Aprendizagem em jogo do tipo solitário
(Folding Blocks)

Nuno Marques
Inteligência Artificial
FEUP
Porto, Portugal
up201708997@fe.up.pt

Ricardo Ferreira
Inteligência Artificial
FEUP
Porto, Portugal
up200305418@fe.up.pt

***Abstract*—Neste documento será apresentada uma comparação entre dois métodos de aprendizagem por reforço para ensinar um agente a jogar o jogo *Folding Blocks*. O objectivo do jogo consiste em preencher um tabuleiro de jogo duplicando as formas das peças existentes. O agente será ensinado com recurso aos algoritmos Q-Learning e SARSA. Todo o trabalho foi realizado na linguagem de programação Python, o que permitiu a utilização da framework Open AI Gym, especifica para aplicação de algoritmos de aprendizagem por reforço.**

***Index Terms*—Aprendizagem por reforço, Q-Learning, SARSA, Folding Blocks**

I. Introducão

Este trabalho procura aplicar técnicas de aprendizagem por reforço para ensinar um agente a jogar o jogo Folding Blocks, [1]. Aprendizagem por reforço consiste em ensinar um agente a agir, escolhendo acções em troca de recompensas, para no final maximar o número da recompensa. Ao agente não é dito o que fazer, este deve descobrir quais as acções que levam às recompensas. As acções irão afectar não só a recompensa actual, como o próximo estado e logo todas as recompensas futuras. Neste trabalho o agente irá tentar maximar a recompensa, tentando terminar o jogo no menor número de jogadas possíveis, uma vez que apenas o estado final, tabuleiro totalmente preenchido, terá uma recompensa positiva. A forma como a escolha das acções é feita é a característica diferenciadora dos algoritmos de aprendizagem por reforço, [2]. Neste trabalho são propostos dois algoritmos para realizar a aprendizagem: *Q-Learning* e *SARSA*. *Q-Learning* é um algoritmo considerado off-topic, ou seja, apesar de existir uma política para a passagem para o estado seguinte, o algoritmo vai sempre fazer as escolhas mais gulosas e que conduzam aos resultados mais rapidamente. Por outro lado, *SARSA* é um algoritmo on-topic, o que significa que valoriza as acções definidas na política e como tal, se a política convergir para o estado final, o algoritmo acabará por chegar lá. Podemos assim dizer que um algoritmo valoriza mais os estados, *Q-Learning*, e o outro valoriza mais as acções, *SARSA*, [2].

A. Descrição do jogo folding blocks

Folding blocks é um jogo que consiste em preencher um tabuleiro com blocos. O tabuleiro incialmente terá apenas um bloco que pode ser duplicado para formar um novo bloco e posteriormente duplicado novamente até o tabuleiro ficar completamente preenchido. Quando um bloco é duplicado, resulta num novo objecto e é esse novo objecto que pode ser novamente duplicado. Para uma melhor compreensão segue um exemplo nas imagens 1 até 3

