Trabalho Prático 2 - Q-Learning
Inteligência Artificial 2019/1

Gabriel Henrique Souto Pires - 2014106384

1 Implementação

Este projeto foi implementado na linguagem Python versão 3.6.7 e testado em ambiente Linux versão Ubuntu 18.04.2 LTS. Foram criadas 2 classes, sendo uma para o tabuleiro e outra para o algoritmo de Q-Learning. Em um terceiro arquivo, os argumentos de execução são recebidos e os algoritmos executados.

1.1 Decisões de Projeto

Ao receber os argumentos, é criado um objeto da classe *Maze* que lê o arquivo contendo o labirinto e joga os caracteres que o compõem em uma matriz $n \times m$ sendo n o número de linhas e m o número de colunas do labirinto, ambas as dimensões também são passadas no arquivo de entrada. A classe *Maze* também tem um método que retorna uma posição aleatória válida no labirinto, ou seja, uma posição aleatória em que o pacman pode ficar, isso será útil mais tarde na hora de rodar o algoritmo de q-learning, uma vez que a posição inicial do pacman no labirinto é sempre aleatória. Existe também um método para mostrar o labirinto na tela, mas esse método foi utilizado apenas para fins de debug.

Na classe referente ao algoritmo de q-learning, primeiro guardamos os argumentos α, ε e n que são a taxa de aprendizado, o fator de exploração da estratégia ε-greedy e a quantidade de episódios que serão executados respectivamente, também é passado o labirinto lido anteriormente e uma posição aleatória é atribuída ao estado inicial pelo método *random_valid_positon*() da classe *Maze*.

Para criar a q-table, foi escolhido usar um dicionário ao invés de uma simples matriz com cada valor, o que tornou o processo de atualização da q-table muito mais rápido, aumentando a velocidade de execução do programa, uma vez que para se acessar uma posição específica usando um dicionário, isso é feito em $O(1)$, ao contrário da matriz que, para cada estado, seria necessário percorrer o vetor de ações até se encontrar qual deveria ser atualizada.

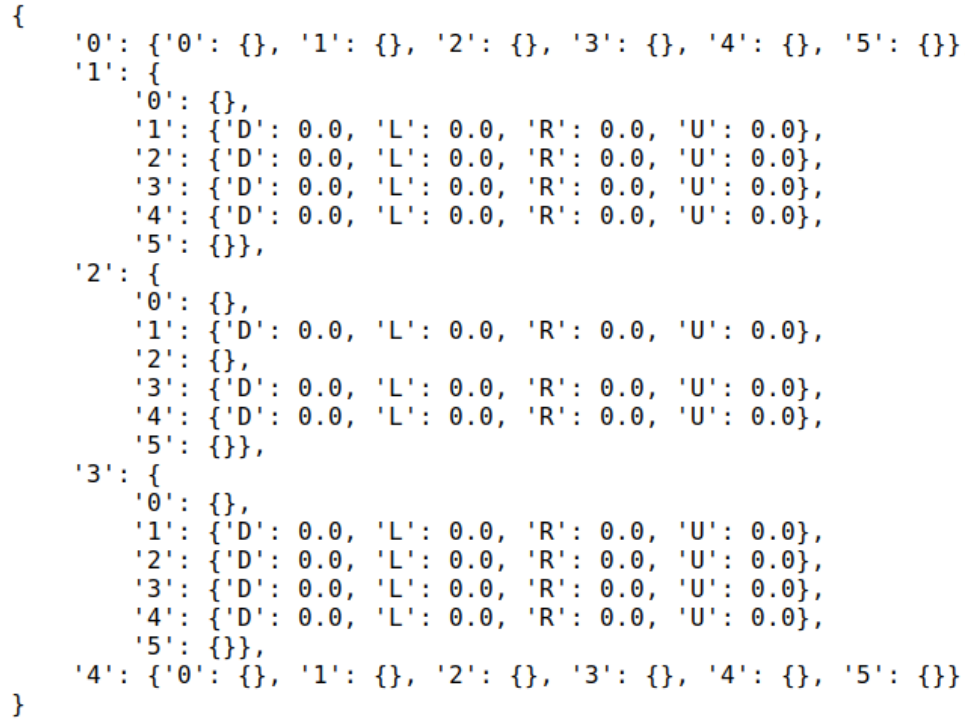

Figura 1: Estrutura da q-table. As primeiras chaves correspondem à linha em que cada estado se encontra, as chaves dentro dessa primeira correspondem à posição da coluna de cada estado, dentro dessa segunda chave ficam os pares de ação e recompensa de cada estado. Dessa forma é possível atualizar um valor na q-table em O(1), uma vez que cada estado e cada ação tem um índice que pode ser acessado diretamente.

