

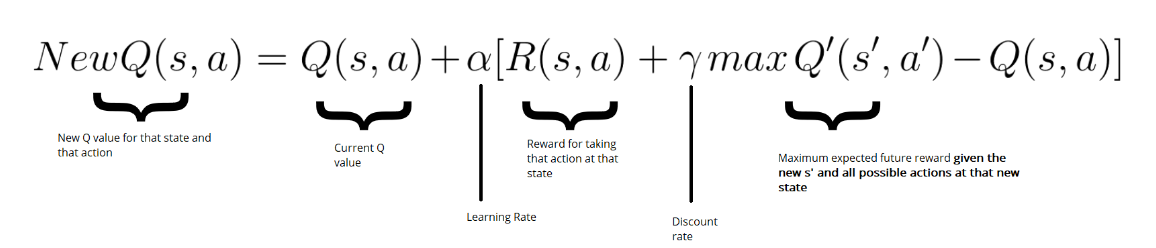
**Figura 2:** Autómatos correspondentes, respetivamente, ao teste 1 e 2 do ficheiro “mainRL.py”

O código produzido no ficheiro “RL.py” permitiu passar todos os testes executados com o ficheiro “mainRL.py”. Apesar disso, nem sempre a aproximação de Q está dentro do previsto – isto é devido ao valor gerado pela função “policy” quando o agente está em modo “exploration” ser aleatório, e não garante que a ação escolhida seja a com maior recompensa. Apesar disso, a trajetória gerada é sempre óptima.

A base do projeto é no conceito de Aprendizagem por Reforço (Reinforcement Learning), mais concretamente, em Q-Learning, que é um algoritmo de aprendizagem por reforço com base em valores numéricos. O algoritmo de Q-Learning baseia-se na ideia de “recompensa”, ou seja, que para cada ação que o agente pode efetuar, ele recebe uma recompensa que tanto pode ser positiva ou negativa. Estes valores da recompensa são mantidos numa matriz com os vários movimentos possíveis – tendo cada posição da matriz 4 possíveis (esquerda, baixo, direita, cima) estados – e tem como objetivo calcular o valor máximo de recompensa esperado.

Existem duas formas de explorar esta matriz e tomar decisões: o agente pode estar em modo “exploitation” – o agente escolhe sempre a decisão com maior nível de recompensa – ou em modo “exploration” – o agente escolhe uma decisão aleatória, que nem sempre pode ter o maior nível de recompensa.

Para calcular a recompensa para cada ação, a função “traces2Q” percorre os vários caminhos possíveis dentro da matriz, e calcula a recompensa para cada ação dado o caminho usando a seguinte fórmula, denominada equação de Bellman.



A aplicação desta fórmula implica a atualização de Q. Eu suma, o algoritmo de QLearning baseia-se no seguinte conjunto de ações: Inicializar a matriz Q -> Escolher uma ação a -> Realizar ação -> Calcular recompensa -> Atualizar Q.

As vantagens do uso deste algoritmo é que permite recolher informação a partir de movimentos parcialmente randomizados, que com o cálculo da recompensa para cada entrada possível da matriz Q, vai permitir, à medida que o “treino” do agente evolve, a aleatoriedade dos seus movimentos diminui, atingindo uma chamada “boa” matriz Q. Apesar disto, como o algoritmo de QLearning se baseia numa de função de aproximação, pode tornar-se instável em fases mais avançadas da sua execução. Além disso, o algoritmo de QLearning não é o algoritmo mais eficiente em termos de tempo de execução.

A complexidade do algoritmo implementado depende do espaço ocupado pelo número de estados presentes. Sendo s o número de estados, a complexidade do algoritmo é O(sn). Este pode ser melhorado, evitando que existam ações que são efetuadas múltiplas vezes.