GCoM
Grupo de Controle e Modelagem
Universidade Federal de São João del-Rei - UFSJ
Departamento de Engenharia Elétrica

UFSJ
UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

CEFET-MG

Rubisson Duarte Lamperti

Aprendizado por Reforço no Domínio do Futebol de Robôs 2D

Dissertação apresentada à banca examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica, associação ampla entre a Universidade Federal de São João del-Rei e o Centro Federal de Educação Tecnológica de Minas Gerais, como parte dos requisitos necessários à obtenção do grau de Mestre em Engenharia Elétrica.

Orientador: Erivelton G. Nepomuceno

São João del-Rei
2013

Agradecimentos

A Universidade Federal de São João Del Rei e ao Programa de Pós-Graduação em Engenharia Elétrica pela oportunidade concedida para a realização do curso.

A Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES pela concessão da bolsa.

Ao professor Erivelton Geraldo Nepomuceno pela orientação, confiança, aprendizado e amizade.

Aos professores membros da banca Alexandre da Silva Simões e Valceres Vieira Rocha, pelas contribuições no trabalho.

Aos professores do Programa de Pós-Graduação em Engenharia Elétrica pelos ensinamentos transmitidos, apoio, estímulo e exemplo de profissionalismo.

Aos funcionários do Programa de Pós-Graduação em Engenharia Elétrica pela amizade e auxílio durante todo o curso.

A todos aqueles que, de alguma forma, contribuíram para a realização deste trabalho.

Epígrafe

Os caminhos que conduzem o homem ao saber são tão maravilhosos quanto o próprio saber.

Johannes Kepler

Sumário

Lista de Símbolos **19**

1 Introdução **1**

2 Revisão de Literatura **5**

2.1 Processos de Decisão de Markov - PDM 6

2.1.1 Época de Decisão 7

2.1.2 Conjunto de Estados e Ações 8

2.1.3 Recompensa e Probabilidade de Transição 9

2.1.4 Política para um PDM 10

2.2 Programação Dinâmica - PD 14

2.2.1 Avaliação de Políticas 15

2.2.2 Melhoria de Políticas 16

2.2.3 Iteração de Políticas 18

2.3 Método de Monte Carlo - MC 20

2.3.1 Avaliação de Política 20

2.3.2 Controle 20

2.4 Diferença Temporal - DT(0) 22

2.5 Q-Learning 23

2.6 Sarsa 25

2.7 Traço de Elegibilidade - λ 25

2.7.1 Controle Off-Policy: Q-Learning (λ) 27
2.7.2 Controle On-Policy: Sarsa (λ) 28
2.8 Generalização de Funções 30
2.8.1 Tile Coding 30
2.9 Robocup - Simulação 2D 32
2.9.1 Cliente 35
2.9.2 Monitor 35
2.9.3 Servidor 36
2.9.4 Treinador 36

3 Metodologia **37**
3.1 Cenário Proposto 37
3.2 O Modelo para o Aprendizado 40
3.2.1 Modelo Discreto 40
3.2.2 Modelo Contínuo 44
3.3 Experimentos 45
3.3.1 Modelo Discreto 46
3.3.2 Modelo Contínuo 46

4 Resultados **49**
4.1 Modelo Discreto 49
4.1.1 Experimento 01: Sarsa(λ) 49
4.1.2 Experimento 02: Q-Learning(λ) 52
4.1.3 Comparação entre Sarsa(λ) e Q-Learning(λ) 55
4.1.4 Discussão 56
4.2 Modelo Contínuo 58
4.2.1 Experimento 03: Sarsa($\lambda = 0{,}3$) 58
4.2.2 Experimento 04: Q-Learning($\lambda = 0{,}3$) 59

4.2.3 Experimento 05: Sarsa(λ) Tile 59
4.2.4 Experimento 06: Q-Learning(λ) 60
4.2.5 Comparação entre Sarsa Tile(λ) e Q-Learning Tile(λ) 61
4.2.6 Comparação entre Sarsa(λ) e Sarsa Tile(λ) 63
4.2.7 Comparação entre Q-Learning(λ) e Q-Learning Tile(λ) 64
4.2.8 Discussão 67

5 Conclusões **69**
5.1 Principais Contribuições 69
5.2 Trabalhos Futuros 71

Referências Bibliográficas **73**

A Simulações - Repetições **81**

Resumo

O Aprendizado por Reforço (AR) é uma técnica em que o agente aprende por meio da interação direta com o ambiente, convergindo para uma situação de equilíbrio. Uma das vantagens do aprendizado por reforço é a possibilidade do agente aprender em um ambiente desconhecido. Porém, a falta de conhecimento do ambiente, a complexidade do sistema e um grande espaço de estados contribuiem para o aumento do tempo necessário de aprendizado. Para minimizar este tempo e diminuir o espaço de estados, outras técnicas como, Traço de Elegibilidade, λ, e a Generalização de Função tem sido utilizadas com o aprendizado por reforço. Dentro do AR existem vários algoritmos, dos quais, destacam-se o Q-Learning e o Sarsa. O presente trabalho teve como objetivo avaliar o aprendizado por reforço, por meio dos algoritmos Q-Learning(λ) e Sarsa(λ) com generalização de função no domínio do futebol de robôs 2D. Os modelos para o aprendizado foram denominados de discreto e contínuo. Para o primeiro, os estados e as ações foram modeladas de forma discreta, para o segundo, os estados foram generalizados por meio do Tile Coding e as ações permaneceram discretas. As variáveis de estados escolhidas para a modelagem foram a posição dos agentes em campo e a distância entre o agente com posse de bola e o jogador adversário mais próximo. Os algoritmos Q-Learning(λ) e Sarsa(λ) foram testados em ambos os modelos, e o parâmetro λ foi variado entre 0,0 e 0,7. Além disso, no modelo contínuo, a função valor Q dos algoritmos Q-Learning(λ) e Sarsa(λ) foi generalizada por meio de Tile Coding, com valores variando entre 1 e 64 tilings. Verificou-se que o Traço de Elegibilidade influenciou no comportamento de ambos os algoritmos, e que Q-Learning(λ) teve melhor desempenho que Sarsa(λ) para saldo de gols, porém, o mesmo não ocorreu quando usada a Generalização de Função, já que Sarsa(λ) destacou-se.

Palavras-chave: inteligência artificial, aprendizado de máquinas, generalização de função, sistemas complexos.

Abstract

The Reinforcement Learning (RL) is a technique in which agent learn by means of interaction with an environment. It is expected that after a finite number of interactions, the learning process converge to a suboptimal solution. One advantage of reinforcement learning is the ability of the agent to learn in an uncertainty environment. However, the lack of knowledge of the environment, the complexity of the system and a large state space contribute to increase the time required for learning. To minimize this time and decrease the state space, as other techniques, Trace Eligibility, λ,Generalization Function have been used with RL. Inside there are several RL algorithms, which stand out the Q-Learning and Sarsa. This study aims at evaluating the Reinforcement Learning, by means an application of Q-learning (λ) and Sarsa (λ) with generalized function to robot soccer 2D. Models for learning were called discrete and continuous. For the first, states and actions were modelled discretely; for the second, the states were generalized through Tile Coding and actions remain discrete. The state variables were chosen for modelling the position of agents in the field and the distance between the agent with the ball and the player closest opponent. The Q-Learning(λ) and Sarsa (λ) were tested in both models, and the parameter λ was tested between 0.0 and 0.7. Moreover, in the continuous model, the function value of the algorithms Q-Learning(λ) and Sarsa(λ) was generalized by means of Tile Coding, with values ranging between 1 and 64 tilings. It was found that the trace Eligibility affect the behaviour of both algorithms and Q-Learning(λ) performed better than Sarsa (λ) to goal deference, however, this did not occur when used Generalization Function, as Sarsa (λ) stood out.

Keywords: artificial intelligence, machine learning, generalization function, complex systems.

Lista de Tabelas

3.1 Tabela de Macro-Ações formadas por ações básicas 42

3.2 Tabela de Ações Básicas 42

3.3 Função valor estado-ação $Q(s,a)$ na forma tabular. Conforme os pares estado-ação foram visitados, a função $Q(s,a)$ foi atualizada. 43

4.1 Sarsa(λ) - Tempo de decaimento (episódio). 51

4.2 Sarsa(λ) - Média de Gols por Episódio. 52

4.3 Q-Learning(λ) - Tempo de decaimento (episódio). 53

4.4 Q-Learning(λ) - Média de Gols por Episódio. 55

4.5 Sarsa(λ) Tile - Média de Gols por Episódio. 60

4.6 Q-Learning(λ) Tile - Média de Gols por Episódio. 63

Lista de Figuras

2.1 Épocas de Decisão 7

2.2 Diagrama de um Processo de Decisão de Markov. 10

2.3 Traço de Elegibilidade: acumulativo e por substituição 26

2.4 Tile Coding Unidimensional - Generalização do espaço de estados utilizando 4 tilings, totalizando 32 tiles. O vetor de características binário para $s = 0{,}40$ é dado por $\phi(s) = \{0_0, ..., 1_4, 0_5, ..., 1_{12}, 0_{13}, ..., 1_{20}, 0_{21}, ..., 1_{28}, 0_{29}, ..., 0_{32}\}$. Os tiles ativos estão representados pela cor cinza. 31

2.5 Arquitetura do Sistema de Simulação [21]. 35

3.1 Representação do Cenário - Os três defensores foram denominados de G, Z_1 e Z_2 (círculo da cor vermelha). Os atacantes foram chamados de A_1 e A_2 (círculo da cor azul). As tomadas de decisão utilizando o aprendizado por reforço foram feitas apenas pelos atacantes. 38

3.2 Fluxograma da rotina computacional 39

3.3 Representação das variáveis de estado - A área limite possui dimensão igual a 52×64 (comprimento $\times$ largura) unidades, na qual, foi dividido em doze subáreas. 41

3.4 Representação das variáveis de estado - A variável D representa a distância entre o atacante com posse de bola e o defensor mais próximo. . . 41

3.5 Representação dos estados discretos - Os estados foram formados pela posição do agente dentro da área limite e a distância D. A combinação das subáreas $AA_1...AA_{12}$ com os intervalos D_1 e D_2 formaram os 24 estados do sistema, por exemplo, $S_1 = [AA_1, d_1]$. 41
3.6 Representação das variáveis de estado contínuas X e Y. 44
3.7 Representação da variável estado contínua D. 44

4.1 Histograma dos pares estado-ação visitados utilizando o algoritmo Sarsa(λ). As barras possuem cores diferentes, as quais, representam as ações a_1 (cor azul), a_2 (cor verde), a_3 (cor vermelha) e a_4 (cor azul claro). 50
4.2 Histograma dos pares estado-ação visitados utilizando o algoritmo Q-Learning(λ). As barras possuem cores diferentes, as quais, representam as ações a_1 (cor azul), a_2 (cor verde), a_3 (cor vermelha) e a_4 (cor azul claro). 50
4.3 Sarsa(λ) - Média da diferença temporal da matriz $Q(s,a)$ para os $\lambda's$ simulados. 51
4.4 Sarsa(λ) - Média dos gols acumulados para os $\lambda's$ simulados. 52
4.5 Sarsa(λ) - Histograma do número de gols para $\lambda = 0{,}3$. 53
4.6 Q-Learning(λ) - Média da diferença temporal da matriz $Q(s,a)$ para os $\lambda's$ simulados. 54
4.7 Q-Learning(λ) - Média dos gols acumulados para os $\lambda's$ simulados. 54
4.8 Q-Learning(λ) - Histograma do número de gols para $\lambda's = 0{,}3$. 55
4.9 Comparação entre Sarsa(λ) e Q-Learning(λ), $\lambda = 0{,}3$. 56
4.10 Teste T-Student: Comparação entre Sarsa(λ) e Q-Learning(λ), $\lambda = 0{,}3$. . 57
4.11 Sarsa - Média dos gols acumulados para $\lambda = 0{,}3$. 58
4.12 Q-Learning(λ) - Média dos gols acumulados para $\lambda = 0{,}3$. 59
4.13 Sarsa(λ) Tile - Média dos gols acumulados para os $\lambda's$ simulados. 60
4.14 Sarsa(λ) Tile - Histograma do número de gols para $\lambda = 0{,}3$. 61

4.15 Q-Learning(λ) Tile - Média dos gols acumulados para os $\lambda's$ simulados. . 62
4.16 Q-Learning(λ) Tile - Histograma do número de gols para $\lambda = 0{,}3$. 62
4.17 Comparação entre Sarsa Tile(λ) e Q-Learning Tile(λ), $\lambda's = 0{,}3$. 63
4.18 Teste T-Student: Comparação entre Sarsa Tile(λ) e Q-Learning Tile(λ), $\lambda = 0{,}3$. 64
4.19 Comparação entre Sarsa(λ) e Sarsa Tile(λ), $\lambda's = 0{,}3$. 65
4.20 Teste T-Student: Comparação entre Sarsa(λ) e Sarsa Tile(λ), $\lambda = 0{,}3$. . . 65
4.21 Comparação entre Q-Learning(λ) e Q-Learning Tile(λ), $\lambda's = 0{,}3$. 66
4.22 Teste T-Student: Comparação entre Q-Learning(λ) e Q-Learning Tile(λ), $\lambda = 0{,}3$. 66

A.1 Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0.0$). 82
A.2 Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0.0$). 82
A.3 Fluxograma da média de gols, Sarsa($\lambda = 0.0$). 83
A.4 Média de gols acumulados, Sarsa($\lambda = 0.0$) 83
A.5 Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}1$) 84
A.6 Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}1$) 84
A.7 Fluxograma da média de gols, Sarsa($\lambda = 0{,}1$). 85
A.8 Média de gols acumulados, Sarsa($\lambda = 0{,}1$). 85
A.9 Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}2$). 86
A.10 Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}2$). 86
A.11 Fluxograma da média de gols, Sarsa($\lambda = 0{,}2$). 87
A.12 Média de gols acumulados, Sarsa($\lambda = 0{,}2$). 87
A.13 Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}3$) 88

A.14 Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa(λ = 0,3). 88
A.15 Fluxograma da média de gols, Sarsa($\lambda = 0{,}3$). 89
A.16 Média de gols acumulados, Sarsa($\lambda = 0{,}3$). 89
A.17 Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}4$). 90
A.18 Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa(λ = 0,4). 90
A.19 Fluxograma da média de gols, Sarsa($\lambda = 0{,}4$). 91
A.20 Média de gols acumulados, Sarsa($\lambda = 0{,}4$). 91
A.21 Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}5$). 92
A.22 Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa(λ = 0,5). 92
A.23 Fluxograma da média de gols, Sarsa($\lambda = 0{,}5$). 93
A.24 Média de gols acumulados, Sarsa($\lambda = 0{,}5$). 93
A.25 Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}6$). 94
A.26 Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa(λ = 0,6). 94
A.27 Fluxograma da média de gols, Sarsa($\lambda = 0{,}6$). 95
A.28 Média de gols acumulados, Sarsa($\lambda = 0{,}6$). 95
A.29 Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}7$). 96
A.30 Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa(λ = 0,7). 96
A.31 Fluxograma da média de gols, Sarsa($\lambda = 0{,}7$). 97
A.32 Média de gols acumulados, Sarsa($\lambda = 0{,}7$). 97

Lista de Símbolos

α	Taxa de aprendizagem;
γ	Fator de desconto para os reforços futuros;
S	Conjunto finito de ações;
T	Função de Transição de Estado;
R	Função de Recompensa;
π	Política;
π^*	Política ótima;
Q	Função valor estado-ação;
Q^*	Função valor estado-ação ótima;
V	Função valor estado;
V^*	Função valor estado ótima;

Capítulo 1

Introdução

O desenvolvimento de máquinas e sistemas inteligentes tem sido o alvo de pesquisadores de diversas partes do mundo [37]. A partir da década de 90, as máquinas inteligentes tornaram-se realidade e estão cada vez mais presentes na sociedade [29, 55]. A Inteligência Artificial (IA) é uma área da ciência que abrange conceitos como, autonomia e inteligência de máquinas. Várias técnicas de IA são adotadas na tentativa de encontrar a melhor solução para uma determinada tarefa, tais como: o Aprendizado por Reforço (AR) [42], o Aprendizado por Reforço Acelerado por Heurísticas [21], as Redes Neurais Artificiais [43], Sistemas Fuzzy [60, 39] e os Algoritmos Genéticos [15].

O AR é uma técnica de aprendizado fundamentada no conceito de Processo de Decisão de Markov (PDM). Nesta técnica, o agente aprende por meio de interação direta com o ambiente, convergindo para uma situação de equilíbrio [50]. Uma das vantagens do uso da técnica de AR está na possibilidade do agente aprender em um ambiente totalmente desconhecido, devido às experiências obtidas por meio de tentativa e erro. Por outro lado, a falta de conhecimento prévio do ambiente, a complexidade do sistema e um grande espaço de estados podem contribuir para o aumento do tempo necessário de aprendizado [52, 40].

Na tentativa de minimizar o tempo de aprendizado e diminuir o espaço de estados, outras técnicas tais como, os Traços de Elegibilidade e a Generalização de Funções tem

sido utilizadas em conjunto com o AR.

A técnica de Traço de Elegibilidade combinada com o AR tem como objetivo acelerar o aprendizado, por meio da memorização dos estados recém-visitados pelo agente [35, 59].

A generalização de funções tem sido amplamente utilizada na redução do espaço de estados, como uma alternativa à discretização do espaço, a qual, apesar de fácil implementação dos algoritmos [52, 22], pode resultar um mal desempenho no aprendizado, devido a uma discretização ruim [17]. Por meio da generalização de função é possível visitar um subconjunto limitado do espaço de estados e produzir uma boa aproximação de todo o espaço [52]. A generalização de funções é capaz de lidar com variáveis contínuas em várias dimensões, diminuindo o custo computacional e viabilizando o aprendizado de algumas tarefas de maneira eficiente [8].

Existem várias formas de generalização de funções, as quais, dividem-se em lineares e não-lineares. As técnicas de generalização frequentemente utilizadas em conjunto com o AR são as lineares, dentre as quais estão o Tile Coding [40], o Kanerva Coding [24], a Função Radial de Base (*Radial Basis Functions - RBFs*) [38]. A teoria de aprendizagem por reforço com aproximação de função não é ainda bem entendida [52, 5, 49]. Entretanto, possui vários exemplos de sucesso [18].

Dentro do AR existem vários algoritmos utilizados para o aprendizado, dos quais, destacam-se o Q-Learning [57] e o Sarsa [52]. Estes algoritmos são conhecidos como sendo de fácil implementação [52, 22], e por isso, têm sido amplamente utilizados [14, 30, 12, 45, 28].

Os algoritmos Q-Learning e Sarsa têm sido utilizados com as técnicas de Traço de Elegibilidade [23] e Generalização de Funções [3, 40], apresentando bons resultados [16], apesar de ambos métodos não garantirem a convergência para a política ótima em todos os casos [19, 56, 44]. Além disso, o algoritmo Q-Learning com generalização de funções pode tornar-se instável em alguns casos [48], ao contrário do Sarsa com generalização

de função linear [52].

Em AR, os ambientes de simulação têm contribuído para o uso da técnica, podendo ser obtidas soluções para o problema antes da aplicação em sistemas reais. Um ambiente que tem sido amplamente estudado e que tem contribuído para o avanço do AR é a simulação de futebol de robôs 2D.

A simulação de futebol de robôs 2D é uma categoria criada pela Robocup, na qual, partidas de futebol entre equipes de robôs virtuais autônomos são realizadas para simular condições reais de jogo, por meio de um simulador que fornece um sistema totalmente distribuído, onde, os agentes (ou jogadores) utilizam sensores e atuadores com ruídos, tornando o ambiente incerto [32]. Além disso, uma comunicação limitada entre os agentes, ciclos de percepção e ação assíncronos fazem do simulador um ambiente realista e complexo.

A complexidade do ambiente 2D é uma das características que torna o domínio de futebol de robôs adequado para a avaliação das técnicas de AR. Além das técnicas de traço de elegibilidade [47] e generalização de funções [48] no domínio de futebol de robôs 2D.

Sabe-se que o aprendizado por reforço apresenta dificuldades de convergência. Essa dificuldade não é geral e não há evidências na literatura que garantam que isso serve para todos os sistemas. Assim, nesse trabalho deseja-se verificar se a aplicação por aprendizado por reforço no futebol 2D apresenta também essas características, utilizando os algoritmos Q-Learning e Sarsa com traço de elegibilidade, e com a generalização de função linear.

-

Capítulo 2

Revisão de Literatura

Neste capítulo, são apresentados algumas teorias e técnicas relacionadas com Aprendizado por Reforço (AR) e o domínio do futebol de robôs 2D.

A primeira teoria é o Processo de Decisão de Markov (PDM), na qual, o AR foi formalizado. Os sistemas modelados por PDM devem obedecer a propriedade de Markov: o efeito de uma ação em um estado depende apenas da ação e do estado atual do processo.

A segunda, Programação Dinâmica (PD), teve como principal colaborador Richard Bellman que estabeleceu uma maneira discreta e estocástica de resolver problemas de controle ótimo, por meio de uma equação recursiva, conhecida como equação de Bellman. A PD é um método recursivo para resolução de problemas sequenciais de decisão.

A terceira, Monte Carlo (MC), utiliza a experiência adquirida no ambiente para a estimação da função. Desta forma, não é necessário um total conhecimento prévio do sistema.

A quarta, Diferença Temporal (DT), permite um aprendizado direto sem a necessidade de conhecimento do ambiente. Além disso, a atualização da função é realizada antes do término de cada episódio. Esse método baseia-se nas principais ideias da PD e MC.

Os algoritmos Q-Learning e Sarsa estão entre os mais utilizados, sendo que o primeiro

tem como característica a capacidade de aprender por meio da iteração de uma política ótima, e o segundo, por meio da política atual (Off-Policy e On-Policy, respectivamente).

A técnica, Traço de Elegibilidade, é utilizada para que o agente memorize os estados recém-visitados. O estado ao ser visitado é marcado como elegível. A partir daí, a recompensa ou punição é atribuída ao estado atual e aos estados elegíveis.

A generalização de funções é um método utilizado para reduzir o espaço de estados do sistema. Este método busca uma aproximação de todo o espaço, visitando apenas um subconjunto limitado do espaço de estados.

A simulação de futebol 2D é o domínio escolhido para a realização deste trabalho. Este domínio oferece um ambiente de simulação multiagente em tempo real, que retrata a complexidade de uma partida de futebol, adicionando ruídos às ações e percepções dos agentes.

2.1 Processos de Decisão de Markov - PDM

Em PDM as transições entre os estados são probabilísticas, e a interferência no processo ocorre periodicamente por meio das ações tomadas pelo agente [7]. As ações sempre estão associadas a uma recompensa (ou custo), a qual, está relacionada ao estado atual do processo.

Um processo é dito *de Markov* quando este obedece à propriedade de Markov, ou seja, o efeito de uma ação em um estado depende apenas da ação e do estado atual do processo; e são chamados de processos *de decisão* porque modelam a possibilidade de um agente interferir periodicamente no sistema executando ações [36].

Um PDM pode ser definido formalmente pela quádrupla (S, A, T, R) [22] em que:

- S: é o conjunto finito de estados do ambiente,
- A: é o conjunto finito das ações que podem ser realizadas por um agente,

- $\mathbb{T} : S \text{ x } A \rightarrow \pi(S)$: é a função de transição de estado, sendo $\pi(S)$ uma distribuição de probabilidade para cada par estado/ação,

- $R : S \text{ x } A \rightarrow R$: é a função de recompensa.

A função de transição $\pi(S)$ representa uma distribuição de probabilidade para cada par estado/ação seguindo uma política π qualquer.

Uma política π é um conjunto de regras que irá determinar quais estados o agente percorrerá, por meio das análises das recompensas.

As tomadas de decisões devem ocorrer em períodos determinados. Seja na forma discreta ou contínua, as decisões ocorrem na chamada *época de decisão*. A seguir será definido o conceito de época de decisão para os casos de tempo discreto e contínuo.

2.1.1 Época de Decisão

As decisões são tomadas em períodos pré-determinados (Figura 2.1), os quais, são denominados de épocas de decisão, T. As épocas de decisão podem ser classificadas como: discreta ou contínua, e, finita ou infinita. Quando discreta, as épocas de decisão ocorrem em intervalos de tempo maiores e as decisões são tomadas em todas as épocas. Quando contínua, as épocas de decisão ocorrem em intervalos de tempo menores, e as decisões podem ser tomadas em todas as épocas, aleatoriamente, ou, a escolha do agente [36].

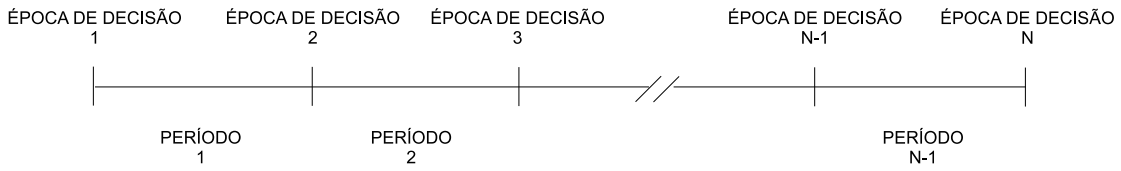

Figura 2.1: Épocas de Decisão

O conjunto de épocas de decisão finito pode ser definido como $T \equiv \{1, 2, ..., N\}$ para qualquer $N \leq \infty$, ou, infinito para $T \equiv \{1, 2, ...\}$. Geralmente, os elementos de T são

denotados por t. O problema de decisão pode ser denominado *problema de horizonte finito* ou *problema de horizonte infinito*, quando N é finito ou infinito, respectivamente [36]. Por convenção, em *problemas de horizonte finito*, as decisões são tomadas até o período $N - 1$.

2.1.2 Conjunto de Estados e Ações

Os problemas modelados pelo PDM possuem um conjunto, S, de estados, s, no qual, a cada época de decisão, o sistema ocupa um estado s_t. Quando em época de decisão, o agente observa o sistema no estado $s \in S$, e escolhe uma ação, $a \in A$, em que A representa o conjunto de ações permitidas [2]. As ações podem ser escolhidas de forma aleatória ou determinística.

Alguns trabalhos sobre PDMs frequentemente assumem que os estados de um modelo são facilmente obtidos do problema [6, 36] e que as variáveis de estado devem ser escolhidas de tal modo que a propriedade de Markov se mantenha. Hasselt (2011) define as variáveis de estado da seguinte forma [20]:

A variável de estado deve ser necessária e suficiente para calcular a função de decisão.