Apesar de ser mais eficiente, o formato de saída do programa não é em forma de dicionário, então foi criada um método para formatar a q-table do jeito que foi pedido para o arquivo de saída. Como pode ser visto abaixo:

```
1,1,R,6.165
1,1,L,4.481
1,1,U,4.379
1,1,D,2.929
1,2,R,7.991
1,2,L,4.505
1,2,U,6.118
.... (demais pares estado-ação)
```

Cada tipo de estado do labirinto tem um valor de recompensa pré-definido, sendo que estados representados por '-' tem valor -1, estados representados por '0' tem valor 10 e estados representados por '&' tem valor -10. Os estados representados por '#' não tem valor de recompensa, uma vez que não é possível andar sobre eles.

Para cada movimento do pacman, a q-table é atualizada seguindo a seguinte fórmula:

$$Q[state, action] = Q[state, action] + alpha * (reward + gamma * max(Q[new\_state, actions]) - Q[state, action]$$

onde $reward$ é a recompensa do estado para o qual se está tentando ir,

$max(Q[new\_state, actions])$ é a recompensa máxima do próximo estado e $Q[state, action]$ é a recompensa do estado em que o pacman está atualmente. Obviamente a fórmula acima é apenas uma abstração, no código são usadas outras estruturas e funções que não cabe mostrar aqui.

Após se atualizar a q-table, é testado se o pacman se encontra em um estado final (0 ou &), em caso positivo, o episódio atual é dado como terminado e o pacman é colocado de volta no labirinto em uma nova posição aleatória. Isso é feito n vezes até que o número de episódios passados como argumento sejam concluídos.

Para se controlar a taxa de exploration e exploitation, foi usada a estratégia ε-greedy, que consiste em fixar um número ε de 0 a 1 que representa a probabilidade de se executar movimentos aleatórios. Geramos então um número aleatório no mesmo intervalo, toda vez que o pacman for realizar um movimento no labirinto, se esse número for menor que o nosso epsilon, o pacman irá explorar o ambiente, realizando um movimento aleatório, se o número gerado for maior que o epsilon, o pacman ira escolher a direção que tem maior recompensa, daí o nome ε-greedy.

No final da execução de todos os episódios, podemos usar a q-table para gerar uma política, que consiste em um mapa com a ação ideal que o pacman deve executar a partir de cada estado, baseado nas recompensas a partir de cada um. Vamos ver as políticas geradas para cada caso de teste a seguir.

2 Saídas obtida

2.1 Caso 1 - Labirinto 5 por 6

Esse caso teste foi incluído junto à especificação do trabalho e serviu de base até que outros testes fossem disponibilizados. É um caso bem simples que contém apenas um estado final bom e um ruim.

2.1.1 Entrada

```
5 6
######
#---0#
#-#-&#
#----#
######
```

2.1.2 Saída

```
######
#RRR0#
#U#U&#
#URUL#
######
```

Neste caso, notamos que a taxa de aprendizado é mais importante que a quantidade de exploitation ou exploration que o agente realiza, como fica evidente no gráfico abaixo. As 3 linhas que mais demoram a convergir são justamente aquelas em que o valor de α foi colocado em 0.1, pouco importando o valor de ε, apesar de ficar evidente que nesse caso realizar menos exploração é mais vantajoso.