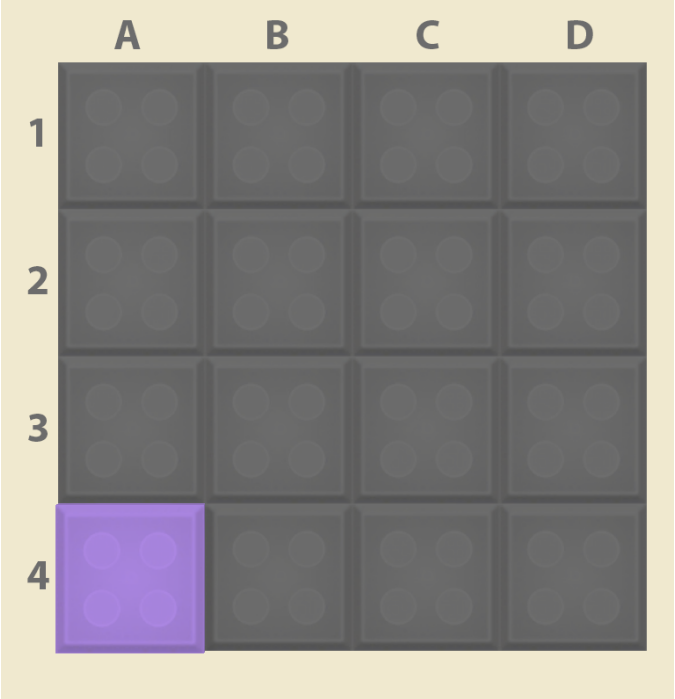

Fig. 1. Exemplo de um tabuleiro de jogo inicial

A figura 1 mostra o estado inicial de um possível jogo, um tabuleiro vazio apenas com um bloco na célula A4. O objectivo será preencher todo o tabuleiro, para isso podemos duplicar o bloco existente formando um rectângulo entre as céclulas A4 e A3 ou A4 e B4. A segunda opção está representada na figura 2.

No estado da figura 2, a peça do jogo é o rectangulo A4-B4 e deve ser esta peça a ser duplicada, neste caso, formando um quadrado entre A3 e B4 ou um rectângulo entre A4 e D4, tal como mostra a figura 3

Continuando a duplicar o objecto resultante é possível preencher todo o tabuleiro e terminar o jogo.