Se o espaço de estados contínuo possuir uma dimensão relativamente pequena, então existe a possibilidade de discretização do espaço de estados na fase de modelagem. Após a discretização dos estados, um algoritmo adequado para AR deve ser utilizado para resolução do problema. A discretização do estado simplifica o problema, contudo, a solução do PDM discreto pode não ser ideal. Em problemas reais, geralmente, os PDMs são modelados por meio de variáveis contínuas [51, 12, 13].

2.1.3 Recompensa e Probabilidade de Transição

A decisão tomada pelo agente ao escolher a ação, $a \in A$, no estado, $s \in S$, em época de decisão, t, resulta no ganho de uma recompensa, $r_t(s,a)$. O estado do sistema na próxima época de decisão, s_{t+1}, é determinado pela distribuição da probabilidade $p_t(.|s,a)$ [36]. A recompensa pode ser:

1. uma quantia recebida em um tempo fixo ou aleatório antes da próxima época de decisão,

2. acumulada continuamente ao longo dos períodos,

3. uma quantidade aleatória que depende do estado atual do sistema no período posterior à decisão, ou,

4. uma combinação das anteriores.

Quando a recompensa depende somente do estado do sistema na próxima época de decisão, a recompensa é definida como $r_t(s,a,j)$, na qual, o estado do sistema na época de decisão, t, é igual a $s \in S$, a ação $a \in A$ é selecionada e o sistema passa para o estado j na época de decisão $t+1$. A recompensa esperada na época de decisão, t, pode ser calculada por [36]:

$$r_t(s,a) = \sum_{j \in S} r_t(s,a,j) p_t(j|s,a) \tag{2.1}$$

Na Equação 2.1, a função não-negativa, $p_t(j|s,a)$, representa a probabilidade do sistema mudar para o estado $j \in S$ no instante de tempo $t+1$, quando o agente escolher a ação $a \in A$ no estado $s \in S$ no tempo t. A função $p_t(j|s,a)$ é denominada *função de transição de probabilidade*, na qual, assume-se que

$$\sum_{j \in S} p_t(j|s,a) = 1$$

Quando no horizonte finito, as decisões não são tomadas na época de decisão N. Por isso, a recompensa neste momento é dada apenas em função do estado s.

2.1.4 Política para um PDM

A dinâmica de um sistema modelado como um PDM pode ser observada na Figura 2.2. O agente verifica o estado atual do sistema, consulta uma política (π) e executa uma ação (a), na qual, pode causar um efeito direto sobre o estado.

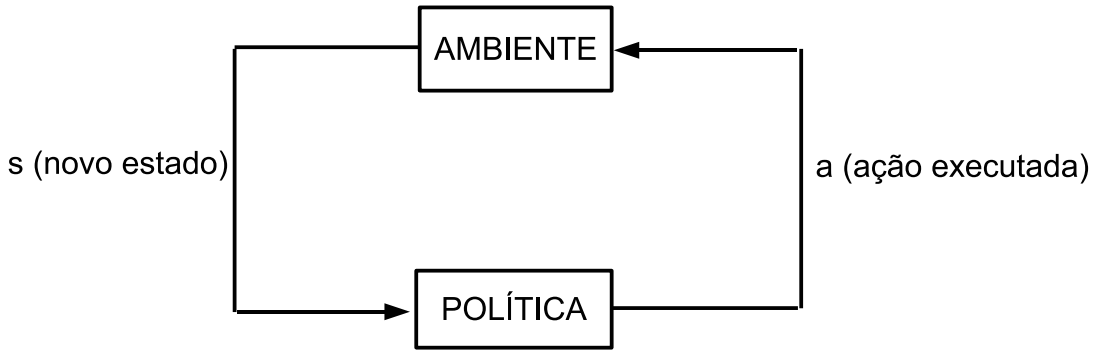

Figura 2.2: Diagrama de um Processo de Decisão de Markov.

A cada época de decisão, o agente usa uma regra de decisão para escolher a próxima ação. Uma forma simples de regra de decisão é um mapeamento direto de estados em ações. O conjunto de todas as regras de decisão (uma para cada época de decisão) é chamado de política. Uma política pode ser definida como sendo [36]:

- total, se cada uma de suas regras de decisão é definida para todos os estados do PDM;
- parcial, se alguma de suas regras de decisão é definida para apenas alguns estados do PDM.

Quanto à sua relação com as épocas de decisão, uma política pode ainda ser classificada como [36]:

- estacionária, se a ação recomendada independe da época de decisão (ou seja, $d_k = d_j$, $\forall$ k e j);
- não-estacionária, se a ação tomada depende da época de decisão.

É comum o uso de políticas estacionárias para horizonte infinito e não-estacionárias para horizonte finito, entretanto, tanto as políticas estacionárias quanto as não estacionárias podem ser usadas com qualquer horizonte [34]. Um sistema de horizonte finito pode ter a mesma política para cada época de decisão e um sistema de horizonte infinito pode mudar a política periodicamente. Uma política também pode ser classificada como:

- determinística, quando cada estado é sempre mapeado em uma única ação;
- não-determinística (estocástica), quando um estado é mapeado em um conjunto de ações, sendo que cada ação tem uma probabilidade de ser escolhida ($d_k : S$ x $A \rightarrow [0, 1]$).

ou,

- Markoviana (ou sem memória), quando a escolha da ação depende apenas do estado corrente;
- não-Markoviana, quando a escolha da ação depende de todo o histórico de ações e estados do sistema. As regras de decisão são definidas então como funções $d_k : H_k \rightarrow A$, onde H_k é o histórico de estados e ações até a época de decisão k.

Ao executar uma política, o agente receberá recompensas em cada época de decisão, as quais determinam o desempenho da mesma. Uma maneira de comparar duas políticas

é por meio de um critério de desempenho. Diversos critérios de desempenho (ou de otimalidade) para PDMs são encontrados na literatura, e entre os mais conhecidos estão [34]:

- a recompensa média por época de decisão, $\frac{1}{z}\sum_{k=0}^{z-1} r_k$;
- a recompensa esperada total, E $\sum_{k=0}^{z-1} r_k$;
- a recompensa esperada descontada, E $[\sum_{k=0}^{z-1} \gamma^k r_k]$.

A recompensa na época de decisão k é denotada por r_k e $\gamma \in]0,1[$ é um fator de desconto. O fator de desconto é usado para um problema de horizonte infinito para garantir a convergência do valor da recompensa total esperada.

Puterman [36] define uma regra de decisão para um PDM em uma época de decisão k como sendo uma função $d_k : S \rightarrow A$, que determina a ação a ser executada, dado o estado do sistema. Uma política para um PDM é uma sequência de regras de decisão $\pi = (d_0, d_1,..., d_{z-1})$ tomadas em épocas de decisão.

A função valor de uma política π para um PDM, $M = (S,A,T,R)$, é uma função $V^\pi : S \rightarrow R$, tal que V^π é o valor esperado da recompensa, de acordo com o critério de otimalidade [52]. O valor esperado da recompensa para horizonte finito com desconto seja $\pi = \{d_0, d_1, ..., d_{z-1}\}$ é dado por:

$$V_k^\pi(s) = R(s, d_k(s)) + \gamma \sum_{s' \in S} T(s, d_k(s),s') V_{k+1}^\pi(s') \tag{2.2}$$

em que $V_z^\pi(s) = 0$ para todo s.

Definição - Espaço de políticas para PDMs

Dado um PDM, $M = (S,A,T,R)$ é um conjunto de todas as políticas para M e $V(M)$, o conjunto de todas as funções valor $\mathbb{V} : S \rightarrow R$ para M. O espaço $\mathbb{V}$ será um espaço linear, se cada função valor for representada por um vetor e as operações de soma e multiplicação por um escalar forem definidas no espaço $\mathbb{R}^n$.

Seja um estado $s \in S$, uma ação $a \in A$ e uma política π para um PDM, pode-se definir o valor da ação a no estado s, considerando a recompensa imediata de a e a recompensa esperada após a, nas outras épocas de decisão, desde que as ações tomadas após a sejam determinadas pela política π. A função que expressa esse valor é denominada por Q, na qual, a esperança da recompensa total descontada para o horizonte finito, é definida como [52],

$$Q_k^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V_{k+1}^{\pi}(s') \tag{2.3}$$

Definição - Política ótima para um PDM

Seja $M = (S, A, T, R)$ o conjunto de todas as política de um PDM com horizonte z. Uma função valor V^* é ótima para M se $\forall\ U \in V$, $\forall\ s \in S$, $V_k(s) \geq U_k(s)$ para todo $k \in \mathbb{N}$ tal que $0 \leq k < z$. Uma política π^* é ótima para M se $\forall \pi' \in \prod$, $\forall s \in S$, $Q_k^{\pi^*}(s,d_k^{\pi^*}(s)) \geq Q_k^{\pi'}(s,d_k^{\pi'}(s))$ para todo $k \in \mathbb{N}$ tal que $0 \leq k < z$ [52].

Provou-se que existe apenas uma função que satisfaz este critério, sendo portanto chamada de função valor ótima. No entanto, é possível que várias políticas tenham a mesma função valor, as quais, são denominadas de políticas otimais. A função ótima em função da política ótima pode ser definida como,

$$\begin{aligned} Q^*(s,a) &= R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V^*(s'), \\ \pi^*(s) &= arg \max_{a \in A} Q^*(s,a), \\ V^*(s) &= \max_{a \in A} Q^*(s,a). \end{aligned} \tag{2.4}$$

onde, $arg \max_{a \in A}$ indica a ação a que maximiza a função valor estado-ação $Q(s,a)$ e $\max_{a \in A}$ representa o valor de Q obtido com a ação a.

2.2 Programação Dinâmica - PD

Programação dinâmica (PD) é um método recursivo para resolver problemas sequenciais de decisão. A partir da teoria de PD é possível construir e resolver uma variedade enorme de problemas de tomada de decisão, tanto em sistemas determinísticos e não-determinísticos [4, 34].

Um dos principais colaboradores da PD foi *Richard Bellman*, o qual propôs várias teorias, entre as quais destacam-se o Príncipio de Otimalidade de Bellman e a Equação de Bellman [6]. O Príncipio de Otimalidade de Bellman afirma que seguir uma política ótima entre um estado inicial e um estado final, passando por um estado intermediário, é equivalente a seguir a melhor política do estado inicial até o intermediário, seguida da melhor política deste até o estado final [8]. A Equação de Bellman é dada pela função [6].

$$V^{\pi}(s) = \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^{\pi}(s')] = \pi(s) \tag{2.5}$$

onde, $P^{a}_{ss'} = P_r\{s_{t+1} = s'|s_t = s, a_t = a\}$ são as probabilidades de transição e $R^{a}_{ss'} = E\{r_{t+1}|a_t = a, s_t = s, s_{t+1} = s'\}$ são as expectativas de recompensa imediata para todo $s \in S$, $a \in A$ e $s' \in S^{+}$ (S^{+} é S união com os estados terminais, se o problema for episódico).

O objetivo da PD é encontrar a melhor política, e para isso, deseja-se encontrar a função valor máxima, ou seja, descobrir o valor da função que forneça a melhor política. Uma maneira seria escolher a ação que forneça o maior valor da função, $V^{\pi}(s)$. A função valor é máxima se a política é ótima, ou seja, se as ações tomadas pela política levam a maximização da função, V. A função valor máxima é expressa por:

$$V^{\pi}(s) = \max_{a} \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^{\pi}(s')] \tag{2.6}$$

onde, $\max_a$ indica o maior valor de V obtido com a ação a.

A função valor e a política são independentes entre si, por isso, os algoritmos procuram melhorar a função valor de maneira a encontrar a política ótima.

2.2.1 Avaliação de Políticas

A obtenção de uma função valor-estado, V^{π}, para uma política π qualquer é considerado por Sutton [52] um *problema de predição*. Em PD, esse problema é denominado de *Avaliação de Política*, na qual, a função V^{π} é expressa por:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi} r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... | s_t = s \\ &= E_{\pi} r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s \\ &= \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^{\pi}(s')] \end{aligned} \tag{2.7}$$

onde, $\pi(s,a)$ são as probabilidades de tomar a ação a no estado s adotando uma política π. $P^{a}_{ss'} = P_r\{s_{t+1} = s' | s_t = s, a_t = a\}$ são as probabilidades de transição e $R^{a}_{ss'} = E\{r_{t+1} | a_t = a, s_t = s, s_{t+1} = s'\}$ são as expectativas de recompensa imediata.

A existência e a unicidade de V^{π} são garantidas quando $\gamma < 1$. A forma recursiva

é dada por,

$$
\begin{aligned}
V_{k+1}(s) &= E_\pi r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s \\
&= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V_k(s')] \qquad (2.8)
\end{aligned}
$$

A sequência V_{k+1} converge para V^π quando $k \to \infty$ sob algumas condições que garantam a existência de V^π [52]. O cálculo iterativo utilizando a expressão 2.8 é realizado no algoritmo 1, o qual, é denominado de *Avaliação de Política Iterativa*. A função valor $V(s)$ é inicializada, e V é calculada até que a condição de parada seja satisfeita ($\Delta \ll 1$).

Entrada: π (política que será avaliada)
Inicialize $V(s) = 0, \forall s \in S$;
repita
  $\Delta \leftarrow 0$;
  para cada $s \in S$ **faça**
    $v \leftarrow V(s)$;
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')]$;
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$;
  fim
até *até* $\Delta \ll 1$;
Saída: $V \approx V^\pi$

Algoritmo 1: Avaliação de Política Iterativa

2.2.2 Melhoria de Políticas

O cálculo da função valor, V, para uma determina política, π, auxilia na tomada de decisão de acordo com o conjunto de regras descritas por π. A escolha de outras ações, isto é, $a \neq \pi(s)$, poderá ou não aumentar os ganhos de recompensas. A função valor estado-ação, $Q^\pi(s,a)$, usada para avaliar a ação a no estado s é dada por,

$$
\begin{aligned}
Q^\pi(s,a) &= E_\pi \{ r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a \} \\
&= \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')] \qquad (2.9)
\end{aligned}
$$

onde, $P^a_{ss'} = P_r\{s_{t+1} = s'|s_t = s, a_t = a\}$ são as probabilidades de transição, $R^a_{ss'} = E\{r_{t+1}|a_t = a, s_t = s, s_{t+1} = s'\}$ são as expectativas de recompensa imediata para todo $s \in S$, $a \in A$ e $s' \in S^+$ (S^+ é S união com os estados terminais, se o problema for episódico) e $\gamma V(s')$ é o valor da função V para o estado s' com o fator de desconto.

Após modificar a política atual π para π', ou seja, $a = \pi'(s)$, é possível avaliar a nova política por meio da equação 2.9 resultando em $Q(s,\pi'(s))$. O *Teorema de Melhoria de Políticas* afirma que para qualquer par de políticas π e π', tal que, para todo $s \in S$, e sendo, a inequação 2.10 verdadeira, então, a política π' é melhor ou igual a π.

$$Q^{\pi}(s,\pi^{'}) \geq V^{\pi}(s) \tag{2.10}$$

onde, $Q^{\pi}(s,\pi^{'})$ é o valor da função Q para o estado s seguindo a política $\pi^{'}$ e $V^{\pi}(s)$ é o valor da função V para o estado s seguindo a política π.

A expectativa do retorno de todos os estados $s \in S$ é dada por, [52]

$$V^{\pi^{'}}(s) \geq V^{\pi}(s) \tag{2.11}$$

Selecionando a melhor ação pertencente a A é possível a escolha de uma política que maximize os ganhos, considerando todos os estados, $s \in S$. Esse conjunto de ações está associado a uma nova política, a qual, é denominada *Política Gulosa*, $\pi'(s)$, dada por,

$$\begin{aligned} \pi^{'}(s) &= arg \max_a Q^{\pi}(s,a) \\ &= arg \max_a E\{r_{t+1} + \gamma V^{\pi}(s_{t+1})|s_t = s, a_t = a\} \\ &= arg \max_a \sum_{s'} P^a_{s's}[R^a_{s's} + \gamma V^{\pi}(s^{'})] \end{aligned} \tag{2.12}$$

onde, $arg \max_a$ indica a ação a que maximiza a função valor estado-ação, $Q^{\pi}(s,a)$.

A *Política Gulosa* seleciona a ação que parece ser a melhor em um curto prazo. A *Melhoria de Política* é definida quando a nova política adotada for a *gulosa*.

Suponha que a nova política gulosa, $\pi^{'}$, é tão boa quanto, mas não melhor do que a política, π. Então, $V^{\pi} = V^{\pi^{'}}$, $\forall s \in S$, logo,

$$\begin{aligned} V^{\pi^{'}}(s) &= \max_{a} E\{r_{t+1} + \gamma V^{\pi^{'}}(s_{t+1}) | s_t = s, a_t = a\} \\ &= \max_{a} \sum_{s'} P^{a}_{s's}[R^{a}_{s's} + \gamma V^{\pi^{'}}(s^{'})] \end{aligned} \tag{2.13}$$

As equações 2.13 e 2.6 são iguais, portanto, $V^{\pi^{'}} = V^{\pi^{*}}$, e as políticas π e $\pi^{'}$ devem ser ótimas. Assim, a *Melhoria de Política* fornece uma política estritamente melhor, exceto quando a política original é ótima [52].

2.2.3 Iteração de Políticas

Na seção anterior foi apresentada uma maneira para melhorar a política π, usando V^{π} para avaliar uma nova política π'. Logo, é possível melhorar a política π', usando $V^{\pi^{'}}$ para avaliar uma nova política π'', e assim, obter uma seqüência monotônica de melhoria de políticas e de funções valor, como ilustrado a seguir:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi^{*} \xrightarrow{E} V^{\pi^{*}}$$

onde, $\xrightarrow{E}$ simboliza a avaliação de política e $\xrightarrow{I}$ a melhoria de política. Esta maneira de encontrar uma política ótima é chamada *política de iteração*. O algoritmo 2 apresenta o procedimento para obtenção da política de iteração.

A ideia do algoritmo 2 é encontrar a política ótima usando a técnica de *Melhoria de Política*, por meio da equação 2.12. Inicialmente, o algoritmo calcula a função valor, $V(s)$, usando a técnica de *Política de Avaliação*, item 2.2.1. Uma vez o critério de

Entrada: π (política que será avaliada)
Inicialize $V(s)$ e π $\forall s \in S$;
repita
  repita
    $\Delta \leftarrow 0$;
    para cada $s \in S$ **faça**
      $v \leftarrow V(s)$;
      $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$;
      $\Delta \leftarrow max(\Delta, |v - V(s)|$;
    fim
  até *até* $\Delta \ll 1$;
  política-estável $\leftarrow$ verdadeira ;
  para cada $s \in S$ **faça**
    $b \leftarrow \pi(s)$;
    $\pi(s) \leftarrow arg\max_a \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$;
    se $b \neq \pi(s)$ **então**
      política-estável $\leftarrow$ falso ;
    fim
  fim
até *política-estável = verdadeira*;
Saída: π^*, V^{π^*}

Algoritmo 2: Avaliação de Política Iterativa

parada seja satisfeito, ocorre a busca da política ótima, de acordo com a equação 2.12, ou seja, a sequência de ações a que maximizam a expressão 2.9.

2.3 Método de Monte Carlo - MC

A função valor $V^*(s)$, e a política ótima π^*, são obtidas pelo método de MC, por meio de experiências simples de estados, ações e reforços, ao contrário dos métodos de PD que exigem um conhecimento prévio do ambiente representado pela matriz probabilidade de transição de estados. Devido a falta de conhecimento do ambiente, o método de MC utiliza a média dos retornos para estimar as funções V^π e Q^π, e a política π^*.

2.3.1 Avaliação de Política

Os métodos apresentados no tópico de PD são frequentemente utilizados em casos que se conhece o mundo. O método de Avaliação de Política tem o objetivo de estimar a função $V^\pi(s)$ no estado, s, sob a polítca, π, dado um conjunto de episódios. A experiência adquirida no ambiente é utilizada na estimação da função, por meio da média dos retornos observados após a visita ao estado s. De acordo com a teoria dos *Grandes Números*, a função $V^\pi(s) \rightarrow V^*(s)$ quando $n \rightarrow \infty$, sendo, n o número de visitas ao estado s, e, $V^*(s)$ o valor ótimo da função valor [44]. O algoritmo 3 mostra os procedimentos para a estimação da função V, por meio do Avaliação de Política.

A partir de valores iniciais da função valor $V(s)$, e da lista $Retorno(s)$, o algoritmo gera episódios que são utilizados para o cálculo da função valor por meio da lista, a qual, armazena as recompensas recebidas após cada visita.

2.3.2 Controle

Neste método, as Políticas de Avaliação e Melhoria são utilizadas na busca da política ótima e de uma função valor estado-ação ótima, $Q^{\pi^*}(s,a)$, por meio de uma sequência

```
Inicialize
V(s), π ∀s ∈ S
Retorno(s) ← lista vazia
repita
    Gere um episódio tomando a política π ;
    para cada s ∈ S visitado no episódio faça
        R ← retorna a primeira visita de s ;
        Acrescenta R a Retorno(s);
        V(s) ← media(Retorno(s)) ;
    fim
até número de episódios desejado;
```

Algoritmo 3: Avaliação de Política

monotônica. A figura abaixo ilustra a sequência de avaliações e melhorias.

$$\pi_0 \xrightarrow{E} Q^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} Q^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} Q^{\pi^*}$$

O símbolo $\xrightarrow{E}$ indica o passo, no qual, a política de avaliação é utilizada e, em seguida, está o símbolo $\xrightarrow{I}$ representando a política de melhoria. O procedimento completo pode ser visto no algoritmo 4.

```
Inicialize
Q(s,a), π ∀s ∈ S, a ∈ A
Retorno(s,a) ← lista vazia
repita
    Gere um episódio tomando a política π e usando a exploração inicial ;
    para cada s, a visitado no episódio faça
        R ← retorna a primeira visita de s, a ;
        Acrescenta R a Retorno(s,a);
        Q(s,a) ← media(Retorno(s,a)) ;
    fim
    para cada s no episódio faça
        π(s) ← arg max_a(Q(s,a)) ;
    fim
até;
```

Algoritmo 4: Controle com exploração inicial

No algoritmo 4, a Política de Avaliação é semelhante à descrita na Seção 2.2.1, exceto, pela Política de Melhoria que adota a Política Gulosa. Como mencionado, a

Política Gulosa toma a ação que fornece o maior valor da função, $Q(s,a)$, conforme a expressão 2.14.

$$\pi(s) \leftarrow arg\max_{a}(Q(s,a)) \tag{2.14}$$

$$Q^{\pi_k}(s,\pi^{k+1}) \geq V^{\pi_k}(s) \tag{2.15}$$

Segundo o *Teorema de Melhoria de Política*, Seção 2.2.2, para qualquer par de políticas π_k e π_{k+1}, tal que, $\forall s \in S$, e sendo, a inequação 2.15 verdadeira, então, a política π_{k+1} é melhor ou igual a π_k. Logo, a escolha de políticas que tem o argumento máximo da função $Q(s,a)$ poderá levar a uma política ótima, π^*.

2.4 Diferença Temporal - DT(0)

O método de Diferença Temporal (DT) baseia-se nas principais ideias dos métodos de PD e de MC. A semelhança entre os métodos de MC e DT está no aprendizado direto com o ambiente sem a necessidade de conhecimento prévio. A PD e o DT compartilham do princípio de atualizar a estimativa da função valor antes do término de cada episódio, utilizando a expressão 2.16.

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2.16}$$

A atualização da função valor $V(s)$ é realizada pelos termos $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$. Devido a diferença $\gamma V(s_{t+1}) - V(s_t)$ nos instantes $t+1$ e t originou-se o nome de DT.

A atualização da estimativa antes do término do episódio é conhecido como bootstrapping, nos quais, os métodos PD e DT estão inseridos. O procedimento que descreve o método de DT é visto no algoritmo 5. Neste algoritmo, a estimativa da função valor

é feita utilizando a equação 2.16, a qual, tem convergência garantida como nos métodos PD e MC.

```
Inicialize
V(s), π ∀s ∈ S, a ∈ A
para cada episódio faça
    Inicialize s;
    para cada passo do episódio faça
        a ← ação conforme π(s);
        Toma a ação a;
        observe a recompensa, r, e o próximo estado, s';
        V(s) ← V(s) + α[r + γV(s') − V(s)] ;
        s ← s' ;
    fim
fim
```

Algoritmo 5: TD(0) tabular para estimação da função valor V^{π}

2.5 Q-Learning

O algoritmo Q-Learning [57] é conhecido por vários autores como sendo de fácil implementação [22], e por isso, tem sido amplamente utilizado [14, 30].

O Q-Learning tem como principal característica a capacidade de aprender por meio da interação de uma política π^* ótima na ausência de um modelo do sistema. A ideia do algoritmo é aprender por meio da interação com o ambiente, utilizando as melhores ações para encontrar a função valor ótima $Q^*(s,a)$ [58].

O conhecimento adquirido é armazenado em uma matriz $Q(s,a)$, que possui os valores estimados para cada par de estado-ação. A função Q é estimada por meio da equação [57]:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{2.17}$$

onde,

- s é o estado atual;
- a é a ação executada em a;
- r é a recompensa dada ao executar a estando em s;
- s' é o próximo estado;
- a' é a ação executada em s';
- α é a taxa de aprendizagem;
- γ é a taxa de desconto ($0 < \gamma < 1$).

O termo $\max_{a'} Q(s',a')$ representa a utilização das melhores ações na busca da função valor ótima $Q^*(s,a)$, pois, se utiliza das ações que fornece o melhor valor da função Q.

O algoritmo Q-Learning pode ser usado com qualquer tipo de estratégia de exploração durante um processo interativo de aproximação da função Q. Entretanto, deve-se garantir que cada par de estados seja visitado inúmeras vezes, para que a convergência de Q para Q^* (função valor ótima) seja garantida de maneira gradual e lenta [20]. O algoritmo procedural pode ser visto em detalhes abaixo:

```
Inicialize Q(s,a) arbitrariamente ;
para todo episodio faça
    Inicialize s (estado inicial) ;
    repita
        a ← ação para s utilizando política derivada de Q ;
        Tome ação a, observe a recompensa r e o próximo estado s' ;
        Q(s,a) ← Q(s,a) + α[r + γ max_a' Q(s',a') − Q(s,a)] ;
        s ← s';
    até s ser terminal;
fim
```

Algoritmo 6: Q-Learning – Controle Off-Policy

Q-learning aprende sobre a política ótima independente da política adotada. Por esta razão, Q-learning é chamado Off-Policy, isso significa que o algoritmo aprende sobre uma política seguindo outra política [52].

