ISEL – Instituto Superior de Engenharia de Lisboa
ADEETC – Área Departamental de Engenharia de Electrónica e Teleconumicações e de Computadores

MERCM
Mestrado em Engenharia de Redes de Comunicação e Multimédia
Dissertação

Aprendizagem por Reforço com Memória Episódica

André Filipe Casaleiro Gomes

Júri

Presidente	*Prof. Doutor* Carlos Jorge Sousa Gonçalves
Vogais	
Arguente	*Prof. Doutor* Paulo Manuel Trigo Cândido da Silva
Orientador	*Prof. Doutor* Luís Filipe Graça Morgado

Setembro, 2019

Resumo

A Inteligência Artificial é uma área de estudo que, com o propósito maior de criar sistemas que apresentem comportamento inteligente, tem vindo a desenvolver várias abordagens que tentam atender a esse mesmo propósito, normalmente focando-se nalguma actividade específica associada a inteligência, como a deliberação ou a aprendizagem.

Especificamente na aprendizagem, existem vários paradigmas utilizados em sistemas inteligentes, sendo uma delas a aprendizagem por reforço, na capacidade de um sistema melhorar o seu desempenho numa tarefa, sem qualquer conhecimento prévio das possíveis configurações do problema ou das melhores acções a escolher, observando as consequências das suas acções sob a forma de reforços positivos ou negativos. Estes reforços são processados de modo a que as próximas acções possam ser mais direccionadas ao objectivo da tarefa em questão.

Têm vindo a ser desenvolvidos ao longo do tempo vários algoritmos para este tipo de aprendizagem, e estes têm sido cada vez mais optimizados à medida que são expostos a problemas progressivamente mais complexos, seja pela adição de subsistemas que contribuam para uma melhoria do desempenho de alguma característica específica do sistema, ou pela total substituição da mesma. A característica em questão nesta dissertação é a estrutura de memória utilizada em sistemas de aprendizagem por reforço, que nos algoritmos clássicos pode vir a apresentar limitações em termos de dimensão e de complexidade.

Pretende-se com esta dissertação apresentar alguns dos métodos clássicos de aprendizagem por reforço, e observar se a integração de estruturas de memória episódica pode ou não trazer melhorias para sistemas desta natureza, através do desenvolvimento de uma biblioteca de aprendizagem por reforço que permita visualizar o funcionamento de algoritmos já conhecidos, assim como algoritmos que integrem estruturas de memória episódica.

Palavras chave: agentes inteligentes, aprendizagem por reforço, memória episódica, política comportamental

Abstract

Artificial Intelligence is a field of study that, with the greater purpose of building systems that show an intelligent behaviour, has been developing a series of approaches that try to attend to that purpose, usually focusing on some specific activity associated with intelligence, like deliberation or learning.

Specifically on the subject of learning, there are several paradigms used in intelligent systems, one of them being reinforcement learning, which consists of a system's capability of improving it's performance on a task, without any prior knowledge of the possible configurations of the problem or of the best actions to choose, observing the consequences of its actions in the form of positive and negative reinforcements. These reinforcements are processed in a way that the following actions can be more directed towards the goal of the task in question.

Several algorithms for this type of learning have been developed over time, and these have been increasingly optimized as they are exposed to progressively more complex problems, either by the addition of subsystems that contribute to improving the performance of some specific feature of the system, or by its full replacement. The feature in question in this dissertation is the memory structure used in reinforcement learning systems, which in classical algorithms may present limitations in terms of size and complexity.

The goal of this dissertation is to present some of the classical reinforcement learning methods, and to observe whether or not the integration of episodic memory structures can bring improvements to such systems, through the development of a reinforcement learning library that allows the visualization of known algorithms, and also algorithms that integrate episodic memory structures.

Keywords: intteligent agents, reinforcement learning, episodic memory, behaviour policy

Agradecimentos

Ao Professor Doutor Luís Morgado, pela orientação não só nesta dissertação, mas nos vários anos de aulas. O seu rigor, a sua disponibilidade e, sobretudo, a sua paciência foram determinantes para o meu percurso académico e para o sucesso deste trabalho.

Ao Instituto Superior de Engenharia de Lisboa – que à boa moda portuguesa, consegue fazer muito com pouco – pela formação excelente, pelo corpo docente excepcional, pelos colegas, pela *Praxe*, e acima de tudo por me levar à Estudantina Académica do ISEL, que me proporcionou uma forma única de viver o Instituto, como uma segunda casa, e que me trouxe amizades e ensinamentos para uma vida.

Aos meus pais, que sacrificaram tudo para que não me faltasse nada, permitindo-me chegar a esta etapa na vida, e pelo apoio que também nunca faltou. À minha irmã, que se não me tivesse ensinado a usar um computador, talvez hoje eu não soubesse programar.

A todos os amigos e colegas que fingiram interesse em Inteligência Artificial durante 10 minutos, apenas para me incentivar.

À Susana, que está sempre do meu lado nos altos, nos baixos, e nos entretantos.

Just as you dont know, how you manage to be conscious, how you manage to grow and shape this body of yours, that doesnt mean to say that you're not doing it. Equally, you don't know how the universe shines the stars, constellates the constellations and galactifies the galaxies. You dont know. And that doesn't mean to say that you aren't doing it in just the same way as you are breathing without knowing how you breathe.

Alan Watts

Índice

Lista de Figuras

Lista de Tabelas

Glossário

PSA	Plataforma de Simulação de Agentes
PAR	Plataforma de Aprendizagem por Reforço
PDM	Processos de Decisão de Markov
SARSA	State–action–reward–state–action (Estado-acção-recompensa-estado-acção)
GPS	Global Positioning System (Sistema de posicionamento global)
DQN	Deep Q-Network

Capítulo 1

Introdução

A aprendizagem por reforço é um processo ao qual nos expomos diariamente. Seja num evento tão complexo como jogar uma partida de xadrez, um problema que envolve uma quantidade elevada de configurações possíveis do tabuleiro, ou tão simples como nos queimarmos a agarrar numa panela quente, um evento cujo resultado pode ser tão negativo que é suficiente para alterar completamente a nossa conduta em situações futuras. Resumidamente, o nosso comportamento é constantemente influenciado pelas consequências que observamos dos nossos actos.

O desenvolvimento de um sistema computacional de aprendizagem por reforço tem por objectivo concretizar esse mesmo processo contínuo de interacção com o mundo, observação dos resultados e actualização do comportamento, de forma a que as interacções futuras sejam mais adequadas a um dado problema.

Um sistema deste tipo define um agente: uma representação da entidade que vai aprender, que está inserida num ambiente onde pode efectuar acções e de onde pode obter estímulos sensoriais. Numa fase inicial, o agente não possuí informação interna acerca do mundo que o rodeia, pelo que antes de poder seleccionar uma acção concreta a executar, este tem primeiro de explorar o ambiente. Ao longo da exploração, o agente pode construir estruturas que armazenem o histórico dos resultados das suas acções e/ou o resultado do processamento das mesmas, permitindo assim definir uma política comportamental — um mapeamento entre estados do ambiente e a respectiva acção a executar baseado na experiência acumulada, reduzindo a necessidade de exploração por parte do agente.

Um dos desafios fundamentais da aprendizagem por reforço é exactamente o do balanço entre exploração/aproveitamento por parte do agente. Quando é que um agente deve considerar que a política comportamental que calculou até a um dado momento é óptima? Na realidade não existe, por enquanto, um método absolutamente infalível para qualquer problema, mas sim uma variedade de opções possíveis a seguir, que devem ser escolhidas adequadamente à situação.

Os métodos clássicos de aprendizagem por reforço, desenvolvidos muito tempo antes do poder computacional actual (ex: *Q-Learning* [1], *SARSA* [2]), ainda hoje funcionam perfeitamente nos cenários habituais nesta área de estudo. Contudo, mesmo com esse poder computacional, estes algoritmos sofrem uma degradação de desempenho significativa quando expostos a problemas do mundo real [3], onde os espaços de estados são esparsos e de dimensões elevadas, resultando em quantidades de dados consideráveis e com uma baixa probabilidade de existirem estados repetidos, dado o carácter contínuo de um ambiente real.

Foi referido anteriormente que um sistema computacional de aprendizagem por reforço tem por analogia o processo de aprendizagem a partir da experiência, tal como observamos nos animais. Consideremos como exemplo a divisão conceptual da memória humana em dois sistemas distintos (esta divisão específica não existe na realidade, é apenas utilizada para facilitar a discussão do tema) [4] : memória semântica, responsável pelos conceitos, símbolos e as suas relações e regras, e memória episódica, responsável pelo armazenamento de eventos no tempo e as respectivas relações espaço-temporais, podemos observar que nos algoritmos clássicos de aprendizagem por reforço as estruturas de memória utilizadas assemelham-se mais à memória semântica, na medida em que é armazenada o resultado do processamento das experiências de um agente num dado ambiente. Deste ponto de vista, faz sentido considerar incluir nestes sistemas uma estrutura de memória episódica, tanto como um sistema independente como uma estrutura adicional de apoio às decisões do agente. Esta poderia ser utilizada, por exemplo, como uma forma de poder "recordar"factos do passado de modo a consolidar o conhecimento geral do agente, ou como uma heurística para aferir a qualidade das soluções.

Com este trabalho pretende-se estudar qual o impacto de adicionar uma

estrutura de memória episódica, que permita ter uma noção temporal e segmentada das acções efectuadas no ambiente e dos seus retornos, tanto aos algoritmos clássicos como em novos algoritmos.

1.1 Motivação

Na área de estudo de Inteligência Artificial, pretende-se estudar e criar sistemas que apresentem comportamento inteligente [5]. Sendo o ser humano o melhor exemplo de ser inteligente que temos (por enquanto), podemos afirmar que seria útil ter por base o conhecimento existente acerca de inteligência humana para desenvolver sistemas inteligentes. Contudo, uma simples pesquisa acerca do cérebro humano permite constatar que este é um órgão de elevada complexidade em termos das várias áreas e subsistemas responsáveis pelas inúmeras funções do corpo humano, mesmo quando considerando apenas a "parte inteligente", ignorando os restantes sistemas relativos a outras funções (ex. função motora, sensorial).

Posto isto, torna-se então provável que, para conceber um sistema inteligente, se torne eventualmente inevitável que este também tenha de depender do funcionamento de vários subsistemas que trabalhem em conjunto para replicar inteligência. Consideremos um problema actual, que já começa a ter soluções em produção mas que ainda está numa fase de desenvolvimento: condução autónoma. Se tivéssemos de desenvolver um sistema para tornar um automóvel capaz de se conduzir sem intervenção de um humano, alguns dos requisitos poderiam incluir:

- Ter capacidade de evitar obstáculos detectados por meio de sensores;
- Ter capacidade de inferir quais as estradas a percorrer para chegar ao destino pretendido (talvez até a evitar trânsito) através de *GPS* (*Global Positioning System*);
- Ter capacidade de aprender a lidar com outros automóveis, sejam estes conduzidos por humanos ou não, principalmente em momentos de volume de tráfego elevado, através da observação do comportamento do mesmo.

Apenas para cumprir estes requisitos, o sistema desenvolvido tem de ser dotado de capacidade reactiva, deliberativa e de aprendizagem, ou seja, uma

combinação de algumas das abordagens que podem ser encontrados na literatura do tema, além da aprendizagem por reforço.

Esta conjunção de vários subsistemas para criar um sistema mais robusto e cada vez mais capaz de resolver problemas reais é a motivação para esta dissertação, sendo este tema mais uma opção para a optimização de sistemas de aprendizagem por reforço, não pela substituição dos mecanismos já existentes, mas pela possibilidade de ter mais um para adicionar a qualquer outro sistema de modo a que este possa ter um melhor desempenho.

1.2 Objectivos

Pretende-se realizar o estudo de métodos de aprendizagem por reforço com memória episódica, de modo a poder tirar conclusões acerca da sua influência no desempenho do agente, e de como podem ser utilizadas para solucionar algumas das limitações associadas a estas técnicas quando aplicadas em problemas reais.

Para tal, pretende-se desenvolver software que permita observar algumas das técnicas actuais em aprendizagem por reforço, assim como desenvolver versões modificadas das mesmas utilizando estruturas de memória episódica.

1.3 Organização

Esta dissertação foi organizada em 7 capítulos, incluindo este primeiro capítulo introdutório do tema:

Capítulo 2: Aprendizagem por Reforço
Introdução à aprendizagem por reforço, dos elementos constituintes, e apresentação de alguns algoritmos clássicos;

Capítulo 3: Aprendizagem por Reforço com Memória Episódica
Formalização do conceito de memória episódica e apresentação das vantagens e desvantagens que esta pode trazer para sistemas de aprendizagem por reforço;

Capítulo 4: Abordagem Proposta
Apresentação das decisões tomadas para estudar o tema e apresentação

de algoritmos a implementar que não foram descritos nos capítulos anteriores;

Capítulo 5: Plataforma de Aprendizagem por Reforço
Apresentação da biblioteca desenvolvida do ponto de vista da engenharia de software;

Capítulo 6: Concretização Experimental
Definição dos cenários de teste para os vários algoritmos implementados e apresentação dos respectivos resultados;

Capítulo 7: Conclusão
Conclusões após observação dos resultados observados no capítulo anterior.

1.4 Convenções de Escrita

- Não foram traduzidas expressões em inglês de uso comum (ex: *hardware*, *software*)
- Os algoritmos em pseudo-código apresentados foram escritos em inglês para manter coerência com as expressões tradicionais em inglês

Capítulo 2

Aprendizagem por Reforço

Vivemos actualmente uma realidade em que o desenvolvimento de sistemas automatizados, quando se tratam de problemas fechados e completamente previsíveis em termos das acções a executar e das suas consequências, é uma área de estudo que se encontra numa fase tão avançada em termos teóricos e de ferramentas de trabalho disponíveis, que se poderia considerar uma tarefa trivial. Do ponto de vista do *software*, existindo familiarização com as bases de engenharia de *software* (o que nos dias de hoje se consegue obter gratuitamente), o desafio, na maioria das vezes, está apenas em escolher a solução correcta de acordo com os padrões conhecidos. Quanto a *hardware*, o acesso ao mesmo está cada vez mais barato e com uma melhor relação entre capacidade computacional e preço, sendo uma prova disto a crescente oferta de equipamentos de prototipagem de baixo custo como o *Arduino* ou o *Raspberry Pi*.

Esta facilidade de acesso tanto a *hardware* como a *software*, assim como à formação técnica e até à operacionalização atingiram níveis em que até uma pessoa a título individual tem a possibilidade de construir sistemas automatizados na sua própria casa a baixo custo, e, no caso industrial, tornou-se um requisito relevante para poder ter hipótese de competitividade no mercado. Posto isto, torna-se imperativo que o passo seguinte no desenvolvimento de sistemas automatizados seja que estes tenham a capacidade de tirar partido da experiência resultante do trabalho efectuado no seu problema, e adaptar o seu funcionamento de acordo com as necessidades, ou até conseguir criar novas rotinas que optimizem o desempenho.

A inteligência artificial é uma área de estudo que, apesar da sua conotação

futurista, tem vindo a ser desenvolvida há várias décadas, muito antes do poder computacional actual. Com o objectivo principal de desenvolver sistemas que repliquem inteligência, nomeadamente, tal como a conhecemos em nós próprios. Estes sistemas podem abordar o problema de várias formas, consoante a estrutura cognitiva nestes implementada, que pode ir desde o mais simples agente que apenas reage ao que observa, até ao mais complexo processo de raciocínio que pode envolver múltiplas fontes de informação sensorial, mecanismos de memória, ou até mecanismos sociais entre múltiplos sistemas. Atendendo ao tema abordado nesta dissertação, apenas nos iremos focar em sistemas de aprendizagem por reforço.

Neste capítulo iremos definir aprendizagem por reforço e o seu propósito, definir os seus elementos constituintes e as suas respectivas representações formais, e como é que estas são utilizadas para atingir soluções, tanto com um modelo do mundo, como sem qualquer informação prévia.

2.1 Propósito

A aprendizagem por reforço é um método de aprendizagem em que o agente aprende o que fazer [6], e tem como propósito a resolução de problemas através da interacção com os mesmos, com informação inicial limitada, utilizando a informação obtida pelas consequências destas para influenciar as seguintes, permitindo assim optimizá-las ao longo do tempo relativamente aos objectivos que se pretendem atingir.

Um sistema computacional de aprendizagem por reforço pretende dotar uma entidade, seja esta física ou lógica, da capacidade de recolher, organizar, e processar as informações recolhidas, de forma a encontrar soluções para os problemas a que este seja exposto. Estes sistemas são actualmente aplicados em vários tipos de problema:

- Aprender a jogar um jogo, como o AlphaGO da Google, capaz de vencer os profissionais de topo em Go, um jogo de tabuleiro de estratégia com um numero de configurações possíveis do tabuleiro muito superior ao jogo de xadrez, que aprendeu inicialmente a partir da experiência pré-compilada de jogadores humanos (aprendizagem supervisionada) e mais tarde utilizando aprendizagem por reforço em jogos contra outras instâncias de si próprio [7];

- Sistemas que precisem de aprender a navegar num espaço físico, como carros de condução autónoma, que para além de todos os mecanismos que possam utilizar antes da navegação, podem usufruir de aprendizagem por reforço para avaliar a execução pré-determinada de modo a poder ajustar o comportamento em deliberações futuras[8];

- Gestão de recursos de um servidor *web*, onde os tempos de resposta do servidor são utilizados para ajustar as configurações do servidor ao longo do tempo, optimizando a utilização de recursos [9];

- Automatização de processos através de robótica, em que a máquina tem como objectivo aprender a efectuar uma tarefa no mundo real, como movimentação num espaço ou transporte de objectos [10], no limite até para aprender a própria acção de andar [11].

2.2 Processos de Decisão de Markov

Os processos de decisão de Markov (PDM) são uma formalização matemática de um processo de decisão sequencial, que definem uma estrutura de dados que representa o agente e as suas capacidades de acção e percepção, o ambiente onde o agente opera, e a forma como este é recompensado durante a interacção com o ambiente. Esta estrutura serve de base aos sistemas de aprendizagem por reforço, ao providenciar uma forma sistemática de representação dos problemas a resolver, permitindo também o desenvolvimento de algoritmos relativos à representação do problema na forma de um processo de decisão de Markov ao invés de algoritmos específicos para o problema.

2.2.1 Conceitos Base

Antes de apresentar especificamente a formalização de um processo de decisão de Markov, é necessário ter noção dos elementos constituintes da estrutura por este definida: agente, acção, estado e reforço.

O agente é o elemento que opera sobre um ambiente através de acções a. Esta abstracção isola as características do operador concreto necessárias para obter a solução do problema. É irrelevante se o agente concreto é um ser humano, um *robot*, ou uma personagem virtual, apenas as acções que é capaz de efectuar e as percepções que consegue obter são relevantes.

Na mesma lógica de abstracção, o ambiente onde o agente se insere é representado pelo conjunto de estados possíveis de atingir pelo agente. Um estado, designado por s, é um vector de valores que especificam uma configuração do problema relativamente ao agente num dado instante de tempo, sem qualquer limitação na quantidade ou qualidade dos valores utilizados. Normalmente são utilizados apenas atributos que de alguma forma possam ser úteis para a operação do agente ou para a valorização do retorno de atingir um determinado estado.

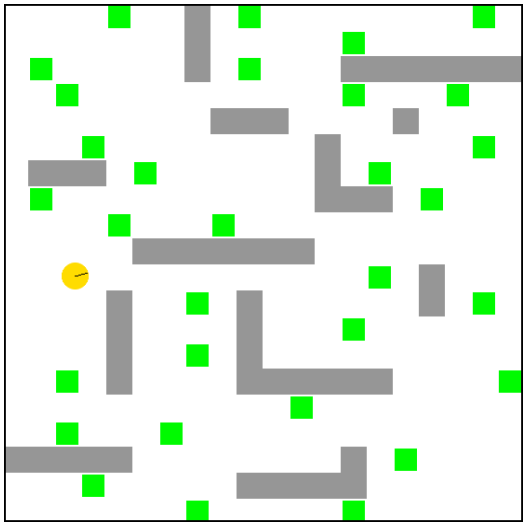

Figura 2.1: Exemplo de agente inserido num ambiente (PSA - Plataforma de Simulação de Agentes)

A figura 2.1 mostra um ambiente onde o conjunto de estados possíveis poderia ser, por exemplo, um conjunto das coordenadas 2D constituintes do ambiente, e o estado do agente seria a posição onde este se encontrar. Para poder transitar entre estados, o agente executa acções, designadas por a. Esta transição de um estado actual s para um estado seguinte s' gera uma recompensa r observável pelo agente. A recompensa é um valor numérico representativo do valor da transição para o agente relativamente ao objectivo a cumprir, sendo positivo quando da chegada a um estado desejado e negativo ao transitar para um estado indesejado, podendo incluir também o custo da acção realizada.

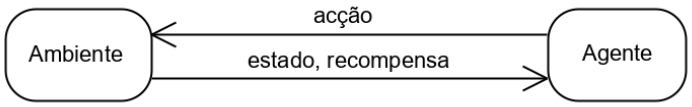

Figura 2.2: Interacção agente-ambiente

A figura 2.2 sintetiza a interacção de um agente com um ambiente, do qual obtém um estado e uma recompensa, que utiliza para escolher uma acção para executar no ambiente, que por sua vez tem como retorno um novo estado e uma nova recompensa.