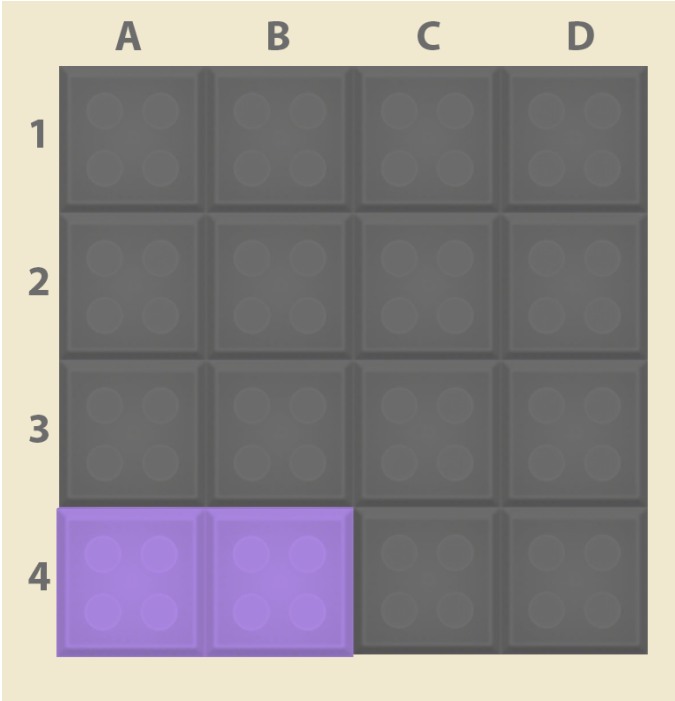

Fig. 2. Exemplo de um tabuleiro de jogo após uma jogada

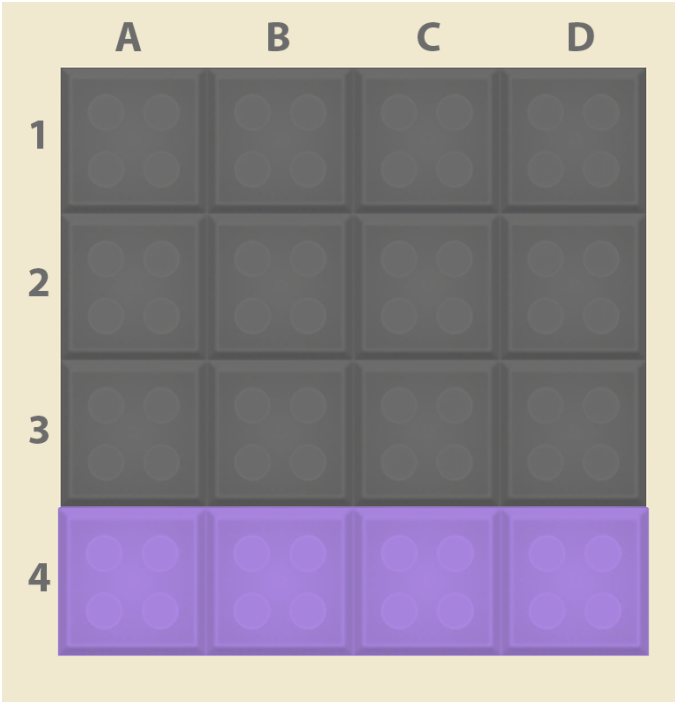

Fig. 3. Exemplo de um tabuleiro de jogo após duas jogada

II. Descrição do problema

Aprendizagem por reforço é aprendizagem focada num objectivo e conseguida através de interacção com o ambiente. No caso específico deste trabalho o ambiente serão os diferentes tabuleiros de jogo e os diferentes estados por que passam até chegarem ao estado final. O agente irá explorar o ambiente em troca de recompensas. Uma vez que os agentes escolhem as acções sem qualquer tipo de intervenção devem procurar as acções que levam a maiores recompensas. As acções por sua vez afectam não só a recompensa imediata mas também situações futuras e próximas recompensas e é este equilibrio entre escolher acções que se sabe que terão uma recompensa e descoberta de acções que levem a estados com recompensa que deve ser feito pelos algoritmos de aprendizagem por reforço. Outro aspecto interessante da aprendizagem por reforço é o facto de não ser necessário um modelo exacto do ambiente, uma vez que o agente irá explorar o ambiente e isto será feito de forma incremental e iterativa. Este aspecto, revela que o jogo em análise neste trabalho, é adequado para aplicar este tipo de aprendizagem, essencialmente porque uma acção conduz rapidamente a um novo estado e o agente obtém rapidamente a recompensa e pode passar para a escolha de uma nova acção.

Os algoritmos de aprendizagem por reforço são constituidos pelos seguintes elementos:

- Policy (π) - Define como o agente deve agir e escolher as acções
- Reward (r) - Define o objectivo da aprendizagem. O ambiente recompensa o agente por cada acção que ele toma e este por sua vez tenta maximizar a recompensa.
- Value function (v) - Permite calcular o valor de cada estado, ou seja, o total de recompensa que se espera receber a partir do estado actual.

A evolução e escolha das diferentes acções baseia-se no processo de decisão de Markov e pode ser representado, de forma bastante simples pelo diagrama da figura 4, onde:

- St é o estado actual
- At é a acção escolhida pelo agente no tempo t
- Rt é a recompensa obtida no tempo t

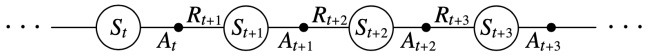

Fig. 4. Sequencia de estados em aprendizagem por reforço.

A. Formulação como um problema de aprendizagem por reforço

Para formularmos o problema em análise como um problema de aprendizagem por reforço, é preciso definir o conjunto de estados, quais as acções possíveis e a sua probabilidade de escolha em cada momento, a probabilidade de uma determinada acção conduzir a um novo estado e a recompensa recebida após a mudança de um estado para outro.

O ambiente e o conjunto de estados serão os tabuleiros de jogo juntamente com as peças de jogo, ou seja, o tabuleiro inicial é o estado inicial, depois de duplicar um dos blocos de jogo passa-se para o estado seguinte e assim sucessivamente até ao estado final que será o tabuleiro totalmente preenchido com as peças de jogo duplicadas. Existe apenas um estado final e vários estados não finais.

As acções que podem ser feitas em cada estado e para cada peça de jogo são:

- DBUP: Duplicar peça para cima
- DBDWN: Duplicar peça para baixo
- DBLFT: Duplicar peça para a esquerda
- DBRGT: Duplicar peça para a direita

Pode-se considerar que numa policy incial em cada estado a probablidade de cada acção será de 0.25.

O sistema de recompensas definido foi o seguinte:

- Escolha de acção válida: -1
- Escolha de acção inválida: -10
- Atingido estado em que não é possível terminar o jogo: -100
- Atingido estado final de sucesso: 100

Utilizando a equação de *Bellman* pode-mos definir a expectativa do estado incial, a função valor.

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right] \quad (1)$$

Considerando então que a probabilidade de escolha de cada acção é de 0.25 e que a recompensa em cada estado não final é de -1, a equação 1 no estado inicial resulta então em:

$$v_\pi(s) = 0.25 * [-1 - 1 - 1 - 1] = -1 \quad (2)$$

Os algoritmos irão então repetir esta equação em cada estado até chegarem ao estado final, onde a recompensa será diferente, levando assim ao máximo valor se se escolher o menor número de passos.