2.6 Sarsa

O algoritmo Sarsa utiliza o quíntuplo (s_t, a_t, r_t, s_{t+1}, a_{t+1}), o qual advém o nome *Sarsa*. A regra de atualização baseada em Sarsa é semelhante à regra de atualização de Q-learning:

$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha[r + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)] \tag{2.18}$$

Diferente do algoritmo Q-Learning que aprende os valores da política ótima, Sarsa aprende os valores da política atual, sendo denominado On-Policy, o que significa que a maior recompensa esperada não é necessariamente selecionada para a atualização da função valor estado-ação $Q(s,a)$. O algoritmo procedural para o Sarsa pode ser visto a seguir:

```
Inicialize Q(s,a) arbitrariamente ;
para todo episodio faça
    Inicialize s (estado inicial) ;
    a ← ação para s utilizando política derivada de Q ;
    repita
        Tome ação a, observe a recompensa r e o próximo estado s' ;
        a' ← ação para s' utilizando política derivada de Q ;
        Q(s,a) ← Q(s,a) + α[r + γQ(s',a') − Q(s,a)] ;
        s ← s'; a ← a';
    até s ser terminal;
fim
```

Algoritmo 7: Sarsa – Controle On-Policy

2.7 Traço de Elegibilidade - λ

Introduzido por Klopf [23], os Traço de Elegibilidade foram usados em uma variedade de sistemas de AR [25, 35, 54]. A ideia do Traço de Elegibilidade é memorizar em um curto tempo os estados visitados, ou seja, um traço, o qual, decai gradualmente ao longo do tempo, marcando o estado como elegível para o aprendizado. Durante

a elegibilidade, se um evento inesperado ocorrer o estado terá um valor atribuído de acordo com a política adotada. Existem dois tipos de traços - o acumulativo e o de substituição. O acumulativo soma um ao traço elegível a cada vez que o estado for visitado, acumulando os valores existentes. O traço por substituição atribui o valor de um para o estado elegível cada vez que for visitado, independente da presença de um traço prévio. A Figura 2.3 ilustra as diferenças entre os traços elegíveis.

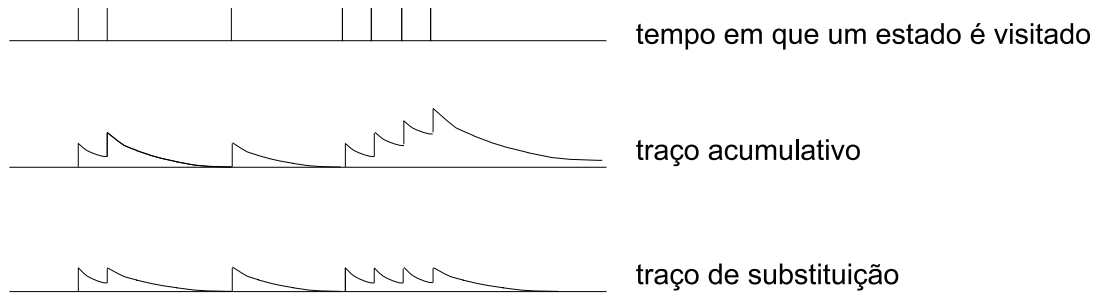

Figura 2.3: Traço de Elegibilidade: acumulativo e por substituição

Geralmente, os Traço de Elegibilidade decaem exponencialmente com o produto entre o parâmetro de decaimento λ, e a taxa de desconto γ. O acúmulo de traços é definido por,

$$e_{t+1}(s) = \begin{cases} \gamma\lambda e_t(s), & s \neq s_t; \\ \gamma\lambda e_t(s) + 1, & s = s_t. \end{cases} \tag{2.19}$$

onde, e_t representa o Traço de Elegibilidade para o estado s no tempo t, e s_t é o estado atual no tempo t.

O traço por substituição é definido por,

$$e_{t+1}(s) = \begin{cases} \gamma\lambda e_t(s), & s \neq s_t; \\ 1, & s = s_t. \end{cases} \tag{2.20}$$

Nos problemas de controle, cada par estado-ação tem um traço distinto. Quando um estado é visitado e uma ação é tomada, apenas no traço do estado para aquela ação é reposto a 1, enquanto os demais traços decaem exponencialmente até zero.

Os métodos de controle Off-Policy, Seção 2.5, e On-Policy, Seção 2.6, sofreram modificações, as quais, insere o Traço de Elegibilidade dando origem aos métodos Q-Learning(λ) e Sarsa(λ), respectivamente.

2.7.1 Controle Off-Policy: Q-Learning (λ)

O algoritmo Q-Learning(λ) é formado pela inserção do Traço de Elegiblidade no algoritmo Q-learning. O traço $e_t(s,a)$ registra a passagem do agente pelo par estado-ação (s,a) no instante t. A regra de atualização da função valor Q para o algoritmo 8 é dada por,

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a), \quad \forall\ s,\ a \tag{2.21}$$

onde,

$$\begin{aligned} \delta_t &= r_{t+1} + \gamma Q_t(s_{t+1}, a^*_{t+1}) - Q_t(s_t, a_t) \\ a^* &= arg \max_a Q(s,a) \end{aligned} \tag{2.22}$$

e,

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1, & \text{se } s = s_t, \text{ e } a = a_t; \\ \gamma\lambda e_{t-1}(s,a), & \text{caso contrário.} \end{cases} \tag{2.23}$$

A atualização do Traço de Elegibilidade no algoritmo Q-Learning (λ) é realizada para os pares estado-ação recém-visitados, exceto os pares estado-ação que possuem a

ação $a' \neq a^*$. Conforme pode ser visto no algoritmo 8, os Traço de Elegibilidade que possuem as ações $a' \neq a^*$ são setados para zero ($e_t(s,a) \leftarrow 0$).

```
Inicialize Q(s,a) arbitrariamente e e(s,a) = 0, ∀s, a ;
para todo episodio faça
    Inicialize s (estado inicial), a (ação inicial) ;
    repita
        Tome ação a, observe a recompensa r e o próximo estado s' ;
        a ← ação para s utilizando política derivada de Q ;
        a* ← arg max_b Q(s,b) ;
        δ ← r + γQ(s',a*) − Q(s,a) ;
        e(s,a) ← e(s,a) + 1 ;
        para todo s, a faça
            Q(s,a) ← Q(s,a) + αδe(s,a) ;
            se a' = a* então
                e(s,a) ← γλe(s,a) ;
                senão
                    e(s,a) ← 0;
                fim
            fim
        fim
        s ← s'
        a ← a'
    até s ser terminal;
fim
```

Algoritmo 8: Q-Learning(λ) – Controle Off-Policy com Traço de Elegibilidade Acumulativo

2.7.2 Controle On-Policy: Sarsa (λ)

O Traço de Elegibilidade pode ser combinado diretamente com o algoritmo apresentado na Seção 2.6, resultando no algoritmo denominado Sarsa(λ) [50]. O traço é direcionado a cada par estado-ação, denotado por $e_t(s,a)$ no instante t. A regra de atualização é dada por,

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a), \;\; \forall\, s,\; a \tag{2.24}$$

onde,

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \tag{2.25}$$

e,

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1, & \text{se } s = s_t, \text{ e } a = a_t; \\ \gamma\lambda e_{t-1}(s,a), & \text{caso contrário.} \end{cases} \tag{2.26}$$

O algoritmo procedural para o Sarsa(λ) pode ser visto a seguir:

```
Inicialize Q(s,a) arbitrariamente e e(s,a) = 0, ∀s, a ;
para todo episodio faça
    Inicialize s (estado inicial) ;
    a ← ação para s utilizando política derivada de Q ;
    repita
        Tome ação a, observe a recompensa r e o próximo estado s' ;
        δ ← r + γQ(s',a') − Q(s,a) ;
        e(s,a) ← e(s,a) + 1 ;
        para todo s, a faça
            Q(s,a) ← Q(s,a) + αδe(s,a) ;
            e(s,a) ← γλe(s,a) ;
        fim
        s ← s'
        a ← a'
    até s ser terminal;
fim
```

Algoritmo 9: Sarsa(λ) – Controle On-Policy com Traço de Elegibilidade Acumulativo

No algoritmo Sarsa(λ), a atualização da função $Q(s,a)$ é realizada para todos os estados recém-visitados, de acordo com a taxa de decaimento, ou seja, por meio do Traço de Elegibilidade que é corrigido pelo produto $\gamma\lambda$ a cada passo do episódio.

2.8 Generalização de Funções

Os sistemas modelados por meio de variáveis de estados discretas são facilmente implementados por meio de representações da função valor estado, $V(s)$, ou função valor estado-ação, $Q(s,a)$, na forma tabular.

No entanto, a escolha de uma discretização ruim do espaço de estados pode introduzir estados ocultos no problema, tornando impossível o aprendizado da política ótima [17]. Já uma discretização dos estados detalhada perde a capacidade de generalizar, aumentando a quantidade de dados para o aprendizado. Como a maioria dos problemas reais é constituída de variáveis de estados contínuas uma enorme quantidade de memória é exigida para armazenar as tabelas e garantir a precisão. Assim, substituir a discretização de estados por uma representação reduzida, por meio da generalização dos estados, tem sido uma alternativa encontrada capaz de lidar com variáveis contínuas em várias dimensões [41]. Por isso, as técnicas de generalização de função têm sido exaustivamente estudadas e combinadas com AR [8].

Por meio da generalização de função é possível visitar um subconjunto limitado do espaço de estados e produzir uma boa aproximação de todo o espaço [52]. Várias abordagens como, Tile Coding [3, 40], também conhecido como CMAC (*Cerebellar Model Articulation Controller*), Árvores de decisão [11], Redes Neurais [10], o Kanerva Coding [24], a Função Radial de Base (*Radial Basis Functions - RBFs*) [38] e resolução de variáveis por meio de PD [27] têm sido utilizadas com sucesso na aproximação de funções.

2.8.1 Tile Coding

O Tile Coding é uma forma de representação grosseira [52, 48] que possui as características agrupadas em partições do espaço de estados. Cada partição é chamada de *tiling*, e cada elemento dessa partição é chamado de *tile*, no qual, representa uma característica binária [17]. Os *tilings* são sobrepostos e deslocados uns dos outros em uma distância

(resolução) r. O número de tilings, t, a largura dos tiles, w, e o deslocamento de cada tiling (r) estão relacionados diretamente, conforme a expressão 2.27[26]:

$$r = w/t. \tag{2.27}$$

Na Figura 2.4, apresenta-se um exemplo de Tile Coding, no qual, representa um espaço de estados unidimensional, S, em que quatro *tilings* foram sobrepostos.

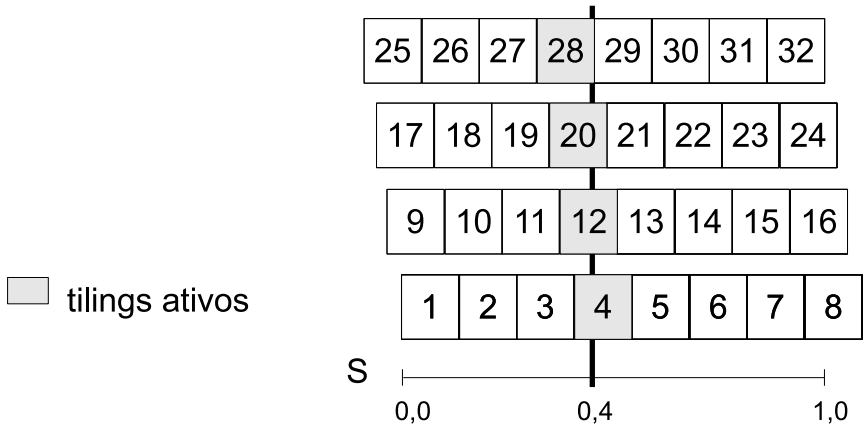

Figura 2.4: Tile Coding Unidimensional - Generalização do espaço de estados utilizando 4 tilings, totalizando 32 tiles. O vetor de características binário para $s = 0{,}40$ é dado por $\phi(s) = \{0_0, ..., 1_4, 0_5, ..., 1_{12}, 0_{13}, ..., 1_{20}, 0_{21}, ..., 1_{28}, 0_{29}, ..., 0_{32}\}$. Os tiles ativos estão representados pela cor cinza.

O conjunto de todos os tiles ativos, um por *tiling* e quatro por estado, é representado pelo vetor de característica binário, $\phi(s)$, o qual, pode ser definido por meio da expressão 2.28.

$$\phi(s) = \begin{cases} 0 & se \quad s \notin tile_t, \\ 1 & se \quad s \in tile_t \end{cases} \tag{2.28}$$

A combinação linear utilizando a função de mapeamento $\phi(S)$ transforma o estado s em um vetor com N características binárias $[\phi_1, \phi_2, ..., \phi_N]^T$. Assim, a função $Q_a(s)$ é obtida por meio do produto escalar $\phi(s) \times \vec{\theta}$, onde $\vec{\theta}$ é um vetor de dimensão igual a N

(Equação 2.29).

$$
\begin{aligned}
Q_a(s) &= \vec{\theta_t^T}\vec{\phi_s} \\
Q_a(s) &= \sum_{i=1}^{N} \theta_t(i)\phi_s(i).
\end{aligned} \tag{2.29}
$$

A generalização da função Q_a ocorre com a sobreposição dos tilings e, para cada tiling são armazenadas as variáveis que indicam o peso de cada tile. Na equação 2.29, o vetor $\vec{\theta_t^T}$ representa o conjunto de pesos de cada tile, e $\vec{\phi_s}$ o vetor binário com os tiles ativos.

Os algoritmos Q-Learning(λ) e Sarsa(λ) quando combinados com a generalização de funções dão origem aos algoritmos de controle Off-Policy (10) e On-Policy (11) . Nestes algoritmos, o vetor de características presentes em (s, a), $\vec{\phi_s}$, é representado pelo vetor binário F_a. A cada episódio, a função Q é obtida por meio da função 2.29, a qual é representada por $Q_a \leftarrow \sum_{i \in F_a} \theta(i)$ nos algoritmos. A regra de atualização do vetor de características (ou, pesos), $\vec{\theta}$, é dada pela expressão $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$, a qual, considera a taxa de aprendizado e o traço de elegibilidade.

2.9 Robocup - Simulação 2D

A categoria de simulação 2D da RoboCup foi desenvolvida pelo Dr. Itsuki Noda em 1993 [9]. A simulação 2D é constituída por 11 jogadores virtuais criados por meio de softwares. A simulação de uma partida é feita em um servidor denominado *Soccer Server* [31]. Este servidor foi desenvolvido para ser um ambiente de simulação multiagentes em tempo real, sendo este um sistema que simula a complexidade de uma partida de futebol com a adição de ruído [32]. A arquitetura do futebol de robôs 2D pode ser dividida em cliente (Client), servidor (Soccer Server) e monitor (Soccer Monitor), como ilustrado na Figura 2.5.

Uma partida realizada no simulador tem uma duração de 6.000 ciclos, ou seja, 10

```
Inicialize θ⃗ arbitrariamente ;
para todo episodio faça
    Inicialize s (estado inicial), a (ação inicial), e, e⃗ = 0⃗ ;
    F_a ← características presentes em s, a ;
    repita
        para todo i ∈ F_a faça
            e(i) ← e(i) + 1 ;
        fim
        Tome ação a, observe a recompensa r e o próximo estado s' ;
        δ ← r − Σ_{i∈F_a} θ(i) ;
        para todo a ∈ A(s) faça
            F_a ← características presentes em s, a ;
            Q_a ← Σ_{i∈F_a} θ(i) ;
        fim
        δ ← δ + γ max_a Q_a ;
        θ⃗ ← θ⃗ + αδe⃗ ;
        se (1 − ε) > rand[0,1] então
            para todo a ∈ A(s) faça
                Q_a ← Σ_{i∈F_a} θ(i) ;
            fim
            a ← arg max_a Q_a ;
            e⃗ ← γλe⃗ ;
            senão
                a ← a' (escolha aleatória de a ∈ A) ;
                e⃗ ← 0⃗ ;
            fim
        fim
    até s ser terminal;
fim
```

Algoritmo 10: Q-Learning(λ) com função de generalização, política gulosa (ε), e traço acumulativo.

```
Inicialize θ⃗ arbitrariamente ;
para todo episodio faça
    Inicialize s (estado inicial), a (ação inicial), e e⃗ = 0⃗ ;
    F_a ← características presentes em s, a ;
    repita
        para todo i ∈ F_a faça
            e(i) ← e(i) + 1 ;
        fim
        Tome ação a, observe a recompensa r e o próximo estado s' ;
        δ ← r − Σ_{i∈F_a} θ(i) ;
        se (1 − ε) > rand[0,1] então
            para todo a ∈ A(s) faça
                F_a ← características presentes em s, a ;
                Q_a ← Σ_{i∈F_a} θ(i) ;
            fim
            a ← arg max_a Q_a ;
            senão
                a ← a' (escolha aleatória de a ∈ A) ;
                F_a ← características presentes em s, a ;
                Q_a ← Σ_{i∈F_a} θ(i) ;
            fim
        fim
        δ ← δ + γQ_a ;
        θ⃗ ← θ⃗ + αδe⃗ ;
        e⃗ ← γλe⃗ ;
    até s ser terminal;
fim
```

Algoritmo 11: Sarsa(λ) com função de generalização, política gulosa (ε), e traço acumulativo.

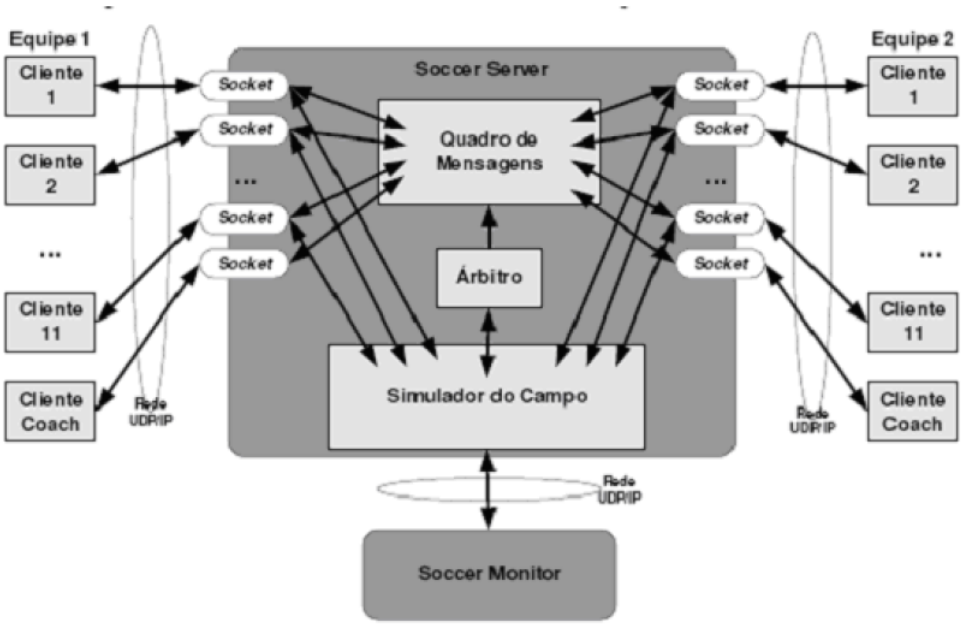

Figura 2.5: Arquitetura do Sistema de Simulação [21].

minutos de simulação. Da mesma forma que uma partida real de futebol, a simulação é dividida em dois tempos de 5 minutos. Em caso de empate, um tempo extra de 3.000 ciclos é adicionado ao tempo normal. Além disso, o *gol de ouro* pode ser definido dando a vitória ao time que marcar o primeiro gol durante a prorrogação.

2.9.1 Cliente

A RoboCup 2D permite a competição entre os dois times de jogadores virtuais, os quais, são vistos como clientes únicos e autônomos, enviando e recebendo informações do servidor. As informações processadas pelo servidor são usadas para a atualização contínua do estado da partida até o fim do jogo. Toda comunicação entre o servidor e os clientes é feita por meio de sockets TCP/IP.

2.9.2 Monitor

As partidas podem ser observadas por meio do *Soccer Monitor*. O Soccer Monitor é responsável pela interface gráfica da RoboCup 2D usada para a visualização em tempo

real dos jogos. Porém, não existe a necessidade de um monitor para executar uma partida, sendo necessária a presença apenas do servidor e do cliente para execução das simulações.

2.9.3 Servidor

O servidor é o responsável por processar todas as ações requeridas pelos clientes. O Soccer Server processa e atualiza o estado da partida e devolve aos clientes as informações atualizadas. Todo o sistema é atualizado em tempo real em intervalos de tempos discretos, denominados ciclos, com duração aproximada de 100ms por ciclo.

2.9.4 Treinador

Na RoboCup 2D, dois tipos de treinadores podem ser aplicados ao time: off-line e on-line. O modo off-line, geralmente, é usado para o treinamento, pois, o treinador tem um controle quase total dos jogadores e do jogo, podendo mudar a posição e a velocidade da bola e dos jogadores. No modo on-line, o treinador pode apenas comunicar-se com os jogadores em campo. E em ambos os modos, o treinador pode receber as informações globais sem ruído e usá-las durante o jogo.

Capítulo 3

Metodologia

3.1 Cenário Proposto

O ambiente de simulação de futebol de robôs 2D da Robocup foi utilizado na realização dos experimentos. O simulador 2D possui um modo exclusivo para o treinamento dos jogadores, o qual, é denominado de *Trainer*. Neste modo foi possível simular a condição de ataque e defesa entre dois times, sendo o primeiro definido pela cor vermelha e formado por um goleiro e dois zagueiros, e o segundo definido pela cor azul constituído por dois atacantes (Figura 3.1).

Normalmente, os agentes possuem sensores limitados e com ruídos, ou seja, cada jogador pode ver apenas os objetos que estiverem dentro de uma área com um formato de um cone com 90 graus. Além disso, a precisão da localização de um objeto dentro dessa área diminui com a distância. Entretanto, neste trabalho, os agentes receberam 360 graus de visão para garantir a eles o conhecimento completo do mundo [48].

Optou-se em limitar a posição dos jogadores entre os intervalos $25 \leq X \leq 52$ e $-32 \leq Y \leq 32$, conforme ilustrado na Figura 3.1. Para cada simulação foram realizadas 140 episódios, sendo cada episódio composto por 12 tentativas a gols, totalizando 1680 tentativas a gols. A determinação do número de tentativas a gols por episódio foi feita com base na metodologia proposta por Takahashi [53]. O início de cada tentativa ocorreu

com o sorteio aleatório dos jogadores e o fim foi definido pelas seguintes situações: a posse de bola por um dos jogadores do time defensor (cor vermelha); após chute a gol; e, a saída da bola de dentro da área limitada. A Figura 3.2 mostra o fluxograma utilizado para a elaboração das rotinas computacionais utilizadas.

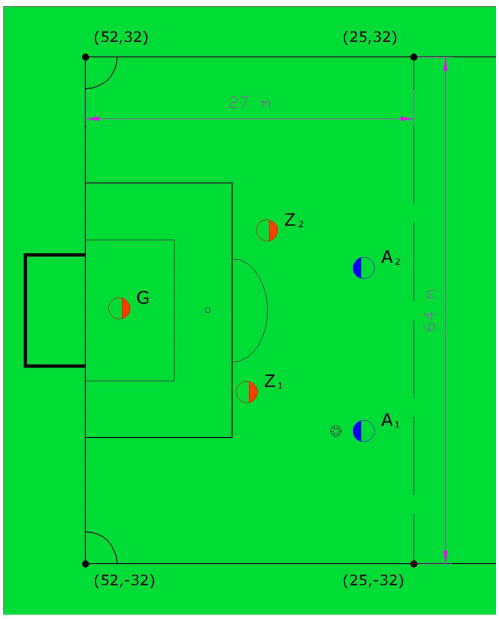

Figura 3.1: Representação do Cenário - Os três defensores foram denominados de G, Z_1 e Z_2 (círculo da cor vermelha). Os atacantes foram chamados de A_1 e A_2 (círculo da cor azul). As tomadas de decisão utilizando o aprendizado por reforço foram feitas apenas pelos atacantes.

Após o início da tentativa, o contador de tentativas (TRAIN) é acrescido de um. Em seguida, os jogadores e a bola são reposicionados. A posição da bola é verificada, e caso ela não esteja na área limite pré-determinada, a tentativa chega ao fim, dando início a outra tentativa. Se a ação de *chute a gol* for escolhida, e posteriormente, for feito um gol é dado o início de outra tentativa. Outra situação que acarreta o fim da tentativa é a posse de bola pelos defensores.

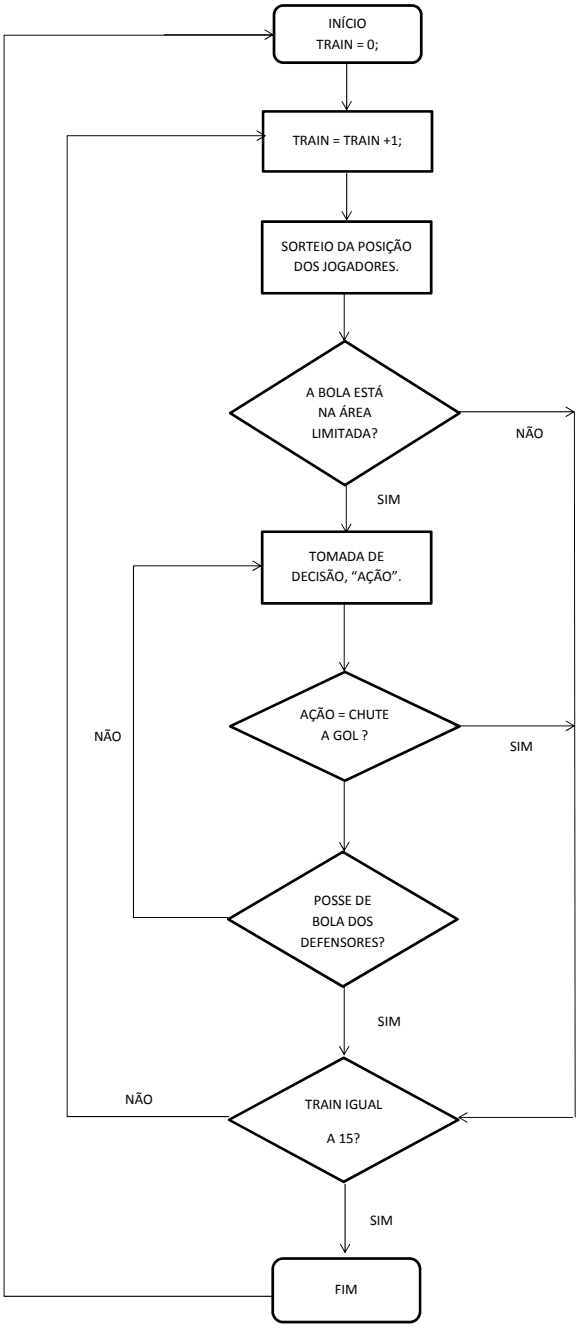

Figura 3.2: Fluxograma da rotina computacional

3.2 O Modelo para o Aprendizado

Neste trabalho foram propostos dois modelos, denominados, para este estudo, de modelo discreto e modelo contínuo. Nos próximos tópicos serão apresentados conceitos como: estados, ações e recompensas para a modelagem do problema na forma discreta e contínua.