2.2.2 Formalização

Para ser considerado um PDM, o problema respectivo deve satisfazer a propriedade de Markov - que indica que num processo estocástico, a distribuição probabilística condicional dos estados futuros de um processo depende exclusivamente do estado presente, ou seja, são apenas válidos para esta modelação problemas em que uma transição de um estado s_t para um estado s_{t+1} é apenas influenciada pelo estado presente s_t , e não pelos estados anteriores s_{t-n}. Este conceito é facilmente observável numa tarefa de movimentação num espaço: o destino de um passo está apenas relacionado com a posição actual do agente, independentemente de qualquer movimentação feita no passado.

Um PDM é definido segundo um tuplo de 4 conjuntos, nomeadamente:

- S - Conjunto dos estados s do mundo
- $A(s)$ - Conjunto de acções a possíveis num estado $s \in S$
- $T(s, a, s')$ - Probabilidade de transição de s para s' através de a
- $R(s, a, s')$ - Recompensa esperada na transição de s para s' através de a

Tendo o problema representado sob a forma de um PDM, resta apenas o desenvolvimento de uma política comportamental para o agente. Designada por π, a política comportamental representa a estratégia que o agente deve seguir em cada estado. Esta pode ser ou não determinista, consoante o

problema em questão. Num problema determinístico, para as transições de estado válidas:

$$T(s, a, s') = 1, s, s' \in S \tag{2.1}$$

A política de um agente num problema determinístico associa a cada estado $s \in$ S uma única acção, dada a garantia de que essa acção vai gerar sempre uma transição para um único estado *s'*:

$$\pi : S \to A(s), s \in S \tag{2.2}$$

Num problema estocástico, uma acção a num estado s pode levar a estados *s'* diferentes, ou seja, $T(s, a, s') < 1$, com multiplos s', torna-se necessário relacionar pares (estado, acção) à probabilidade de transição para cada estado sucessor possível:

$$\pi : S \times A(s) \to [0, 1], s \in S \tag{2.3}$$

A política óptima π^* é a política em que todos os estados estão associados às acções que maximizem a recompensa relativamente ao objectivo a cumprir. Regressando ao exemplo referido anteriormente, um mundo 2D onde o agente tem como objectivo percorrer o mapa de uma posição inicial P_{ini} a uma posição objectivo P_{obj}, e onde o movimento não tenha qualquer influência em termos de recompensa, sendo que o agente apenas recebe uma recompensa positiva quando efectuar a transição para o estado objectivo, e negativo quando forem atingidos obstáculos, como apresentado na figura seguinte. Se o agente for explorando o ambiente e se guiar apenas pelas recompensas recebidas, serão apenas valorizados positivamente os estados vizinhos de P_{obj}, dado que nos restantes estados não existem transições relevantes, excepto as que levam a obstáculos, que evidentemente devem ser evitados. Desta forma, uma valorização de estado baseada apenas em recompensas imediatas não pode resultar numa política óptima, pois não são tidas em conta recompensas possíveis no futuro.

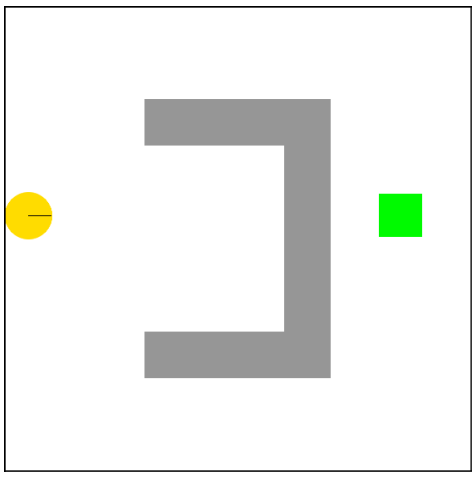

Figura 2.3: Exemplo de um agente inserido num ambiente

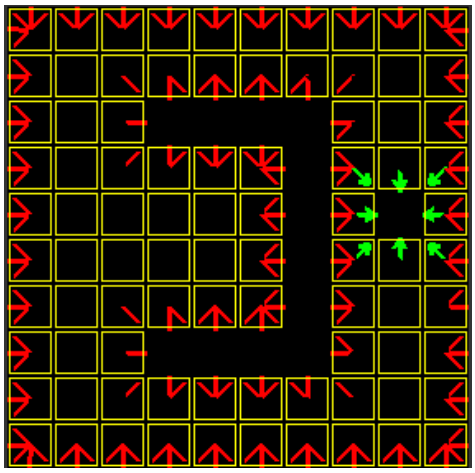

Figura 2.4: Recompensas imediatas das acções possíveis

A figura 2.3 apresenta um agente inserido num ambiente, onde se pode deslocar nos espaços brancos, e é impedido nos espaços a cinzento. O estado objectivo é representado pela quadrícula verde. A figura 2.4 apresenta uma valorização das acções em cada estado no mesmo ambiente, calculadas utilizando apenas as recompensas imediatas das transições de estado, sendo cada quadrícula amarela representativo de um estado possível no problema em questão, mais especificamente uma coordenada (x, y). As setas em cada quadrícula representam acções possíveis de executar em cada estado, apresentando uma cor vermelha ou verde consoante uma recompensa negativa ou positiva, variando também na intensidade consoante o valor. Apesar de em todos os estados que não sejam obstáculo ser possível para o agente se movimentar em 8 direcções diferentes, não existindo custo de movimento, não apresentam qualquer cor, ou seja, têm retorno nulo. Com este método de valorização das acções, não é possível gerar uma política comportamental que cumpra o objectivo, porque nos estados não adjacentes ao objectivo não é possível escolher uma acção óptima de acordo com o mesmo.

A política óptima pode ser obtida, por exemplo, recorrendo a programação dinâmica. Esta consiste na utilização de algoritmos que recorrem a funções de valor de estado (utilidade) para encontrar os estados mais úteis para atingir o objectivo e as acções a tomar para atingir os mesmos.

A utilidade de efectuar uma acção num estado s tem em conta não só o reforço gerado pela transição, mas também a utilidade do estado seguinte,

s'. No entanto, a utilidade do estado futuro não pode ser apenas adicionada à do estado presente, dado que desta forma estaríamos a atribuir igual importância a uma recompensa garantida (a recebida imediatamente pela transição de estado), e a uma recompensa num futuro não garantido. Desta forma, torna-se necessário adicionar um factor de desconto γ, que permite definir qual a influência das recompensas futuras no presente. A utilidade $U(s)$, considerando a política óptima, é dada por:

$$U(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U(s')], \forall s \in S \tag{2.4}$$

Quando é encontrada a função de valor óptima, é encontrada também a política óptima, satisfazendo o princípio da optimização de Bellman, que indica que a política óptima é aquela que independentemente do estado e decisão iniciais, todas as decisões seguintes devem ser óptimas. A política óptima π^* para um determinado estado s é a acção a cujo estado seguinte *s'* maximize a recompensa cumulativa, como indicado pela expressão:

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U(s')] \tag{2.5}$$

Um algoritmo de programação dinâmica que utiliza esta noção de utilidade é o *Iterative Policy Evaluation*, que consiste na iteração de todos os estados s ∈ S e na actualização do seu valor de utilidade. Os reforços recebidos ao longo do tempo são acumulados na função U(s) utilizando a utilidade de estados futuros em conjunto com a recompensa recebida para determinar a utilidade de um estado, descontados com um factor γ. Dado que a utilidade de um estado depende da utilidade de estados futuros (que inicialmente é desconhecida), o espaço de estados é iterado várias vezes, para que estes possam ser actualizados com a informação obtida de iterações anteriores. Este processo é mantido até ser atingido um limiar de convergência pré-definido, isto é, quando a diferença entre a utilidade actual e anterior seja baixa o suficiente para se considerar que se chegou ao fim da deliberação. No fim deste processo, o agente deverá possuir uma política comportamental óptima para solucionar o problema, e pode simplesmente seguir a mesma para chegar ao objectivo.

Algoritmo 1 Iterative Policy Evaluation [6]

Input π, the policy to be evaluated
Output $U \approx u_\pi$
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $U(s)$, for all $s \in S$, arbitrarily except that $U(terminal) = 0$
loop until $\Delta < \theta$
    $\Delta \leftarrow 0$
    loop for each $s \in S$
        $u \leftarrow U(s)$
        $U(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} T(s,a,s')[R(s,a) + \gamma U(s')]$
        $\Delta \leftarrow max(\Delta, |u - U(s)|)$
    end loop
end loop

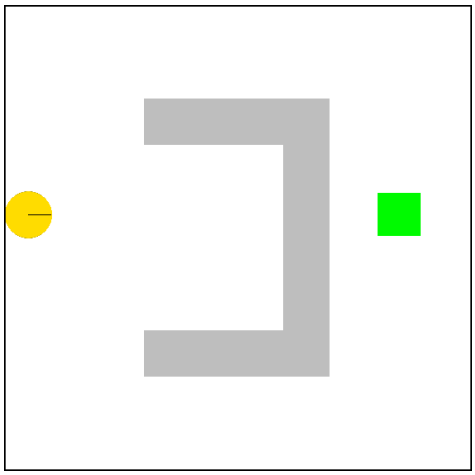

Figura 2.5: Exemplo de um agente inserido num ambiente

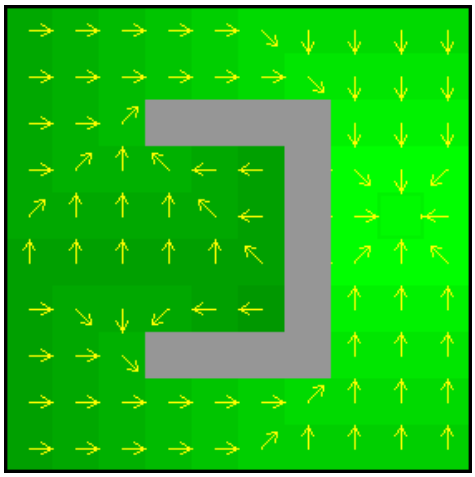

Figura 2.6: Política comportamental

As figuras acima apresentam uma política comportamental óptima, calculada utilizando o algoritmo apresentado. Neste caso, ao contrário da figura 2.3 que mostrava o valor de cada acção possível em cada estado, na figura 2.6 podemos observar o valor da utilidade de cada estado representada a verde, variando a intensidade de acordo com o valor, e a política resultante da iteração dos estados, ou seja, apenas a acção com maior valor em cada estado, representado pelas setas amarelas.

Deve ser sublinhado que este e outros algoritmos de programação dinâmica

assumem um conhecimento completo do mundo, através dos conjuntos *S, A* e funções T, R apresentadas anteriormente, permitindo assim a formulação da política antes da execução propriamente dita do agente. Deste modo, estes algoritmos não constituem métodos de aprendizagem por reforço, mas funcionam como uma base de conhecimento para a definição destes sistemas, como iremos observar a seguir.

2.3 Aprendizagem Baseada em Experiência

Naturalmente, nem todos os problemas possibilitam ter *à priori* toda a informação do mundo, o que implica que o agente tenha de aprender à medida que interage com o ambiente.

Uma possibilidade seria recorrer aos métodos de Monte Carlo [6], que ao invés de iterar todas as possibilidades, itera episódios constituídos pelas experiências do agente *(st,at,rt)*, construindo uma função de valor que associa a cada estado do episódio a média das recompensas obtidas nos episódios que levaram ao dito estado.

Outra alternativa está nos métodos de aprendizagem por diferença temporal, que de forma semelhante aos métodos de Monte Carlo, aprendem através da experiência ao longo do tempo, mas diferem na actualização da função de valor, na medida em que não esperam pelo resultado final para actualizar a função, e actualizam as suas estimativas utilizando estimativas anteriores [6]. Este método torna-se vantajoso pois mesmo não sendo possível obter soluções óptimas nos passos de execução iniciais, estas podem ser corrigidas ao longo do tempo, à medida que se acumula experiência. Outra vantagem está na distribuição mais eficiente da carga computacional, pois a informação obtida em cada passo é integrada imediatamente após a execução, ao invés de efectuar todo o processamento no fim do episódio de execução.

2.3.1 Aprendizagem sem Modelo

Apresentado por Watkins em 1989 [1], o *Q-Learning* é um algoritmo de aprendizagem por diferença temporal. Ao contrário dos PDM e e de alguns métodos de Monte Carlo, utiliza uma função de valor estado-acção em vez de apenas estado, nomeadamente:

$$Q : SxA(s) \rightarrow R \tag{2.6}$$

A qual é actualizada iterativamente da seguinte forma:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.7}$$

Onde $\alpha \in [0, 1]$ e $\gamma \in [0, 1]$.

A cada passo de execução, a função Q para um estado s e uma acção a evolui em função do valor da diferença entre o retorno estimado actualizado, composto pelo reforço recebido e pelo máximo valor estimado no estado seguinte da transição, e o valor actual conhecido.

O parâmetro γ serve o mesmo propósito que nos PDM: descontar o valor de acções futuras, enfatizando recompensas imediatas. O parâmetro α define o factor de aprendizagem, compreendido entre 0 e 1, representa a importância atribuída às novas experiências. Este providencia um mecanismo para o sistema poder suprimir a aprendizagem para quando, por exemplo, atinge uma fase em que a política comportamental desenvolvida até ao momento é considerada satisfatória para a resolução do problema. Quando não existir mais informação a extrair, como a actualização do valor de um par (estado,acção) é realizada a partir da diferença entre retorno estimado conhecido e actualizado, no infinito esta diferença deverá ser 0, tendo sido atingida a política comportamental óptima para esse estado.

A característica fundamental que diferencia este algoritmo do seu semelhante *SARSA* está no cálculo do retorno estimado, nomeadamente na selecção do valor do estado seguinte: o *Q-Learning* é um algoritmo *off-policy*, na medida em que qualquer que seja a política de selecção de acção utilizada no sistema, é sempre escolhida a acção a' que maximize o valor Q do estado s', e o algoritmo *SARSA* é *on-policy* que, como o nome indica, escolhe a acção a' mediante a política de selecção de acção do sistema. Este algoritmo não será apresentado em profundidade ou implementado nesta dissertação, mas considerou-se importante a sua referência para mostrar esta diferenciação entre algoritmos na valorização de um par (estado,acção)

No cenário dos PDM finitos com informação completa, a política comportamental pode ser determinada directamente a partir do modelo do mundo, mesmo antes de qualquer interacção com o ambiente. Desde que a dimensão

do problema seja compatível com a capacidade computacional do sistema, e que exista uma solução possível, eventualmente esta pode ser apurada. No entanto, em sistemas em que o mundo é desconhecido, existe obviamente a necessidade de explorá-lo, de modo a poder desenvolver uma política comportamental. O que pode não ser tão óbvio, sendo até um dos principais desafios na configuração destes algoritmos, é a quantidade de exploração que deve ser feita. Se um agente executar um determinado número de acções aleatórias e encontrar uma solução para o problema, deve considerar que a atingiu a política comportamental óptima? Podemos concordar que provavelmente não é possível numa só iteração chegar a uma solução óptima, mas não é impossível. Mas quantas iterações é que devemos considerar suficientes? Na realidade, não se pode atribuir um valor ao número de iterações necessárias, pois vão sempre variar consoante o problema.

Para lidar com o equilíbrio entre exploração e aproveitamento, ao longo do tempo foram sendo desenvolvidos vários mecanismos de política de selecção de acção, que podem ser mais ou menos adequadas consoante o problema em questão, ou que podem até ser variadas em sistemas mais complexos, de acordo com as necessidades do sistema. Algumas delas:

- *greedy* - Apenas escolhe a acção que maximize *Q(s,a)*, nunca explorando.

- ε*-greedy* - Com uma probabilidade de ε, segue uma política *greedy*, escolhendo a acção que maximize *Q(s,a)*. Com probabilidade 1 - ε, explora, no sentido em que escolhe uma acção aleatória $a \in A$

- *softmax* - Semelhante ao ε*-greedy* na sua base, com a diferença na exploração, atribuíndo pesos diferentes às várias acções na altura da escolha , consoante a sua função de valor, a função de distribuição utilizada e um parâmetro designado por temperatura τ, que define o quão diferenciados devem ser os pesos atribuídos a cada acção

A parametrização destas políticas pode ser fixa, no sentido em que é definida no inicio da execução e mantida durante todo o tempo, mas pode também ser variada ao longo da execução, de modo a optimizar o processo. Por exemplo, o valor de ε pode ser variado ao longo do tempo para acompanhar a acumulação de conhecimento.

Em suma, como demonstrado pelo pseudo-código apresentado em seguida, a cada passo de execução do agente são recolhidas todas as informações necessárias para actualizar a função Q: estado actual *s*, acção efectuada *a*, estado sucessor *s'*, recompensa recebida *r*. Com estes dados, o retorno esperado conhecido é actualizado com a diferença entre o próprio valor e o retorno estimado actualizado, baseado na recompensa e no retorno esperado máximo conhecido do estado *s'*. Quando é atingido um estado terminal, o ambiente deve ser reiniciado, mantendo a informação acumulada na função Q. Ao longo do tempo e de várias tentativas a resolver o problema, o sistema deve-se aproximar da solução óptima.

Algoritmo 2 *Q-Learning*

Initialize $\alpha \in [0, 1]$
Initialize $\varepsilon \in [0, 1]$ ▷ if action policy is ε based
Initialize $\gamma \in [0, 1]$
Initialize $Q(s, a) = 0, \forall s \in S, \forall a \in A$
Initialize s
while s is not a terminal state **do**
    Choose action $a \in A$ using chosen action selection policy
    Execute action a, observe reinforcement r and arrived state s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
end while

2.3.2 Aprendizagem com Modelo

Nos métodos vistos até agora, apesar das funções de valor serem actualizadas com base na experiência, a informação especifica da experiência é descartada, sendo apenas actualizado o valor dos pares estado-acção. Este tipo de algoritmos são designados *sem modelo* (*model-free*), ou seja, não criam um modelo do mundo.

O algoritmo *Dyna-Q* [17] acrescenta ao algoritmo *Q-Learning* um modelo do mundo, como definido nos PDM, nomeadamente as funções T e R. A cada passo, aquando da actualização da função Q, as mesmas informações são utilizadas para actualizar as funções T e R. Com um modelo, passam a existir duas formas de actualização da função de valor: a experiência real, e

os resultados de simulações utilizando o modelo do mundo. Dada a propriedade de Markov, que indica que o valor de um estado não depende de estados passados, uma simulação pode ser uma selecção aleatória de uma transição de estado e respectiva recompensa do modelo, e utilizar essa informação para actualizar a função Q. Sendo uma tarefa de custo computacional muito reduzido comparativamente à de uma experiência real e independente da própria execução do agente, esta pode ser repetida tantas vezes quanto possível e em paralelo com a execução, permitindo a convergência para uma política óptima a uma velocidade significativamente superior.

Algoritmo 3 Dyna-Q (com modelo determinista)

Initialize $\alpha \in [0, 1]$
Initialize $\varepsilon \in [0, 1]$ ▷ if action policy is ε based
Initialize $\gamma \in [0, 1]$
Initialize $n >= 0$
Initialize $Q(s, a) = 0, \forall s \in S, \forall a \in A$
Initialize $Model\ T(s, a) : s'$
Initialize $Model\ R(s, a) : r$
while s is not a terminal state **do**
    Choose action $a \in A$ using chosen action selection policy
    Execute action a, observe reinforcement r and arrived state s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
    Update Model $T(s, a) \leftarrow s'$
    Update Model $R(s, a) \leftarrow r$
    loopn times
        Select random $(s, a) \in T$ and respective s'
        Get r from $R(s, a)$
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    end loop
end while

É importante sublinhar que nesta especificação do algoritmo o modelo assume um ambiente determinista, sendo que a função T associa tuplos (s, a) a um estado seguinte s' ao invés de uma probabilidade, como visto na função $T(s, a, s')$ nos MDP. Num ambiente estocástico, como nunca se pode saber uma probabilidade real para atribuir às transições, é utilizada a frequência das ocorrências como uma probabilidade aproximada, sob a forma das funções

$\hat{T}$ e $\hat{R}$, que substituem as funções homólogas respectivas T e R.

$$\hat{T}(s,a,s') = \frac{\text{Quantidade de experiências (s,a,s')}}{\text{Quantidade de experiências (s,a)}} \tag{2.8}$$

$$\hat{R}(s,a) = \frac{\text{Total das recompensas recebidas por executar acção a no estado s}}{\text{Quantidade de experiências (s,a)}} \tag{2.9}$$

2.4 Sumário

Foram apresentadas as bases da aprendizagem por reforço, começando pelos seus elementos e pelas respectivas formalizações matemáticas, e como estas podem ser utilizadas para a resolução de problemas em que exista conhecimento absoluto do modelo dos mesmos, o que não constitui de facto aprendizagem, mas sim uma deliberação sobre o modelo. Em seguida, foram apresentados dois algoritmos utilizados quando não se possuí modelo, onde é necessário aprender com as observações parciais do mundo adquiridas ao longo do tempo. Esta visão geral da aprendizagem por reforço permite a compreensão do tema da dissertação, que se assenta nestas bases.