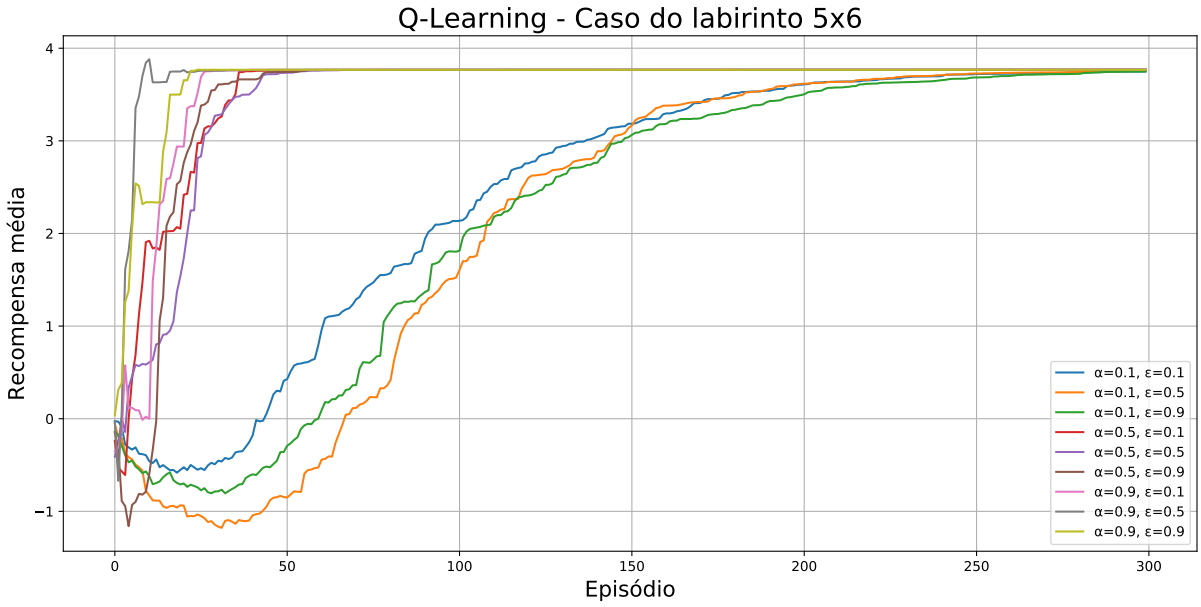

Figura 2: Recompensas médias ao longo do tempo no labirinto de tamanho 5x6

2.2 Caso 2 - Labirinto 7 por 14

2.2.1 Entrada

```
7 14
##############
#-----0------#
#######-##-#-#
#-------&--#-#
#-##########&#
#------------#
##############
```

2.2.2 Saída

```
##############
#RRRRR0LLLLLL#
#######U##U#U#
#RRRRRRU&RU#U#
#U##########&#
#ULLLLLLLLLL#
##############
```

Novamente fica claro que a taxa de aprendizado é muito importante para a velocidade de convergência. Mas dessa vez podemos ver também que em casos com muita exploração (a linha marrom por exemplo), o valor médio das recompensas tende a cair mesmo depois de vários episódios passarem, isso se deve ao fato de o agente escolher caminhos ruins aleatoriamente, e eventualmente cair em estados finais com recompensas muito baixas. Mas à medida que o tempo passa, uma vez que a q-table é ajustada melhor, o agente começa a tomar caminhos melhores, evitando cair em estados que prejudicam a recompensa média.

