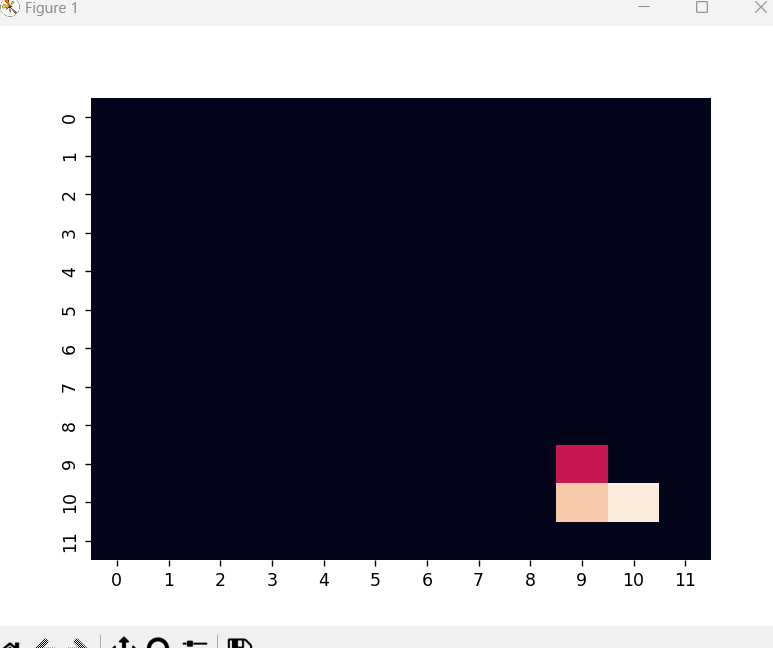
No exercício 2b a simulação foi alterada de maneira que escolha a melhor posição seguinte.

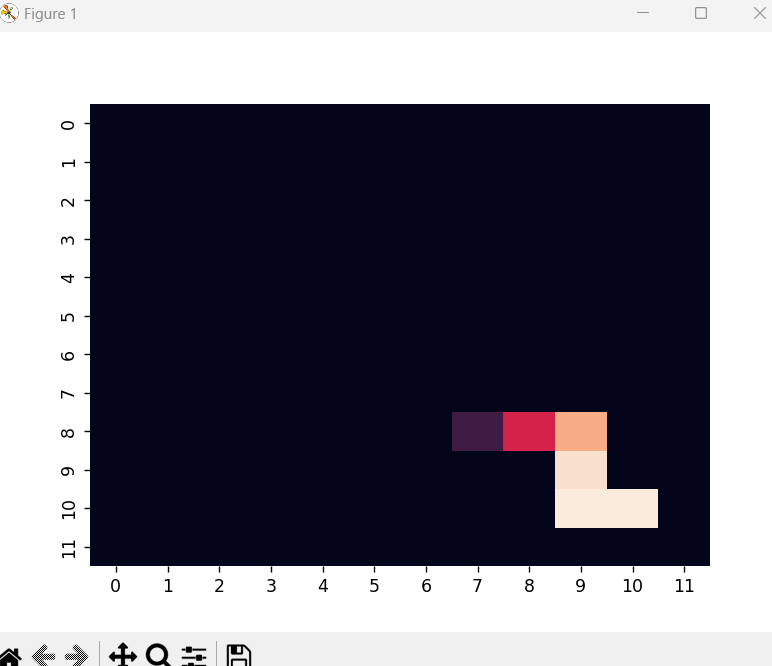
E para se obter resultados validos os runs foram alterados para 40 e os steps para 300.

Uma imagem com texto

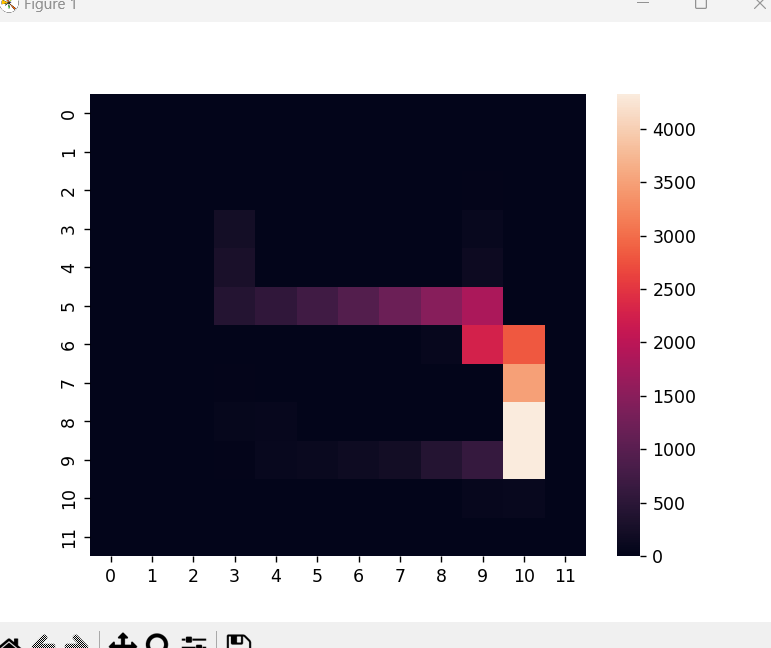
Descrição gerada automaticamente

Resultado da primeira run.