Capítulo 3

Aprendizagem por Reforço com Memória Episódica

Apesar de algoritmos como o *Q-Learning* se mostrarem muito eficientes em exemplos académicos controlados, a sua implementação em problemas no mundo real complica-se drasticamente, começando pela definição de estado. Como visto anteriormente, num problema de aprendizagem por reforço o estado define as características do ambiente relevantes para o objectivo, às quais o agente tem acesso na sua percepção do ambiente. Nos problemas tradicionais o espaço de estados é discreto, limitado e desenhado de forma a obedecer à propriedade de Markov, tal como nos exemplos mostrados até agora, como a movimentação de um agente num espaço bidimensional, onde o estado representa as coordenadas de uma posição no ambiente. No entanto, no mundo real, mesmo considerando apenas a posição do agente, o espaço de estados é contínuo e como tal computacionalmente incomportável e, além disso, a probabilidade de voltar a encontrar exactamente o mesmo estado é quase nula. Embora uma solução possível seja a discretização das percepções do mundo de modo a ter algum controlo sobre a dimensão do problema, não existe uma regra geral para esse processo, o que significa que apesar de se tratar de uma abordagem válida para alguns problemas, não o será para todos, podendo surgir problemas, por exemplo, se o nível de discretização for tal que ignore pormenores fulcrais ou demasiado pormenorizado para ser computacionalmente viável em termos de processamento. Ademais, no mundo real nem todos os acontecimentos têm a propriedade de Markov, dado que podem depender de estados anteriores.

Para abordar estas questões, consideremos o conceito de memória episódica. A noção operacional de episódio não é nova: é um termo utilizado regularmente na bibliografia de aprendizagem por reforço para denominar um conjunto de passos de execução de um agente num ambiente, por vezes utilizado como heurística para aferir a qualidade de um sistema, por exemplo, pela quantidade de episódios necessários para atingir a política óptima, ou pela dimensão dos episódios individuais, caso estes terminem num estado objectivo. É também utilizada nos métodos de Monte Carlo, como referido anteriormente. O uso que aqui se quer expor é o aproveitamento de certas características específicas desta estrutura de memória (ex: noção temporal) face às estruturas tradicionais para colmatar alguns dos problemas destas e também como uma forma de melhoria de desempenho.

De acordo com Gershman e Daw [3], um possível passo seguinte na evolução da aprendizagem por reforço pode ser a utilização de uma estrutura de memória episódica, não necessariamente substituindo as estruturas de memória semântica e procedimentais (apesar de possível), mas como uma estrutura adicional que permita colmatar os problemas referidos anteriormente. Isto faz sentido tanto de um ponto de vista operacional, na medida em que os sistemas de inteligência artificial têm de ser inevitavelmente compostos por múltiplas camadas de processamento, cada qual com a sua responsabilidade, sendo esta estrutura de memória uma proposta de uma nova camada, como do ponto de vista da analogia com a nossa compreensão da mente humana, cuja memória também é definida como um conjunto de vários subsistemas [4] (entre os quais a memória episódica e semântica, tal como aqui proposto para os nossos sistemas de aprendizagem por reforço).

Neste capítulo pretende-se definir a noção de episódio e memória episódica, algumas das possíveis vantagens e desvantagens desta estrutura, as formalizações matemáticas propostas, e como esta nova estrutura deverá operar num sistema de aprendizagem por reforço, seja de forma independente ou integrada em sistemas conhecidos.

3.1 Conceito de Episódio

Um episódio, semelhante ao observado nos métodos de Monte Carlo, trata-se de uma lista ordenada de tuplos *(estado, acção, recompensa)* que

representam uma sequência de estados pelo qual o agente passou durante a sua execução, as acções que executou e as recompensas que recebeu, sem qualquer tipo de processamento.

Este pode ser limitado temporalmente por um valor de passos de execução, ou por algum evento de maior relevância na execução, como receber um reforço positivo. A escolha é da responsabilidade de quem desenvolve o sistema: se o espaço de estados for de dimensões muito elevadas, terminar os episódios apenas aquando de uma recompensa positiva poderá ser um problema, tal como o é nos sistemas clássicos vistos até agora, pois dificulta a propagação de valor de estado. Nestes casos, a limitação temporal poderá ser vantajosa.

Esta estrutura não é fechada, sendo possível a extensão dos atributos armazenados em cada passo, para reduzir a carga computacional e/ou criar algoritmos diferentes. Um exemplo disto seria guardar o estado sucessor s' de um passo de execução, apesar implícito no passo seguinte, de modo a conter todo o processamento de um passo em si mesmo, evitando ter de obter o passo seguinte no episódio para saber o estado seguinte.

3.2 Formalização do Processo de Aprendizagem

Gershman e Daw [3] definem possíveis implementações de sistemas de aprendizagem por reforço, as quais foram consideradas como base de conhecimento para o estudo desta estrutura de memória, dado que expõem algumas das dificuldades que podem surgir e recomendam soluções para as mesmas.

A implementação mais simples de um sistema de aprendizagem por reforço com memória episódica consiste na acumulação de episódios ao longo da execução e, ao atingir um estado conhecido, estimar o valor das acções possíveis a partir dos episódios que contenham o dito estado:

$$Q_\pi(s_1, a) = E_\pi \left[\sum_{n=1}^{N} \gamma^{n-1} r_n | s_1, a \right] \approx \frac{1}{M} \sum_{m=1}^{M} R_m \tag{3.1}$$

A variável M representa a quantidade de trajectórias em memória que contenham o estado s_1 Como indicado, o valor de $Q(s, a)$ neste sistema de memória episódica é o retorno esperado acumulado e descontado Rm de todos os episódios relacionados com o estado s e com a acção a. Não é necessário

ter uma função de valor armazenada em memória, dado que os cálculos têm ser efectuados em tempo de execução, com a lista mais recente de episódios.

Surgem dois problemas com esta implementação: primeiro, dada a forma de calcular o valor estimado de um episódio, torna-se fundamental que estes tenham uma dimensão reduzida, caso contrário os episódios terão os reforços relevantes muito atenuados. Contudo, uma segmentação excessiva dos episódios pode resultar na perda de recompensas a longo prazo. O segundo problema envolve a generalização de estado, que surge principalmente quanto o agente é exposto a um ambiente contínuo: se a implementação pressupõe a passagem repetida num mesmo estado, e num ambiente deste género a probabilidade de encontrar o mesmo estado em múltiplas ocasiões é reduzida, então a probabilidade de encontrar episódios para valorizar um estado também será reduzida.

As secções seguintes apresentam as soluções referidas pelos autores para a resolução destes problemas, assim como outras possíveis questões que foram identificadas durante o desenvolvimento deste projecto.

3.2.1 Miopia do Futuro

Devido à segmentação de episódios e à forma como a recompensa acumulada é calculada, o agente pode se tornar míope relativamente a recompensas futuras, se estas se encontrarem noutros episódios. Desta forma, torna-se necessário acrescentar ao cálculo do valor de estado uma noção do retorno esperado após terminar um episódio, acrescentando a equação de Bellman à formulação inicial:

$$Q_\pi(s_1, a) = \frac{1}{M} \sum_{m=1}^{M} \left[R_m + \gamma^N \sum_s P(s_{N+1} = s | s_{mN}, \pi(s_{mN})) Q_\pi(S_{N+1}, \pi(s_{N+1})) \right] \tag{3.2}$$

A primeira parte da expressão é idêntica à primeira implementação, representando o retorno acumulado estimado do episódio, a segunda parte representa o retorno esperado após terminar o episódio. Este cálculo pode ser feito utilizando outros episódios que comecem onde o primeiro acaba, ou através de algoritmos clássicos com ou sem modelo do mundo. Esta implementação permite tornar os episódios mais curtos, e como tal obter valores de reforço mais precisos, e ao mesmo tempo sem ignorar recompensas a longo prazo.

3.2.2 Generalização

Quando o sistema é exposto a um problema do mundo real, onde os ambientes são contínuos e esparsos, existe uma probabilidade muito reduzida do agente atingir o mesmo estado múltiplas vezes. Consequentemente, torna-se necessário que o método de valorização de estados possua um mecanismo de generalização, para que o agente possa inferir valor em estados novos a partir da experiência de estados semelhantes, o qual pode ser definido da seguinte forma:

$$Q_\pi(s_1, a) = \frac{\sum_{m=1}^{M} R_m K(s_1, s_{m1})}{\sum_{m=1}^{M} K(s_1, s_{m1})} \tag{3.3}$$

Para um dado estado s_1, o valor é estimado a partir do retorno esperado de um episódio e de uma função de semelhança K, que deve devolver um valor de correspondência entre o estado s_1 e o estado inicial s_{m1} do episódio. A função é apresentada genericamente pois não está limitada a nenhuma função especifica, nem sequer limitada na parametrização: a comparação não tem de ser feita apenas entre estados, podendo ser acrescentadas as acções ou as recompensas, de acordo com as necessidades do problema. Uma função comum seria a gaussiana:

$$K(s, s') = \exp\left(- \frac{|| \ s - s' \ ||^2}{2\sigma^2} \right) \tag{3.4}$$

O parâmetro σ^2 determina o factor de generalização, isto é, valores mais baixos resultam numa generalização menor de estados, afastando-os mais, e valores mais altos geram maior semelhança entre estados.

3.3 Vantagens da Memória Episódica

3.3.1 Noção Temporal

Dado que estas estruturas de memória são constituídas pelos acontecimentos ordenados no tempo, sem qualquer tipo de processamento, torna-se possível a qualquer momento da execução a revisão dos factos ocorridos no passado. A partir do instante em que exista um episódio que cumpra os objectivos, este poderia ser perpetuamente repetido, providenciando uma

solução provavelmente longe de ser óptima, mas de reduzida complexidade computacional. Outra possibilidade seria a utilização da informação para actualizar uma função de valor.

3.3.2 Quasi-modelo

Como referido, um modelo do mundo é caracterizado pelas funções $T(s, a, s')$ e $R(s, a, s')$, representando, respectivamente, a probabilidade de transição de estado (T = 1 para todos os estados se o ambiente for determinístico) e o reforço gerado pelas transições. Toda esta informação pode ser recolhida numa estrutura episódica, dado que cada passo contém um estado inicial e estado sucessor, juntamente com a acção relativa à transição (equivalente à função T) e o respectivo reforço (equivalente à função R), pelo que apesar de não constituir realmente um modelo por definição, poderia ser utilizado como tal num algoritmo que envolva simulações, como o DynaQ.

3.3.3 Factor Urgência

Supondo um algoritmo de aprendizagem do agente que define um parâmetro $\lambda \in [0, 1]$, representativo da urgência do agente em obter a acção seguinte a executar [13], sendo que $\lambda = 0$ significa nenhuma urgência, ou seja, é irrelevante para o agente o tempo de execução necessário para obter a acção seguinte, e $\lambda = 1$ urgência total, sendo necessário reduzir ao máximo o tempo de processamento. Se nesse algoritmo fosse utilizado um mecanismo de memória episódica, o parâmetro λ poderia influenciar, por exemplo, a quantidade de episódios a processar, sendo que no limite, em estado de urgência máxima, o agente poderia repetir o passo de execução mais recente e relacionado com o seu estado actual, independentemente do ganho ou perda resultante dessa escolha.

3.3.4 Sub-episódios

Os episódios devem representar os factos relativos à operação do agente e não devem ser manipulados, de modo a mantê-los como tal. No entanto, isso não torna impeditivo tirar proveito da propriedade de Markov, quando o problema permite, que indica que o valor de um estado não depende dos es-

tados que o antecedem, e trabalhar com sub-episódios, ignorando informação que seja redundante ou até mesmo prejudicial para a aprendizagem.

3.3.5 Múltiplos Objectivos

Sendo o propósito de um sistema de aprendizagem por reforço maximizar uma estimativa de valor em todos os estados do problema relativamente a um estado objectivo, através de múltiplas iterações do problema, é expectável que, após uma certa quantidade de iterações, a maioria dos estados estejam valorizados tal que a política comportamental resultante leve o agente na direcção do objectivo. Contudo, num cenário com múltiplos objectivos utilizando, por exemplo, *QLearning*, o primeiro estado objectivo atingido após exploração irá gerar um óptimo local em memória, reduzindo drasticamente a probabilidade de atingir os restantes objectivos.

Ainda no algoritmo *Q-Learning*, uma solução possível seria a criação de uma função Q para cada objectivo individual [14], no entanto esta solução pode se tornar demasiado complexa em termos de escala: se já foi constatada a possibilidade de uma única função Q chegar a dimensões incomportáveis em determinados problemas, então torna-se seguro afirmar que provavelmente ter múltiplas funções de valor também se pode tornar problemático (e a uma escala muito superior).

3.3.6 Estatísticas

Ao contrário da função Q, que integra ao longo do tempo o valor estimado a partir das observações do agente, com esta estrutura de memória são armazenadas todas as observações do agente, mesmo que redundantes entre episódios. Isto permite contabilizar quantas vezes foi atingido o mesmo estado ou foi efectuada uma acção pelo agente, podendo ser útil como uma heurística para a selecção de acção, ou como um factor de ajuste dos parâmetros de aprendizagem ao longo do tempo. Esta contabilização também poderia ser feita a par com o algoritmo *Q-Learning* por meio de estruturas de armazenamento adicionais, mas no caso de utilização de estruturas de memória episódica, essa contagem por ser feita directamente sobre os episódios armazenados.

Estas estatísticas poderiam ser utilizadas, por exemplo, para influenciar

o parâmetro α, mediante a quantidade de visitas a um dado estado, idealmente a tender para 0. Após um determinado numero de visitas, o agente deixaria de aprender novas informações acerca desse estado, considerando apenas o valor estimado até ao momento. Poderia também ser efectuado o mesmo tratamento ao parâmetro ε no caso de uma política de selecção de acção baseada nesse valor, reduzindo ao longo do tempo a probabilidade de escolher uma acção aleatória, atribuindo maior importância à acção com o valor estimado máximo num dado estado.

3.4 Desvantagens

3.4.1 Memória Limitada

No algoritmo *Q-Learning*, a dimensão da função $Q(s, a)$ está limitada pela dimensão combinada dos conjuntos de estados e acções disponíveis. No entanto, num cenário com memória episódica, esta não está limitada pelas dimensões do problema, dado que são acumulados todos os passos de execução. Neste cenário, deverá sempre existir um limite na quantidade de informação que um agente com memória episódica pode reter, sendo que a solução não pode ser tão simples como eliminar episódios antigos, pois a perda dessa informação constitui na realidade um processo oposto ao de aprendizagem. Poderíamos argumentar que dada a ordem temporal dos episódios, que os mais antigos deverão conter mais informação redundante ou irrelevante para atingir o objectivo, pois habitualmente são nestes onde ocorre mais exploração, e que como tal faria todo o sentido descartar essa informação. Contudo, esses mesmos episódios, por conterem mais informação resultante de exploração, devem provavelmente conter mais informação acerca do ambiente além do objectivo, como por exemplo obstáculos a evitar. Ao eliminar esses episódios, o agente pode não ser afectado do ponto de vista de atingir o objectivo, mas provavelmente será afectado o conhecimento do restante ambiente, o que pode contribuir para uma solução sub-óptima ou até inaceitável.

Outra solução possível poderia ser a filtragem de episódios de acordo com alguma função de valorização de episódio (valorização no sentido de relevância do episódio para o problema ao invés de valorização relativamente ao objectivo como discutido em temas anteriores), o que mais uma vez iria resultar em perda de informação como referido anteriormente, a menos que,

por exemplo, essa função de valorização classificasse os episódios em termos de semelhança, de modo a eliminar episódios redundantes, mas isso serviria apenas para adiar o limite.

3.4.2 Tempo de Resposta

Como visto anteriormente, ao longo do tempo são armazenados os passos de execução, que podem posteriormente ser processados no momento de selecção de acção. Se esta acumulação não for explicitamente limitada, e os cálculos efectuados em tempo real, como nos algoritmos propostos por Gershwan e Daw [3], a complexidade de processamento irá crescer proporcionalmente à quantidade de episódios, resultando em respostas mais demoradas, mesmo antes de atingir qualquer limite máximo de memória computacional. Uma solução possível seria o processamento de cada episódio quando este é armazenado, criando uma estrutura adicional para guardar o resultado do processamento. No momento de decisão, o agente poderia optar entre consultar os valores já processados, ou efectuar um novo processamento dos episódios, consoante a necessidade.

3.5 Sumário

Foram identificados alguns problemas que podem surgir com as implementações clássicas de aprendizagem por reforço, e apresentadas abordagens possíveis para colmatar os mesmos. Foi apresentada a noção de episódio e de memória episódica, e enumeradas algumas das vantagens e desvantagens identificadas na utilização de mecanismos de memória episódica no contexto da aprendizagem por reforço.

Capítulo 4

Abordagem Proposta

Determinou-se que a abordagem ideal para observar e aferir a viabilidade da utilização de estruturas de memória episódica em sistemas de aprendizagem por reforço, seria o desenvolvimento de software, especificamente uma biblioteca, que colocasse em prática alguns algoritmos clássicos do tema, e algoritmos baseados em memória episódica, de modo a poderem ser comparados entre si, assim como testar e identificar possíveis sinergias entre uma estrutura de memória episódica e métodos já conhecidos, tendo por base os algoritmos apresentados por Gershwan e Daw [3] descritos no capítulo anterior.

Pretende-se neste capítulo definir os objectivos que se esperam cumprir com o desenvolvimento da biblioteca referida, assim como apresentar os algoritmos a implementar além dos já vistos.

4.1 Propósito

Pressupondo sistemas já existentes responsáveis por proporcionar um agente e um ambiente em que este possa observar e operar, pretendeu-se desenvolver uma biblioteca capaz de, a partir das informações obtidas por um agente resultantes da sua execução, processá-las num algoritmo de aprendizagem por reforço e retornar acções para o agente executar, baseadas no resultado desse processamento. Seria ideal que esta fosse tão modular quanto possível, tanto em termos do algoritmo base a utilizar, como nas estruturas de armazenamento de informação e em toda a parametrização permitida pelo algoritmo, de forma a permitir um controlo granular sobre os cenários de

teste. Em termos de algoritmos, pretende-se desenvolver dois dos algoritmos clássicos apresentados anteriormente: o *Q-Learning* e o *Dyna-Q*, como demonstração das implementações clássicas e já conhecidas, assim como as implementações sugeridas por Gershwan e Daw, e também algumas tentativas experimentais, apresentadas em seguida.

4.2 *Q-Learning* com Simulação Episódica

Apesar de uma estrutura de memória episódica não se tratar de um modelo de mundo, esta contém as mesmas informações – as funções de transição T e de recompensa R podem ser observadas num único passo de execução se este incluir o estado seguinte s', ou em dois passos, aproveitando a ordenação temporal de um episódio.

Desta forma, à semelhança do que é feito nos algoritmos de *Deep Q-Network (DQN)* [15] torna-se possível desenvolver uma variação do algoritmo *Dyna-Q*, aqui designado por *Q-Learning* com Simulação Episódica, que mantém o modo de funcionamento base, na medida em que é mantida uma função de valor que não só é actualizada com base na experiência, como também através de simulações baseadas na informação de um modelo interno do mundo, com a diferença apenas na fonte de informação para simulação: ao invés de ser consultado um modelo para obter um estado aleatório, é consultada uma instância de memória episódica, sendo seleccionado um passo aleatório de um episódio aleatório.

No caso de um problema estocástico, o algoritmo *Dyna-Q* precisa, como já referido, de uma estrutura adicional para armazenar as estatísticas de execução, de modo a poder calcular as funções $\hat{T}$ e $\hat{R}$, que indicam a probabilidade de transição de estado. Neste caso específico, a memória episódica é mais eficiente, na medida em que dado que são armazenados todos os episódios (mesmo que redundantes), ao seleccionar um passo aleatório de um episódio aleatório, a probabilidade de transição está implícita na própria selecção.

4.2.1 Objectivo Único

Num cenário em que o agente tenha apenas um objectivo único para resolver o problema, por exemplo, atingir uma determinada posição num ambi-

ente bidimensional, este algoritmo deve comportar-se de forma semelhante ao algoritmo *Dyna-Q* em ambientes deterministas, diferenciando-se apenas na forma como as informações do ambiente são armazenadas. A cada passo de execução, são seleccionados n passos aleatórios de episódios aleatórios para efectuar simulações de execução para actualizar a função Q.

Algoritmo 4 *Q-Learning* with Episodic Simulation

Initialize $\alpha \in [0, 1]$
Initialize $\varepsilon \in [0, 1]$ ▷ if action policy is ε based
Initialize $\gamma \in [0, 1]$
Initialize $n >= 0$
Initialize $Q(s, a) = 0, \forall s \in S, \forall a \in A$
Initialize $EpisodicMemory$
while s is not a terminal state **do**
    Choose action $a \in A$ using chosen action selection policy
    Execute action a, observe reinforcement r and arrived state s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
    Update $EpisodicMemory$ current episode with (s, a, r, sn)
    if $r > 0$ **then** ▷ goal state
        Finish current episode, store in $EpisodicMemory$
        Reset current episode
    end if
    loop n times
        Select random step (s, a, r, sn) from a random
        episode in $EpisodicMemory$
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    end loop
end while

4.2.2 Múltiplos Objectivos

Num cenário com múltiplos objectivos, pode ocorrer um de dois problemas na função $Q(s, a)$: se os estados objectivo tiverem recompensa única, ou seja, só geram reforço positivo ao serem atingidos uma vez (ex: um agente que esteja a apanhar objectos num espaço), quando o objectivo é atingido, a função de valor passa a estar desactualizada, dado que as estimativas de valor foram feitas com base na existência de um objectivo que deixou de existir. O segundo problema surge no cenário oposto, onde os estados objectivo geram reforço positivo em todas as vezes que são atingidos: os estados mais próximos desse objectivo irão sempre valorizar mais acções que levem a esse estado, impedindo o agente de atingir objectivos mais afastados.