III. Implementação

Neste trabalho é utilizada a framework Open AI Gym, [3]. Esta framework é utilizada para desenvolver algoritmos de aprendizagem por reforço, permitindo a integração de ambientes costumizáveis para a aprendizagem. Para este trabalho foi gerado um ambiente que permite criar diferentes tabuleiros de jogo para o agente solucionar. Este ambiente define não só os tabuleiros de jogo mas também toda a lógica de funcionamento do jogo, acções permitidas, mudanças de estado entre outros aspectos.

Serão utilizados três tabuleiros diferentes para a aprendizagem, que tentam simular níveis de diferentes dificuldades (fácil, intermédio e difícil). Um tabuleiro muito simples e de pequenas dimensões, que foi utilizado para iniciar a aprendizagem do agente e outros dois tabuleiros que são incrementos na dificuldade e servirão para verificar como o agente se comporta com o aumento da complexidade do problema. Desta forma o agente fica exposto a tabuleiros apenas com uma peça, com várias peças e com variações de células a preencher e não preencher, cobrindo todas as variações dos níveis do jogo.

A. Q-Learning

O agente irá utilizar uma tabela de recompensa, *qTable*, para aprender quais as melhores acções a tomar e ter mais recompensa. Esta tabela irá mapear estados e acções, dando informação sobre a "qualidade" de uma acção tomada a partir de um determinado estado. Melhores valores na tabela, indicam maior probabilidade de obter uma recompensa mais alta.

A tabela foi inicializada com todos os valores a zero e depois ao longo da execução é actualizada seguindo a fórmula 3.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma max \alpha, Q(s_{t+1}, a_t) - Q(s_t, a_t)] \quad (3)$$

Onde:

- α: representa a taxa de aprendizagem (0 ¡ α¡= 1), ou seja, a taxa com que os valores da tabela são actualizados após cada iterecção. Um valor muito baixo poderá resultar em não aprendizagem.
- γ: é um factor de desconto (0 ¡ γ¡= 1) que permite determinar a importância dada ás recompensas futuras. Valores mais baixos preveligiam as recompensas mais imediatas.

Esta equação mostra que a tabela é actualizada tendo em conta o valor anterior, somando-lhe o valor resultante da aprendizagem e escolha da acção. A aprendizagem é resultado da escolha de uma acção no estado actual e a recompensa máxima que será obtida, a menos do factor de desconto. Ou seja, o agente está a aprender qual a melhor acção a tomar olhando para o estado e acções actuais e a máxima recompensa obtida no estado seguinte. Isto levará a que eventualmente o agente finalize o jogo. Na tabela, *qtable*, um par estado-acção será então a soma da recompensa actual e da recompensa futura.

O algoritmo irá então realizar as seguintes acções:

```
Initialize Q(s,a) (all values = 0)
for(each episode)
    Initialize s
    for (each step of the episode,
    until s is terminal)
        choose a from s using policy
        from Q (e.g. greedy)
        take action a, observe r and s'
        update qTable using the equation
```

No algoritmo foi ainda incluido um parâmetro (ε) para prevenir que o agente siga sempre as mesmas acções nas diferentes simulações. Este parâmetro permite definir com que frequência o agente irá preferir escolhas aleatórias em vez de escolhas com o melhor valor na tabela *qTable*. Este parâmetro vai sendo alterado ao longo da simulação.

B. SARSA

A implementação deste algoritmo é muito semelhante à implementação do algoritmo Q-Learning, sendo a maior diferença a forma como é actualizada a tabela. Neste caso

a tabela é actualiza com os valores de recompensa actuais e não com estimativas das recompensas que se irão receber em futuras acções. Isto implica que o SARSA não olhe para as recompensas futuras mas apenas para a próxima recompensa, o que se poderá traduzir numa aprendizagem mais consistente.

A equação 4 descreve a fórmula utilizada para actualizar a tabela *qTable*.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{4}$$

```
Initialize Q(s,a) (all values = 0)
for(each episode)
    Initialize s
    choose a from s using policy
        from Q (e.g. greedy)
    for (each step of the episode,
    until s is terminal)
        take action a, observe r and s'
        choose a' from s' using policy
        from Q (e.g. greedy)
        update qTable using equation
```

Tal como no algoritmo anterior foi incluido o parâmetro (ε) para permitir valorizar escolhas aleatórias ou escolhas com base na tabela.