3.2.1 Modelo Discreto

Estados

As informações escolhidas para compor o espaço de estados foram a posição do agente, e a distância D entre o agente com a posse de bola e o adversário mais próximo (ver Figura 3.7). A posição do agente foi definida utilizando as subáreas, AA_1, AA_2, ..., AA_{12}, como mostra a Figura 3.3. A variável D pertence ao intervalo $[0, 20m[$, a qual, foi discretizada por meio das variáveis d_1 e d_2, conforme pode ser visto no sistema 3.1.

$$D = \begin{cases} 0 \leq & d_1 & < 10, \\ 10 \leq & d_2 & < 20. \end{cases} \tag{3.1}$$

Os estados s_i foram formados por uma combinação entre as subáreas, AA_i ($i = 1, 2, ..., 12$) e as variáveis d_j ($j = 1, 2$), como mostra a Figura 3.5. Neste trabalho, o modelo proposto, por meio de estados discretos, foi formado por vinte e quatro ($12x2 = 24$) estados e quatro ações.

Ações

As tomadas de decisões devem ser feitas por meio de ações pré-determinadas, as quais, fazem parte de um conjunto A de ações. Por definição, a ação a_i deverá pertencer ao conjunto A de ações para que a_i seja tomada. No modelo de aprendizagem proposto, as ações contidas em A foram definidas como macro-ações (Tabela 3.1). Uma macro-ação representa um conjunto de ações básicas conforme descritas na Tabela 3.2.

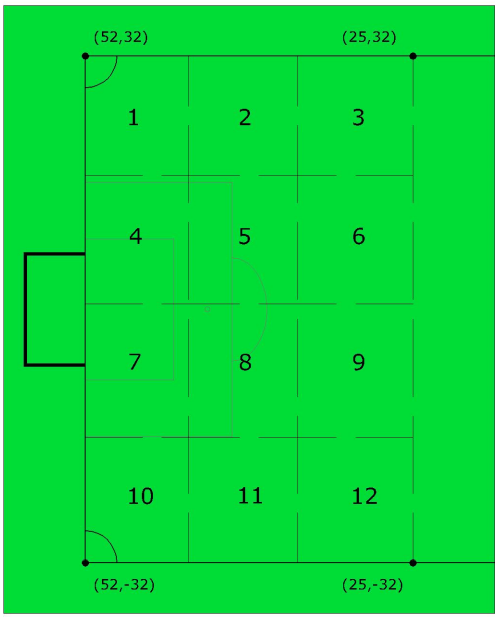

Figura 3.3: Representação das variáveis de estado - A área limite possui dimensão igual a 52×64 (comprimento $\times$ largura) unidades, na qual, foi dividido em doze subáreas.

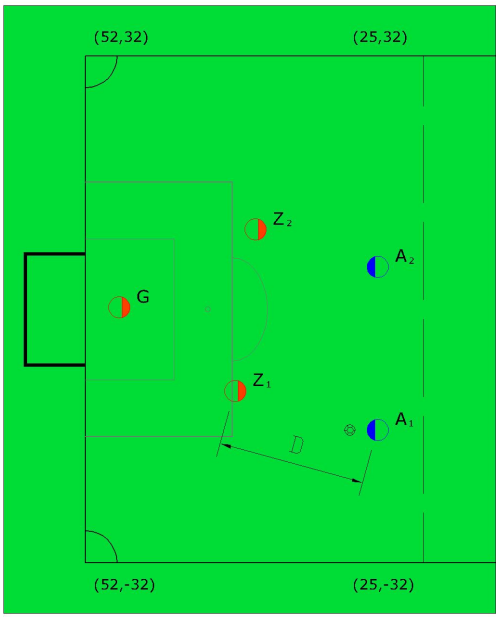

Figura 3.4: Representação das variáveis de estado - A variável D representa a distância entre o atacante com posse de bola e o defensor mais próximo.

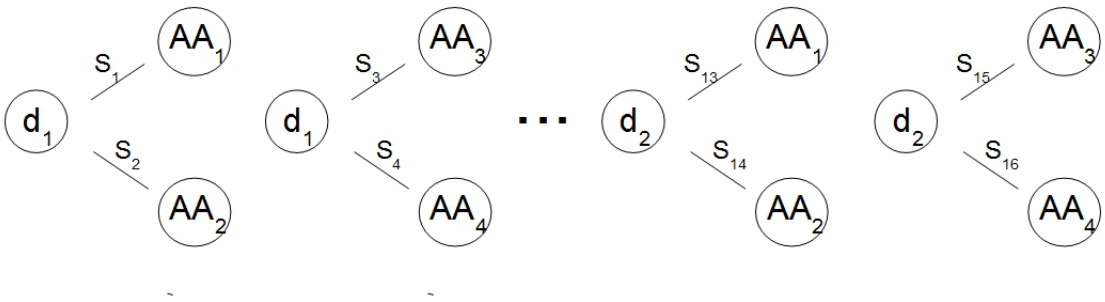

Figura 3.5: Representação dos estados discretos - Os estados foram formados pela posição do agente dentro da área limite e a distância D. A combinação das subáreas $AA_1 ... AA_{12}$ com os intervalos D_1 e D_2 formaram os 24 estados do sistema, por exemplo, $S_1 = [AA_1, d_1]$.

Tabela 3.1: Tabela de Macro-Ações formadas por ações básicas

Macro-Ação	**Descrição**	**Ações Básicas**
a_1 - Segurar a bola	Ficar parado com a bola	b_1
a_2 - Chute a gol	Chutar a bola em direção ao gol	b_2, b_3 e b_5
a_3 - Passe	Tocar a bola para o outro jogador	b_3 e b_4
a_4 - Carregar a bola	Deslocar com a bola	b_2, b_3 e b_4

As macro-ações ajudam a diminuir o número total de ações necessárias para a execução de uma mesma tarefa, diminuindo o número de ações ou pares estado-ação [21].

Tabela 3.2: Tabela de Ações Básicas

Ação Básica	**Descrição**
b_1 - Parar	Parar junto à bola.
b_2 - Correr	Correr com uma velocidade V.
b_3 - Definir Ponto	Definir um ponto para passar a bola ou chutar.
b_4 - Chute	Chutar a bola com uma força F.
b_5 - Chute Forte	Chutar a bola com uma força máxima.

Recompensas

A decisão tomada pelo agente ao escolher a ação $a \in A$, no estado $s \in S$ resulta no ganho de uma recompensa r. No caso discreto, as recompensas r_t são dadas em função do par estado-ação atual (s_t, a_t) do agente.

A função de recompensa foi utilizada na forma tabular, a qual, é denominada de matriz de recompensa R. Os elementos da matriz R, $r_t(s,a)$, informam o valor da recompensa para cada par estado-ação. A matriz R, 24×4, teve seus elementos todos iguais a -1, exceto, nos casos em que a ação terminou em gol, $r_t(s,a) = 50$, ou, com a perda da bola, $r_t(s,a) = -10$.

Função Valor - $Q(s,a)$

A função valor estado-ação Q foi estimada segundo os algoritmos Sarsa(λ)(2.24) e Q-Learning(λ)(2.24). No caso discreto, a função Q foi representada na forma tabular,

conforme Tabela 3.3. Cada elemento da matriz, $Q(s,a)$, representa o valor da função para cada par estado-ação. No início de cada simulação, o valor dos elementos $Q(s,a)$ foram iguais a 500 unidades. A Matriz Função Valor Q possui dimensão igual a 24×4, totalizando 96 elementos.

Tabela 3.3: Função valor estado-ação $Q(s,a)$ na forma tabular. Conforme os pares estado-ação foram visitados, a função $Q(s,a)$ foi atualizada.

Estado/Ação	a_1	a_2	a_3	a_4
s_1	500	500	500	500
s_2	500	500	500	500
s_3	500	500	500	500
s_4	500	500	500	500
s_5	500	500	500	500
s_6	500	500	500	500
s_7	500	500	500	500
s_8	500	500	500	500
s_9	500	500	500	500
s_{10}	500	500	500	500
s_{11}	500	500	500	500
s_{12}	500	500	500	500
s_{13}	500	500	500	500
s_{14}	500	500	500	500
s_{15}	500	500	500	500
s_{16}	500	500	500	500
s_{17}	500	500	500	500
s_{18}	500	500	500	500
s_{19}	500	500	500	500
s_{20}	500	500	500	500
s_{21}	500	500	500	500
s_{22}	500	500	500	500
s_{23}	500	500	500	500
s_{24}	500	500	500	500

3.2.2 Modelo Contínuo

Estados

As variáveis de estado contínuas foram descritas em relação à posição do agente, por meio da coordenada X e Y, e a variável D que representa a distância entre o agente com posse de bola e o adversário mais próximo. As Figuras 3.6 e 3.7 ilustram as variáveis de estados X, Y e D, as quais, são definidas como:

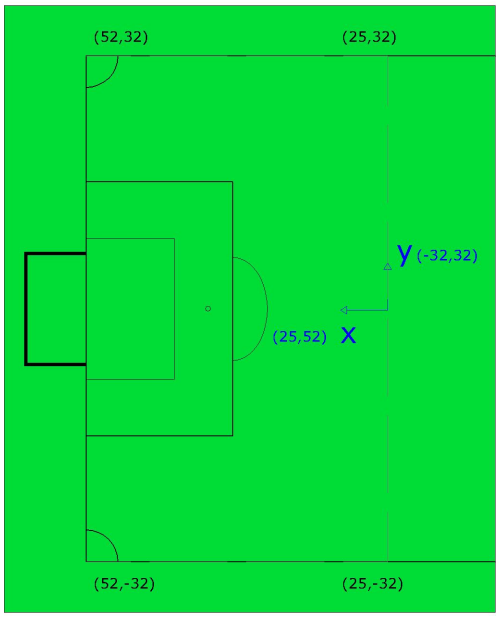

Figura 3.6: Representação das variáveis de estado contínuas X e Y.

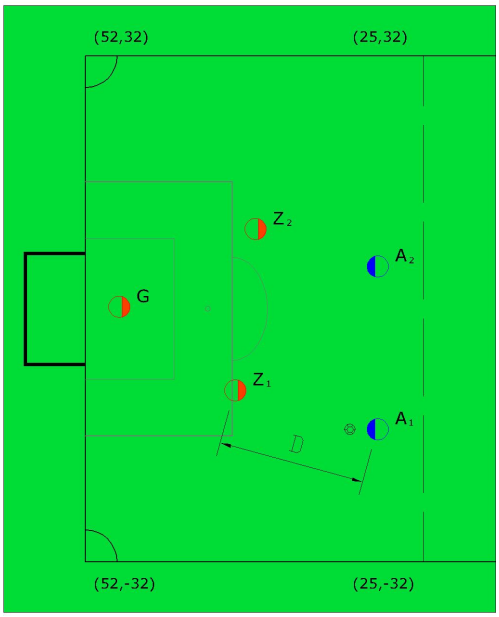

Figura 3.7: Representação da variável estado contínua D.

$$S = \begin{cases} 25 \leq & X & \leq 52, \\ -32 \leq & Y & \leq 32, \\ 0 \leq & D & \leq 20. \end{cases} \tag{3.2}$$

Ações

No modelo contínuo, as ações foram discretizadas, pré-determinadas e utilizadas da mesma maneira que no modelo discreto. No modelo contínuo, as ações foram definidas como macro-ações descritas na Tabela 3.1.

Recompensas

No modelo contínuo, as recompensas não foram utilizadas na forma tabular, sendo feitas por meio de uma lógica simples, a qual, dependia do gol marcado e a posse de bola. As recompensas foram iguais a -1 para as ações que não resultaram em gol. No caso de gol, a recompensa foi igual a 50. Quando ocorreu a perda de bola para o adversário, o agente recebeu uma recompensa igual a -10.

Função Valor - $Q_a(s)$

A função valor estado-ação $Q_a(s)$ foi estimada utilizando os algoritmos Q-Learning(λ) e Sarsa(λ) com generalização de função (Seção 2.8.1). Neste trabalho, os algoritmos combinados com a generalização de função por meio do Tile Coding são denominados de Q-Learning(λ) Tile e Sarsa(λ) Tile.

3.3 Experimentos

Todos os experimentos foram realizados com os seguintes parâmetros: $\gamma = 0{,}9$, $\alpha = 0{,}125$ [48], e taxa de exploração igual a 0,05. Os parâmetros γ e exploração são os mesmos utilizados por alguns autores [21, 8, 33]. Para o modelo contínuo, foram realizadas simulações variando o número de tilings entre 1 e 64, e o número de tiles igual a 4.

Quando necessário foi feita a comparação entre os algoritmos para os valores médios de número de gols acumulados por episódio pelo teste t-student ($p = 1\%$ e $p = 5\%$) [46].

Os experimentos foram codificados em Linguagem C++ (compilador GNU g++), baseado no código fonte disponível por Hidehisa Akiyama [1], e executados em um Microcomputador Intel Core i5-2430M @ 2,40 GHz, com 4,0 Gb de memória RAM e sistema operacional Linux.

3.3.1 Modelo Discreto

O método descrito na Seção 3.2.1 foi utilizado nos experimentos 01 e 02, os quais, tiveram os algoritmos Sarsa (λ) e Q-Learning (λ) implementados. Nestes experimentos, foram realizadas simulações para $\lambda = 0{,}0;\ 0{,}1;\ \ldots\ ; 0{,}7$. O número total de repetições foi igual a 10 (dez), totalizando 80 (oitenta) simulações ($8x10$) para cada experimento.

Nos experimentos 01 e 02, avaliou-se a aprendizagem dos agentes em função da variação do Traço de Elegibilidade utilizando o algoritmo Sarsa (λ) e Q-Learning (λ), descrito na Seção 2.7.2, respectivamente.

3.3.2 Modelo Contínuo

O modelo contínuo descrito na Seção 3.2.2 foi utilizado nos experimentos 03, 04, 05 e 06, os quais, tiveram os algoritmos Sarsa (λ)-Tile e Q-Learning (λ)-Tile implementados.

Nos experimentos 03 e 04 foi avaliada a influência do número de tilings necessário para a generalização da função valor $Q_a(s)$. Nestes experimentos, todos os parâmentros se mantiveram fixos, exceto, o número de tilings. O número total de repetições foi igual a 10 (dez), totalizando 60 (sessenta) simulações ($6x10$) para cada experimento.

Para o experimento 03, variou-se o número de tilings para a generalização da função $Q_a(s)$. As simulações foram realizadas para 2^n tilings, no qual, $n \in \mathbb{N} \subset [0{,}5]$. O algoritmo utilizado foi o Sarsa (λ)-Tile, descrito na Seção 2.8.1.

No experimento 04, o comportamento dos agentes foi observado utilizando o algoritmo Q-Learning (λ)-Tile, descrito na Seção 2.8.1, variando o número de tilings. As simulações foram realizadas para 2^n tilings, no qual, $n \in \mathbb{N} \subset [0{,}5]$.

Nos experimentos 05 e 06, foram realizadas simulações para $\lambda = 0{,}0;\ 0{,}1;\ \ldots\ ; 0{,}7$. O número total de repetições foi igual a 10 (dez), totalizando 80 (oitenta) simulações ($8x10$) para cada experimento. O número de tilings foi determinado a partir dos experimentos 03 e 04, mantendo fixos durante todo o experimento. Foi avaliada a aprendizagem dos agentes em função da variação do Traço de Elegibilidade utilizando os

algoritmo Sarsa (λ) Tile e Q-Learning (λ) Tile.

Capítulo 4

Resultados

4.1 Modelo Discreto

Inicialmente, o número de pares estado-ação visitados foi observado na execução do algoritmo Sarsa(λ) e Q-learning(λ). Verificou-se que todos os pares foram visitados, e que existiu uma maior concentração de visitas a partir do estado 13 em ambos os algoritmos (Figuras 4.1 e 4.2). A ocorrência das visitas aos estados segue uma das premissas para o aprendizado, a qual, determina que todos os pares estado-ação devem ser visitados infinitas vezes [57].

4.1.1 Experimento 01: Sarsa(λ)

Após a determinação do número de tentativas por episódio, foram realizadas as simulações, nas quais, foram obtidos os resultados da média da diferença temporal ($Q_{t-1} - Q_t$) da matriz $Q(s,a)$ para os diferentes valores do parâmetro λ (Figura 4.3). Como ilustrado, o comportamento das curvas foi semelhante para os parâmetros $\lambda's$, diferindo apenas no tempo de decaimento da diferença $Q_{t-1} - Q_t$. As curvas para $\lambda = 0{,}1$ e $\lambda = 0{,}2$ apresentaram o maior e o menor tempo, respectivamente, para o decaimento da curva até o ponto de equilíbrio. O número de episódios para que a diferença temporal

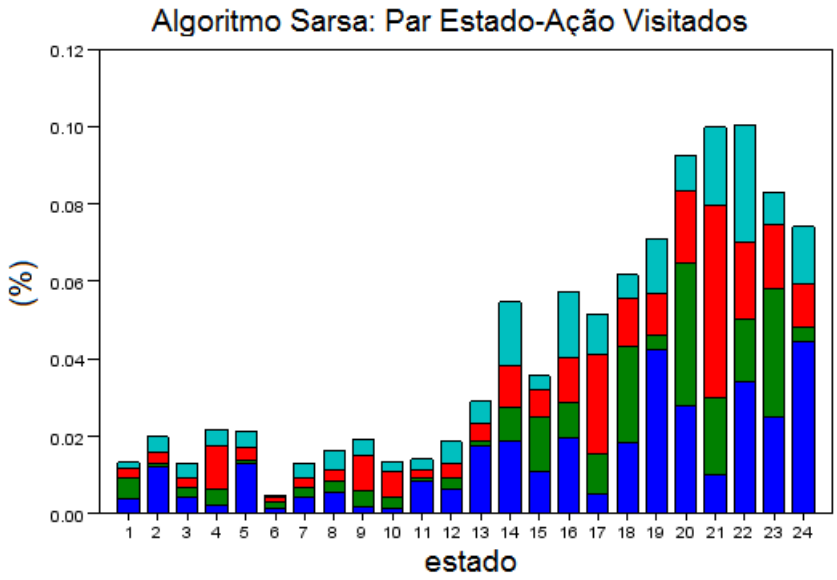

Figura 4.1: Histograma dos pares estado-ação visitados utilizando o algoritmo Sarsa(λ). As barras possuem cores diferentes, as quais, representam as ações a_1 (cor azul), a_2 (cor verde), a_3 (cor vermelha) e a_4 (cor azul claro).

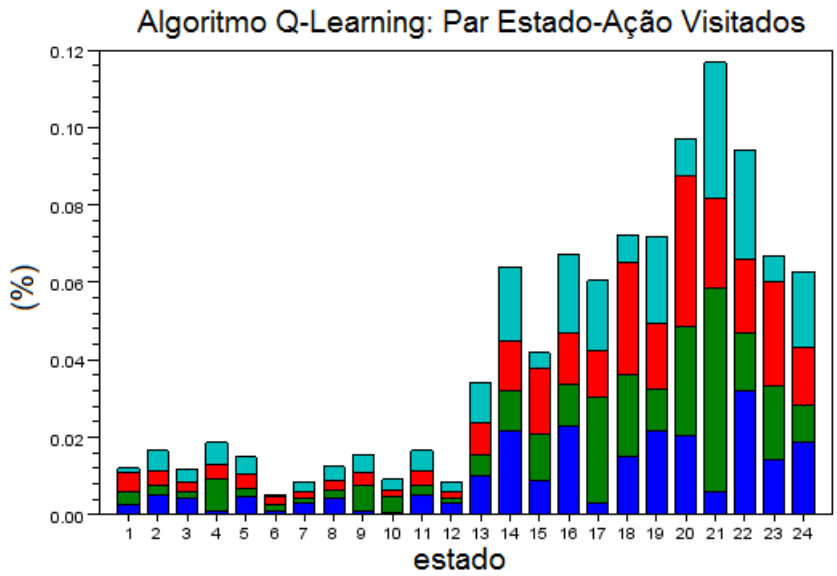

Figura 4.2: Histograma dos pares estado-ação visitados utilizando o algoritmo Q-Learning(λ). As barras possuem cores diferentes, as quais, representam as ações a_1 (cor azul), a_2 (cor verde), a_3 (cor vermelha) e a_4 (cor azul claro).

estabilizasse em torno do ponto de equilíbrio foi igual a 118 e 92 para $\lambda = 0{,}1$ e $\lambda = 0{,}2$, respectivamente (Tabela 4.1).

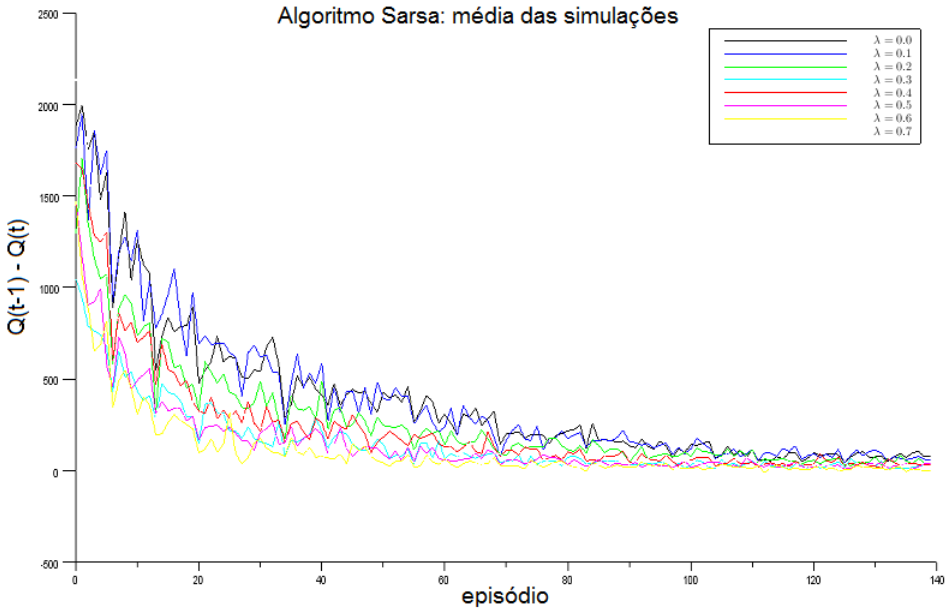

Figura 4.3: Sarsa(λ) - Média da diferença temporal da matriz $Q(s,a)$ para os $\lambda's$ simulados.

Tabela 4.1: Sarsa(λ) - Tempo de decaimento (episódio).

λ	**Tempo de decaimento (episódio)**
0,0	116
0,1	118
0,2	92
0,3	101
0,4	106
0,5	113
0,6	104
0,7	103

Ao analisar a média de gols acumulados para os diversos valores de λ verificou-se que todas as curvas foram crescentes durante os 140 episódios (Figura 4.4), apesar de apresentarem saldos de gols diferentes (Tabela 4.2), sendo que o maior (241,60 gols) foi para $\lambda = 0{,}3$ e o menor (198,5 gols) para $\lambda = 0{,}2$.

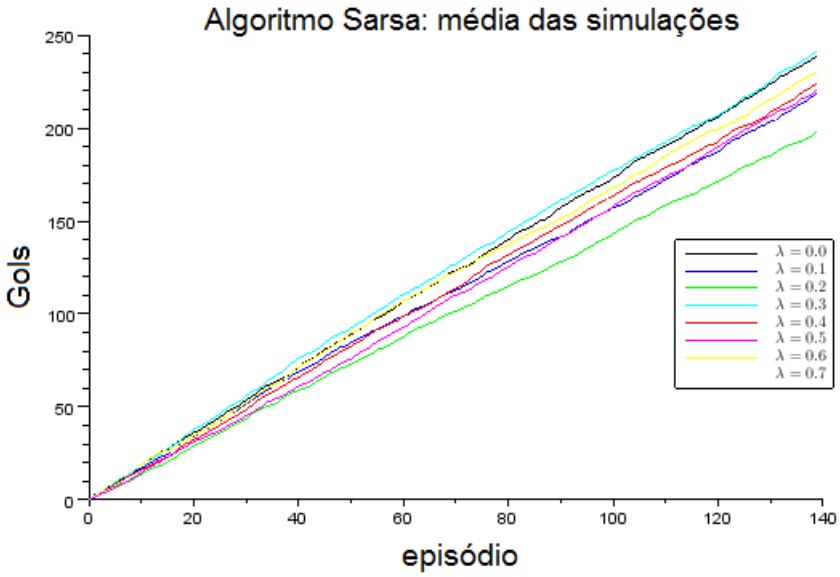

Figura 4.4: Sarsa(λ) - Média dos gols acumulados para os $\lambda's$ simulados.

O parâmetro com maior saldo de gols (λ = 0,3) apresentou o valor médio de 1,93 gols por episódio (Tabela 4.2) com variação entre 0,61 e 3,3 gols (Figura 4.5). Para os demais valores de λ os resultados podem ser vistos no apêndice A.

Tabela 4.2: Sarsa(λ) - Média de Gols por Episódio.