Uma solução possível para estes problemas passa por utilizar memória episódica para reconstruir a função de valor. À medida que o agente atinge estados objectivo, a memória da função de valor para todos os estados é eliminada, e o respectivo estado passa a ser ignorado. Todas as simulações posteriores para alimentar a função de valor utilizam apenas episódios que não contenham esse estado.

Algoritmo 5 *Q-Learning* with Episodic Simulation (Multiple Goals)

Initialize $\alpha \in [0, 1]$
Initialize $\varepsilon \in [0, 1]$ ▷ if action policy is ε based
Initialize $\gamma \in [0, 1]$
Initialize $n >= 0$
Initialize $Q(s, a) = 0, \forall s \in S, \forall a \in A$
Initialize $EpisodicMemory$
Initialize *List of ignored states*
while s is not a terminal state **do**
    Choose action $a \in A$ using chosen action selection policy
    Execute action a, observe reinforcement r and arrived state s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
    Update $EpisodicMemory$ current episode with (s, a, r, sn)
    if $r > 0$ **then** ▷ goal state
        Finish current episode, store in $EpisodicMemory$
        Reset current episode
        Reset the Q function
        if more goals available **then**
            Add goal state to list of ignored states
        else
            Reset ignored states list
        end if
    end if
    loopn times
        Select random step (s, a, r, sn) from a random episode in the
        $EpisodicMemory$ that does not contain any of the ignored states
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    end loop
end while

4.3 *Q-Learning* com Propagação Episódica

Proposto por Morgado e Gaspar [13], este algoritmo define uma estrutura de memória ligeiramente diferente das vistas até agora, de modo a combinar memória episódica com a estrutura de memória do *Q-Learning*. É definida uma lista de todos os estados visitados, e a cada estado é associado a uma lista das acções efectuadas no mesmo. Cada acção por sua vez é associada ao túplo (estado seguinte, recompensa, valor Q). No fim de cada episódio, o mesmo é iterado no sentido inverso, actualizado o valor Q em cada estado percorrido.

Algoritmo 6 *Q-Learning* with Episodic Propagation

Initialize $\alpha \in [0, 1]$
Initialize $\in [0, 1]$ ▷ if action policy is ε based
Initialize $\gamma \in [0, 1]$
Initialize $Q(s, a) = 0, \forall s \in S, \forall a \in A$
Initialize $StructuredMemory$
Initialize $CurrentEpisode$
while s is not a terminal state **do**
    Choose action $a \in A$ using chosen action selection policy
    Execute action a, observe reinforcement r and arrived state s'
    Append current state s to the current episode
    Insert state s in StructuredMemory
    Insert action a in StructuredMemory[s] and create/update tuple (s', r, Q) ▷ Q calculated as Q-Learning
    if s' is a goal state **then**
        Append arrived state s' to the current episode
        Iterate current episode in reverse order and update Q values in StructuredMemory for each state (using the already known transitions and rewards)
        Reset the current episode
    end if
end while

Para tornar a exploração mais eficiente, foi proposta também uma variação da política de selecção de acção $\varepsilon - greedy$, com a funcionalidade adicional de filtragem da lista de acções disponíveis, de modo a priorizar as que ainda não tenham sido tomadas, em situações em que para um determinado estado não seja conhecida uma acção com retorno estimado positivo, tornando a exploração um pouco mais eficiente ao invés de ser completamente aleatória.

Algoritmo 7 E-Not-so-greedy policy

```
1: if no actions have a positive value then
2:     Perform ε-greedy selection with list of non explore actions
3: else
4:     Perform ε-greedy selection with list of all available actions
5: end if
```

4.4 Sumário

Foi apresentada a abordagem determinada para verificar se a utilização de mecanismos de memória episódica podem ser úteis para sistemas de aprendizagem por reforço, que consiste no desenvolvimento de uma biblioteca de aprendizagem por reforço. Esta deve definir algoritmos clássicos de aprendizagem por reforço e os algoritmos apresentados no capítulo 3 baseados em memória episódica. Foram expostos outros algoritmos a implementar que tiram partido de uma estrutura de memória episódica como uma fonte de informação adicional.

Capítulo 5

Plataforma de Aprendizagem por Reforço

Como referido no capítulo anterior, foi desenvolvida uma biblioteca de aprendizagem por reforço, para colocar em prática os algoritmos apresentados até agora, designada por Plataforma de Aprendizagem por Reforço (PAR).

Foi escolhida a linguagem *Python* para a construção da biblioteca, primeiramente devido a esta também ser utilizada pela plataforma de simulação de ambientes utilizada, que será apresentada no próximo capítulo, e também por ser uma ferramenta que permite o desenvolvimento rápido de protótipos, providenciando uma forma rápida de aferir se os algoritmos testados são viáveis de incluir na biblioteca. De acordo com um estudo efectuado pela a empresa GitHub [16], em 2018 esta foi a linguagem mais utilizada em projectos de *Machine Learning* nos projectos alojados no seu serviço, demonstrando que o mercado também se está a direccionar para esta linguagem, o que por sua vez contribui para o desenvolvimento de mais bibliotecas para apoiar este tipo de projectos.

Neste capítulo pretende-se apresentar a biblioteca desenvolvida e descrever a organização do ponto de vista da engenharia de software, identificando os objectos do domínio do problema e detalhando as implementações específicas das várias componentes do sistema.

5.1 Objectos de Domínio

De modo a manter uma estrutura robusta, capaz de suportar a implementação de novos algoritmos e de fácil reconfiguração de parâmetros teve de ser definida uma arquitectura base sobre a qual seriam implementados os elementos concretos do sistema. Como tal, foi necessário identificar os objectos do domínio do problema em questão, para criar as respectivas abstracções.

De acordo com o que foi visto no capítulo de aprendizagem por reforço, um sistema desta natureza consiste em três elementos base: um algoritmo de aprendizagem para processamento das experiências do agente, uma estrutura de memória para armazenar os resultados desse processamento e um mecanismo de selecção de acção. Além destes, foi necessário definir também a noção de modelo do mundo, para suportar algoritmos que dependam do mesmo.

5.1.1 Algoritmo de Aprendizagem

Esta classe abstracta define a estrutura base de um algoritmo de aprendizagem por reforço: este deve definir um método de aprendizagem baseado nas observações do ambiente, assim como ter acesso a uma estrutura de memória e a uma política de selecção de acção. É a partir desta abstracção que devem ser implementados os algoritmos concretos.

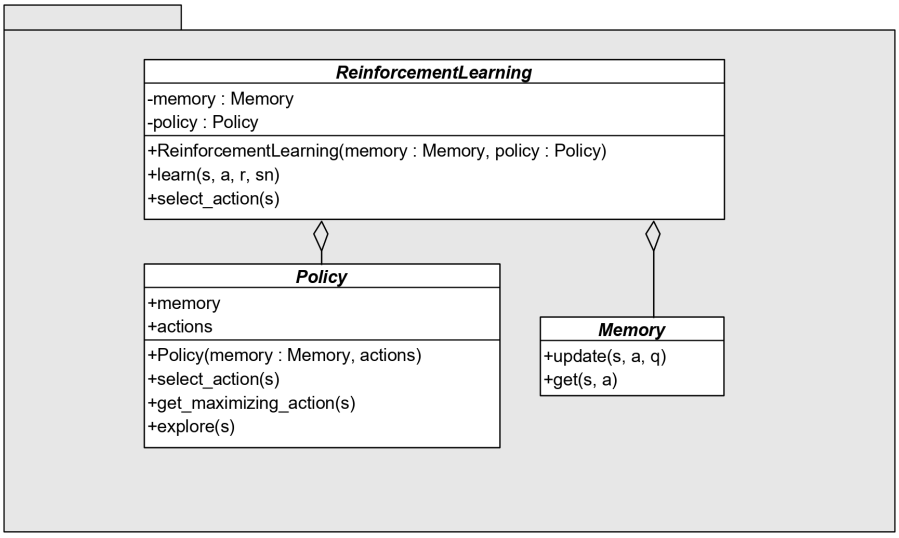

Figura 5.1: Vista geral da arquitectura da Plataforma de Aprendizagem por Reforço

5.1.2 Memória

Esta abstracção define a implementação mínima que um mecanismo de memória deve especificar. A estrutura específica da memória não foi definida na classe abstracta porque esta deverá variar entre concretizações. Foram apenas definidas as funções base que todas as estruturas devem ter: uma função de armazenamento, e uma função de obtenção de dados.

5.1.3 Política de Selecção de Acção

A política de selecção de acção é um mecanismo que deve ser independente da restante configuração do sistema, ou seja, tanto quanto possível, qualquer política deve ser compatível com qualquer algoritmo e estrutura de memória. Como tal, tornou-se necessário definir uma especificação mínima comum para todas as implementações serem compatíveis com a função de selecção de acção do mecanismo de aprendizagem geral. Uma política deve ter acesso à memória e à lista de acções possíveis do agente, e deve no mínimo disponibilizar três funções: selecção de acção de acordo com o algoritmo, uma função que retorne a acção que maximize a função de valor, e uma função de exploração.

5.2 Tipos de Memória

Este subsistema engloba a abstracção de memória e as suas respectivas extensões, desenvolvidas para os algoritmos implementados: memória esparsa (*SparseMemory*), memória estruturada (*StructuredMemory*) e memória episódica (*EpisodicMemory*). Inclui também a definição de episódio (*Episode*), utilizada pela memória episódica.

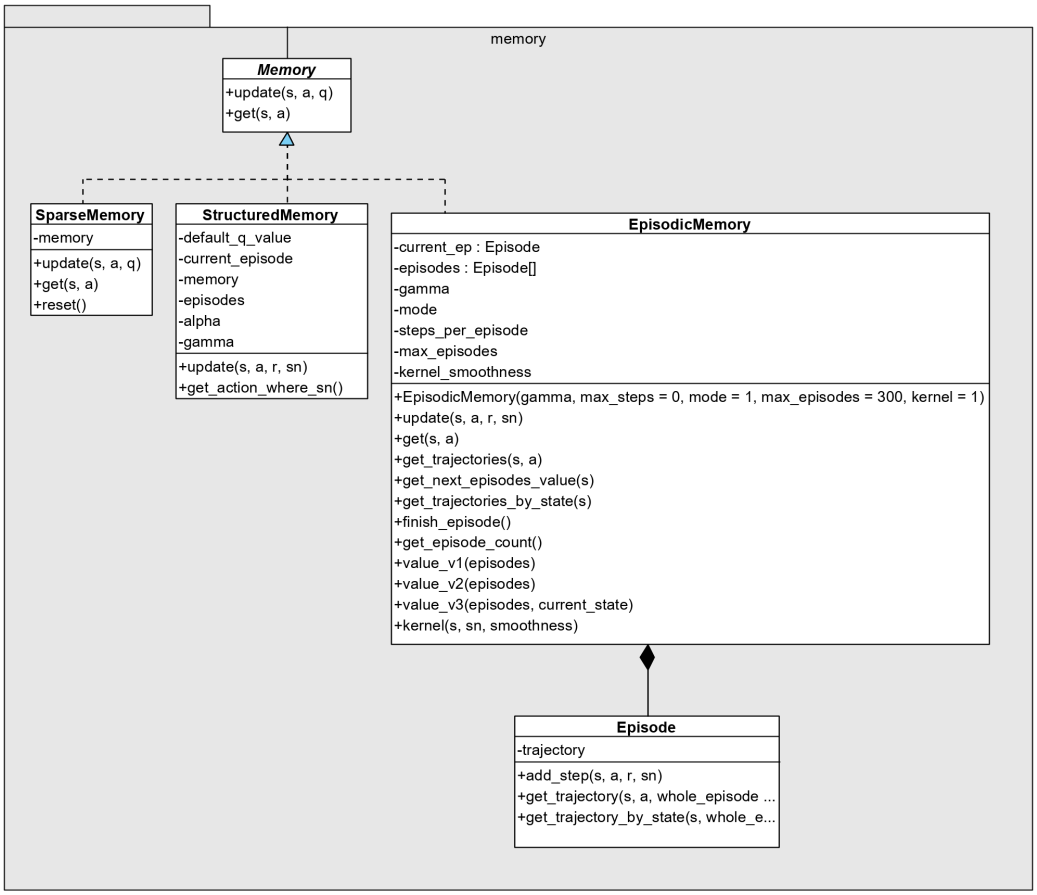

Figura 5.2: Arquitectura do subsistema de memória

5.2.1 Memória Esparsa

Para diferenciar os tipos de memória, foi designada como memória esparsa a estrutura clássica definida pelos algoritmos de diferença temporal, onde é associado um valor Q a pares (*estado,acção*), neste caso representados por um dicionário que mapeia tuplos $(s, a) \rightarrow \mathbb{R}$.

As funções de actualização e obtenção de informação foram definidas aproveitando as funções nativas de um dicionário em *Python*, sendo que em casos em que se procura um tuplo ainda não mapeado, é retornado um valor por omissão.

5.2.2 Memória Estruturada

Proposta por Morgado [13], esta estrutura combina memória episódica com um dicionário que mapeia estados em dicionários de acções, sendo estes associados ao passo de execução mais recente, representados pelo tuplo $(estadosucessor, recompensa, valorQ)$. Este tipo de memória é exclusivo ao algoritmo *Q-Learning* com propagação episódica.

5.2.3 Memória Episódica

A memória episódica foi definida como duas listas ordenadas: uma para armazenar o episódio actual, e outra para armazenar episódios concluídos.

A função de actualização foi configurada para armazenar toda a informação disponibilizada pelo agente como um passo de execução no episódio actual. Se for indicada uma recompensa positiva, o episódio é concluído e guardado na lista.

A função de pesquisa procura em todos os episódios por trajectórias que contenham o estado e acção em questão e calcula o valor destes segundo uma função de valorização, que foi definida segundo a implementação base referida por Gershwan e Daw. [3]

5.3 Algoritmos de Aprendizagem

O subsistema de algoritmos de aprendizagem contém a abstracção *ReinforcementLearning*, que define a implementação base de um algoritmo de aprendizagem, e a sua respectiva concretização no algoritmo *Q-Learning*. A partir deste, são definidas extensões que definem os restantes algoritmos utilizados neste trabalho: *Dyna-Q*, *Q-Learning* com Simulação Episódica (*QLearningEpisodicSimulation*) e Aprendizagem com Memória Episódica (*QLearningEpisodic*). É também implementado neste subsistema o conceito de episódio para uso no algoritmo *Dyna-Q*

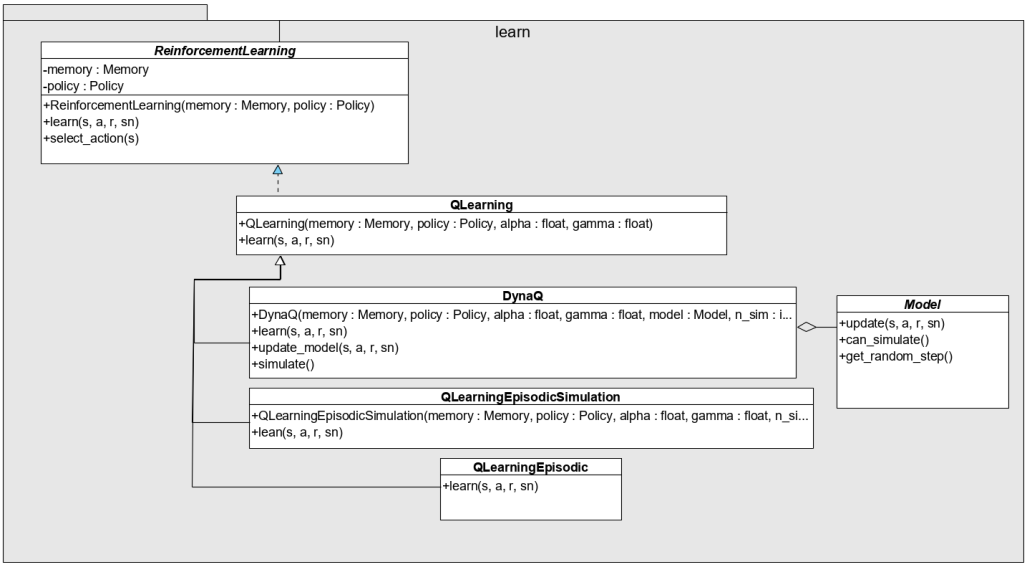

Figura 5.3: Arquitectura do subsistema de algoritmos de aprendizagem

5.3.1 *Q-Learning*

Foi estendida a abstracção base com os parâmetros *alpha* e *gama*, para que possam ser alterados aquando da configuração do sistema. A função de aprendizagem foi implementada segundo o algoritmo em pseudo-código apresentado no capítulo 2, obtendo a' a partir da política de selecção de acção, e os valores de $Q(s, a)$ e $Q(s', a')$ a partir da memória, armazenando na mesma a função Q actualizada.

5.3.2 Dyna-Q

Este algoritmo estende o algorimo *Q-Learning*, dado que o funcionamento base é idêntico. Foi acrescentado um parâmetro *nsim* representativo da quantidade de simulações a efectuar e os atributos representativos de um modelo do mundo, assim como as respectivas funções de simulação e actualização. O modelo do mundo é composto pelas funções $T(s, a) : s'$ e $R(s, a) : r$, sob a forma de dicionários.

A cada passo, para além da actualização da função Q, é actualizado o modelo com as informações da percepção do agente. Após as actualizações, são feitas tantas pesquisas no modelo quantas as definidas pelo parâmetro

de quantidade de simulações, sendo os resultados utilizados também para actualizar a função Q.

5.3.3 Aprendizagem com Memória Episódica

Seguindo o exemplo apresentado por Gershwan e Daw [3], este foi o primeiro algoritmo implementado para validar o funcionamento da memória episódica. A função de aprendizagem neste caso apenas encaminha as percepções do agente para armazenamento em memória, sendo que todos cálculos estão contidos na implementação de memória episódica, dado que todas as valorizações de estado são calculadas em tempo real a partir dos episódios armazenados.

5.3.4 *Q-Learning* com Simulação Episódica

À semelhança do algoritmo *Dyna-Q*, este começa como uma extensão do algoritmo *Q-Learning*, com a adição de dois parâmetros: um de quantidade de simuações e efectuar, e um segundo para indicar se existem múltiplos objectivos. Ao invés de ser instanciado um modelo, é criada uma instância de memória episódica (a mesma utilizada nos algoritmos anteriores), que é utilizada da mesma forma que o modelo: a cada passo de execução, além de ser actualizada a função Q, o mesmo é adicionado à instância de memória episódica. Após as actualizações, é feita a simulação sobre os episódios, sendo seleccionado um passo aleatório de um episódio aleatório, e essa informação utilizada para actualizar a função Q.

Quando é indicado que o problema contém múltiplos objectivos, é instanciada uma lista de estados a ignorar. Quando é atingido um estado objectivo, se ainda existirem outros, este é acrescentado à referida lista, e a função Q é reiniciada. Todas as simulações seguintes passam a ser feitas apenas com episódios que não contenham os estados ignorados. Quando todos os objectivos são atingidos e o ambiente é reiniciado a função Q também é reiniciada, assim como a lista de estados ignorados, permitindo que as simulações passem a ser feitas novamente com todos os episódios.

5.3.5 *Q-Learning* com Propagação Episódica

Este algoritmo estende o algoritmo *Q-Learning*, ocorrendo neste caso a cada passo duas actualizações na memória: primeiramente, à semelhança do que é feito nos algoritmos com memória episódica, é adicionada a informação do passo ao episódio actual, assim como à estrutura de memória proposta pelos autores do algoritmo, que consiste num dicionário que mapeia estados como dicionários de acções, que por sua vez contém a informação de estado sucessor, recompensa, e valor Q. Por último, é feita a actualização do valor Q para a transição ocorrida, tal como no *Q-Learning*.

No fim de cada episódio, este é percorrido inversamente e a informação utilizada também para actualizar os valores Q de cada estado percorrido.

5.4 Políticas de Selecção de Acção

Uma política deve ter acesso à memória e à lista de acções possíveis do agente, e deve no mínimo disponibilizar três funções: selecção de acção de acordo com o algoritmo, uma função que retorne a acção que maximize a função de valor, e uma função de exploração.

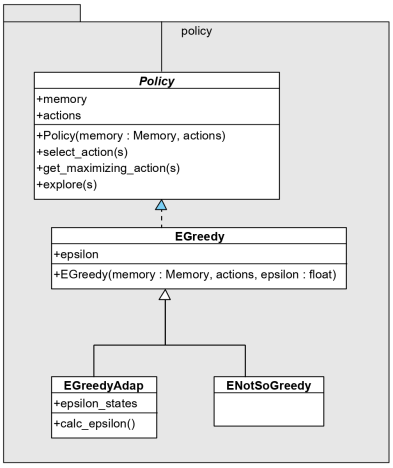

Figura 5.4: Arquitectura do subsistema de política de selecção de acção

5.4.1 ε-greedy

A função de selecção de acção gera um valor aleatório entre 0 e 1, e é comparado a um parâmetro $\varepsilon \in [0, 1]$ definido na configuração. Se o valor aleatório for superior a ε, é escolhida a acção que maximize o valor armazenado em memória, caso contrário é seleccionada uma acção aleatória, leia-se, explora.