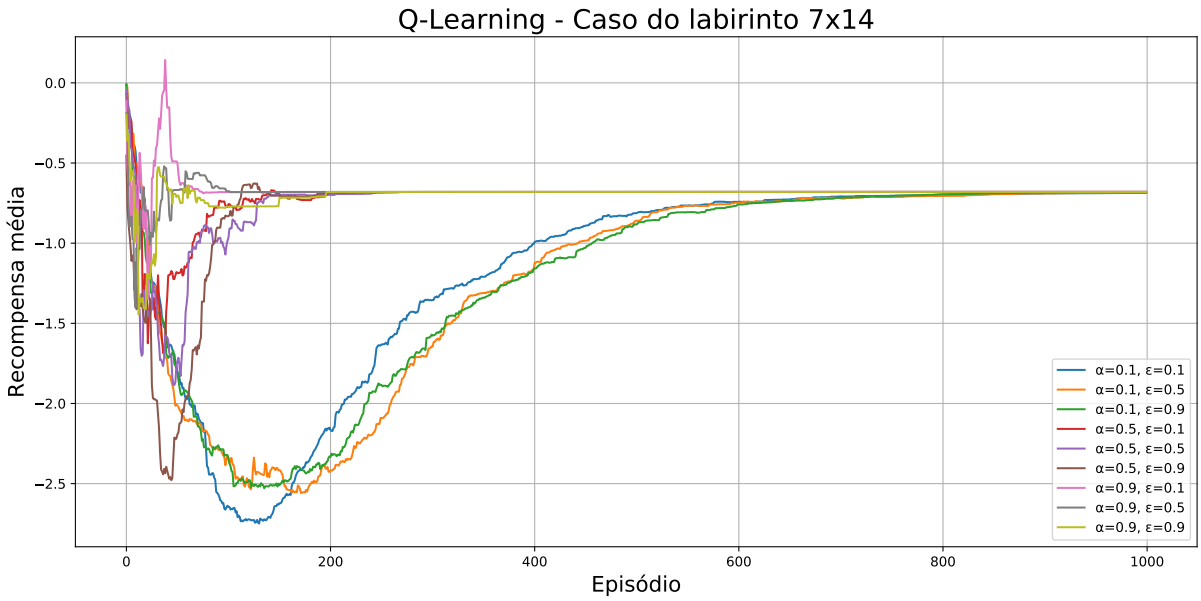

Figura 3: Recompensas médias ao longo do tempo no labirinto de tamanho 7x14

2.3 Caso 3 - Labirinto 14 por 20

2.3.1 Entrada

```
14 20
####################
#--------------#---#
#---&--&--&----#---#
#--------------#-#-#
#-###############-#-#
#----------------#-#
###################-#
#--------&---------#
#-&--########---&--#
#-----------------#
#-################-#
#-#-----#0#--------#
#----#----#--------#
####################
```

2.3.2 Saída

```
####################
#DLLLLLLLLLLLL#RRD#
#DLL&DL&DL&DLLL#RRD#
#DLLLLLLLLLLLL#U#D#
#D##############U#D#
#RRRRRRRRRRRRRU#D#
#################D#
#DLLLLLL&RRRDLLLLL#
#D&DL########DLL&DL#
#DLLLLLLLLLLLLLLLL#
#D################U#
#D#RRRRD#0#RRRRRRU#
#RRRU#RRRU#RRRRRRU#
####################
```

Esse é um caso interessante, no sentido que ao invés da recompensa média aumentar, ela começa em zero e só cai à medida que os episódios passam. Isso é esperado, uma vez que, o labirinto tem apenas um estado com recompensa positiva, e este está em uma posição muito difícil de encontrar, além de ser um labirinto relativamente grande e com muitos estados finais com recompensa negativa. À medida que os episódios passam, o agente só consegue encontrar estados finais ruins ou até mesmo explorar o labirinto sem chegar a nenhum estado final, o que prejudica a recompensa média. Assim como nos outros casos, pode-se notar que os casos com baixa taxa de aprendizado demoram mais para convergir.

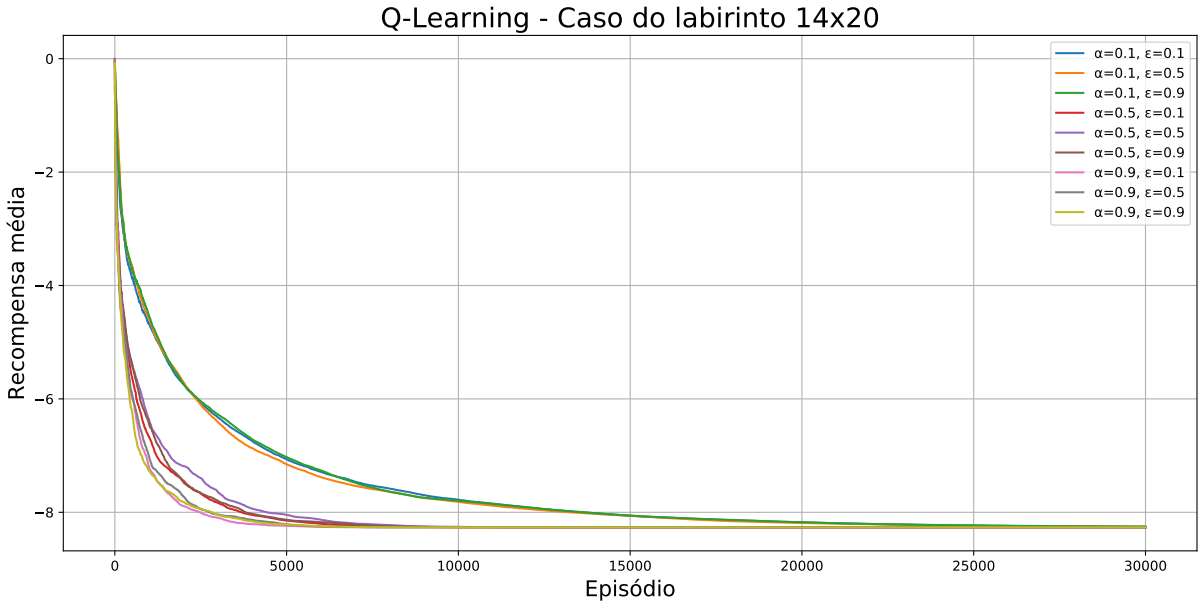

Figura 4: Recompensas médias ao longo do tempo no labirinto de tamanho 14x20

2.4 Caso 4 - Labirinto 11 por 20

2.4.1 Entrada

```
11 20
####################
#----&--------&----#
#-&&&&&&&-&&&&&&&&-#
#-------&-&--------#
#---&---&-&---&----#
#-------&-&--------#
#-----&-----&------#
#-------&-&--------#
#---&---&-&---&----#
#-------&0&--------#
####################
```