IV. Resultados experimentais

Os testes foram divididos em três ambientes diferentes, cada ambiente a representar um nível de dificuldade diferente do jogo.

A. Ambiente 1 - Nível Fácil

O layout utilizado neste nível é um tabuleiro de 3x3 com apenas uma peça, como pode ser visto na figura 5.

Na figura 6 pode ser visto o número de passos necessários por cada um dos algoritmos em análise para se atingir a solução e na figura 7 as recompensas obtidas ao longo da simulação.

Como este ambiente é bastante simples, o agente consegue encontrar solução em poucos passos em ambos os algoritmos, sendo que o algoritmo SARSA apresenta melhores resultados. Isto pode indicar que o algoritmo Q-Learning neste ambiente tende a escolher soluções sub-optimas.

Fig. 5. Layout do tabuleiro para o nível fácil.

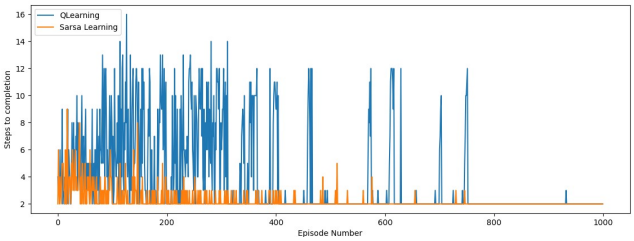

Fig. 6. Passos necessários para resolver o tabuleiro no nível fácil.

B. Ambiente 1 - Nível Intermédio

O layout utilizado neste nível é um tabuleiro de 4x6 com três peças, como pode ser visto na figura 8.

Na figura 9 pode ser visto o número de passos necessários por cada um dos algoritmos em análise para se atingir a solução e na figura 10 as recompensas obtidas ao longo da simulação.

As diferenças entre os algoritmos não são muito significativas, mas o Q-Learning tem um comportamento mais estável, convergindo para o resultado final de forma mais controlada. É também interessante verificar que devido ao facto de o algoritmo Q-Learning tentar maximizar as recompensas, no

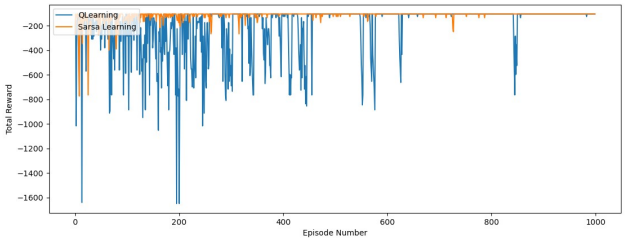

Fig. 7. Recompensas obtidas durante o treino no nível fácil.

Fig. 8. Layout do tabuleiro para o nível intermédio.

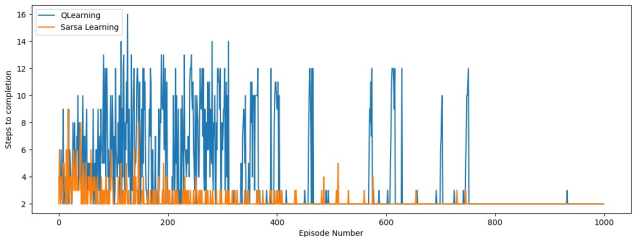

Fig. 9. Passos necessários para resolver o tabuleiro no nível fácil.

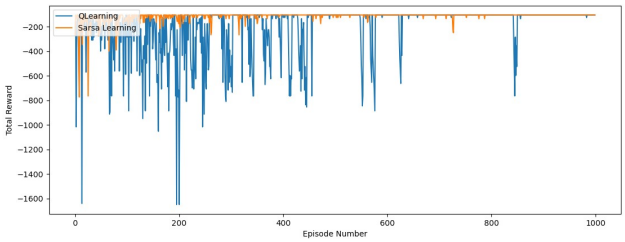

Fig. 10. Recompensas obtidas durante o treino no nível fácil.

geral obtém recompensas melhores que o SARSA. Contudo, a convergência para o valor final é quase igual nos dois algoritmos, não sendo possível verificar qual o mais rápido.