Resultado da segunda run.

Resultado quinta run.

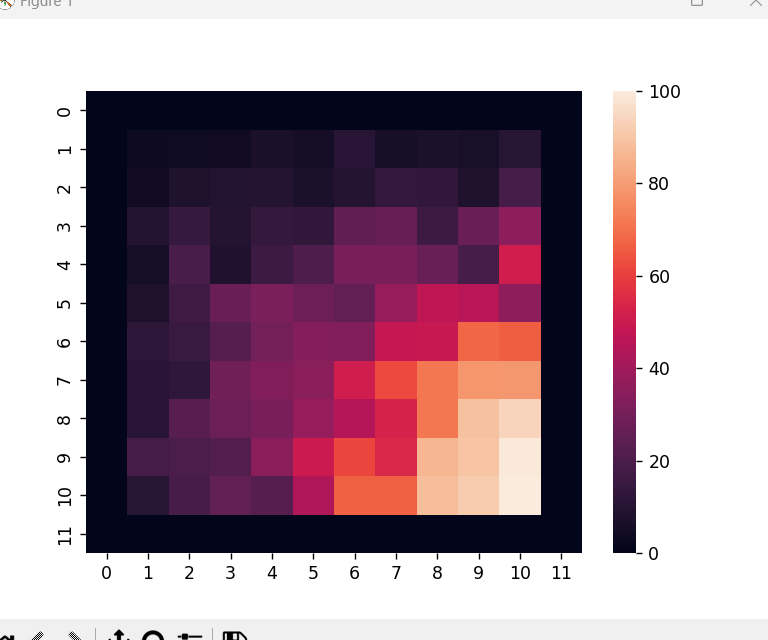
Podemos concluir que comparativamente à simulação onde não se escolhe a melhor opção seguinte, o resultado do melhor caminho possível é mais rápido a ser calculado.



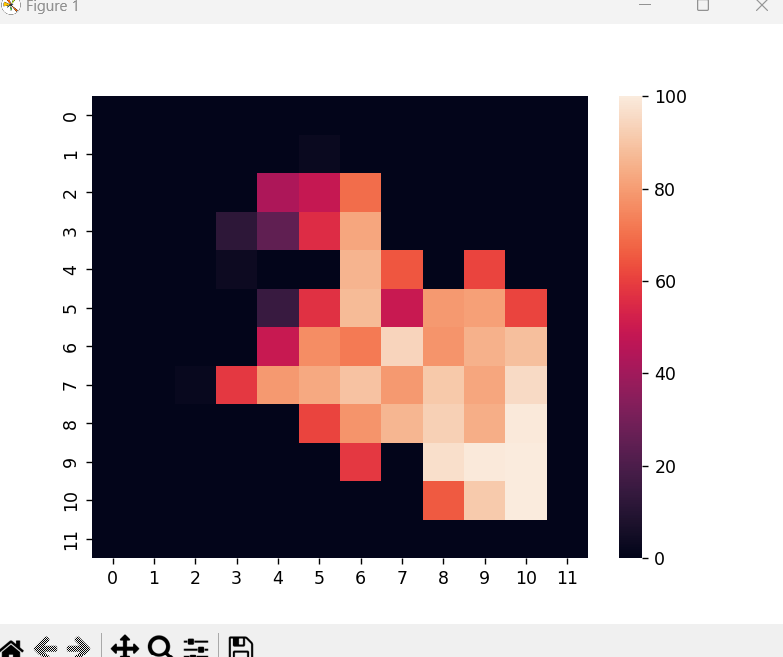
Resultado após os 40 runs com 300 steps cada uma.

## Exercício 3

No exercício 3 foi adicionada na simulação o cálculo probabilístico que permite escolher se a ação é aleatória ou não:



Neste gráfico a probabilidade de chamarmos o método StateTransition é de 90%, enquanto o método GetBestStepAction tem apenas 10%.



Ao alterarmos as probabilidades, tendo o StateTransition 10% e o GetBestStepAction de 90% de probabilidade, podemos observar diferenças.

### Resultados exercício 3

Podemos concluir que quando o método que escolhe a melhor ação seguinte, tem mais probabilidade de acontecer, o caminho acaba por ser mais linear. Ao contrário, quando o método que escolhe uma ação aleatório, tem mais probabilidade de acontecer, o caminho fica menos linear dispersando-se.

## Exercício 4

No exercício 4 foi criado um método para construir as paredes no tabuleiro e dar-lhes recompensas negativas:

Uma imagem com texto

Descrição gerada automaticamente

### Resultados exercício 4

Como podemos ver no heatmap as simulações chegaram ao caminho contornando as paredes criadas.