5.4.2 ε-not-so-greedy

Antes de ser gerado um valor de ε, é verificado se existem em memória pares (estado actual, acção) com um valor superior a 0. Caso existam, é aplicado o algoritmo ε-greedy clássico tendo em conta todas as acções possíveis, caso contrário o mesmo é aplicado apenas com acções que não tenham sido exploradas.

5.5 Instanciação dos Mecanismos

Para facilitar a instanciação do sistema de aprendizagem (e também para facilitar a integração do sistema de visualização de estatísticas, descrito no capítulo seguinte), foi criada uma classe apenas com funções estáticas, concentrando toda a parametrização numa única chamada.

5.6 Sumário

Neste capítulo foi apresentada a Plataforma de Aprendizagem por Reforço (PAR), a biblioteca de aprendizagem por reforço desenvolvida no contexto da dissertação, do ponto de vista do desenvolvimento do software em concreto. Foram vistos os objectos de domínio do problema identificados e a sua consequente transformação em subsistemas que comunicam entre si, para colocar em prática os algoritmos de aprendizagem por reforço que se pretendiam estudar.

Capítulo 6

Concretização Experimental

Com base na biblioteca desenvolvida, foi realizada a integração com um agente e um ambiente para observação de resultados experimentais. Para tal, foi utilizada a PSA (Plataforma de Simulação de Agentes) [12], desenvolvida pelo Professor Doutor Luís Morgado, um sistema que efectua a simulação de um ambiente 2D que pode ser pré-configurado com alguns elementos, como estados obstáculo e estados objectivo, e de um agente capaz de executar acções de movimento no mundo. A PSA permite inicializar uma instância de ambiente e agente, e aceder às percepções do agente (de onde se extrai o estado actual), assim como passar acções ao agente para as executar, permitindo desta forma criar algoritmos que efectuem essa selecção de acção. A integração de um sistema de aprendizagem com a PSA é feita através da implementação de uma classe de agente e uma classe de controlo pré-determinadas pela documentação da mesma.

Neste capítulo pretende-se descrever o sistema agente/ambiente, os cenários de teste considerados e os seus respectivos resultados experimentais.

6.1 PSA — Plataforma de Simulação de Agentes

A PSA é uma aplicação que simula a execução um ou mais agentes num ambiente configurado pelo utilizador. Os ambientes são criados em ficheiros de texto e a sua sintaxe pode ser consultada no anexo A.

Para interagir com o ambiente, foi implementado um agente, estendendo a classe Agente definida pela PSA e definindo uma função de execução do

mesmo. A implementação de um agente implica também a definição de uma classe Controlo, responsável pelo processamento das percepções do Agente. É nesta classe que são instanciados os mecanismos de motivação e de aprendizagem, sendo o primeiro responsável por transformar a percepção do agente num valor de reforço, e o segundo pela aprendizagem a partir da informação da percepção, recompensa e acção anterior, assim como por gerar acções para o agente executar.

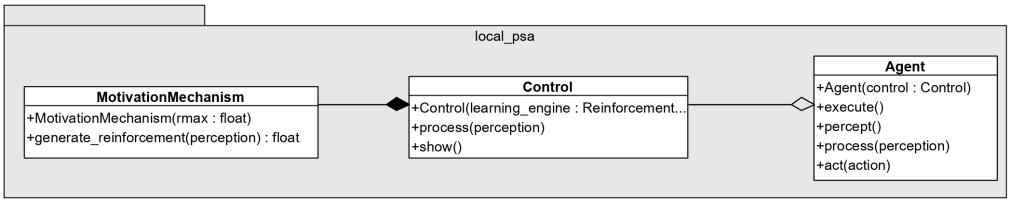

Figura 6.1: Arquitectura do subsistema relativo à PSA

Em termos de visualização de informação, a PSA permite a observação em tempo real da execução do agente, incluindo informação de sensores que este possui. São também disponibilizados espaços no ecrã onde pode ser visualizada a estrutura de memória do agente, mediante o desenvolvimento de uma lista de valores compatível com a mesma. No fim da execução podem também ser recolhidos dados estatísticos da mesma, tais como o número total de episódios ou de passos de execução.

6.2 Visualização de Estatísticas

6.2.1 RLBackend

Utilizando a *framework Flask* em *Python*, foi criado um serviço que recorre a introspecção sobre a classe estática referida anteriormente para listar os métodos de aprendizagem disponíveis e os respectivos parâmetros, assim como lançar instâncias da PSA segundo uma determinada parametrização e devolver os resultados da execução.

Esta foi uma das razões que contribuiu para a criação da classe responsável pela instanciação dos sistemas de aprendizagem: através de reflexão, são recolhidas as listagens dos sistemas disponíveis e da sua respectiva parametrização, que podem ser posteriormente enviadas para a aplicação. Desta

forma, se surgir a necessidade de implementar novos algoritmos, estes estarão automaticamente disponíveis neste serviço, o que era fulcral para optimizar o processo de recolha de resultados. A listagem dos métodos fornecidos pelo serviço foi incluída no Apêndice B.

6.2.2 RLFrontend

Recorrendo à *framework AngularJS*, foi criada uma aplicação *web* que comunica com o serviço *RLBackend* criado para providenciar uma interface gráfica para a parametrização de sistemas e visualização dos respectivos resultados.

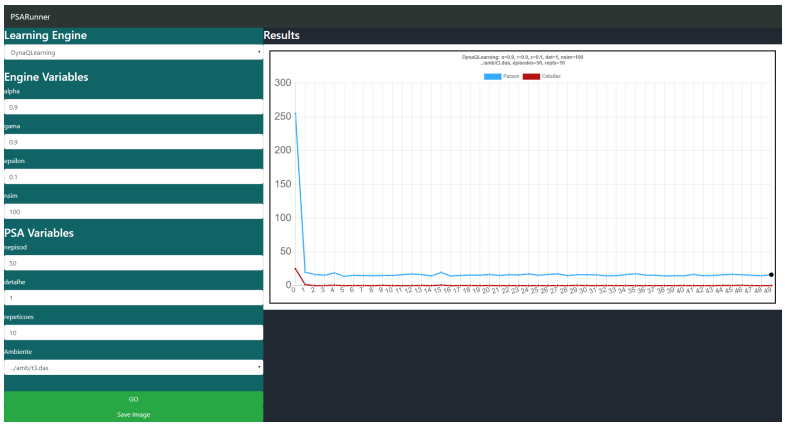

Figura 6.2: Ferramenta RLFrontend

A aplicação comunica com o *RLBackend* para obter informações de parametrização dos algoritmos, e para enviar as respectivas concretizações desses parâmetros para criar instâncias da PSA e do PAR para serem executadas num determinado ambiente. No fim da execução, a PSA gera um objecto *JSON* com as estatísticas da execução em termos de quantidade de passos e colisões por episódio e envia-o para a aplicação. A PSA tem uma definição própria de episódio, isolada do PAR, que considera como episódio a sequência de passos desde o início da execução até ser atingido o objectivo, sendo uma métrica mais consistente para avaliação do desempenho de um algoritmo. Com estas estatísticas, a aplicação gera gráficos para permitir a visualização das mesmas.

6.3 Cenários Propostos

Pretende-se nesta secção apresentar os ambientes desenvolvidos para os testes à biblioteca, assim como a parametrização utilizada nos vários algoritmos.

6.3.1 Ambientes

As figuras seguintes mostram os ambientes utilizados para testar os algoritmos desenvolvidos. Os ambientes designados por T_n são de objectivo único, e são os utilizados para testar todos os algoritmos excepto o *Q-Learning* com Simulação Episódica para múltiplos objectivos, sendo neste caso utilizados os ambientes M_n.

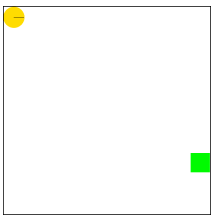

Figura 6.3: Ambiente de teste T_1

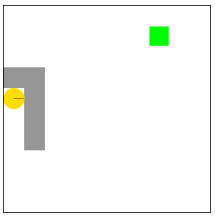

Figura 6.4: Ambiente de teste T_2

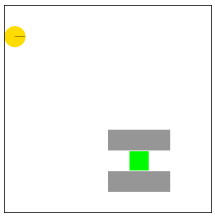

Figura 6.5: Ambiente de teste T_3

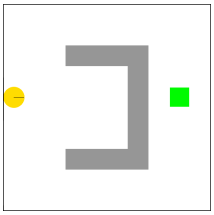

Figura 6.6: Ambiente de teste T_4

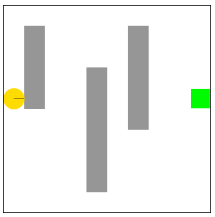

Figura 6.7: Ambiente de teste T_5

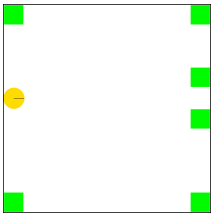

Figura 6.8: Ambiente de teste M_1

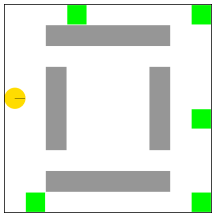

Figura 6.9: Ambiente de teste M_2

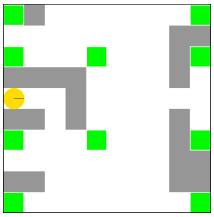

Figura 6.10: Ambiente de teste M_3

6.3.2 Parametrização

A lista seguinte descreve as parametrizações do problema definida para os testes aos algoritmos implementados:

Parâmetros Gerais

- 4 direcções de movimento possíveis
- Reforço por atingir objectivo $r_{obj} = 100$
- Reforço por atingir obstáculo $r_{obs} = -100$
- Custo de movimento $r_{mov} = -1$
- 50 iterações do problema (atingir o obstáculo 50 vezes)
- 10 execuções de cada conjunto de iterações do problema, sendo os resultados finais baseados na média dos resultados em cada execução

A parametrização específica de cada algoritmo pode ser consultada no Anexo C.

6.4 Resultados

A secção seguinte apresenta os resultados da execução dos algoritmos de aprendizagem por reforço em cada ambiente de teste, sob a forma de gráficos gerados pela aplicação *RLFrontend*.

6.4.1 *Q-Learning*

Como previsto, o algoritmo *Q-Learning* neste tipo de problemas converge rapidamente para uma solução muito próxima de óptima. Observa-se também que nunca existe convergência logo após a primeira chegada ao objectivo, como era expectável, pois são necessárias múltiplas iterações para o reforço positivo obtido no estado objectivo ser propagados para os estados antecedentes, através da estimativa de valor.

No ambiente T_1, a quantidade de passos por episódio diminui à medida que é acumulada informação. com pequenas variações, resultantes do carácter aleatório da exploração, que igualmente diminuem até o algoritmo convergir.

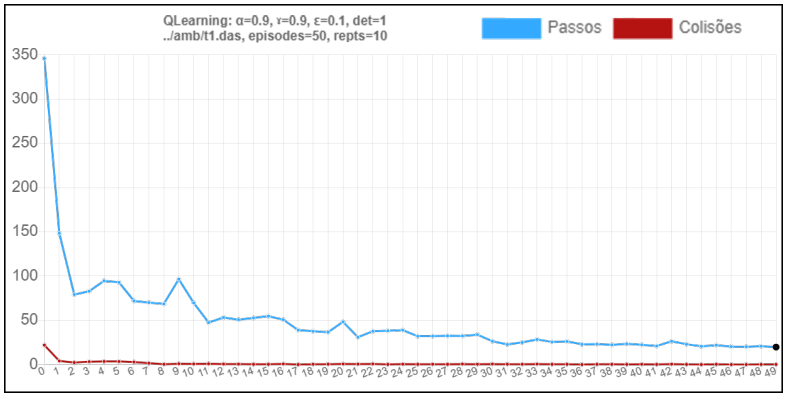

Figura 6.11: Resultado execução *Q-Learning* Ambiente de teste T_1

No ambiente T_2, após alguns episódios, o número de passos aumenta, contrariando a tendência a decrescer como observado noutros testes, voltando depois a decrescer. Este pico tem a mesma origem que as pequenas variações observadas neste e noutros testes, só que neste caso de uma forma muito mais acentuada, provavelmente dada a configuração específica deste ambiente.

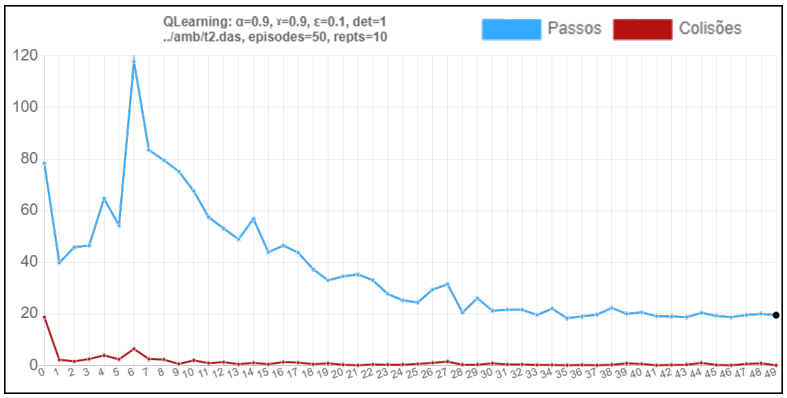

Figura 6.12: Resultado execução *Q-Learning* Ambiente de teste T_2

No ambiente T_3 a evolução do algoritmo foi semelhante à do T_1, reduzindo o tamanho episódios ao longo da execução.

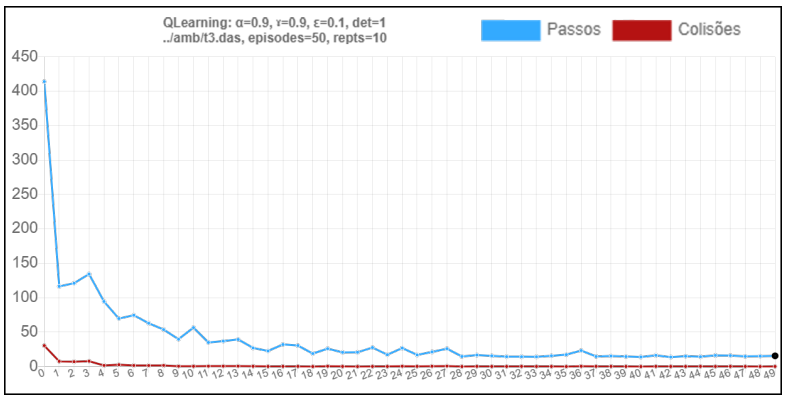

Figura 6.13: Resultado execução *Q-Learning* Ambiente de teste T_3

Nos ambientes mais complexos (T_4, T_5), é também possível observar de uma forma mais óbvia as elevações repentinas na quantidade de passos até ao objectivo, mesmo depois de claramente ter percorrido caminhos mas curtos e com menos colisões em episódios anteriores. Isto deve-se ao factor de exploração ε estar fixo num valor superior a 0, o que faz com que em algumas ocasiões este levar o agente a estados que ainda não possuam estimativa de valor, o que implica ter de explorar partes desconhecidas do ambiente.

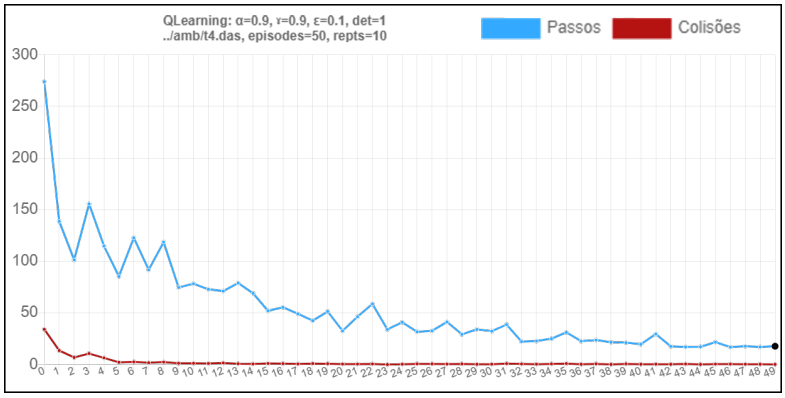

Figura 6.14: Resultado execução *Q-Learning* Ambiente de teste T_4

Isto não é problemático para o algoritmo, e até pode ser considerado vantajoso dado que a informação adicional leva a uma função de valor mais completa.

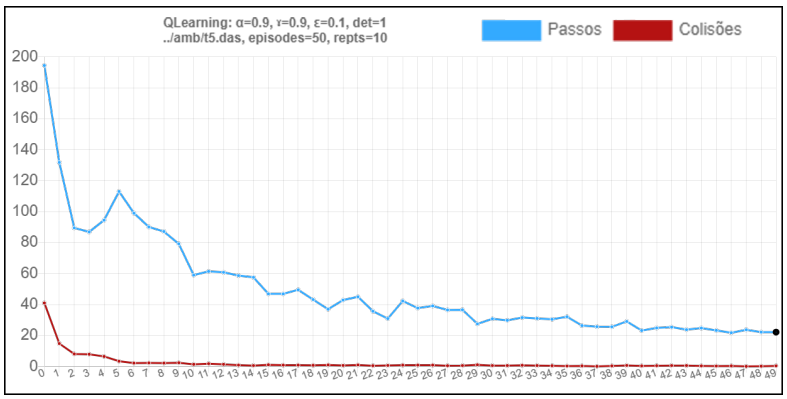

Figura 6.15: Resultado execução *Q-Learning* Ambiente de teste T_5

6.4.2 *DynaQ*

Também como esperado, o algoritmo *Dyna-Q* converge para uma solução muito próxima de óptima, com a vantagem de conseguir chegar a essa solução mais rapidamente devido às simulações efectuadas utilizando o modelo do mundo. Observou-se também uma minimização das elevações na quantidade de passos como ocorreu no *Q-Learning*, pois mesmo quando atingido um estado desconhecido, a estimativa do valor deste é rapidamente actualizada pelas simulações de execução.

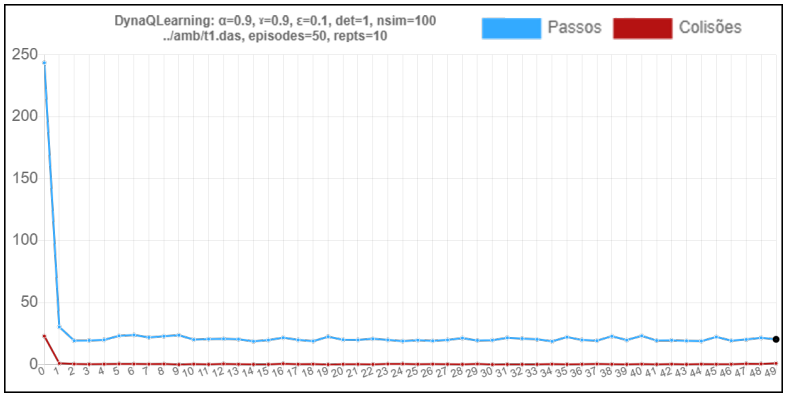

Figura 6.16: Resultado execução *Dyna-Q* Ambiente de teste T_1

No ambiente T_2, observou-se que as variações no valor de passos, apesar de minimizadas, não deixam de ocorrer, sendo mais notório neste ambiente, dado que o agente inicia a sua execução sempre adjacente a vários obstáculos

excepto numa direcção, e que o factor de exploração foi definido como um valor fixo, pelo que mesmo após o algoritmo convergir para uma solução, o valor de ε será sempre um valor de probabilidade de atingir um obstáculo nos passos iniciais de cada episódio

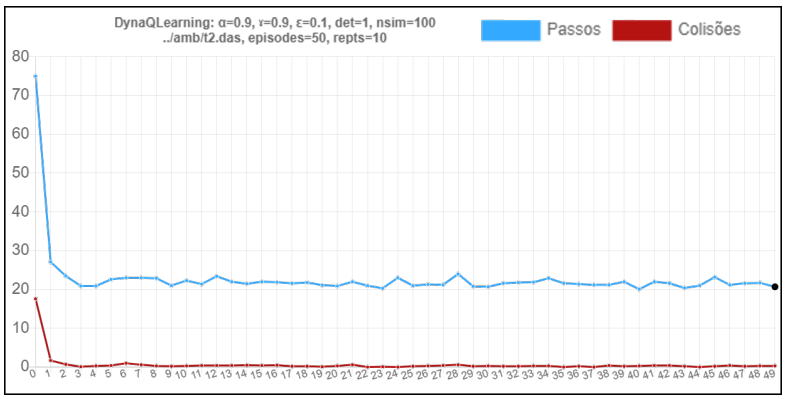

Figura 6.17: Resultado execução *Dyna-Q* Ambiente de teste T_2

Nos restantes ambientes constatou-se a referida rapidez em convergir para uma solução. Logo após o primeiro episódio, o agente possui imediatamente uma solução que, mesmo que sub-óptima, pode ser reforçada e optimizada pelas simulações internas, a par com a exploração.