C. Ambiente 2 - Nível Difícil

O layout utilizado para este nível é um tabuleiro de 5x8 com 3 peças e células que não devem ser preenchidas. A figura 11 mostra este layout.

Na figura 12 pode ser visto o número de passos necessários por cada um dos algoritmos em análise para se atingir a solução e na figura 13 as recompensas obtidas ao longo da simulação.

Neste ambiente as diferenças entre SARSA e Q-Learning são novamente mínimas, mostrando um melhor desempenho do SARSA. As curvas do algoritmo SARSA convergem mais rapidamente apesar, tal como esperado, apresentar mais penalizações que o Q-Learning.

V. Conclusões

Neste trabalho foi possível perceber como funciona a aprendizagem por reforço e como pode ser aplicado através de dois algoritmos diferentes: Q-Learning e SARSA. Ambos os algoritmos se revelaram válidos para o ambiente proposto, jogar o jogo Folding Blocks. Os dois algoritmos estudados apresentam performances semelhantes, sendo que o SARSA é ligeiramente melhor especialmente em situações mais complexas. Isto pode ser explicado porque o SARSA segue a política de escolha das acções enquanto que o Q-Learning ao tentar maximizar as recompensas pode escolher soluções não optimas. Esta diferença nas escolhas das acções reflete-se nas recompensas obtidas por cada um dos algoritmos, o Q-Learning aprensenta de forma consistente valores de recompensa melhores. Relativamente ao número de passos para o agente a aprender a solucionar o jogo, em ambos os algoritmos se obtêm valores praticamente

Fig. 11. Layout do tabuleiro para o nível difícil.

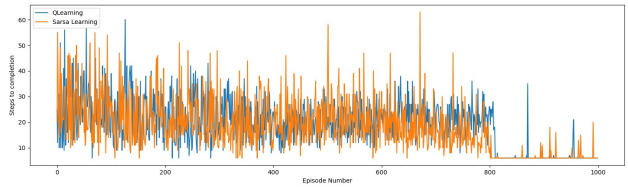

Fig. 12. Passos necessários para resolver o tabuleiro no nível difícil.

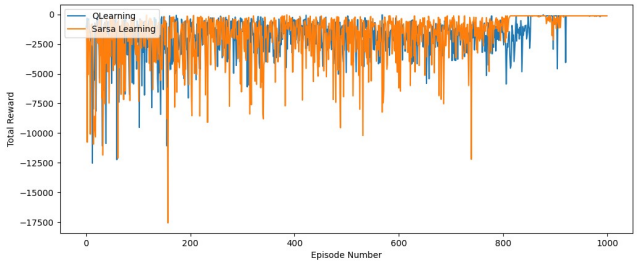

Fig. 13. Recompensas obtidas durante o treino no nível difícil.

identicos, no entanto o Q-Learning mostra mais picos ao longo da aprendizagem mostrando que a natureza das suas escolhas é mais aleatória que o SARSA. No nível mais fácil, o Q-Learning precisa de mais passos, isto pode ser explicado pela natureza off-topic deste algoritmo, ou seja, como não segue a politica o seu comportamento é um pouco mais aleatório e num ambiente bastante fácil isso resulta em maior instabilidade, o que, em ambientes mais complexos não é tão perceptível pois as escolhas tendem a convergir para o resultado final.

REFERENCES

[1] Pop Core Games, "Folding Blocks: Puzzle Game" URL: https://play.google.com/store/apps/details?id=com.popcore.foldingblocks, [Online; acedido em 25-05-2020]

[2] R. Sutton and A. Barto, "Reinforcement Learning: An Introduction", Second Edition, November 2017

[3] Open AI, "Open AI GYM" URL: https://gym.openai.com/docs/, [Online; acedido em 27-05-2020]

[4] Arxiv Insigts, "An introduction to Reinforcement Learning" Recurso multimédia: https://www.youtube.com/watch?v=JgvyzIkgxF0, [acedido em 27-05-2020]

[5] Thomas Simonini, "Diving deeper into Reinforcement Learning with Q-Learning" URL: https://www.freecodecamp.org/news/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe/, [Online; acedido em 27-05-2020]

[6] Kung-Hsiang, "Introduction to Various Reinforcement Learning Algorithms." URL: https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg
[Online; acedido em 25-05-2020]