λ	**Gols Acumulados**	**Máximo de Gols/Episódio**	**Mínimo de Gols/Episódio**
0,0	239,30 ± 22,69	2,92	0,62
0,1	219,10 ± 12,70	2,91	0,89
0,2	198,50 ± 23,30	3,81	0,82
0,3	241,60 ± 18,12	3,30	0,61
0,4	224,50 ± 20,17	2,95	0,57
0,5	220,80 ± 17,92	3,12	0,67
0,6	230,00 ± 23,49	2,94	0,80
0,7	226,30 ± 23,86	2,89	0,70

4.1.2 Experimento 02: Q-Learning(λ)

Neste experimento, os resultados obtidos podem ser vistos por meio da média da diferença temporal ($Q_{t-1} - Q_t$) da matriz $Q(s,a)$ e a média do número de gols (acumulados e por episódios) para os diferentes valores do parâmetro λ. Para o algoritmo Q-Learning(λ),

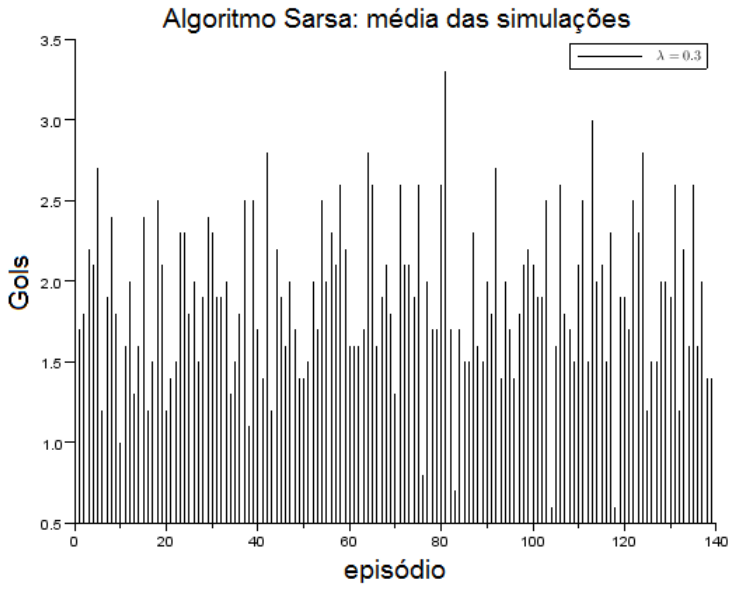

Figura 4.5: Sarsa(λ) - Histograma do número de gols para $\lambda = 0{,}3$.

as curvas que apresentaram o maior e o menor tempo para o decaimento até o ponto de equilíbrio foram para $\lambda = 0{,}1$ e $\lambda = 0{,}3$, respectivamente. Para estes valores foram necessários 126 ($\lambda = 0{,}1$) e 75 ($\lambda = 0{,}6$) episódios para que a diferença temporal estabilizasse em torno do ponto de equilíbrio.

Tabela 4.3: Q-Learning(λ) - Tempo de decaimento (episódio).

λ	**Tempo de decaimento (episódio)**
0,0	105
0,1	126
0,2	92
0,3	88
0,4	106
0,5	84
0,6	45
0,7	97

Do mesmo modo que no experimento 01, as curvas que representam a média de gols acumulados apresentaram um comportamento crescente (Figura 4.7) com variações no saldo de gols. O maior e menor número de gols acumulados foi obtido para $\lambda = 0{,}3$ (280) e $\lambda = 0{,}6$ (220), respectivamente (Tabela 4.3).

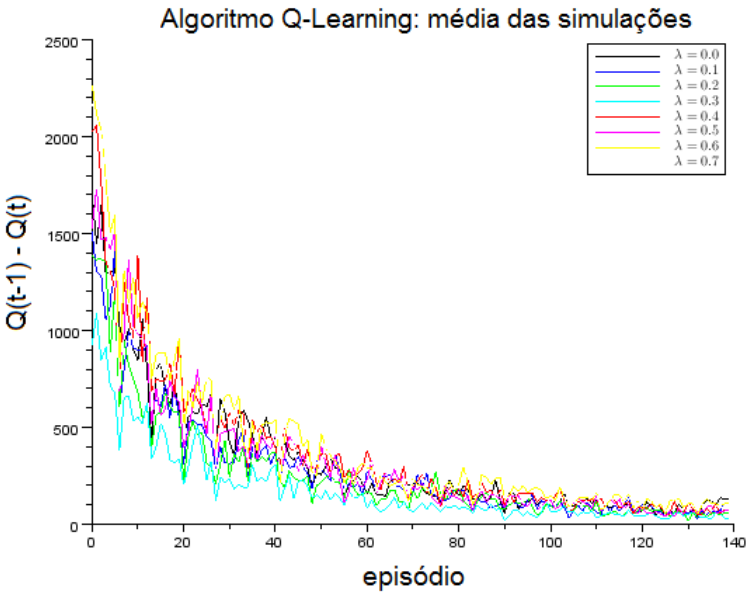

Figura 4.6: Q-Learning(λ) - Média da diferença temporal da matriz $Q(s,a)$ para os $\lambda's$ simulados.

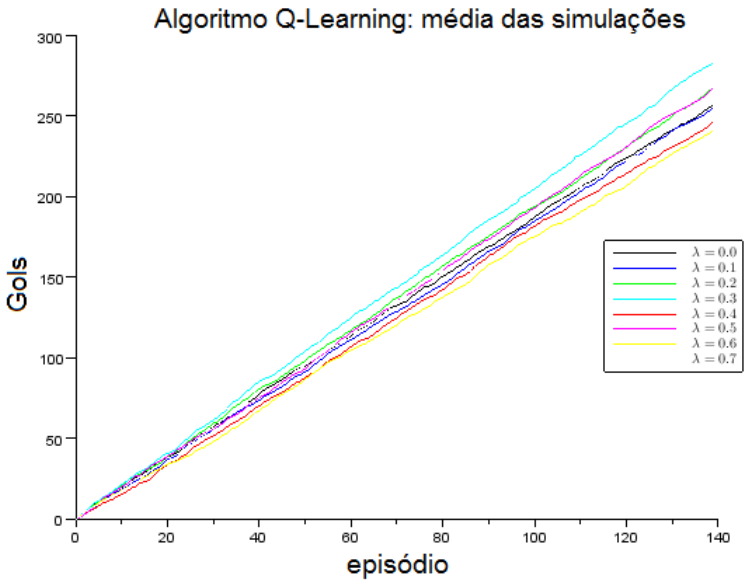

Figura 4.7: Q-Learning(λ) - Média dos gols acumulados para os $\lambda's$ simulados.

O número de gols por episódio para $\lambda = 0{,}3$ variou entre 0,6 e 3,6 (Tabela 4.4), como pode ser observado na Figura 4.8 (demais parâmetros no apêndice A).

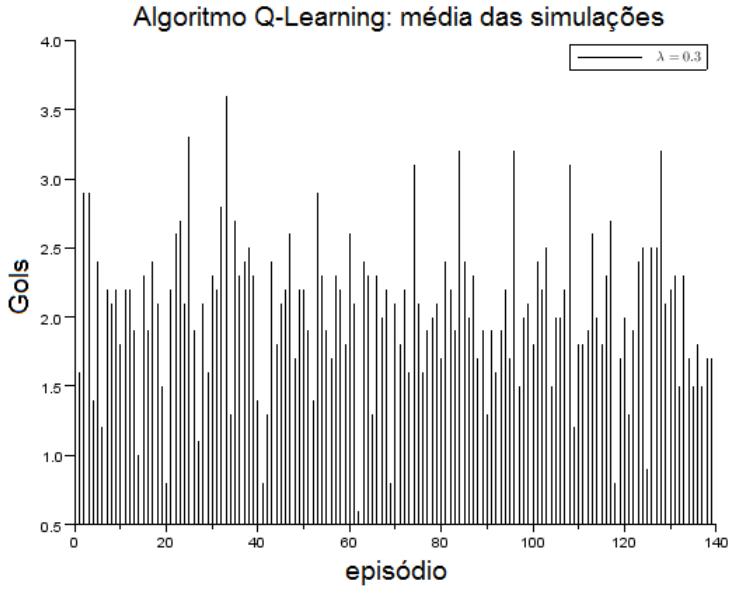

Figura 4.8: Q-Learning(λ) - Histograma do número de gols para $\lambda's = 0{,}3$.

Tabela 4.4: Q-Learning(λ) - Média de Gols por Episódio.

λ	**Gols Acumulados**	**Máximo de Gols/Episódio**	**Mínimo de Gols/Episódio**
0,0	257,60 ± 8,00	3,00	0,72
0,1	254,90 ± 6,12	3,10	0,75
0,2	267,30 ± 8,67	3,00	0,80
0,3	283,20 ± 6,70	3,60	0,60
0,4	246,50 ± 7,38	3,10	0,80
0,5	267,30 ± 7,12	3,10	0,80
0,6	241,40 ± 6,20	3,20	0,70
0,7	254,80 ± 5,33	3,50	0,60

4.1.3 Comparação entre Sarsa(λ) e Q-Learning(λ)

Ao comparar o algoritmo Sarsa(λ) com o Q-Learning(λ) foi possível observar, de acordo com o teste t-student ($p = 1\%$ e $p = 5\%$), que o último apresentou maior número de gols

durante todos os episódios (Figura 4.9 e 4.10). Em relação ao total de gols acumulados, o algoritmo Q-Learning(λ) apresentou uma superioridade de aproximadamente de 8%.

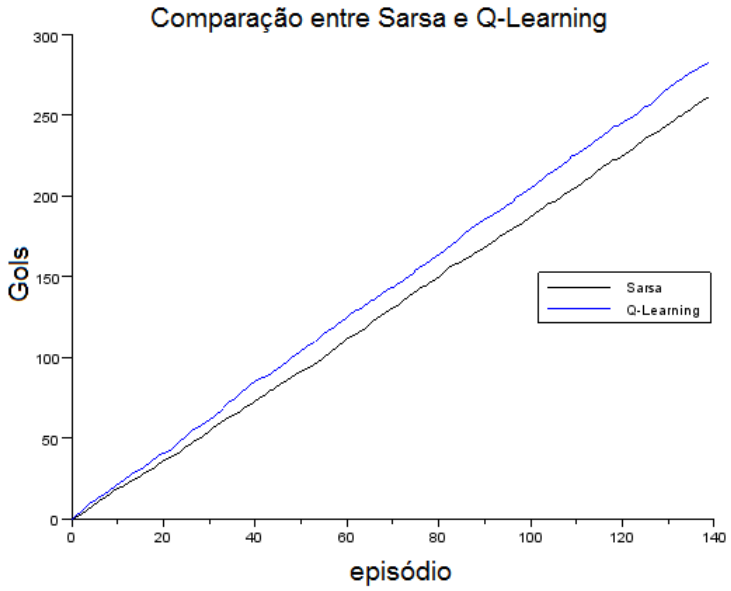

Figura 4.9: Comparação entre Sarsa(λ) e Q-Learning(λ), $\lambda = 0{,}3$.

4.1.4 Discussão

As curvas que representam a diferença temporal da matriz Q (Figura 4.3 e 4.6) para o modelo discreto indicam a convergência para uma política π' qualquer. Além disso, os resultados analisados na forma de gols acumulados e gols por episódio não apresentaram indicações de aprendizagem satisfatória. De acordo com as Figuras 4.4 e 4.7, é possível observar que as curvas aproximadamente lineares independente do λ utilizado, seguindo uma relação constante entre o número de episódios e o número de gols acumulados.

Os resultados obtidos nesse modelo demonstram um baixo desempenho para o aprendizado, o qual, pode ser justificado por três hipotéses - a primeira, pouco tempo para o aprendizado (lenta convergência), a segunda, uma superestimação causada pelo estimador Q dos respectivos algoritmos, e a terceira, uma dificuldade devido a natureza do sistema multiagente.

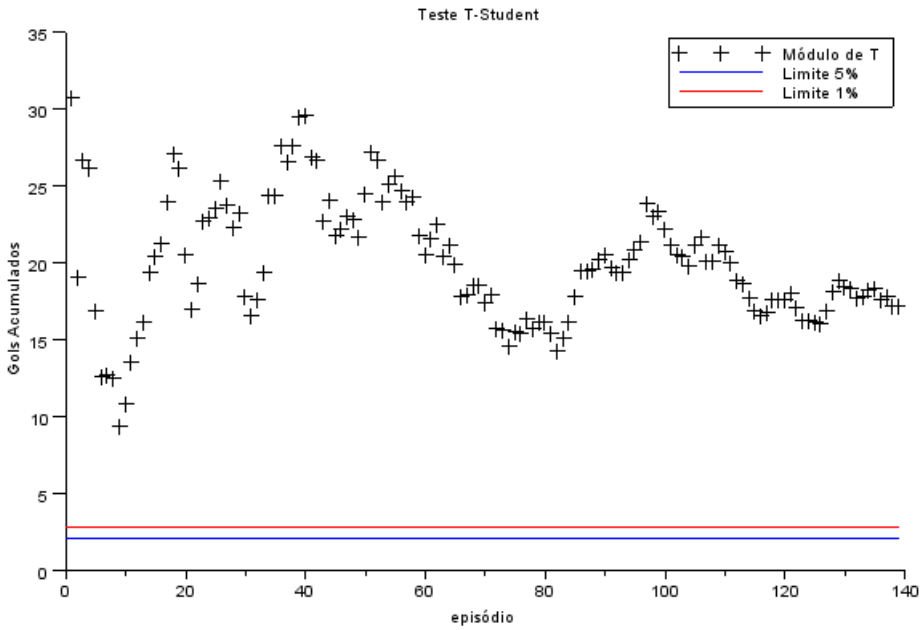

Figura 4.10: Teste T-Student: Comparação entre Sarsa(λ) e Q-Learning(λ), $\lambda = 0{,}3$.

A primeira hipótese vem da afirmação de Watkins e Dayan [58] - desde que cada par estado-ação seja visitado infinitas vezes, a convergência dos valores de Q para Q^* é garantida. Porém, essa convergência pode ser extremamente lenta, necessitando de um maior número de episódios por simulação para atingir a política ótima π^*.

De acordo com a segunda hipotése, a existência de ruídos no ambiente e a modelagem de um PDM estocástico podem resultar em superestimativas dos algoritmos Sarsa(λ) e Q-Learning(λ) [20], dificultando o aprendizado. No presente trabalho, essa hipótese pode ter ocorrido devido ao ruído inserido pelo simulador e pela escolha aleatória da posição dos jogadores no início de cada tentativa.

A terceira hipótese trata sobre a dificuldade de aprendizado em sistemas multiagentes, a qual questiona se o desafio de aprendizagem resulta da interação entre os agentes, os quais tentam aprender e interagir um com o outro, ou da dificuldade da tarefa em si [48]. No cenário proposto, os dois atacantes (agentes) aprendem simultaneamente, pois, seguem a mesma política π', de forma independente, sendo esta uma das

prováveis causas do desempenho apresentado pelos algoritmos.

Comparando o total de gols acumulados, verificou-se um melhor desempenho do algoritmo Q-Learning(λ) em relação ao Sarsa(λ) para o modelo discreto, ou seja, seguir a política que fornece o maior valor da função Q, Off-Policy, gerou melhores resultados.

4.2 Modelo Contínuo

4.2.1 Experimento 03: Sarsa($\lambda = 0{,}3$)

Neste experimento foi observada a relação entre o número de gols acumulados e o número de tilings necessário para a generalização da função $Q_a(s)$ para λ igual a 0,3. Independente do número de tilings utilizado, houve um aumento crescente no número de gols dentro do intervalo estudado (Figura 4.11). Além disso, o aumento do número de tilings não teve relação com o aumento do número de gols acumulados. Assim, a simulação com 32 tilings gerou o maior número de gols acumulados (270), seguido da curva com 8 tilings (260). Já as simulações que forneceram menor número de gols acumulados foram com 1 tiling (176) e 4 tilings (160).

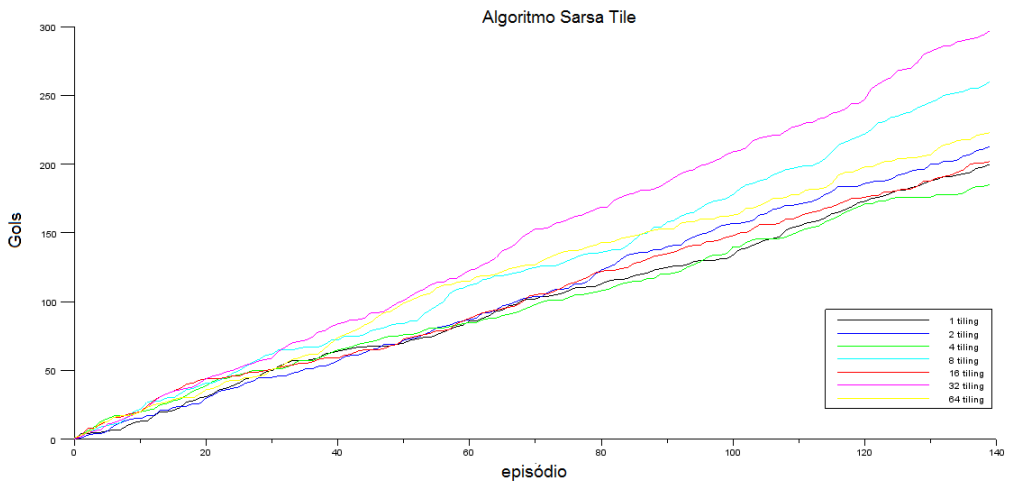

Figura 4.11: Sarsa - Média dos gols acumulados para $\lambda = 0{,}3$.

4.2.2 Experimento 04: Q-Learning($\lambda = 0{,}3$)

Variando o número de tilings e mantendo λ igual a 0,3 no algoritmo Q-Learning(λ) foi possível verificar em todas as curvas um aumento crescente no número de gols durante todo o período de simulação (Figura 4.12), semelhante ao observado no experimento 03. Outra característica comum aos dois algoritmos foi a ausência de relação entre o número de tilings com o aumento do número de gols acumulados. A simulação com maior saldo de gols foi conseguida utilizando 32 tilings (213) e com 1 tiling (155) foi obtido o menor saldo de gols.

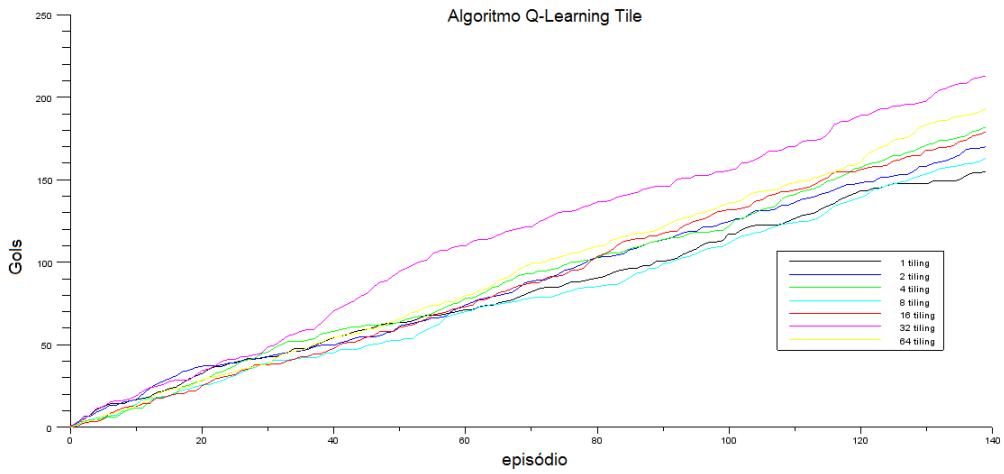

Figura 4.12: Q-Learning(λ) - Média dos gols acumulados para $\lambda = 0{,}3$.

4.2.3 Experimento 05: Sarsa(λ) Tile

A variação do parâmetro λ teve influência sobre o número de gols acumulados. Entretanto, não foi possível afirmar a existência de qualquer relação entre o aumento (ou diminuição) do valor de λ com o acréscimo (ou decréscimo) do número de gols. Todos os valores de λ simulados forneceram um aumento crescente no número de gols dentro do intervalo estudado, sendo o maior saldo de gols (270) obtido com $\lambda = 0{,}3$ e o menor (210,10) com $\lambda = 0{,}1$.

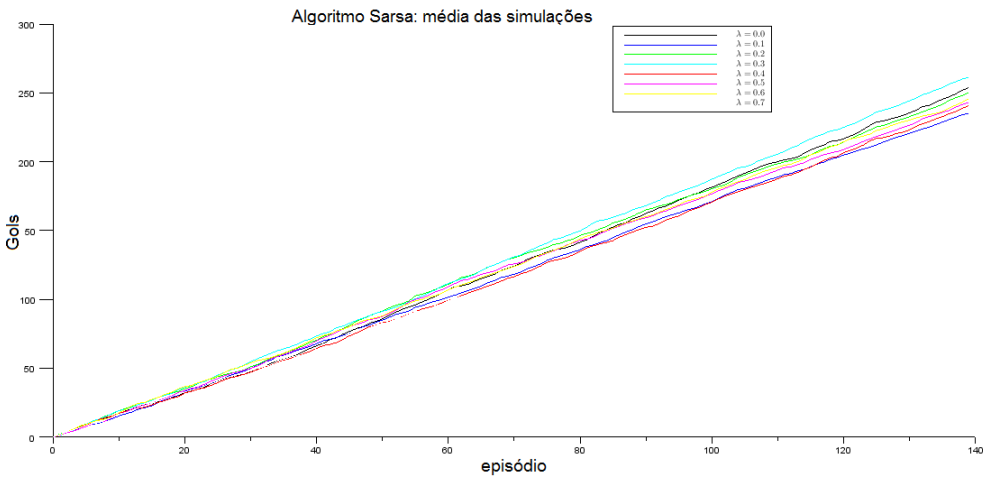

Figura 4.13: Sarsa(λ) Tile - Média dos gols acumulados para os $\lambda's$ simulados.

Os resultados da simulação com λ = 0,3 podem ser observados na Figura 4.14. Essa simulação forneceu um valor médio de 2,8 gols variando entre 0,61 e 3,30 gols por episódio (Tabela 4.5). Para os demais valores de λ os resultados podem ser vistos no apêndice A.

Tabela 4.5: Sarsa(λ) Tile - Média de Gols por Episódio.

λ	**Gols Acumulados**	**Máximo de Gols/Episódio**	**Mínimo de Gols/Episódio**
0,0	255,40 ± 17,30	2,92	0,62
0,1	230,10 ± 12,50	2,91	0,89
0,2	250,20 ± 19,00	3,81	0,82
0,3	270,00 ± 16,40	3,30	0,61
0,4	232,70 ± 13,20	2,95	0,57
0,5	238,30 ± 18,40	3,12	0,67
0,6	245,60 ± 12,70	2,94	0,80
0,7	240,40 ± 11,80	2,89	0,70

4.2.4 Experimento 06: Q-Learning(λ)

Os resultados obtidos com Q-Learning(λ) apresentaram variações no total de gols de acordo com o parâmetro λ utilizado. Estas variações foram semelhantes às encontradas com Sarsa(λ). Da mesma forma, não foi possível afirmar a existência de uma relação

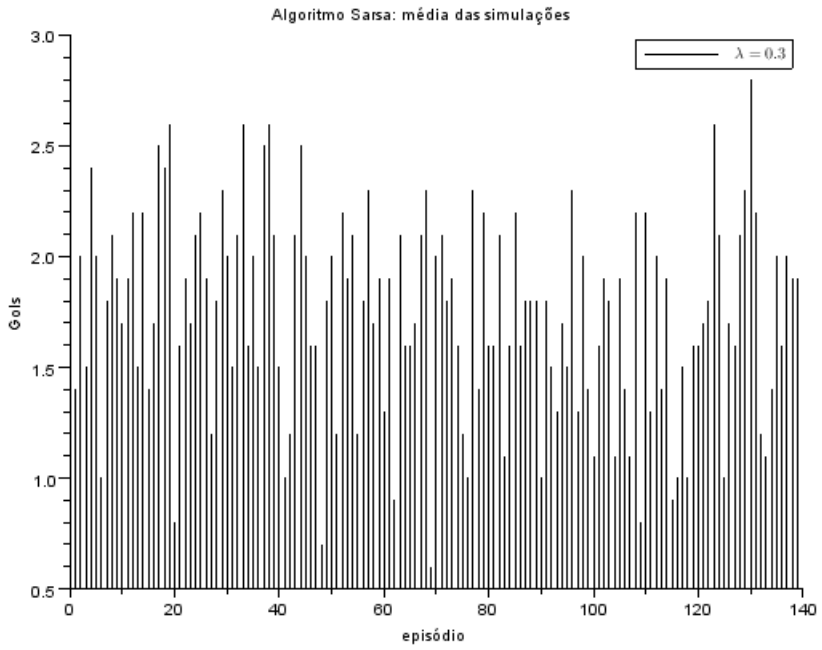

Figura 4.14: Sarsa(λ) Tile - Histograma do número de gols para $\lambda = 0{,}3$.

entre o aumento (ou diminuição) do valor de λ com o acréscimo (ou decréscimo) do número de gols.

Todos os valores de λ forneceram um aumento crescente de gols durante o intervalo estudado, sendo o maior saldo de gols (213) obtido com $\lambda = 0{,}3$ e o menor (197,8) com $\lambda = 0{,}6$.

Os resultados da simulação para $\lambda = 0{,}3$ podem ser observados na Figura 4.14. Essa simulação forneceu um valor médio de 2,3 gols variando entre 0,8 e 2,85 gols por episódio (Tabela 4.5). Para os demais valores de λ, os resultados podem ser vistos no apêndice A.

4.2.5 Comparação entre Sarsa Tile(λ) e Q-Learning Tile(λ)

Os gols marcados durante a simulação do algoritmo Sarsa Tile(λ) e Q-Learning Tile(λ), utilizando λ igual a 0,3, resultou em um maior número de gols (Figura 4.17). Ao analisar

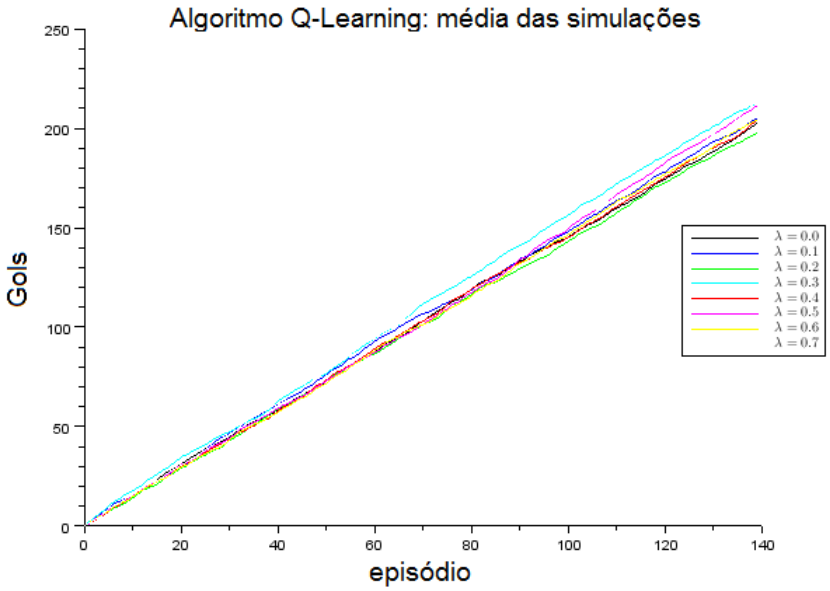

Figura 4.15: Q-Learning(λ) Tile - Média dos gols acumulados para os $\lambda's$ simulados.