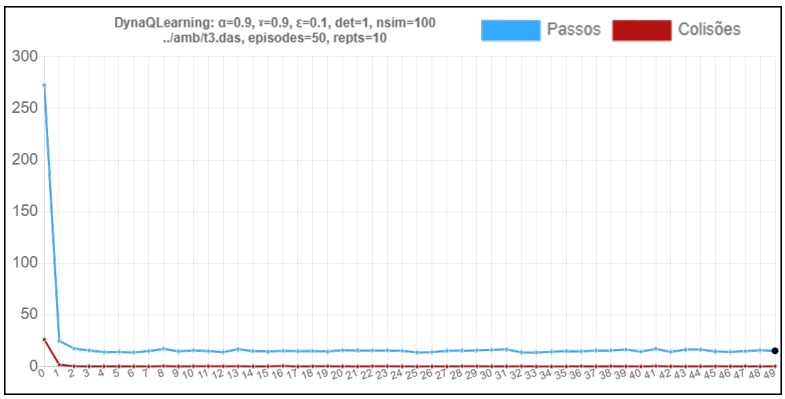

Figura 6.18: Resultado execução *Dyna-Q* Ambiente de teste T_3

Como foi utilizado um número elevado de simulações (100) relativamente ao número de estados total, a partir do segundo episódio a solução encontrada já é praticamente a óptima.

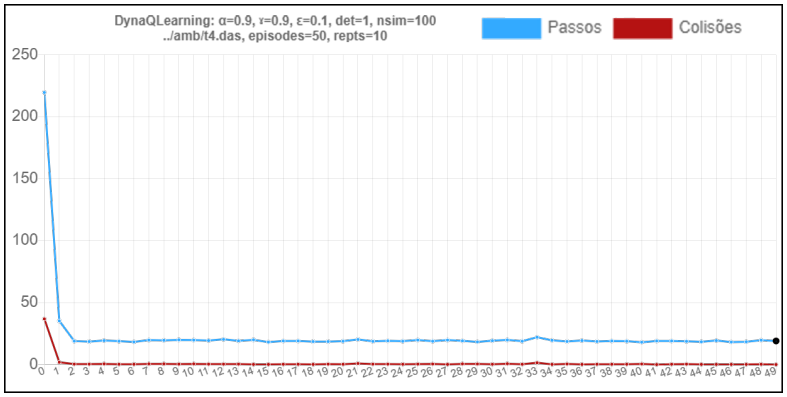

Figura 6.19: Resultado execução *Dyna-Q* Ambiente de teste T_4

Observou-se também que a quantidade de colisões mantém-se praticamente nula após o primeiro episódio, também devido às simulações internas do agente, que fazem propagar o valor negativo destas colisões para os estados adjacentes, contribuindo para a redução das colisões.

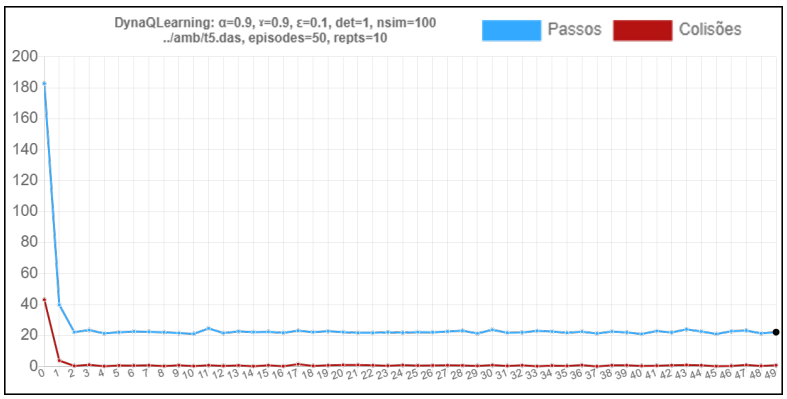

Figura 6.20: Resultado execução *Dyna-Q* Ambiente de teste T_5

6.4.3 Aprendizagem com Memória Episódica

Não foi possível completar os testes em todos os ambientes, sendo a razão principal a estimativa de valor para cada par (estado, acção) ser efectuada em tempo de execução, a cada passo do agente. À medida que são acumulados episódios em memória, o tempo necessário para completar os cálculos aumenta, o que por sua vez torna a execução mais lenta, impedindo o agente de atingir o objectivo em tempo útil.

Modo 1 - Valorização de episódios relevantes

Neste modo, independentemente de quaisquer limitações em termos de memória, era esperada a dificuldade em lidar com a dimensão dos ambientes. Tal como referido no capítulo 3, A dimensão máxima de cada episódio foi um parâmetro difícil de definir, dado o risco de atenuação da estimativa de valor se a dimensão definida for muito elevada e, no caso oposto, o risco de perda de informação acerca de recompensas no futuro, devido a uma segmentação elevada de episódios.

Os testes que foram bem sucedidos permitem verificar que, apesar das limitações, é possível chegar a uma solução apenas com esta forma de cálculo de valor, no entanto com uma quantidade de passos muito superior em comparação ao *Q-Learning*.

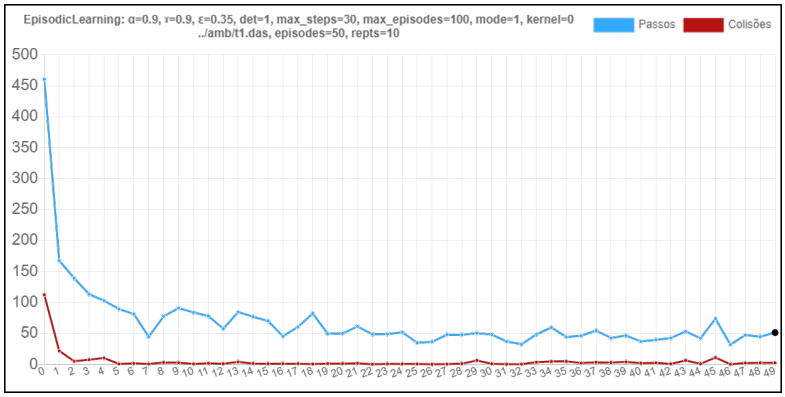

Figura 6.21: Resultado execução *Aprendizagem com Memória Episódica modo 1* Ambiente de teste T_1

No ambiente T_2, começaram a ser mais notórias as dificuldades deste algoritmo. Como esperado, a segmentação de episódios impede uma propagação eficiente dos valores de reforço, sendo necessária uma quantidade maior de exploração.

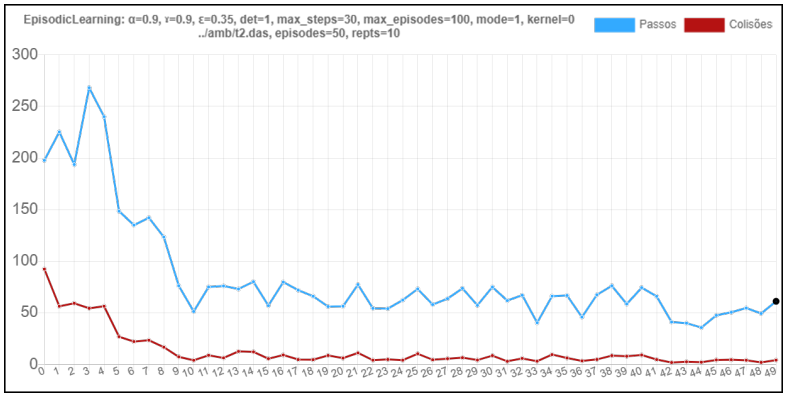

Figura 6.22: Resultado execução *Aprendizagem com Memória Episódica modo 1* Ambiente de teste T_2

No ambiente T_3 observou-se o mesmo problema de uma forma mais acentuada. Mesmo ao transitar para um estado conhecido, se os episódios relativos a este contidos na memória do agente não possuírem o estado objectivo ou um estado com valorização positiva, a informação destes não contribui directamente para atingir o objectivo.

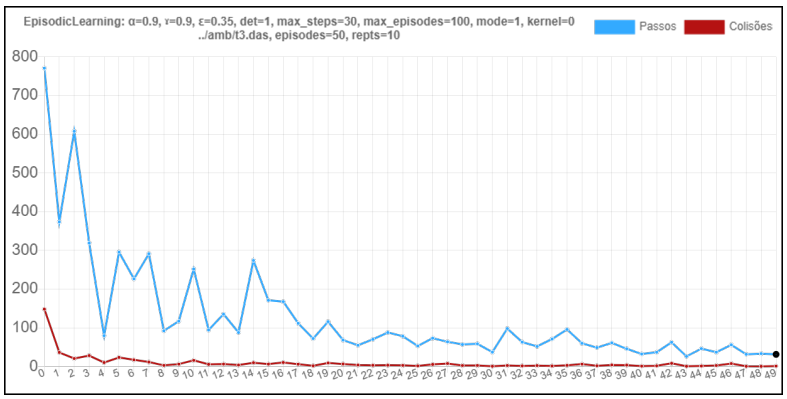

Figura 6.23: Resultado execução *Aprendizagem com Memória Episódica modo 1* Ambiente de teste T_3

Nos ambientes T_4 e T_5, foi possível observar que a quantidade média de passos por episódio de execução é muito superior que nos ambientes anteriores, não existindo um momento óbvio de convergência do algoritmo numa solução, como foi observado nos resultados dos testes anteriores.

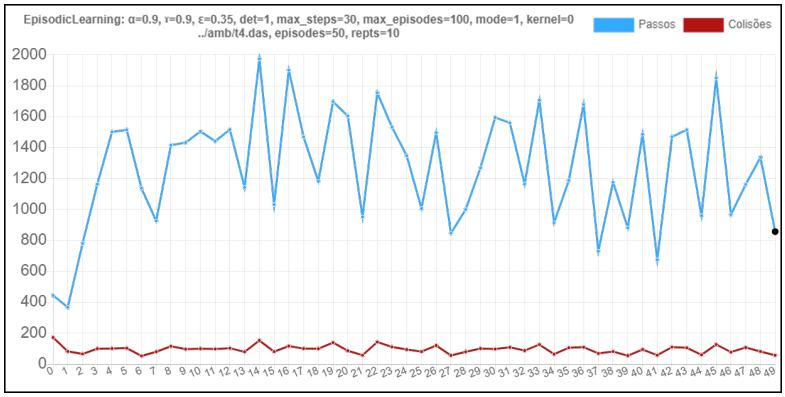

Figura 6.24: Resultado execução *Aprendizagem com Memória Episódica modo 1* Ambiente de teste T_4

Em ambos os ambientes, considerou-se que não foi atingida uma solução. É importante referir que apesar de o tempo de execução não ter sido tido em conta para aferir o desempenho do algoritmo, os resultados de execução apresentados demoraram algumas horas a serem obtidos, ao contrário dos ambientes anteriores em que a execução ocorreu em poucos minutos, demonstrando a incapacidade deste modo, como previsto.

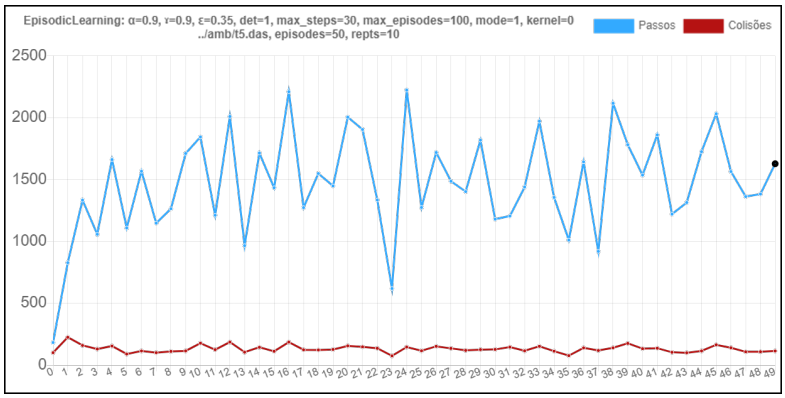

Figura 6.25: Resultado execução *Aprendizagem com Memória Episódica modo 1* Ambiente de teste T_5

Modo 2 - Valorização de episódios relevantes e episódios seguintes

Nos ambientes em que foi possível completar os testes, é visível a melhoria relativamente ao modo anterior, pois são tidos em conta os retornos de episódios além dos em que se encontra o estado actual do agente. Neste

modo, a limitação que causou a falha nos restantes testes foi a referida no inicio desta secção: a iteração dos episódios atinge um tempo de execução tal que não permite chegar a uma solução em tempo útil, agravada do facto de terem de ser iterados não só os episódios relativos ao estado actual, como também os episódios seguintes.

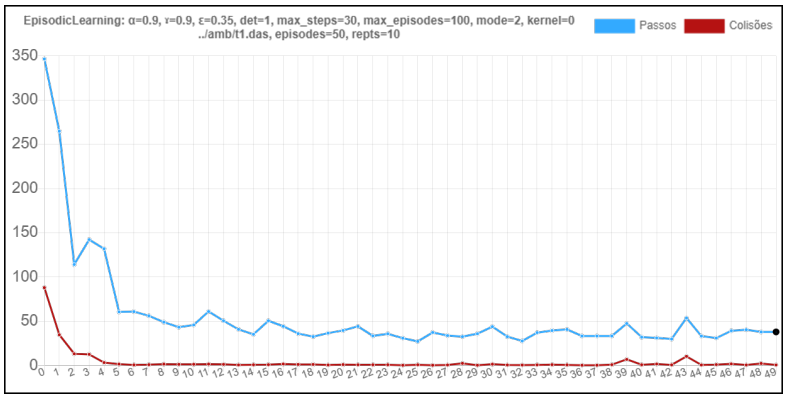

Figura 6.26: Resultado execução *Aprendizagem com Memória Episódica modo 2* Ambiente de teste T_1

A inclusão dos episódios seguintes, apesar de aumentar a complexidade computacional, permitiu atingir soluções aceitáveis com uma quantidade menor de exploração relativamente ao modo anterior.

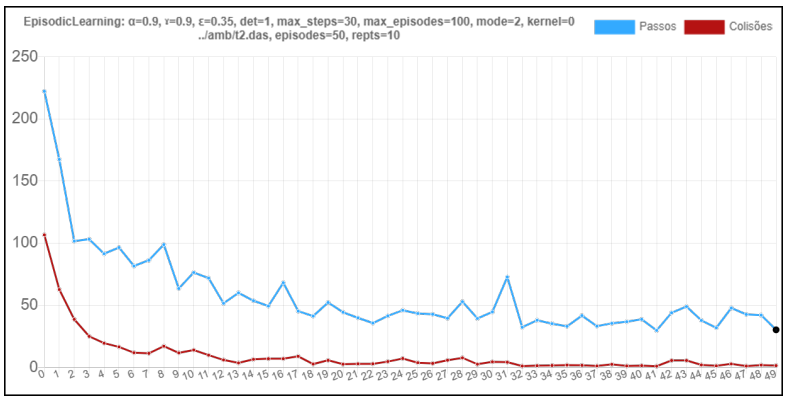

Figura 6.27: Resultado execução *Aprendizagem com Memória Episódica modo 2* Ambiente de teste T_2

Essas melhorias foram também observadas no ambiente T_3, apesar de algumas dificuldades nos episódios iniciais.

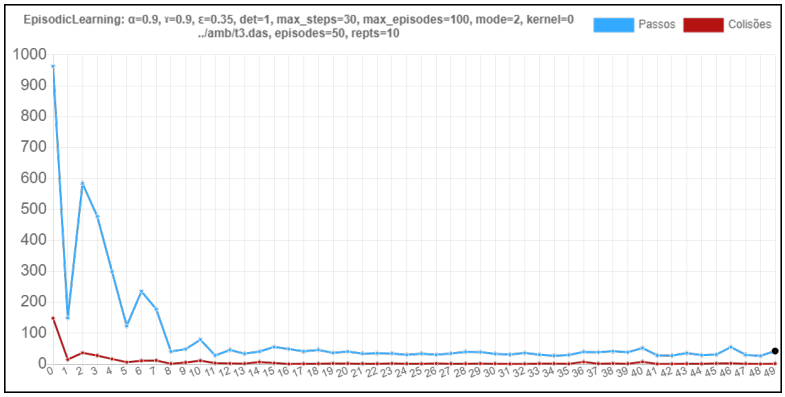

Figura 6.28: Resultado execução *Aprendizagem com Memória Episódica modo 2* Ambiente de teste T_3

Neste modo também não foi possível atingir uma solução nos ambientes T_4 e T_5, indicando a possibilidade de se estarem a formar óptimos locais com a informação acumulada.

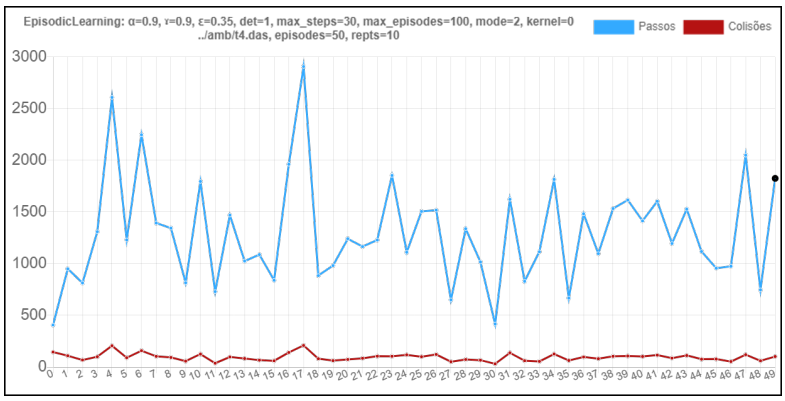

Figura 6.29: Resultado execução *Aprendizagem com Memória Episódica modo 2* Ambiente de teste T_4

Tal como no modo anterior, o tempo necessário para concluir ambos os testes foi muito elevado, agravado pelo cálculo adicional neste modo. Se não tivesse sido colocado um limite máximo de episódios em memória, provavelmente não teria sido possível concluir estes testes, dada a elevada quantidade de dados a processar a cada passo de execução.

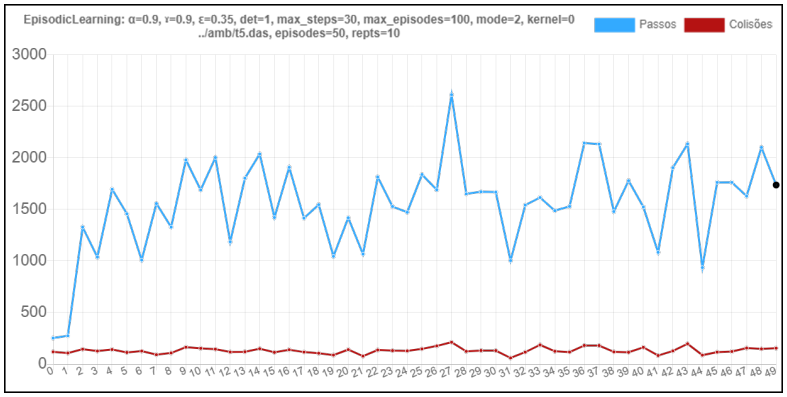

Figura 6.30: Resultado execução *Aprendizagem com Memória Episódica modo 2* Ambiente de teste T_5

Modo 3 - Generalização

Este modo partilha as limitações observadas nos anteriores, com a complexidade adicional de ser necessário efectuar cálculos na lista completa de episódios quando é atingido um estado novo de modo a poder efectuar a generalização de estado. Ademais, é provável que existam erros na valorização dos estados causados pela função de generalização utilizada não ser inteiramente adequada ao problema. Considerou-se que este tema estaria fora do âmbito da dissertação, pelo que não foram efectuados ajustes a esse nível.

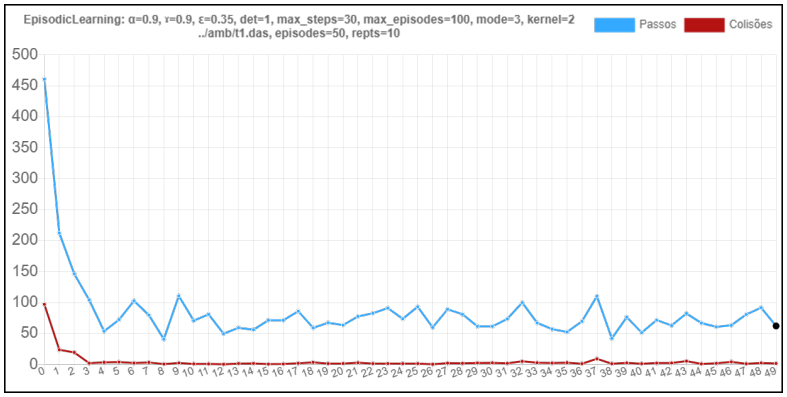

Figura 6.31: Resultado execução *Aprendizagem com Memória Episódica modo 3* Ambiente de teste T_1

Apesar do algoritmo convergir para uma solução nos primeiros 3 ambientes, neste modo a quantidade de passos foi superior ao anterior.