2.4.2 Saída

```
####################
#DLLL&RRRDLLLL&RRRD#
#D&&&&&&&D&&&&&&&&D#
#RRRRRRD&D&DLLLLLLL#
#RRD&RRD&D&DLL&DLLL#
#RRRRRRD&D&DLLLLLLL#
#RRRRU&RRDLL&ULLLLL#
#RRRRRRU&D&ULLLLLLL#
#RRU&RRU&D&ULL&ULLL#
#RRRRRRU&0&ULLLLLLL#
####################
```

Esse caso é muito semelhante ao anterior, porém, em um labirinto menor. Aqui vemos a mesma tendência de começar com a média das recompensas em zero e cair, mantendo o valor negativo. Apesar de esse labirinto aparecer muito mais ameaçador do que o anterior, com as “paredes” praticamente todas feitas de estados finais ruins, deixando o estado bom rodeado deles, aqui o agente tem a chance de aprender sobre o labirinto, o que não acontecia no labirinto anterior, já que o agente ficava a maior parte do tempo preso nas paredes sem conseguir se movimentar. Isso faz com que o agente comece muito mal, caindo em estados ruins, daí a queda abrupta visível no início do gráfico, porém, à medida que o agente aprende onde não deve ir, fica muito mais fácil seguir por caminhos mais seguros. Novamente as execuções onde a taxa de aprendizado é alta são os que convergem mais rapidamente.

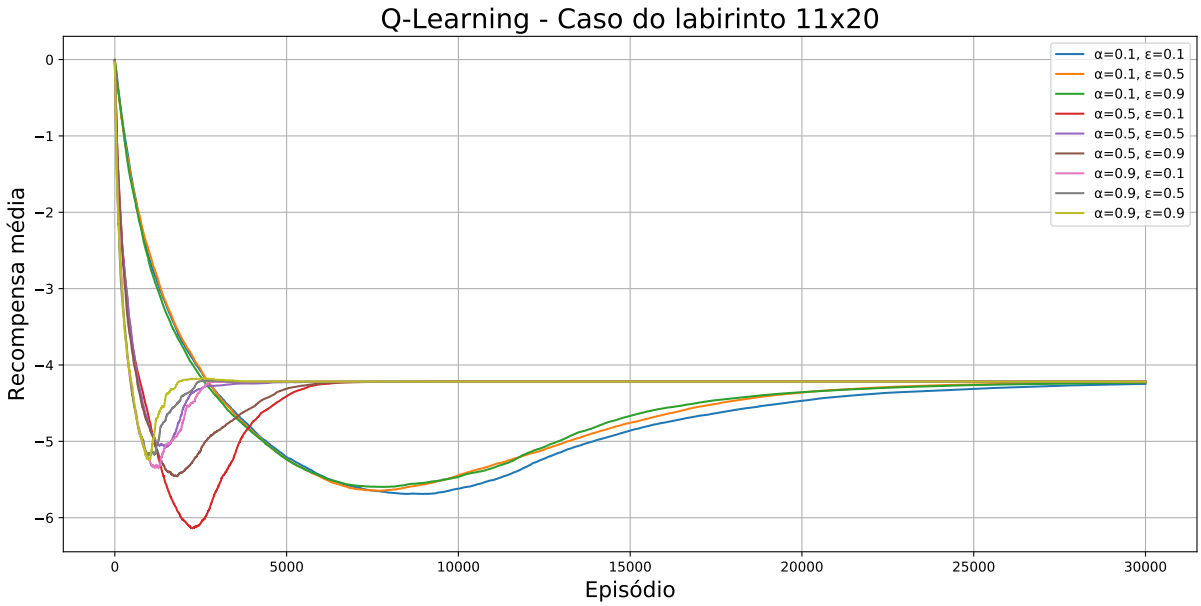

Figura 5: Recompensas médias ao longo do tempo no labirinto de tamanho 11x20

3 Principais dificuldades

A tarefa que mais tomou tempo nesse trabalho foi a elaboração dos gráficos, mas creio que tenha valido a pena, pois, com os gráficos, foi possível visualizar muito melhor como o algoritmo se comporta variando os argumentos e o ambiente em que o agente se encontra.

4 Referências

Simple Reinforcement Learning: Q-learning. Disponível em: <https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56>. Acesso em: 27 jul. 2019.