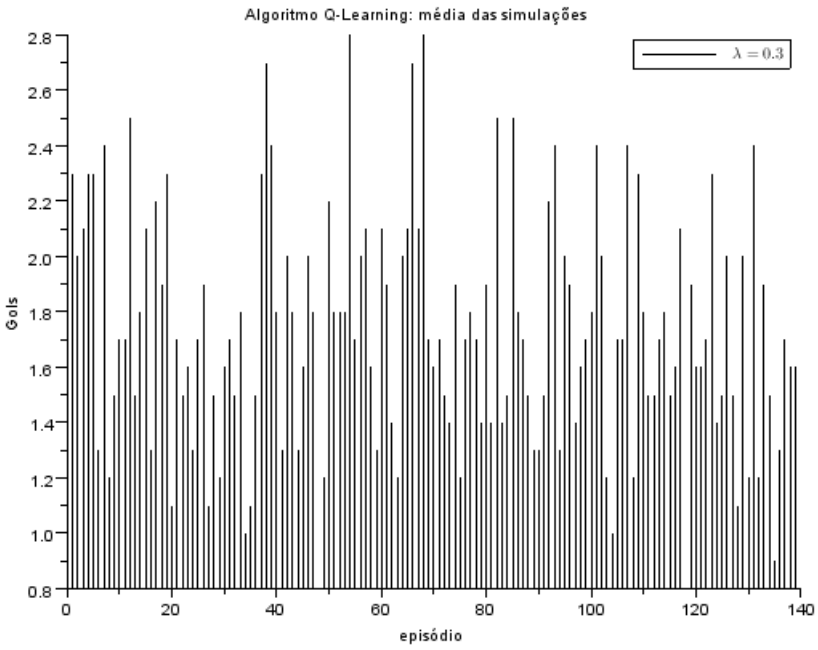

Figura 4.16: Q-Learning(λ) Tile - Histograma do número de gols para $\lambda = 0{,}3$.

Tabela 4.6: Q-Learning(λ) Tile - Média de Gols por Episódio.

λ	**Gols Acumulados**	**Máximo de Gols/Episódio**	**Mínimo de Gols/Episódio**
0,0	202,60 ± 17,40	2,50	0,68
0,1	204,90 ± 18,85	2,90	0,70
0,2	197,80 ± 19,04	2,08	0,70
0,3	213,00 ± 17,85	2,85	0,80
0,4	204,30 ± 11,68	2,50	0,80
0,5	211,50 ± 32,56	2,80	0,80
0,6	204,10 ± 23,33	2,90	0,60
0,7	206,30 ± 24,51	2,90	0,70

este gráfico, é possível observar que o algoritmo Sarsa Tile(λ) obteve o maior saldo de gols que Q-Learning Tile(λ), a partir do 28 episódio pelo teste t-student ($p = 1\%$ e $p = 5\%$)(Figura 4.18).

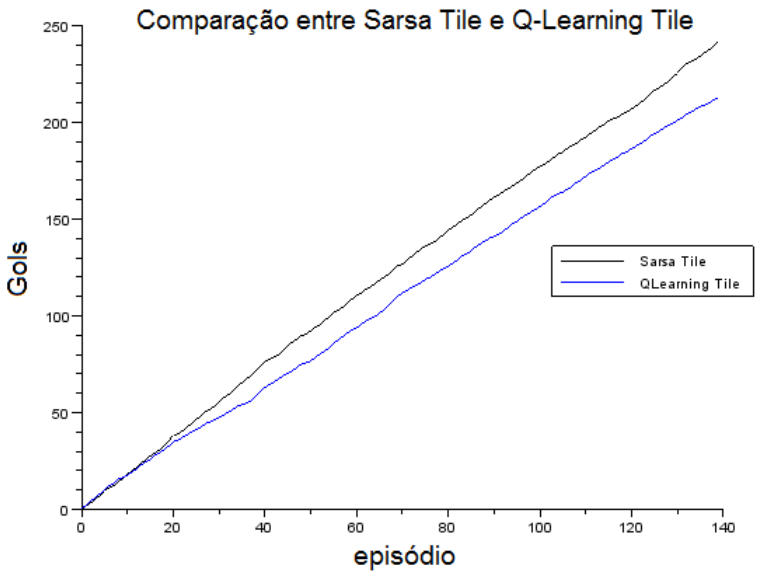

Figura 4.17: Comparação entre Sarsa Tile(λ) e Q-Learning Tile(λ), $\lambda's = 0{,}3$.

4.2.6 Comparação entre Sarsa(λ) e Sarsa Tile(λ)

Analisando os resultados das simulações do algoritmo Sarsa(λ) e do Sarsa Tile(λ) foi possível observar, de acordo com o teste t-student ($p = 1\%$ e $p = 5\%$), que o Sarsa

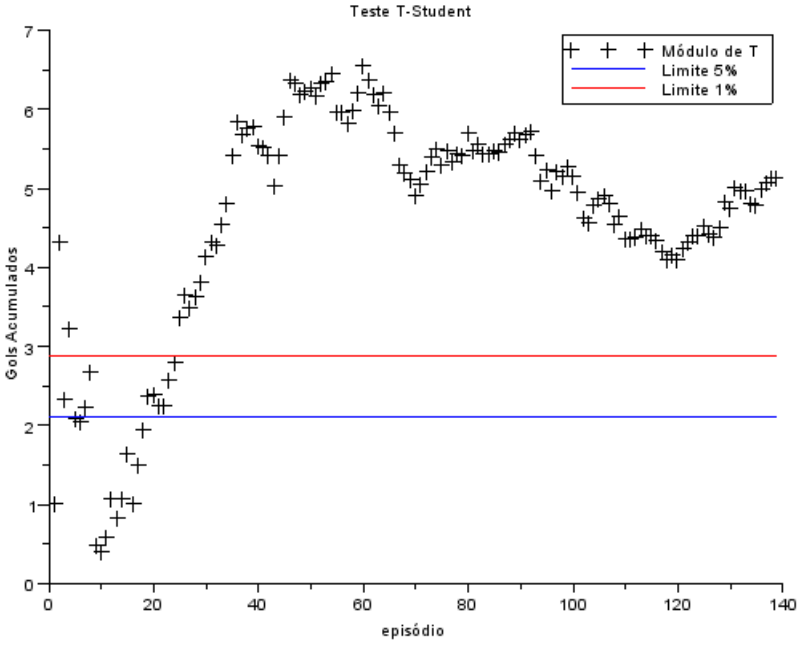

Figura 4.18: Teste T-Student: Comparação entre Sarsa Tile(λ) e Q-Learning Tile(λ), $\lambda = 0{,}3$.

Tile(λ) apresentou um maior número de gols a partir do 67º episódio (Figura 4.19 e 4.20), resultando em uma diferença de aproximadamente de 9,2% (nove vírgula dois por cento) do total de gols marcados.

4.2.7 Comparação entre Q-Learning(λ) e Q-Learning Tile(λ)

A comparação entre Q-Learning(λ) e Q-Learning Tile(λ) foi realizada utilizando o λ igual a 0,3, que resultou em um maior número de gols (Figura 4.21). O teste t-student ($p = 1\%$ e $p = 5\%$) foi utilizado para testar a hipótes de igualdade entre as médias. Como pode ser visto na Figura 4.20, as médias testadas foram consideradas diferentes a partir do 7º episódio, com o algoritmo Q-Learning(λ) apresentando até 42% de gols acumulados a mais que o Q-Learning Tile(λ).

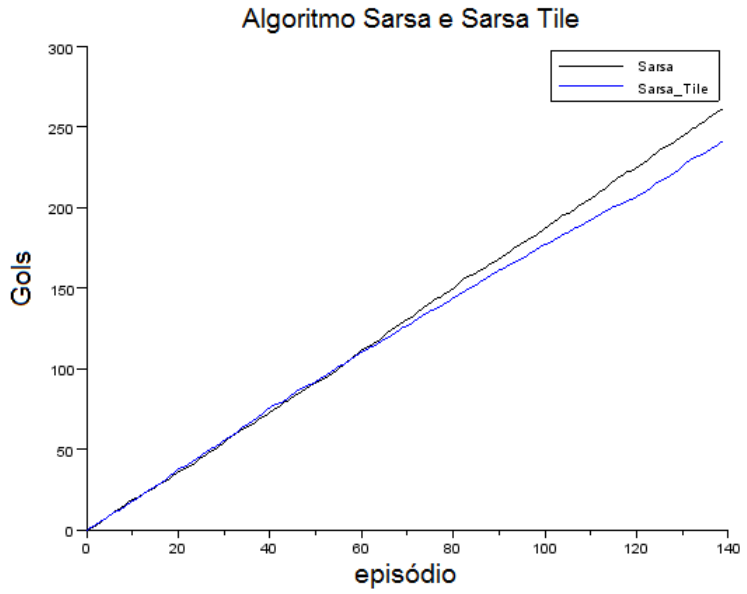

Figura 4.19: Comparação entre Sarsa(λ) e Sarsa Tile(λ), $\lambda's = 0{,}3$.

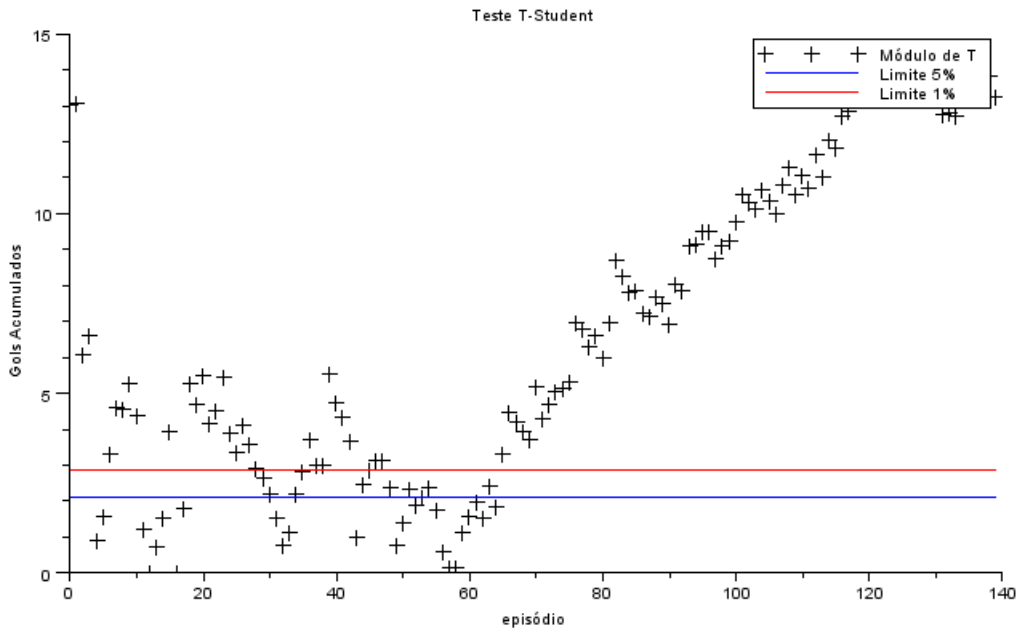

Figura 4.20: Teste T-Student: Comparação entre Sarsa(λ) e Sarsa Tile(λ), $\lambda = 0{,}3$.

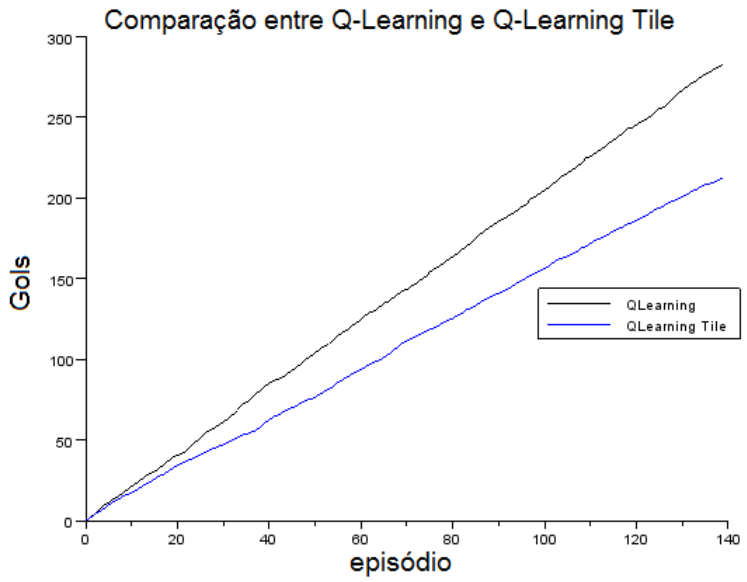

Figura 4.21: Comparação entre Q-Learning(λ) e Q-Learning Tile(λ), $\lambda's = 0{,}3$.

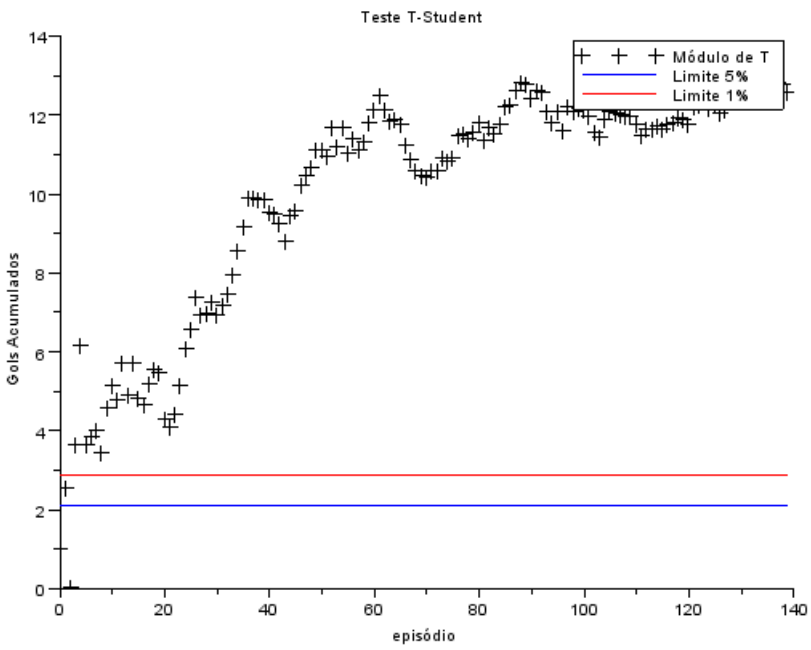

Figura 4.22: Teste T-Student: Comparação entre Q-Learning(λ) e Q-Learning Tile(λ), $\lambda = 0{,}3$.

4.2.8 Discussão

Os resultados obtidos com os experimentos 03 a 06, utilizando a generalização de função linear (Seção 2.8.1), foram qualitativamente semelhantes aos testes realizados com o modelo discreto. O baixo desempenho na aprendizagem manteve-se presente quando utilizado o modelo contínuo. As hipóteses apontadas para o modelo discreto (Seção 4.1.4), como justificativa para o desempenho da aprendizagem, também podem explicar os resultados obtidos com o modelo contínuo.

Comparando o saldo de gols entre os algoritmos do modelo contínuo, verificou-se que Sarsa Tile(λ) apresentou melhor desempenho que Q-Learning Tile(λ), ao contrário do observado na comparação dos algoritmos discretos, que teve melhor resultado quando utilizado o Q-Learning(λ).

Ao comparar o algoritmo Q-Learning(λ) com Q-Learning Tile(λ) pode-se observar um melhor resultado utilizando o algoritmo discreto. Este fato, talvez venha confirmar a opinião de alguns pesquisadores, de que, os métodos Off-Policy como Q-Learning(λ) oferecem vantagens em relação aos métodos On-policy como Sarsa(λ). Entretanto, Q-Learning(λ) pode tornar-se instável quando utilizado com uma função de aproximação linear [48].

O algoritmo Sarsa(λ) Tile, por sua vez, não tem apresentado relatos de problemas com instabilidade causada pela generalização de função. Neste trabalho, o experimento utilizando Sarsa(λ) com o aproximador de funções Tile Coding (2.8.1) apresentou melhores resultados em comparação com Sarsa discreto, demonstrando a eficiência da técnica de generalização de funções. Os resultados obtidos neste trabalho confirmam o sucesso atribuído à utilização de Sarsa(λ) Tile em vez de Q-learning(λ) Tile [48].

Capítulo 5

Conclusões

5.1 Principais Contribuições

Neste trabalho foi investigada a técnica de aprendizado por reforço no domínio de futebol de robôs 2D.

O domínio de futebol de robôs 2D apresenta vários desafios, entre eles, o tamanho do ambiente, os ruídos inseridos pelo servidor e a tarefa multiagente de aprendizado. Estas características fazem do domínio 2D um ambiente complexo, oferecendo ao agente dificuldade para o aprendizado.

Na tentativa de aprender, e além disso, acelerar o aprendizado, os algoritmos Q-Learning (Off-Policy) e Sarsa (On-Policy) foram utilizados nos modelos discreto e contínuo, assim definidos para este trabalho.

No modelo discreto, as variáveis de estado foram discretizadas. Neste modelo, os algoritmos Q-Learning(λ) e Sarsa(λ) foram implementados e simulados para obtenção de informações sobre o Traço de Elegibilidade. O principal parâmetro do Traço de Elegibilidade, λ, foi testado, variando-o entre 0,0 e 0,7. Os melhores resultados foram obtidos para $\lambda = 0{,}3$ em ambos os algoritmos. A comparação feita entre as curvas (gols acumulados) dos algoritmos Q-Learning(λ) e Sarsa(λ) para $\lambda = 0{,}3$ demonstrou a superioridade do algoritmo Off-Policy, constada pelo teste t-student.

No modelo contínuo, as variáveis de estado foram generalizadas por meio do Tile Coding. Um dos principais fatores que influencia na generalização de função é o número de tilings utilizado. Por isso, os algoritmos Q-Learning(λ) Tile e Sarsa(λ) Tile foram avaliados, variando apenas o número de tilings em 1, 2, 4, ..., 64 (2^n, em que $n = 0, 1$,...,6) e o parâmetro λ igual a 0,3 (fixo). O número de tilings que forneceu um maior número de gols acumulados foi igual a 32, para ambos os algoritmos.

Definido o número de tilings (32) para a generalização, os algoritmos Q-Learning(λ) Tile e Sarsa(λ) Tile foram testados quanto ao uso do Traço de Elegibilidade. Da mesma forma que no modelo discreto, o parâmetro λ foi variado entre 0,0 e 0,7. Os resultados destes experimentos mostraram que ambos os algoritmos com generalização de função apresentaram melhores resultados para $\lambda = 0{,}3$. Com isso, concluiu-se que tanto para Q-Learning(λ) quanto para Sarsa(λ) os melhores resultados são obtidos para $\lambda = 0{,}3$, independentemente do tipo de representação das variáveis de estado (discretizada ou generalizada). Além disso, ao comparar as curvas (gols acumulados) obtidas por meio do Q-Learning(λ) Tile e Sarsa(λ) Tile, utilizando 32 tilings e $\lambda = 0{,}3$, foi possível afirmar a superioridade do algoritmo On-Policy (teste t-student).

Outra conclusão importante foi obtida por meio da comparação entre as curvas (gols acumulados), utilizando os modelos discreto e contínuo para o mesmo algoritmo. Utilizando o teste t-student foi possível garantir que o algoritmo Q-Learning(λ) teve o melhor desempenho, quando comparado com o Q-Learning(λ) Tile. De forma contrária, o algoritmo Sarsa(λ) apresentou melhores resultados com o modelo contínuo.

Neste trabalho, de maneira geral, o aprendizado por reforço apresentou dificuldades de convergência no domínio de futebol de robôs 2D. Como já mencionado, a complexidade deste domínio torna a tarefa de aprendizado, sendo necessários novos estudos a partir da experiência adquirida neste trabalho.

5.2 Trabalhos Futuros

Durante este trabalho, algumas hipóteses foram levantadas, as quais, podem ser investigadas. Entre elas, pode-se destacar:

- A superestimação da função Q - Utilizar o cenário proposto para testar a hipotése de superestimação, por meio da comparação dos algoritmos Q-Learning [57] e QV-Learning [20];

- A influência do número de agentes - Os experimentos foram realizados para um cenário, no qual, o número de agentes foi fixo. Por isso, variar o número de adversários e atacantes, mantendo a área limite, ajudaria a entender o comportamento deste sistema multiagente;

- A influência quantitativa e qualitativa na escolha das variáveis de estado - Caso exista, obter o número mínimo de variáveis de estado necessárias para um bom aprendizado. Além disso, avaliar a influência do aprendizado quando inseridas variáveis aleatórias no sistema;

- Comparar o desempenho entre diferentes funções de generalização: linear (Tile Coding) e não-linear (Redes Neurais);

- Avaliar o número de tiles - As simulaçãoes foram realizadas utilizando 4 tiles, os quais, mantiveram-se fixos. Comparar diferentes simulações variando o número de tiles e tilings ajudaria na compreensão do Tile Coding no domínio estudado;

Referências Bibliográficas

[1] Akiyama, H. and Noda, I. (2008). Multi-agent positioning mechanism in the dynamic environment. *RoboCup 2007: Robot Soccer World Cup XI*, pages 377–384.

[2] Alagoz, O., Hsu, H., Schaefer, A. J., and Roberts, M. S. (2010). Markov decision processes: A tool for sequential decision making under uncertainty. *Medical Decision Making*, 30(4):474–483.

[3] Albus, J. S. (1975). A new approach to manipulator control: The cerebellar model articulation controller (cmac). *Trans. of the ASME, J. Dynamic Systems, Measurement, and Control*, 97(3):220–227.

[4] Atici, F. M. and Turhan, N. (2012). Sequential decision problems on isolated time domains. *Journal of Mathematical Analysis and Applications*, 388(2):753–759.

[5] Baird, L. C. and Moore, A. W. (1999). Gradient descent for general reinforcement learning. *In M. J. Kearns, S. A. Solla, and D. A. Cohn (Eds.) Advances in neural information processing systems*, 11:968–974.

[6] Bellman, R. E. (1957). *Dynamic Programming.* Princeton University Press, Princeton, NJ.

[7] Bertsekas, D. (2001). *Dynamic programming and optimal control.* Belmont, MA: Athena Scientific. ISBN 1-886259-08-6.

[8] Bianch, R. A. C. (2004). *Uso de Heurística para a Aceleração do Aprendizado por Reforço*. PhD thesis, Escola Politécnica da Universidade de São Paulo.

[9] Boer and Kok (2002). The incremental development of a synthetic multi-agente system: The uva trilearn 2001 robocup soccer simulation team. Master's thesis.

[10] Cetina, V. U. (2008). Multilayer perceptrons with radial basis functions as value functions in reinforcement learning. *European Symposium on Artificial Neural Networks - Avances in Computational Intelligence and Learning (ESANN'2008)*, pages 23–25.

[11] Chapman, D. and Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. *Morgan Kaufmann*, 1:726 – 731.

[12] Crites, R. H. and Barto, A. G. (1996). Improving elevator performance using reinforcement learning. *In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, Advances in Neural Information Processing Systems*, 8:1017–1023.

[13] Deisenroth, M. P., Rasmussen, C. E., and Peters, J. (2008). Model-based reinforcement learning with continuous states and actions. *in Proceedings of the 16th European Symposium on Artificial Neural Networks (ESANN 2008)*, pages 19–24.

[14] F., S., Tizhoosh, H. R., and Salama, M. (2008). Application of reinforcement learning for segmentation of transrectal ultrasound images. *BMC Medical Imaging*, 8:1–10.

[15] Fraccaroli, E. S. (2010). Análise de desempenho de algoritmos evolutivos no domínio do futebol de robôs. Master's thesis, Escola de Engenharia de São Carlos da Universidade de São Paulo.

[16] Framling, K. (2007). Replacing eligibility trace for action-value learning with function approximation. *Proceedings - European Symposium on Artificial Neural Networks (ESANN)*, pages 25–27.

[17] Garcia, J., Bueno, I. L., Fernandez, F., and Borrajo, D. (2009). *A Comparative Study of Discretization Approaches*, chapter A Comparative Study of Discretization Approaches for State Space Generalization in the Keepaway Soccer Task, pages 1 – 31. Computer Science Department, Universidad Carlos III de Madrid Avenida de la Universidad 30, 28911 Leganes, Madrid, Spain.

[18] Geramifard, A., Bowling, M., and Sutton, R. S. (2006). Incremental least-squares temporal difference learning. *In Proceedings of the 21st national conference on Artificial intelligence*, 1:356–361.

[19] Gordon, G. (2001). Reinforcement learning with function approximation converges to a region. *In T. K. Leen, T. G. Dietterich, & V. Tresp (Eds.), Advances in neural information processing systems*, 13:1040–1046.

[20] Hasselt, H. P. V. (2011). *Insights in Reinforcement Learning: Formal analysis and empirical evaluation of temporal-difference learning algorithms.* Isbn 978-90-39354964, School for Information and Knowledge Systems (SIKS).

[21] Junior, L. A. C. (2007). Aprendizado por reforço acelerado por heurísticas no domínio do futebol de robôs simulado. Master's thesis, Centro Universitário da FEI.

[22] Kaelbling, L., Littman, M. L., and Moore, W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

[23] Klopf, A. H. (1972). Brain function and adaptive systems: A heterostatic theory. *Technical Report AFCRL*, .:72–164.

[24] Kostiadis, K. and Hu, H. (2001). Kabage - rl: Kanerva - based generalization and reinforcement learning for possession football. *Proceedings of IEEE/RSJ International Conference on Intelligente Robots and Systems.*

[25] Lin, L. J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321.

[26] Loiacono, D. and Lanzi, P. L. (2008). *Tile Coding Based on Hyperplane Tiles*, pages 179–190. S. Girgin et al.

[27] Moore, A. W. (1991). Variable resolution dynamic programming: Efficiently learning action maps in multivariate real valued state spaces. *Machine Learning: Proceedings of the Eighth International Workshop*, 6:1–5.