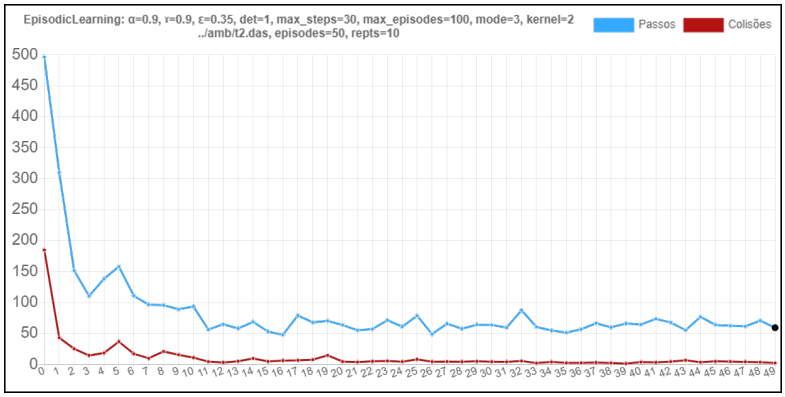

Figura 6.32: Resultado execução *Aprendizagem com Memória Episódica modo 3* Ambiente de teste T_2

Este modo teve a dificuldade adicional de ter de processar todos os episódios a cada passo de execução, em vez de apenas os relacionados, de modo a fazer uso da função de generalização, tornando o tempo de processamento para cada passo mais elevado que os restantes algoritmos.

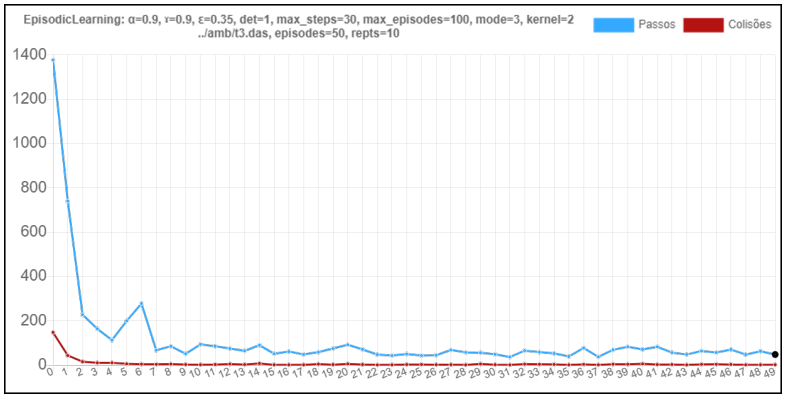

Figura 6.33: Resultado execução *Aprendizagem com Memória Episódica modo 3* Ambiente de teste T_3

Tal como nos modos anteriores, não foi obtida uma solução nos ambientes T_4 e T_5, não existindo um momento de convergência do algoritmo como observado em testes anteriores, como demonstrado pelas figuras seguintes.

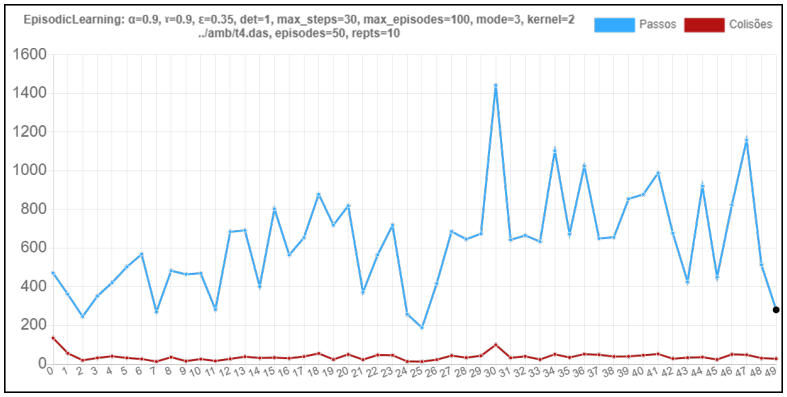

Figura 6.34: Resultado execução *Aprendizagem com Memória Episódica modo 3* Ambiente de teste T_4

Ao contrário do esperado, este modo não melhorou o desempenho do agente, provavelmente devido a uma função de generalização inadequada, como foi referido.

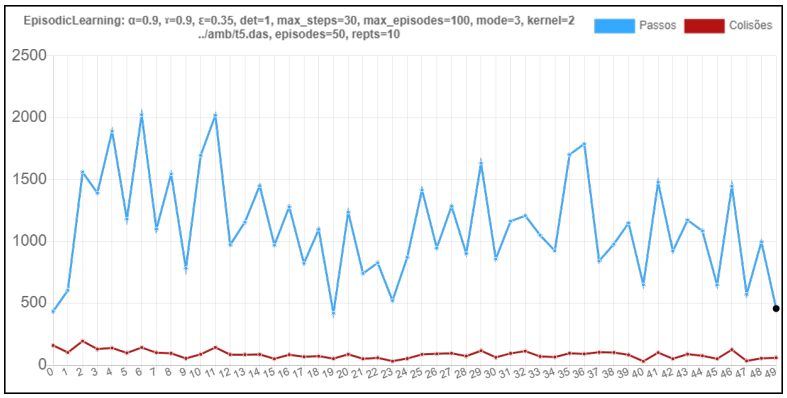

Figura 6.35: Resultado execução *Aprendizagem com Memória Episódica modo 3* Ambiente de teste T_5

6.4.4 *Q-Learning* com Simulação Episódica

Objectivo Único

Tanto nos ambientes de objectivo único como de múltiplos objectivos, os testes foram bem sucedidos, convergindo para uma solução rapidamente, à semelhança do *Dyna-Q*.

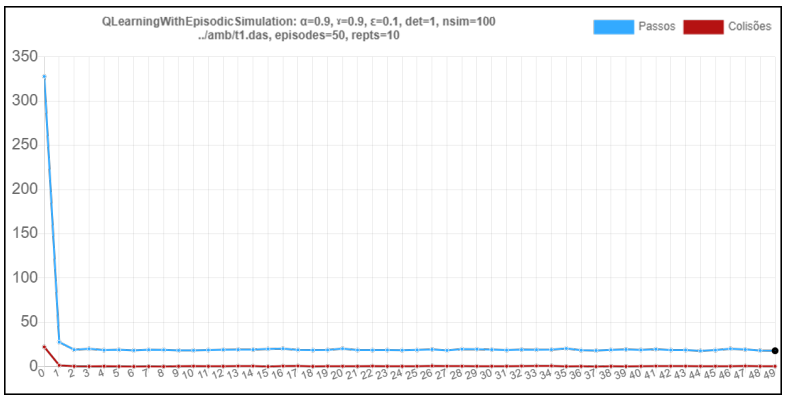

Figura 6.36: Resultado execução *Q-Learning com Simulação Episódica* Ambiente de teste T_1

No ambiente T_2, tal como em todos os restantes algoritmos, observaram-se variações no número de passos até ao objectivo devido à estrutura específica deste ambiente.

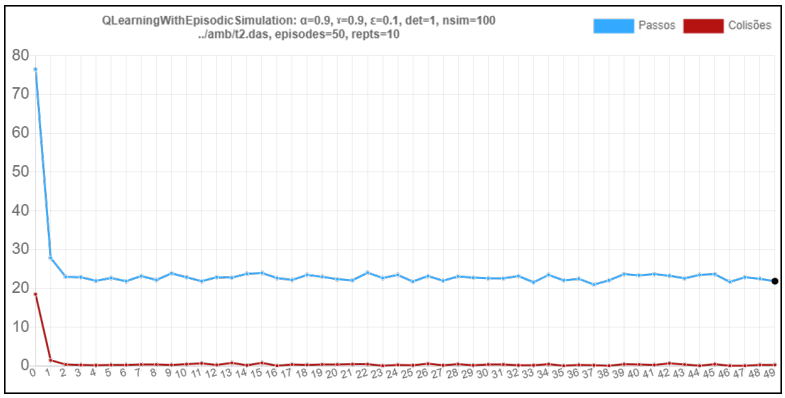

Figura 6.37: Resultado execução*Q-Learning com Simulação Episódica* Ambiente de teste T_2

A única diferenciação possível de ser feita em termos dos resultados deste algoritmo e dos do algoritmo *Dyna-Q* são os episódios iniciais serem por vezes ligeiramente mais longos, sendo no entanto esta diferença desprezível, dado que é encontrada uma solução com o mesmo grau de qualidade e eficiência em ambos os algoritmos.

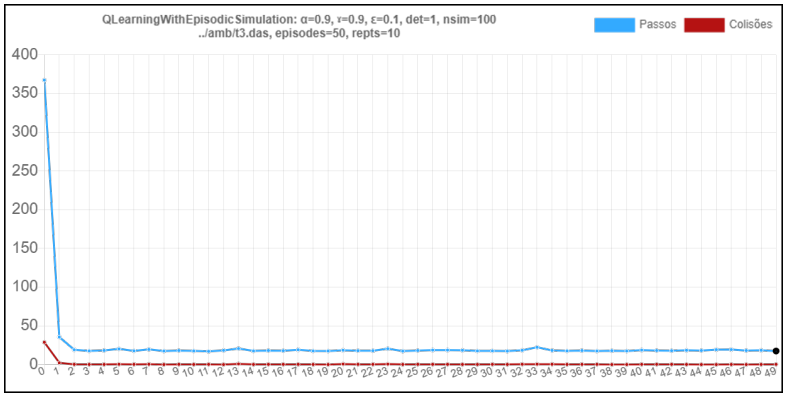

Figura 6.38: Resultado execução *Q-Learning com Simulação Episódica* Ambiente de teste T_3

À medida que são acumulados episódios, estes são cada vez mais redundantes ao longo do tempo, dado que o agente vai seguindo o caminho que maximize o valor. Dado que as simulações são efectuadas a partir de passos aleatórios de episódios aleatórios, ao longo do tempo existe um aumento da probabilidade dos pares (estado, acção) com maior valor serem escolhidos para simulação, dado que existem mais episódios que os contêm.

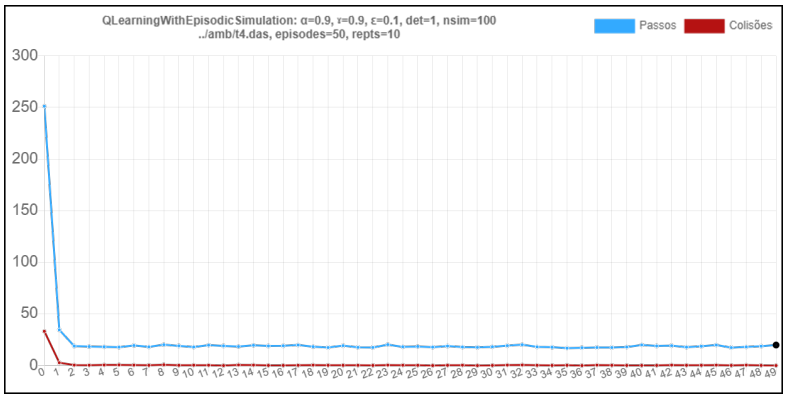

Figura 6.39: Resultado execução *Q-Learning com Simulação Episódica* Ambiente de teste T_4

Nos ambientes T_3 e T_4 o comportamento foi também idêntico ao algoritmo *Dyna-Q*.

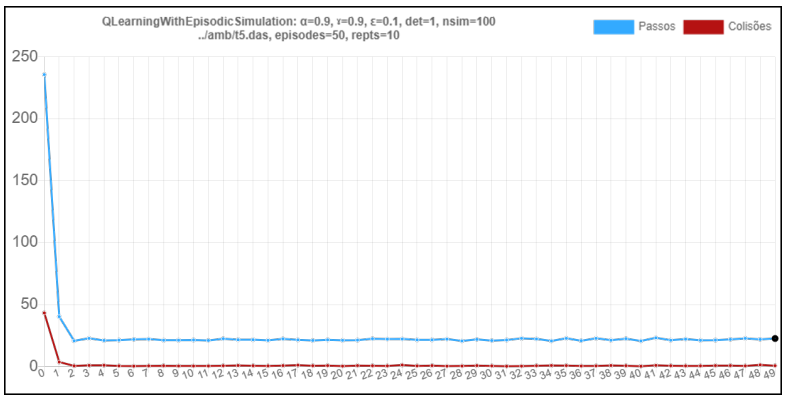

Figura 6.40: Resultado execução *Q-Learning com Simulação Episódica* Ambiente de teste T_5

Múltiplos objectivos

Para fins de estatística e para este algoritmo em especifico, foi considerado como um episódio a sequência de passos de execução até todos os objectivos do ambiente serem atingidos. Observou-se que, apesar do algoritmo convergir rapidamente, existem sempre colisões, ao contrário de algoritmos como o *Q-Learning* ou o *Dyna-Q*, que tendem a reduzir a quantidade de colisões à medida que acumulam mais informação acerca do problema. Estas colisões provavelmente ocorrem no curto espaço de tempo em que a função Q é reiniciada (a execução nunca é parada), e ainda não existe de facto informação de valor de estado para que o agente possa ter uma política comportamental que evite as colisões, até serem efectuadas novas simulações de execução.

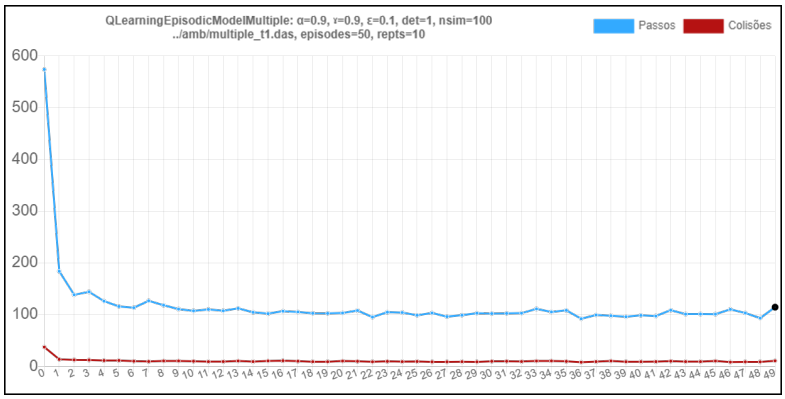

Figura 6.41: Resultado execução *Q-Learning com Simulação Episódica* Ambiente de teste M_1

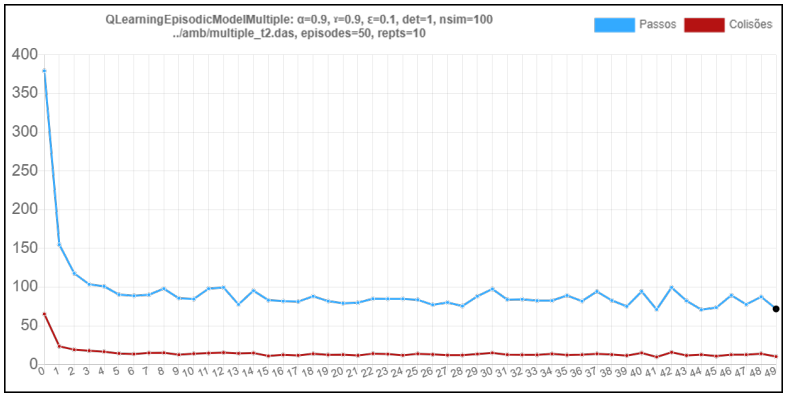

Figura 6.42: Resultado execução *Q-Learning com Simulação Episódica* Ambiente de teste M_2

Observou-se um comportamento semelhante à versão deste algoritmo para um único objectivo e ao *Dyna-Q*, como esperado. Não é possível, no entanto, comparar este algoritmo com os restantes, em termos de resultados, dado que opera em ambientes diferentes com um objectivo diferente.

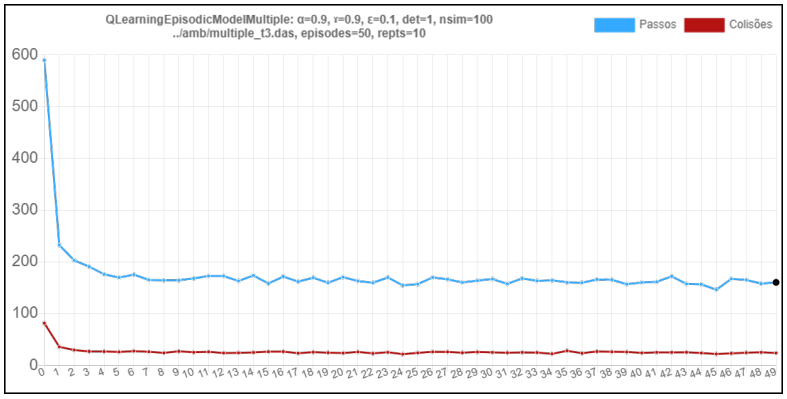

Figura 6.43: Resultado execução *Q-Learning com Simulação Episódica* Ambiente de teste M_3

Contudo, é possível observar as semelhanças em termos da forma dos gráficos: no episódio inicial é dada a maior quantidade de passos, decrescendo drasticamente nos episódios seguintes, até convergir numa solução.

6.4.5 *Q-Learning* com Propagação Episódica

O método de valorização de estado deste algoritmo em particular possibilita que seja encontrada uma solução imediatamente após a primeira vez que é atingido o estado objectivo, dado que a iteração do episódio no sentido inverso ao tempo durante a actualização de valor resulta numa propagação do valor positivo com uma maior intensidade nos estados anteriores.

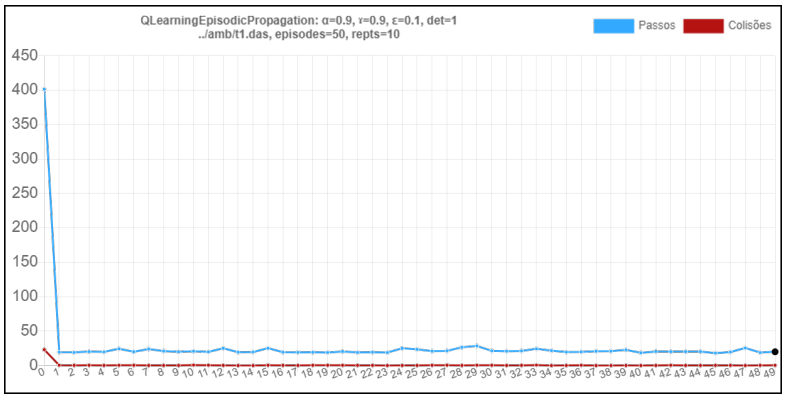

Figura 6.44: Resultado execução *Q-Learning com Propagação Episódica* Ambiente de teste T_1

Novamente, dada a configuração do ambiente T_2, observam-se algumas variações no número de passos mesmo após o algoritmo convergir, devido a colisões causadas pelo factor de exploração.

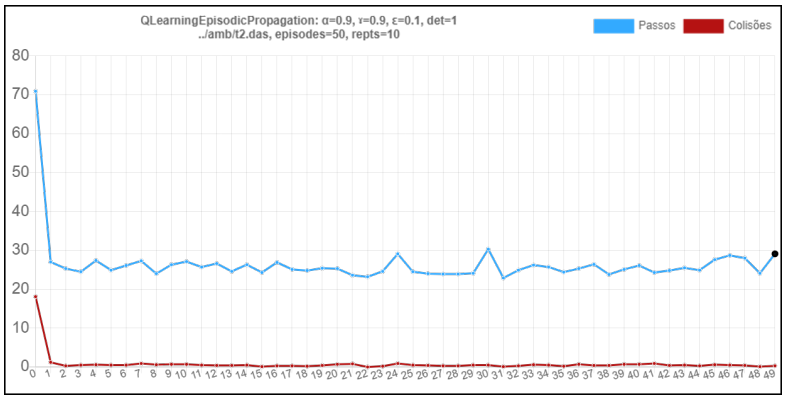

Figura 6.45: Resultado execução *Q-Learning com Propagação Episódica* Ambiente de teste T_2

Numa primeira análise, os resultados deste algoritmo parecem semelhantes aos do algoritmo *Dyna-Q*, contudo estes diferem, como é possível observar pelos gráficos, no facto do algoritmo convergir para uma solução imediatamente após o primeiro episódio, em vez de decrescer ao longo de vários episódios.

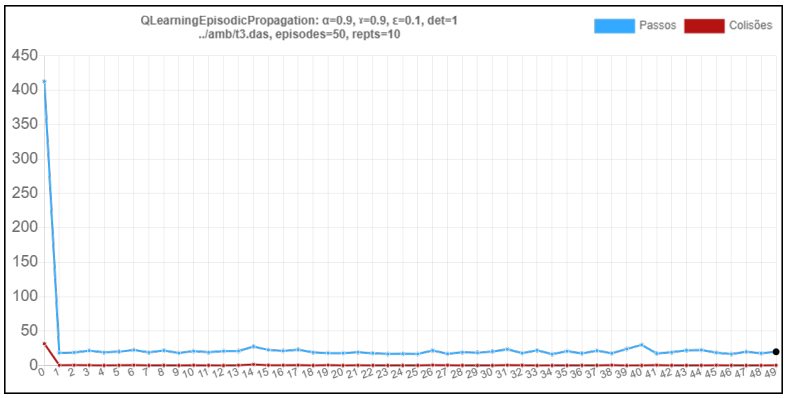

Figura 6.46: Resultado execução *Q-Learning com Propagação Episódica* Ambiente de teste T_3

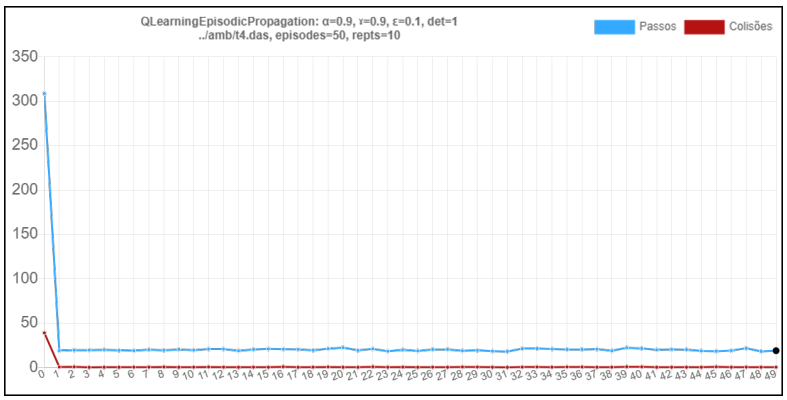

Figura 6.47: Resultado execução *Q-Learning com Propagação Episódica* Ambiente de teste T_4

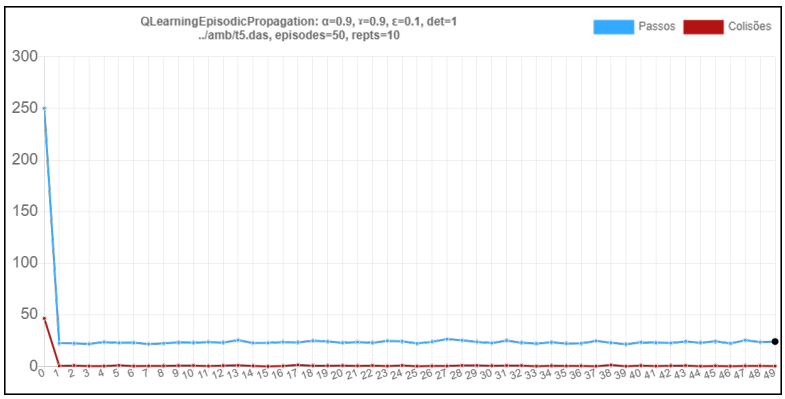

Figura 6.48: Resultado execução *Q-Learning com Propagação Episódica* Ambiente de teste T_5

6.5 Síntese de Resultados

Pretende-se nesta secção resumir os resultados obtidos pelos algoritmos em cada ambiente de teste de objectivo único e apresentar as considerações finais acerca dos testes efectuados.

6.5.1 Ambiente T_1

Em todos os algoritmos, o primeiro episódio foi onde foi efectuada a maioria da exploração, convergindo para uma solução após alguns episódios. No algoritmo *Aprendizagem com Memória Episódica*, em todos os modos, apesar de ser observado um decréscimo na quantidade de passos até valores semelhantes aos algoritmos com resultados melhores, foram sempre observadas variações mais elevadas ao longo do tempo, sem nenhum momento de convergência para um valor. Contudo, considerou-se que todos os algoritmos atingiram soluções satisfatórias.

6.5.2 Ambiente T_2

De igual forma que o ambiente anterior, todos os algoritmos atingiram uma solução, sendo que o algoritmo *Aprendizagem com Memória Episódica* foi o que apresentou mais instabilidades e falta de convergência numa solução específica. Dada a configuração específica deste ambiente (estado inicial apenas com uma acção que gera uma transição para outro estado), e o factor

de exploração ser um valor fixo ao longo de toda a execução, verificou-se sempre uma variação na quantidade de passos, mesmo após os algoritmos convergirem para uma solução, dadas as colisões inevitáveis no estado inicial geradas pelo factor de exploração.

6.5.3 Ambiente T_3

Todos os algoritmos atingiram uma solução, sendo que neste caso o algoritmo *Aprendizagem com Memória Episódica*, apesar dos valores de quantidade de passos até ao objectivo com uma variação elevada demonstrada nos ambientes anteriores, apenas neste ambiente apresentou o comportamento que era esperado pelos pressupostos teóricos apresentados, ao convergir mais rapidamente para uma solução entre cada modo, sendo o modo 1 o mais lento e o modo 3 o mais rápido a convergir para uma solução aceitável em termos de quantidade de episódios.

6.5.4 Ambiente T_4

Neste ambiente, o algoritmo *Aprendizagem com Memória Episódica* não atingiu uma solução para o problema em nenhum dos modos de execução. Os restantes algoritmos atingiram uma solução próxima de óptima após alguns episódios, tendo o algoritmo *Q-Learning* com Propagação Episódica novamente sido o mais rápido a convergir para uma solução.

6.5.5 Ambiente T_5

Igualmente ao ambiente anterior, o algoritmo *Aprendizagem com Memória Episódica* não atingiu uma solução para o problema em nenhum dos modos de execução. Os restantes algoritmos atingiram uma solução próxima de óptima após alguns episódios, tendo o algoritmo *Q-Learning* com Propagação Episódica novamente sido o mais rápido a convergir para uma solução.

6.5.6 Considerações Finais

O algoritmo mais eficiente nos testes efectuados foi o *Q-Learning* com Propagação Episódica, tendo convergido sempre para uma solução após o

primeiro episódio, devido à iteração inversa do episódio após atingir o objectivo e actualização dos valores de estado.

Quanto aos ambientes com múltiplos objectivos, dado que apenas foi utilizado um algoritmo para os mesmos, não existe uma análise comparativa a ser feita. Deve-se no entanto referir que o algoritmo conseguiu gerar soluções para todos os ambientes, tendo sido observadas algumas colisões derivadas da reinicialização da função Q em tempo de execução.

Ambos estes factos levam à conclusão de que uma estrutura de memória episódica traz benefícios para um sistema de aprendizagem por reforço, como uma estrutura de informação adicional para melhorar o desempenho do sistema.

6.6 Sumário

Neste capítulo foi apresentada a Plataforma de Simulação de Agentes (PSA), responsável por proporcionar uma concretização de um agente e de um ambiente onde este é capaz de operar, através de um mecanismo de controlo definido pelo utilizador. Este foi o sistema utilizado para colocar em funcionamento a Plataforma de Aprendizagem por Reforço (PAR), exposta no capítulo anterior, sendo esta responsável por gerar acções para passar ao agente na PSA através do mecanismo de controlo. Foram também apresentadas as aplicações desenvolvidas para proporcionar uma interface gráfica para manipulação dos parâmetros da PAR e posterior visualização dos resultados de execução da PSA relativos à respectiva parametrização. Por último foram apresentados os cenários de teste propostos e os respectivos resultados, tendo sido realizada a sua análise.

Capítulo 7

Conclusões e Trabalho Futuro

Foi proposto no início deste trabalho um estudo acerca do uso de estruturas de memória episódica em sistemas de aprendizagem por reforço. Para tal, foram estudados os fundamentos da aprendizagem por reforço e alguns dos seus algoritmos [6], assim como propostas teóricas de sistemas de aprendizagem por reforço com memória episódica [3],[13], com o proposto de desenvolver de uma biblioteca de *software* que implementasse tanto os algoritmos clássicos de aprendizagem por reforço, como algoritmos baseados nas propostas dos autores para sistemas que fizessem uso de memória episódica, de modo a poder fazer uma análise comparativa entre os algoritmos.

O facto de ter sido providenciada uma aplicação responsável pela criação e execução de um agente num ambiente pré-configurável (PSA) foi um muito importante para o sucesso desta dissertação. Sendo o foco principal deste área de estudo os algoritmos e como estes podem ser melhorados, possuir uma ferramenta que permita visualizar não só a execução do agente em tempo real como as funções de valor foi crucial para manter esse mesmo foco.

A implementação dos algoritmos clássicos de aprendizagem por reforço foi bem sucedida e permitiu obter uma boa base de comparação para os restantes algoritmos implementados, como demonstrado pelos testes efectuados. Ambos atingiram soluções válidas em poucos episódios, sendo que o *Dyna-Q* consegue convergir mais rapidamente dadas as simulações que são efectuadas.

A utilização de apenas memória episódica, na implementação que foi desenvolvida, apresentou problemas em termos do processamento dos episódios para obter os valores estimados das acções em cada estado, após a acumulação de uma grande quantidade de episódios. Não descurando a possibilidade de

minimização deste problema através da optimização desta implementação destes algoritmos, a realidade é que independentemente da forma como forem processados os episódios, se a quantidade de episódios a processar não for de alguma forma limitada, eventualmente irão existir demasiados para que os cálculos sejam efectuados em tempo útil. Esta limitação também não pode ser tão simples como definir um valor máximo de episódios a processar, ignorando ou eliminado os mais antigos, pois nesse caso poderão ocorrer problemas, como por exemplo, nunca chegar atingir o estado objectivo devido ao "esquecimento"de obstáculos ultrapassados anteriormente, ou após atingir o objectivo tantas vezes quanto o valor de quantidade máxima de episódios começar a perder informação acerca de estados com reforços negativos, o que pode parecer vantajoso à primeira vista, mas na realidade resulta numa política comportamental sub-óptima, na medida em que se se perdeu a informação de que acções a evitar.

Após visualização da execução do algoritmo de Aprendizagem com Memória Episódica no modo de generalização de estado, utilizando o visualizador da PSA para observar em tempo de real os resultados das estimativas de valor para cada acção, foi possível confirmar que a função de semelhança de estado estava a funcionar como esperado: ao atingir um estado desconhecido, os valores as várias acções foram estimados utilizando todos os episódios em memória, pesados por um factor de semelhança entre o estado actual do agente e o estado inicial de cada episódio. Contudo, dada a exploração inicial que é necessária para o agente aprender, e como tal, maior probabilidade de transição para estados que resultam em reforço negativo (custo de movimento, obstáculos), esses valores negativos são propagados para os estados vizinhos devido à generalização de estado, o que resulta num desempenho fraco. Desta forma, conclui-se que provavelmente a função de semelhança de estado utilizada não foi a melhor para este tipo de problema em específico, e que não se enquadra no âmbito da dissertação um estudo mais aprofundado acerca deste tema, pelo que não foram efectuados mais testes ou tentativas de optimização.

O algoritmo *Q-Learning* com Simulação Episódica apresentou um desempenho equiparável ao algoritmo *Dyna-Q*, o que era expectável dadas as suas semelhanças. Nos ambientes de objectivo único, este algoritmo poderia ter vantagem relativamente ao algoritmo *Dyna-Q* se estes fossem problemas es-

tocásticos, uma vez que neste último seria necessário alterar a forma como o modelo é construído e acrescentadas estruturas adicionais para calcular as probabilidades aproximadas de transição, ao passo que com simulação baseada em episódios, ao ser escolhido um passo aleatório, essa mesma probabilidade de transição já está implícita na quantidade de vezes que a transição ocorreu nos vários episódios. Quanto aos ambientes de múltiplos objectivos, apesar dos testes terem sido bem sucedidos, não foi observada qualquer vantagem em utilizar memória episódica para as simulações internas, dado que o mesmo mecanismo de ignorar estados objectivo já atingidos poderia ser implementado no algoritmo *Dyna-Q*.

Os resultados do algoritmo *Q-Learning* com Propagação Episódica foram os melhores de entre todos os algoritmos, dado que convergiu para uma solução após o primeiro episódio em todos os ambientes. Isto ocorreu dado o processo de propagação de valor dos episódios no sentido inverso ao atingir um objectivo. A formulação base do *Q-Learning* indica que o valor de um estado depende do valor do estado seguinte, e é por esta razão que a implementação base do *Q-Learning* precisa de atingir o objectivo várias vezes até que a função de valor leve a uma política comportamental que conduza o agente sempre ao objectivo. Se o valor de um estado depende do estado seguinte, a primeira actualização do estado não tem informação do estado seguinte, pelo que é necessária uma nova transição para esse estado para existir uma nova actualização, desta vez com informação do valor do estado seguinte. Existindo apenas um estado objectivo, o seu valor apenas será propagado para um estado adjacente, e sendo este processo repetido para cada estado adjacente do último com um valor relevante, sendo necessários alguns episódios de execução até convergir numa solução. O propósito do algoritmo *Dyna-Q* é colmatar esta necessidade de múltiplas iterações através da simulação de execução. Este algoritmo tira partido de uma estrutura de memória episódica para resolver o problema de uma forma mais eficiente, actualizando a função de valor utilizando a informação ordenada dos episódios, que neste caso terminam sempre apenas quando é atingido um estado objectivo. Ao iterar o episódio no sentido inverso, a parte da actualização da função Q que depende do valor do estado seguinte tem sempre a informação de valor mais actualizada possível.

Desta forma, conclui-se que a integração de uma estrutura de memória

episódica num sistema de aprendizagem por reforço, como um mecanismo de informação adicional, poderá ser benéfica para o desempenho do sistema, na medida em que providencia métodos adicionais de estimação de valor de estado e que, existindo pelo menos um episódio que termine num estado objectivo, uma solução sub-óptima mas de baixo custo computacional.

7.1 Trabalho Futuro

Nesta área de estudo, as possibilidades em termos de profundidade de pesquisa são vastas, podendo um pequeno pormenor ser, por si só, expandido numa dissertação à parte. Desta forma, tornou-se necessária uma limitação do âmbito do trabalho para garantir o foco no tema. No entanto, são descritas nas secções seguintes algumas possibilidades de continuação deste estudo.

7.1.1 Urgência de Valor

Como proposto por Morgado e Gaspar [13], suponha-se um sistema de aprendizagem com reforço com um parâmetro $\lambda \in [0, 1]$ representativo da urgência do agente em obter uma acção para executar, sendo que $\lambda = 0$ significa urgência mínima e não existe limitação em termos do tempo necessário para estimar qual a melhor acção a executar, e $\lambda = 1$ significa urgência máxima, e como tal o tempo de cálculo deve ser mínimo. Nesta situação, uma estrutura de memória episódica poderia ser vantajosa, pois em caso de urgência máxima poderia apenas ser escolhido um episódio aleatório e as respectivas acções passadas ao agente, ou então o valor de λ poderia controlar a quantidade de episódios a ter em conta para o cálculo.

7.1.2 Funções de Generalização

Como referido nas conclusões, possivelmente a função de generalização utilizada não foi a mais indicada para o problema em questão. Desta forma, poderia ser interessante um estudo acerca das várias funções de generalização possíveis e dos seus efeitos no algoritmo com memória episódica que foi implementado no âmbito desta dissertação.

7.1.3 Outros Ambientes

Todos os ambientes utilizados nos testes eram discretos e determinísticos, pois de outra forma teria sido adicionada uma camada de complexidade adicional ao problema que não fazia parte do propósito da dissertação. No entanto, no mundo real o ambiente é contínuo e imprevisível, o que implica que estes algoritmos, para poderem ser colocados eventualmente em funcionamento em problemas reais, devem ser capazes de lidar com os mesmos. Assim, outra possibilidade de aprofundar este trabalho seria a adaptação destes algoritmos para ambientes contínuos e/ou estocásticos, para avaliar se nesses casos se beneficiaria de uma estrutura de memória episódica.

Bibliografia

[1] **Watkins C.J.C.H.** Learning from Delayed Rewards. - Cambridge : Cambridge University,1989.

[2] **Rummery G.A. e Niranjan M.** Online Q-Learning using connectionist systems. - Cambridge : Tech. report CUED/F-INFENG/TR166, Cambridge University, 1994.

[3] **Gershman S.J. e Daw N.D.** Reinforcement Learning and Episodic Memory in Humans and Animals: An Integrative Framework. - Annual Review of Psychology, 2017

[4] **Tulving E.** Episodic and semantic memory. In Tulving, E.; Donaldson, W. (eds.). Organization of Memory. New York: Academic Press. pp. 381–402, 1972

[5] **Russell S., Norvig, P.** Artificial Intelligence: A Modern Approach. 3rd ed, Prentice Hall/Pearson Education, 2003.

[6] **Sutton R., Barto A.** Reinforcement Learning An Introduction, 2018

[7] **Silver D. et al** Mastering the game of Go without human knowledge. Nature, 550, 354, 2017

[8] **Wang, S. et al** Deep Reinforcement Learning for Autonomous Driving, 2018

[9] **Bu et all** A Reinforcement Learning Approach to Online Web Systems Auto-configuration, 2009

[10] **Kober J., Peters J.** (2014) Reinforcement Learning in Robotics: A Survey. In: Learning Motor Skills. Springer Tracts in Advanced Robotics, vol 97. Springer, Cham

[11] **Haarnoja, T., et al.** "Learning to Walk via Deep Reinforcement Learning." ArXiv:1812.11103 [Cs, Stat], Dec. 2018. arXiv.org, http://arxiv.org/abs/1812.11103.

[12] **Morgado L.** Plataforma de Simulação de Agentes v5.1, 2012

[13] **Morgado L., Gaspar G.** Adaptation and Decision-Making Driven by Emotional Memories. In: Bento C., Cardoso A., Dias G. (eds) Progress in Artificial Intelligence. EPIA 2005. Lecture Notes in Computer Science, vol 3808. Springer, Berlin, Heidelberg, 2005

[14] **Zhao, R., et al.** "Maximum Entropy-Regularized Multi-Goal Reinforcement Learning." ArXiv:1905.08786 [Cs, Stat], May 2019. arXiv.org, http://arxiv.org/abs/1905.08786.

[15] **Lin, Z., et al.** "Episodic Memory Deep Q-Networks." ArXiv:1805.07603 [Cs, Stat], May 2018. arXiv.org, http://arxiv.org/abs/1805.07603.

[16] **Github** (24 de Janeiro, 2019) The State of the Octoverse: machine learning. Consultado a 21 de Julho de 2019, em https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/

[17] **Sutton, R.** Dyna, an Integrated Architecture for Learning, Planning and Reacting. In Working Notes of the 1991 AAAI Spring Symposium, pp. 151-155, and SIGART Bul letin, 2, pp. 160-163, 1991

Apêndice A

Sintaxe de Construção de Ambientes na PSA

Um ambiente é definido através de um ficheiro de texto, utilizando diferentes caracteres para representar elementos do ambiente, nomeadamente:

- . — Espaço vazio
- @ — Posição inicial de um agente
- A — Alvo
- O — Obstáculo

As figuras seguintes apresentam um exemplo de um ficheiro e do ambiente resultante:

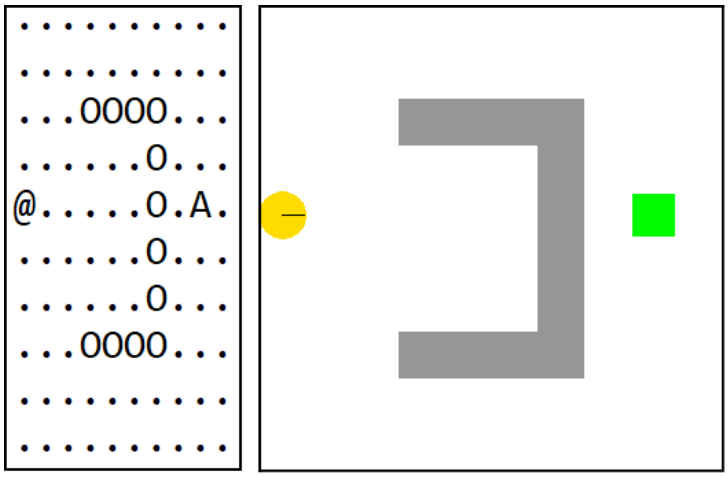

Figura A.1: Exemplo de ambiente e o ficheiro que o gerou

Apêndice B

Endpoints RLBackend

A tabela seguinte apresenta as funções disponibilizadas pela aplicação *RLBackend*:

URL	Método	Função
/methods	GET	Listar métodos disponíveis
/methods/[enginename]	GET	Listar argumentos de um método específico [enginename]
/launch	POST	Executar PSA
/logs	GET	Listar ficheiros log
/logs/[filename]	GET	Abrir ficheiro log com o nome [filename]

Tabela B.1

Apêndice C

Parametrização dos Algoritmos de Teste

As listagens seguintes apresentam a configuração efectuada relativamente a cada algoritmo utilizado nos testes efectuados:

Q-Learning

- Factor de aprendizagem $\alpha = 0.9$
- Factor de desconto temporal $\gamma = 0.9$
- Factor de exploração $\varepsilon = 0.1$
- Mecanismo de selecção de acção $\varepsilon - greedy$

Dyna-Q

- Factor de aprendizagem $\alpha = 0.9$
- Factor de desconto temporal $\gamma = 0.9$
- Factor de exploração $\varepsilon = 0.1$
- Quantidade de simulações $nsim = 100$
- Mecanismo de selecção de acção $\varepsilon - greedy$

Aprendizagem com Memória Episódica

- Factor de desconto temporal $\gamma = 0.9$
- Factor de exploração $\varepsilon = 0.35$
- Máximo de passos de execução por episódio $max\_steps = 30$
- Quantidade máxima de episódios em memória $max\_episodes = 100$
- Três modos de cálculo de valor de acções dado um estado, como expostas no capítulo 3:
  1. Valorização baseada no retorno acumulado e descontado de episódios onde se encontra o par $(estado, acção)$
  2. Modo anterior combinado com o retorno esperado de episódios seguintes
  3. Modo anterior combinado com uma função de generalização de estado
- Factor de generalização (Modo 3) $kernel = 2$
- Mecanismo de selecção de acção $\varepsilon - greedy$ adaptativo

***Q-Learning* com Simulação Episódica**

- Factor de aprendizagem $\alpha = 0.9$
- Factor de desconto temporal $\gamma = 0.9$
- Factor de exploração $\varepsilon = 0.1$
- Quantidade de simulações $nsim = 100$
- Mecanismo de selecção de acção $\varepsilon - greedy$

***Q-Learning* com Propagação Episódica**

- Factor de aprendizagem $\alpha = 0.9$
- Factor de desconto temporal $\gamma = 0.9$

- Factor de exploração $\varepsilon = 0.1$
- Mecanismo de selecção de acção ε-not-so-greedy