[28] Naghibi, M. B., Akbarzadeh, M. R., Bayaz, M. H. J. D., and Mashhadi, H. (2006). Application of q-learning with temperature variation for bidding strategies in market based power systems. *Energy Conversion and Management*, 47:1529–1538.

[29] Nascimeno, C. L. J. and Yoneyama, T. (2009). Inteligência artificial em controle e automação. *Edgard Blucher*, 1.

[30] Netto, S. M. B., Leite, V. R. C., Silva, A. C., Paiva, A. C., and Neto, A. A. (2008). *Reinforcement Learning: Theory and Applications*, chapter 20, pages 379–398. I-Tech Education and Publishing.

[31] Noda, I. (1995). Soccer server: A simulator of robocup. *Proceedings of AI symposium 95, Japanese Society for Artificial Intelligence*, pages 29–34.

[32] Noda, I. and Stone, P. (2002). The robocup soccer server and cmunited clients: Implemented infrastructure for mas research. *Autonomous Agents and Multi-Agent Systems.*

[33] Pegoraro, R. (2001). *Agilizando Aprendizagem por Reforço em Robótica Móvel Através do Uso de Conhecimento Sobre o Domínio.* PhD thesis, EPUSP/PCS.

[34] Pellegrini, J. and Wainer, J. (2007). *Processos de Deciso de Markov: um tutorial.* Revista de Informtica Terica e Aplicada - RITA, Rio Grande do Sul, RS, volume xiv edition.

[35] Peng, J. and Williams, R. J. (1994). Incremental multi-step q-learning. *Machine Learning: Proceedings of the Eleventh International Conference*, .:226–232.

[36] Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* NY: Wiley - Interscience. ISBN 0471619779.

[37] Russell, S. J. and Norvig, P. (2004). *Inteligência Artificial.*

[38] Setiawan, k. (2012). Reinforcement learning combined with radial basis function neural network to solve job-shop scheduling problem. *Progress in business innovation and technology management (APBITM)*, 2:31–38.

[39] Shaw, I. S. and Simões, M. G. (1999). *Controle e Modelagem Fuzzy.*

[40] Sherstov, A. A. and Stone, P. (2005). Function approximation via tile coding: Automating parameter choice. *In J.D. Zucker and I. Saitta, editors, SARA 2005, Lecture Notes in Artificial Intelligence*, pages 194–205.

[41] Siginam, B. (2012). *Adaptive Tile Coding Methods for the generalization of value functions in the RL state space.* PhD thesis, University of Mminnesota.

[42] Silva, A. T. R., Silva, H. G., Santos, E. G., Ferreira, G. B., Santos, T. D., and Silva, V. S. (2010). ibots: Descrição do time. *In Competição Latino Americana de Robótica.*

[43] Simões, M. A. C., Silva, H., Meyer, J., Oliveira, J. D., Cruz, L., and Pessoa, H. (2007). Bahia 2d: Descrição do time. *In XIII Simpósio Brasileiro de Automação Inteligente.*

[44] Singh, S. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158.

[45] Smart, W. D. and Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots. *In Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pages 3404–3410.

[46] Spiegel, M. R. (1984). *Estatística.* São Paulo: McGraw-Hill, 2 edition.

[47] Stone, P. (2000). *Layered learning in multiagent systems: A winning approach to robotic soccer.* Cambridge, MA: The MIT Press.

[48] Stone, P., Sutton, R. S., and Kuhlmann, G. (2005). Reinforcement learning for robocup - soccer keepaway. *Adaptive Behavior*, 13(3):165 –188.

[49] Sutton, R., McAllester, D., Singh, S., and Mansour, Y. (2000). Policy approximation. *In S. A. Solla, T. K. Leen, & K. R. Muller (Eds.) Advances in neural information processing systems*, 12:1057–1063.

[50] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

[51] Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *In Advances in Neural Information Processing Systems 8*, MIT Press:1038 – 1044.

[52] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press., no. 1 edition.

[53] Takahashi, R. H. C. (2007). *Otimização Escalar e Vetorial.* Volume 2: Otimização Escalar.

[54] Tesauro, G. J. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8(3/4):257–277.

[55] Texeira, W. W. M., Teles, F., Barbosa, A. M., Silva, M. A., Nepomuceno, E. G., and Queiroz, M. F. A. (2009). Modelagem e controle de um sistema multiagente cooperativo: visão global e local aplicadas ao problema da alocação de recursos. *In IX Simpósio Brasileiro de Automação Inteligente.*

[56] Tsitsiklis, J. N. and Roy, B. V. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:674–690.

[57] Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* PhD thesis, King's College, Cambridge.

[58] Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.

[59] Wiering, M. and Schimidhuber, J. (1998). Fast on-line q(lambda). *Machine Learning*, 33:105–115.

[60] Zadech, L. A. (1965). Fuzzy sets. *Information and Control.*, 8:338–353.

Apêndice A

Simulações - Repetições

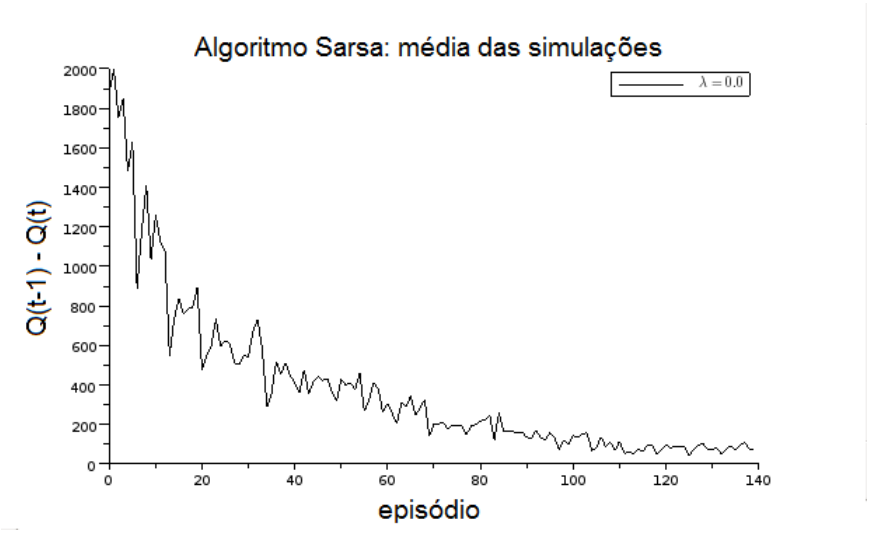

Figura A.1: Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0.0$).

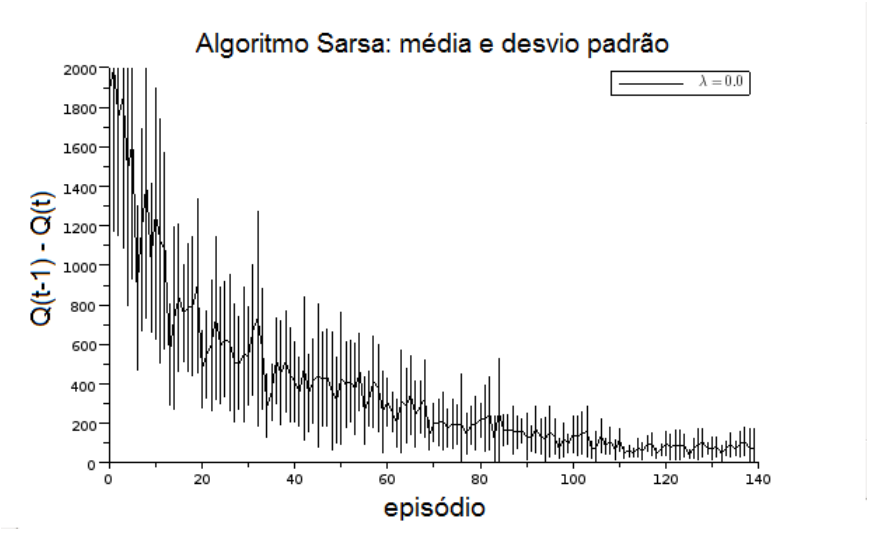

Figura A.2: Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0.0$).

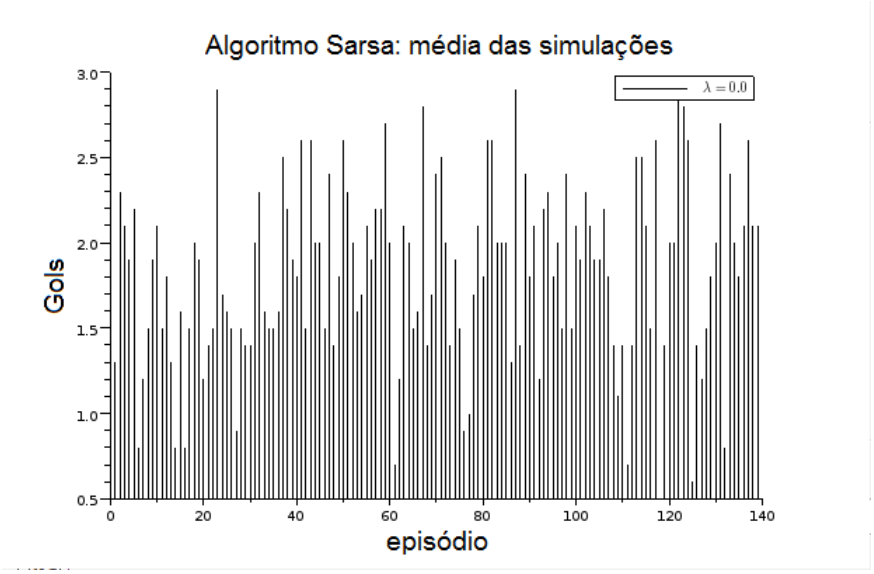

Figura A.3: Fluxograma da média de gols, Sarsa($\lambda = 0.0$).

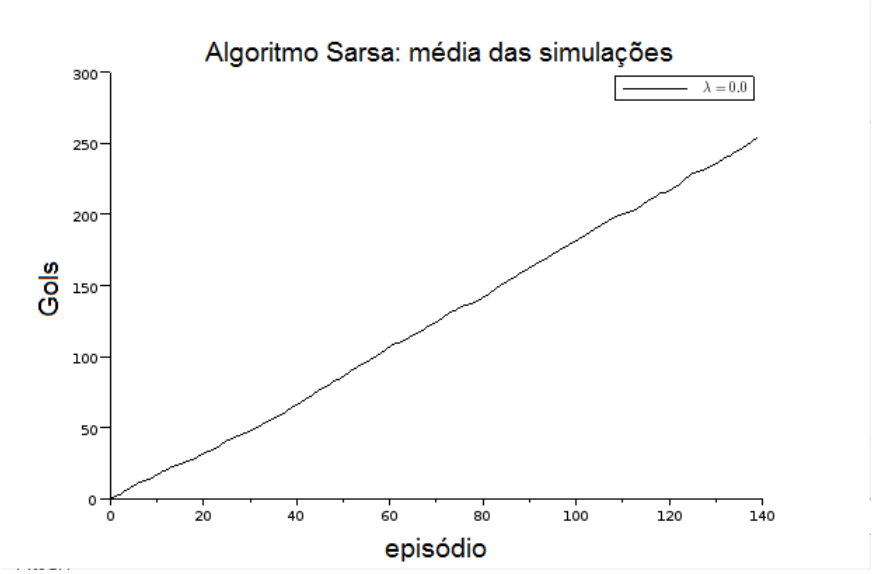

Figura A.4: Média de gols acumulados, Sarsa($\lambda = 0.0$)

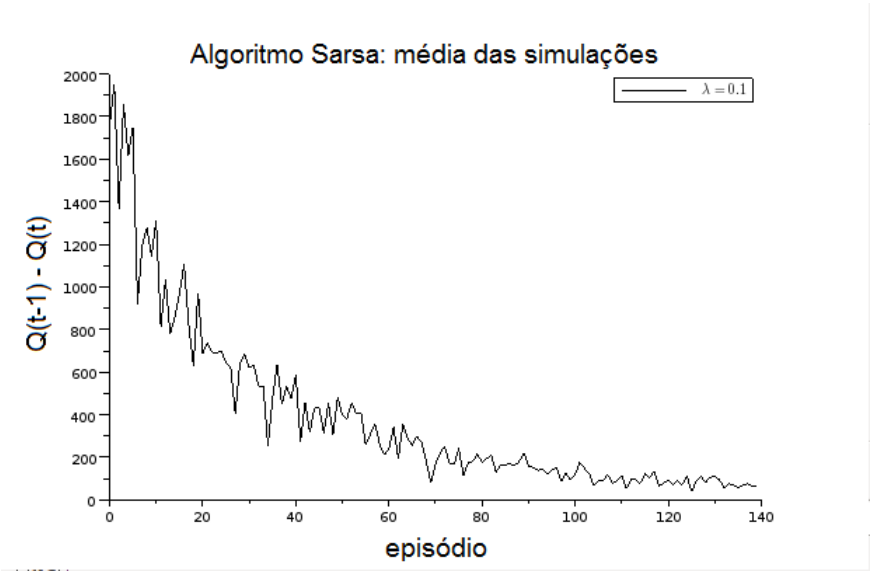

Figura A.5: Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}1$)

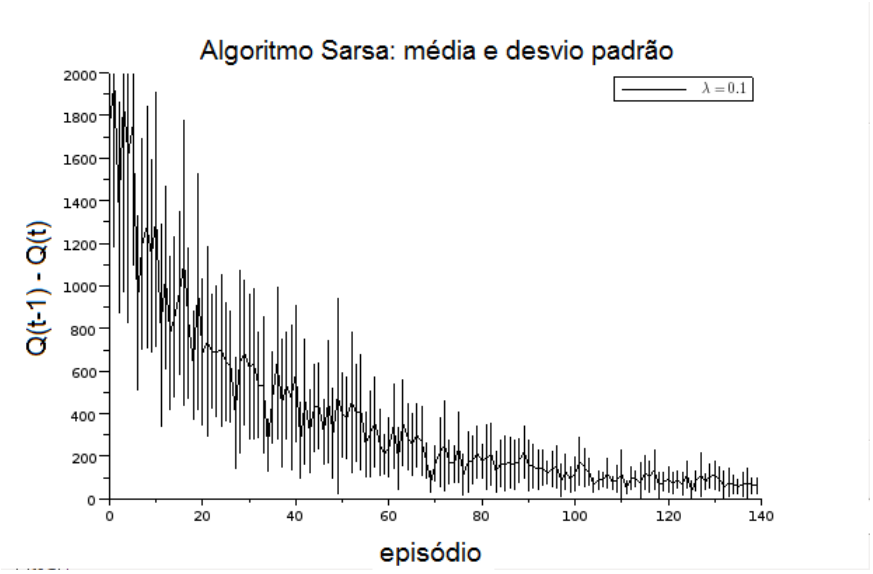

Figura A.6: Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}1$)

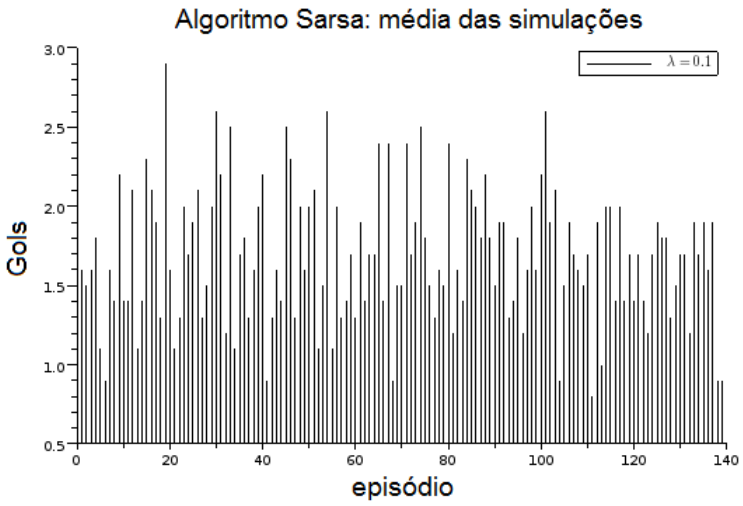

Figura A.7: Fluxograma da média de gols, Sarsa($\lambda = 0{,}1$).

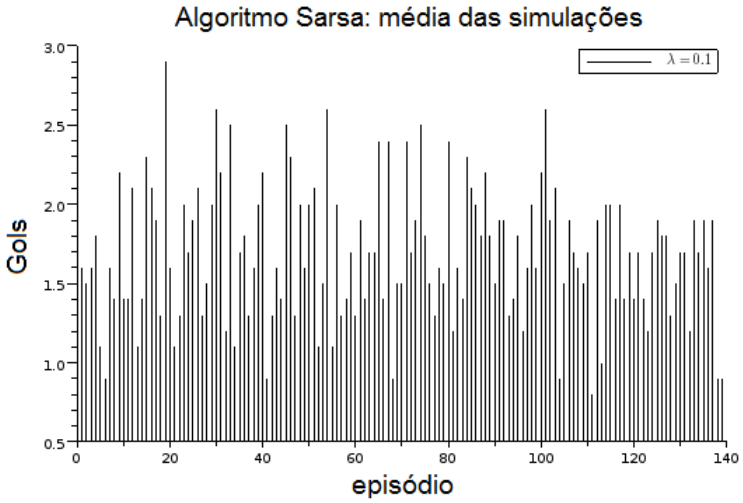

Figura A.8: Média de gols acumulados, Sarsa($\lambda = 0{,}1$).

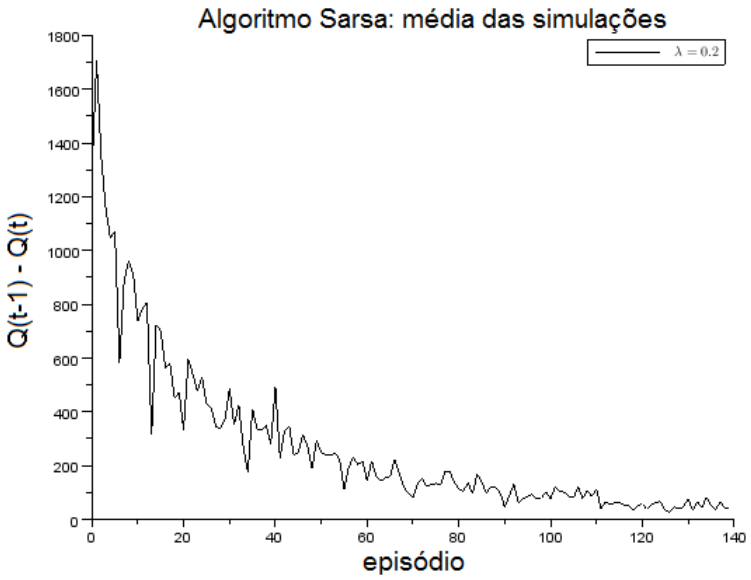

Figura A.9: Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}2$).

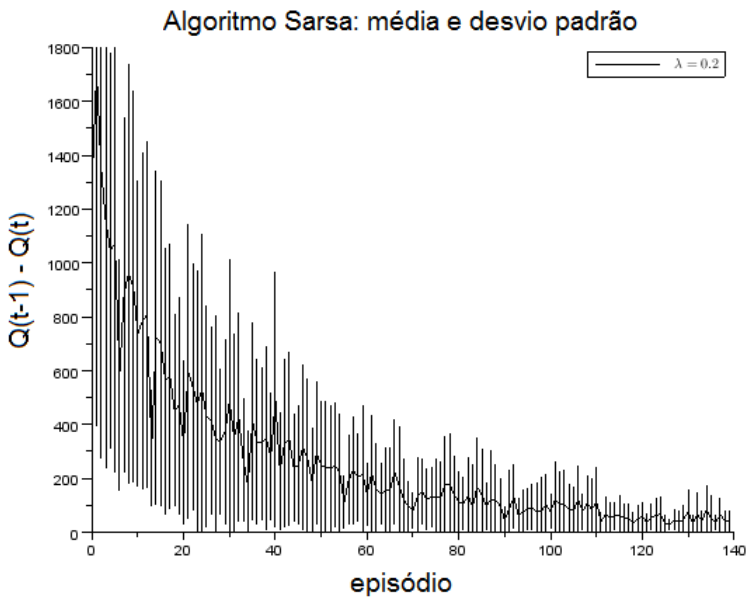

Figura A.10: Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}2$).

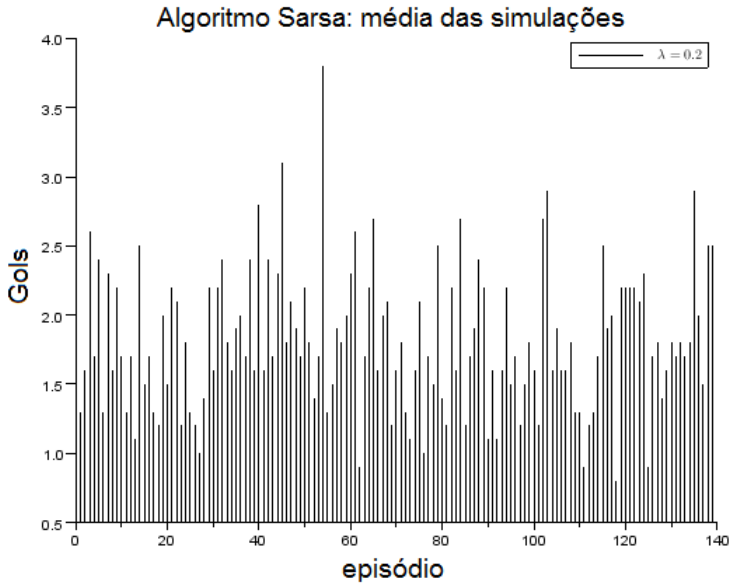

Figura A.11: Fluxograma da média de gols, Sarsa($\lambda = 0{,}2$).

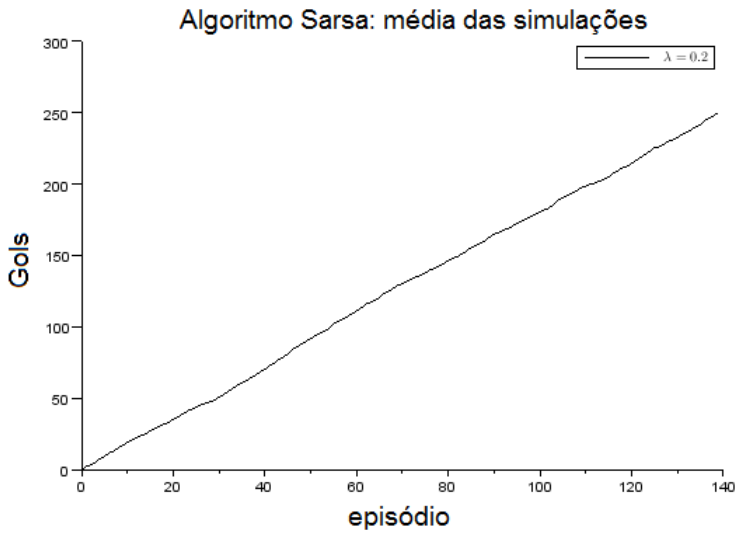

Figura A.12: Média de gols acumulados, Sarsa($\lambda = 0{,}2$).

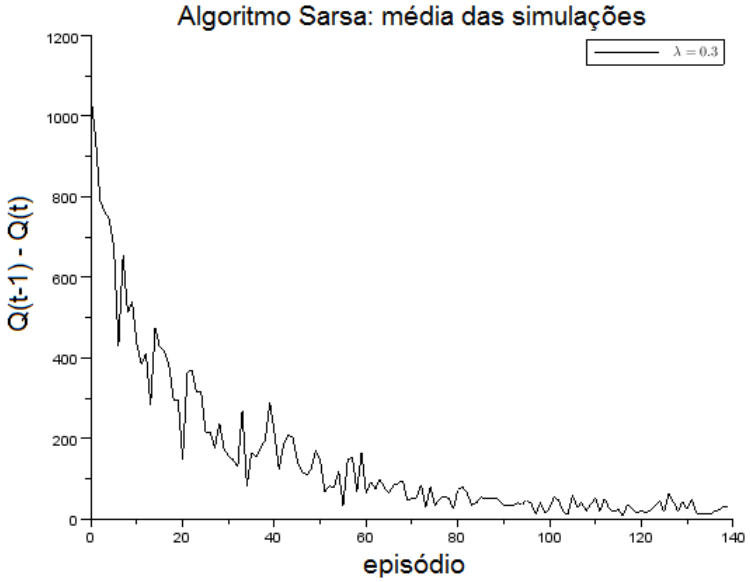

Figura A.13: Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}3$)

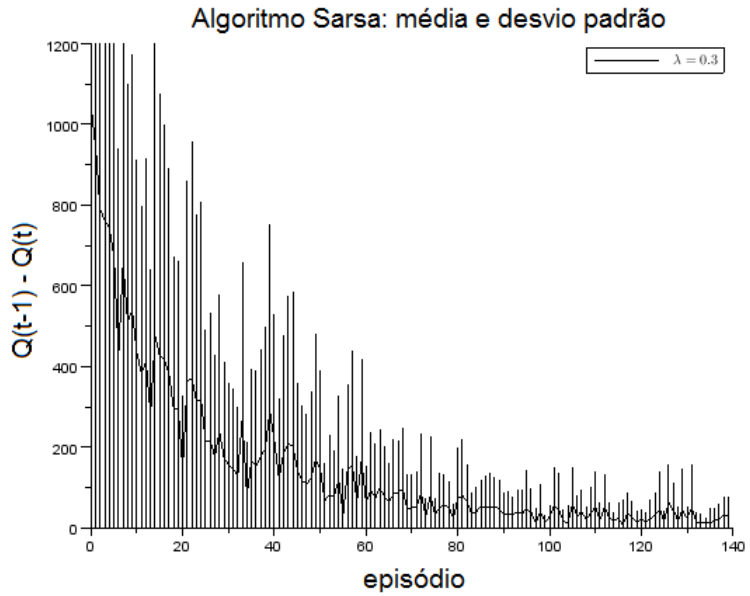

Figura A.14: Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}3$).

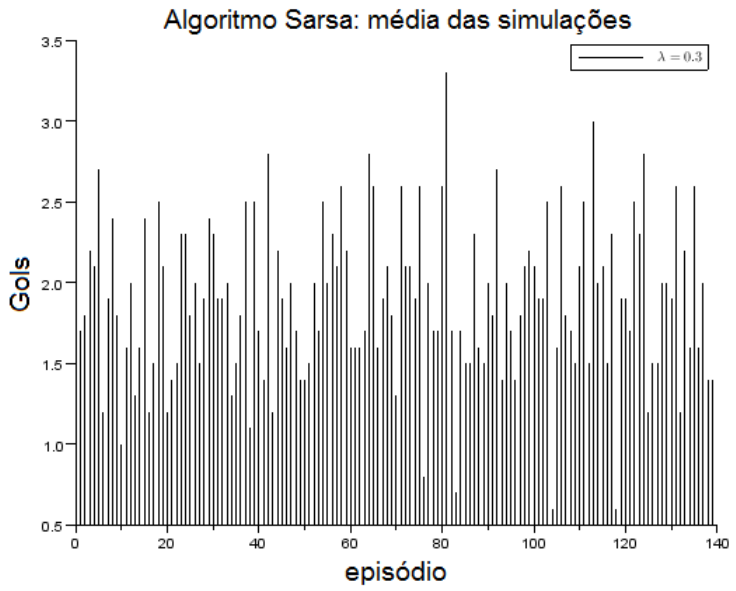

Figura A.15: Fluxograma da média de gols, Sarsa($\lambda = 0{,}3$).

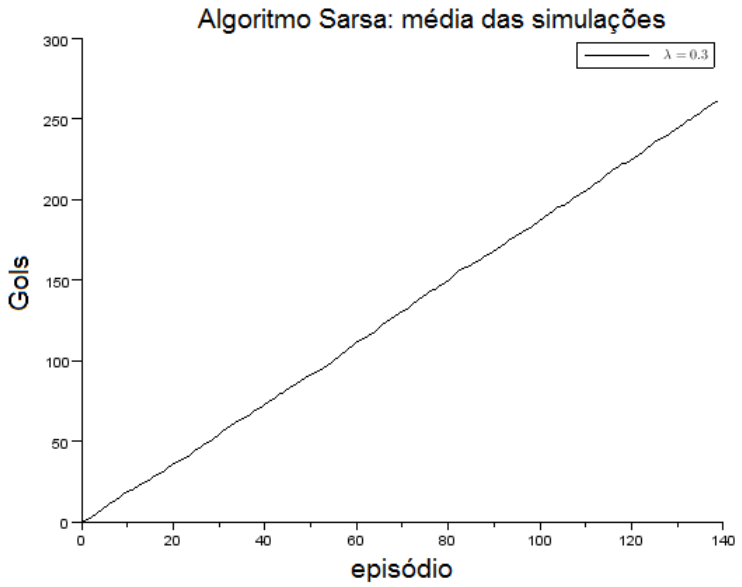

Figura A.16: Média de gols acumulados, Sarsa($\lambda = 0{,}3$).

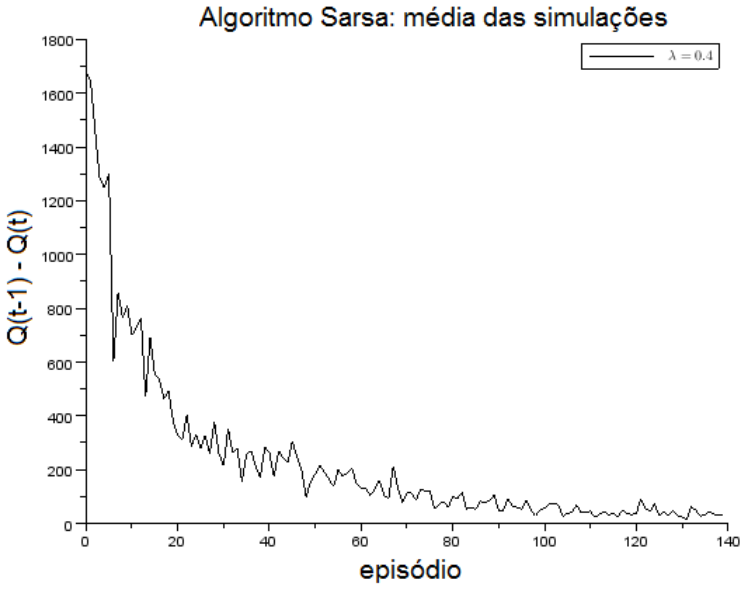

Figura A.17: Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}4$).

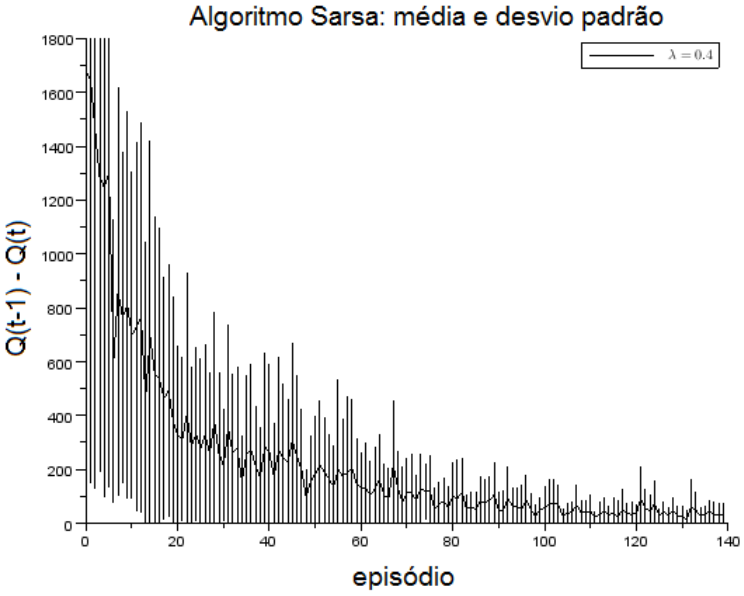

Figura A.18: Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}4$).

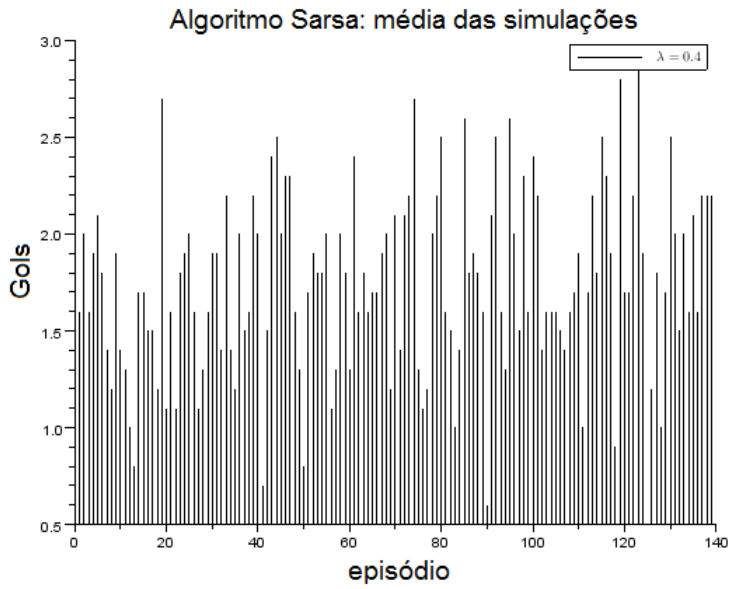

Figura A.19: Fluxograma da média de gols, Sarsa($\lambda = 0{,}4$).

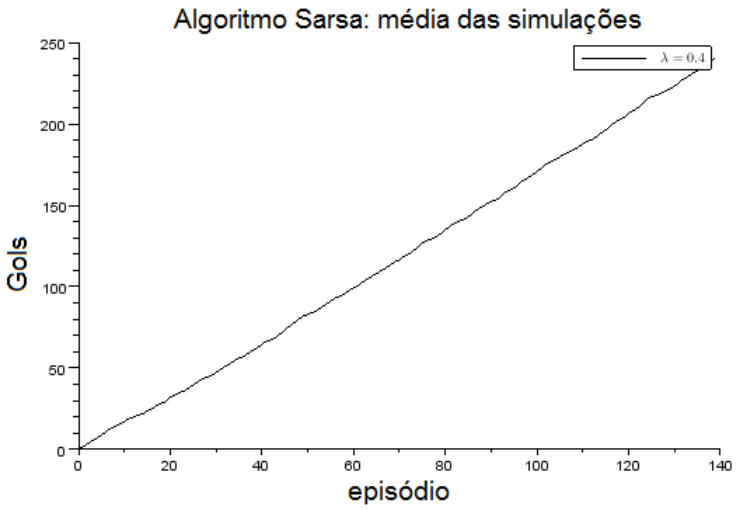

Figura A.20: Média de gols acumulados, Sarsa($\lambda = 0{,}4$).

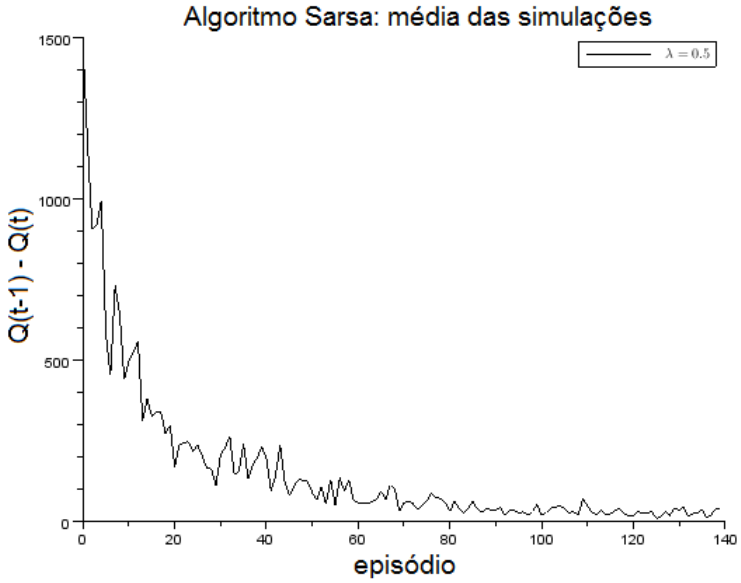

Figura A.21: Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}5$).

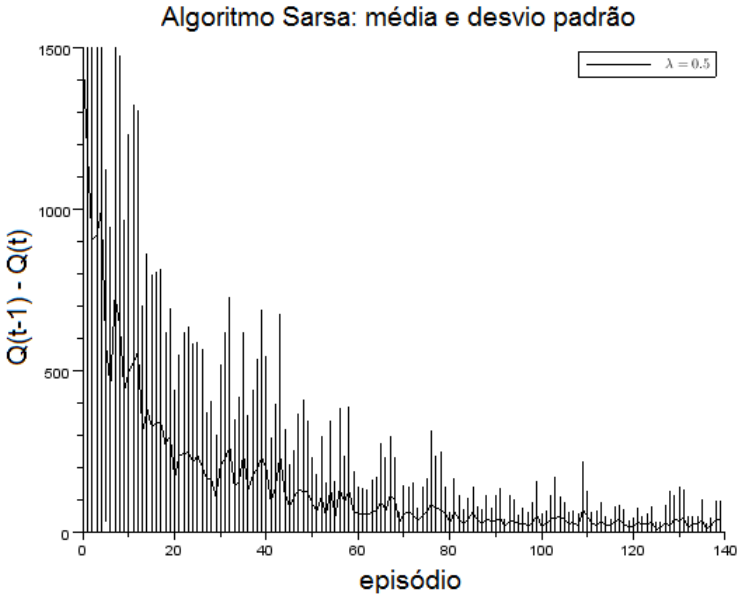

Figura A.22: Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}5$).

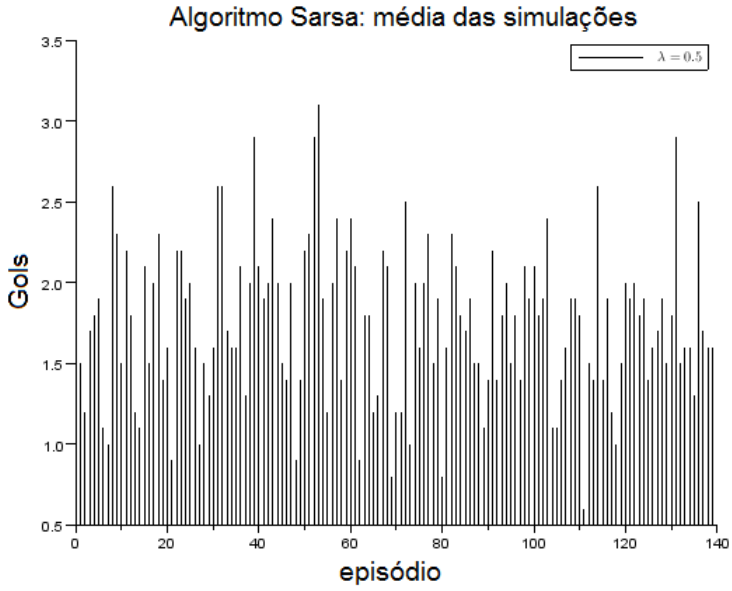

Figura A.23: Fluxograma da média de gols, Sarsa($\lambda = 0{,}5$).

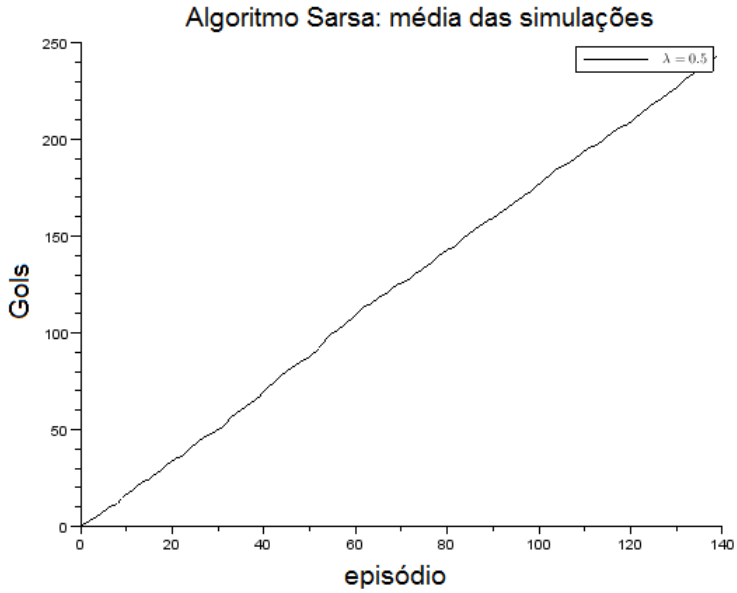

Figura A.24: Média de gols acumulados, Sarsa($\lambda = 0{,}5$).

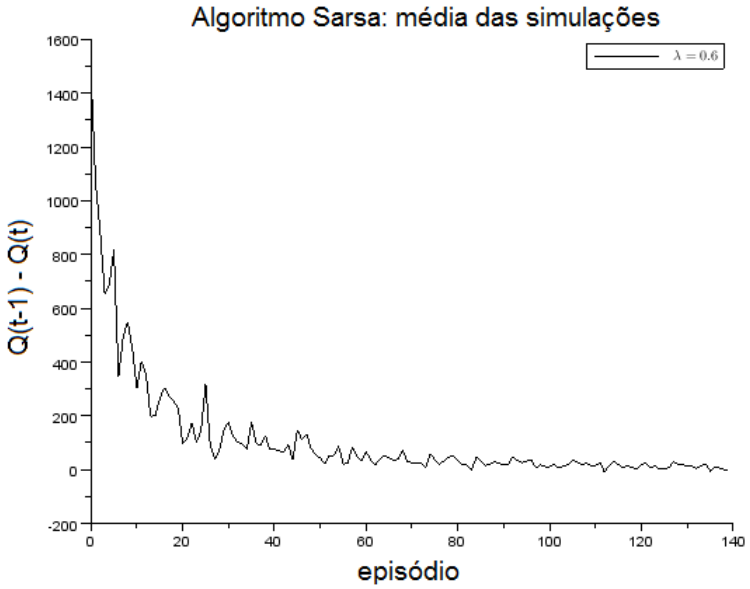

Figura A.25: Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}6$).

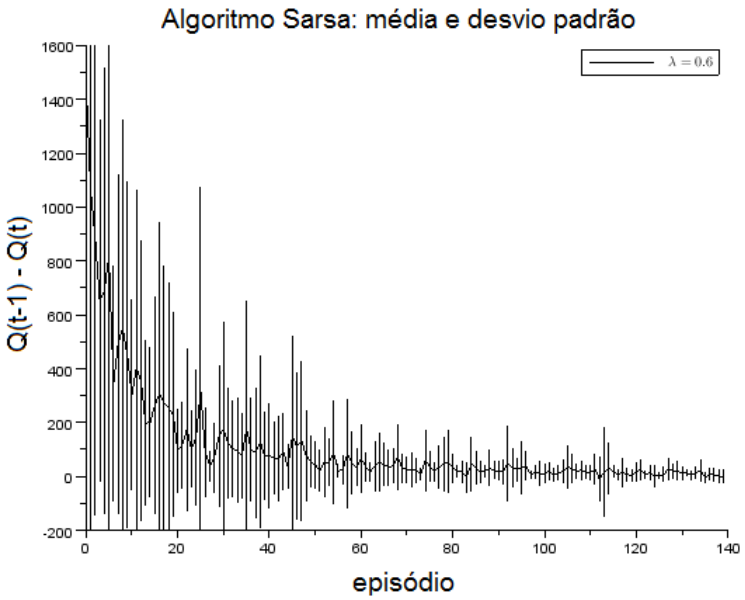

Figura A.26: Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}6$).

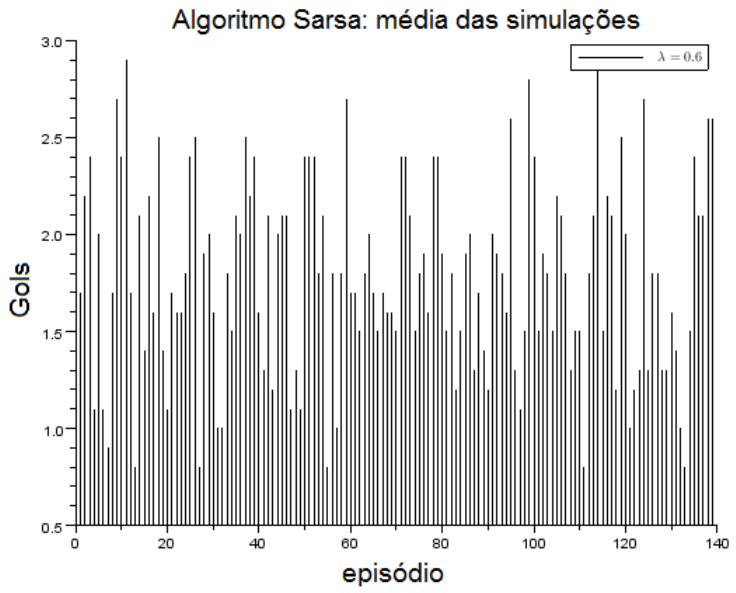

Figura A.27: Fluxograma da média de gols, Sarsa($\lambda = 0{,}6$).

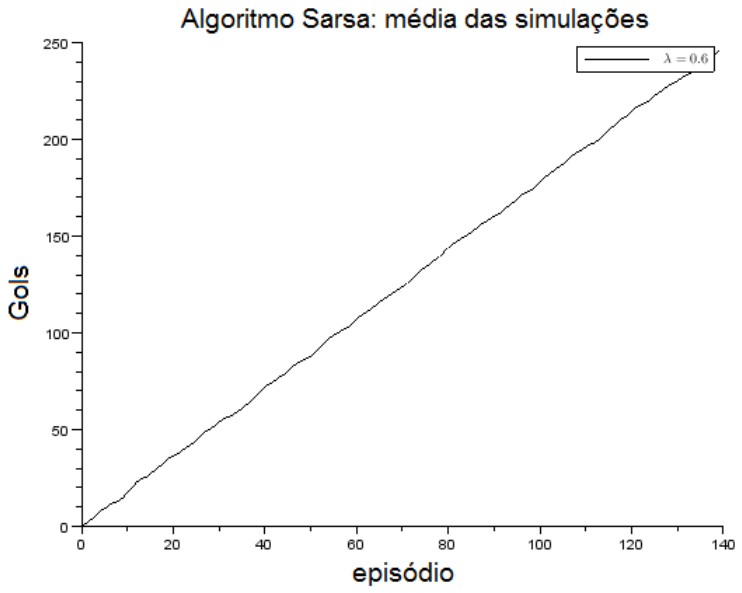

Figura A.28: Média de gols acumulados, Sarsa($\lambda = 0{,}6$).

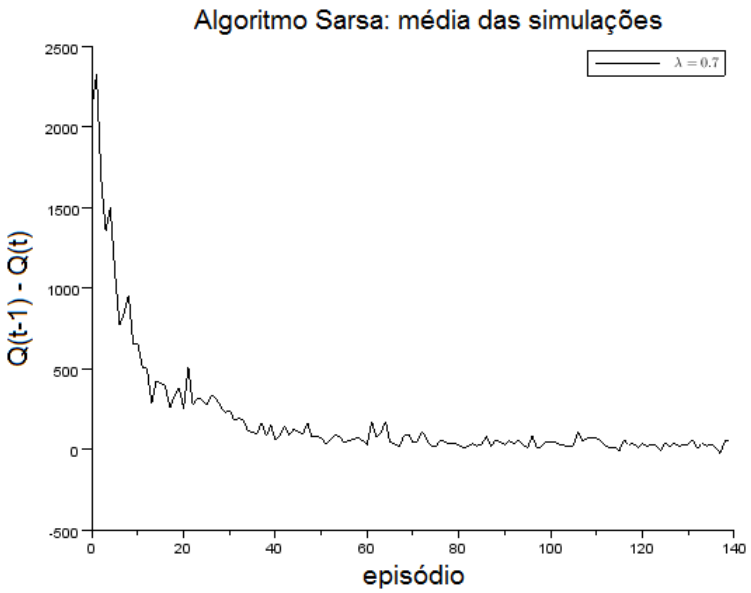

Figura A.29: Média da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}7$).

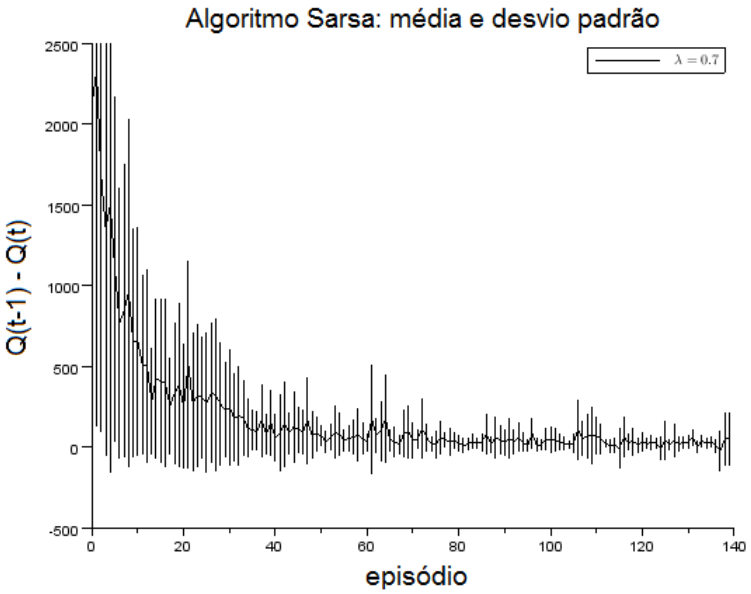

Figura A.30: Média e desvio padrão da diferença temporal da matriz Q(s,a), Sarsa($\lambda = 0{,}7$).

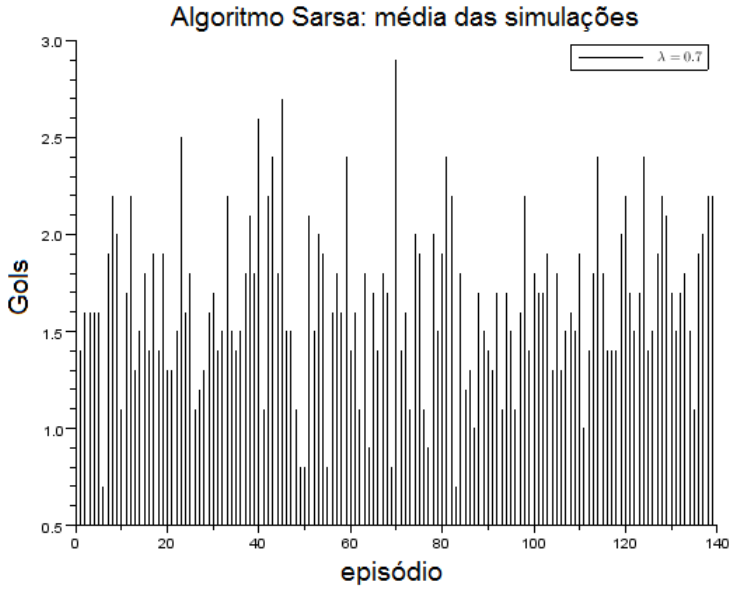

Figura A.31: Fluxograma da média de gols, Sarsa($\lambda = 0{,}7$).

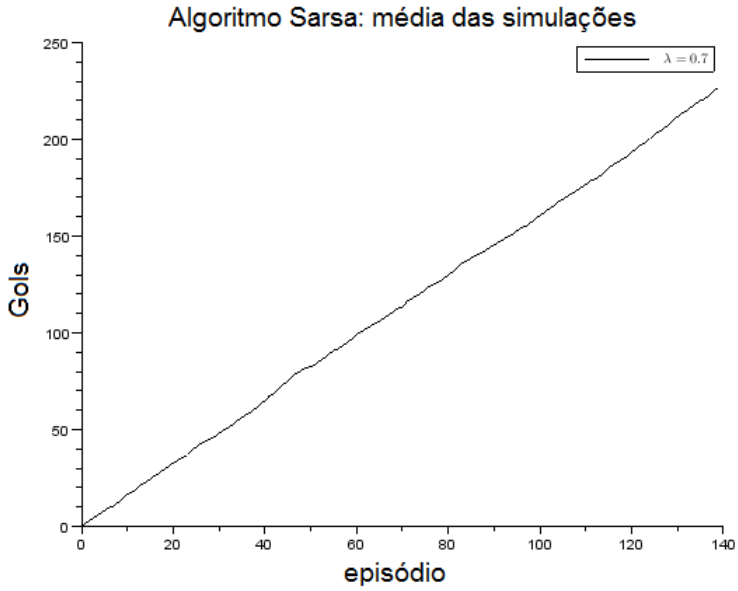

Figura A.32: Média de gols acumulados, Sarsa($\lambda = 0{,}7$).