CENTRO UNIVERSITÁRIO DA FEI

Luiz Antonio Celiberto Junior

APRENDIZADO POR REFORÇO ACELERADO POR HEURÍSTICAS NO DOMÍNIO DO FUTEBOL DE ROBÔS SIMULADO

São Bernardo do Campo

2007

Luiz Antonio Celiberto Junior

APRENDIZADO POR REFORÇO ACELERADO POR HEURÍSTICAS NO DOMÍNIO DO FUTEBOL DE ROBÔS SIMULADO

Dissertação de Mestrado apresentada ao Centro Universitário da FEI como parte dos requisitos necessários para a obtenção do título de Mestre em Engenharia Elétrica.
Orientador: Prof. Dr. Reinaldo A. C. Bianchi

São Bernardo do Campo

2007

Celiberto Jr, Luiz Antonio
Aprendizado por Reforço Acelerado por Heurísticas no Domínio do Futebol de Robôs Simulado / Luiz Antonio Celiberto Junior – São Bernardo do Campo, 2007.
122 f.: il.

Trabalho de Conclusão de Curso – Centro Universitário da FEI.
Orientador: Prof. Dr. Reinaldo Augusto da Costa Bianchi.

1. Inteligência artificial. 2. Robótica inteligente. 3. Robótica móvel. 4. Aprendizado por reforço. 5. RoboCup 2D. I. Título.

CDU 331

A Deus, meus pais e todos os companheiros de todas as horas.

AGRADECIMENTOS

Agradeço a todos os que me deram apoio e ensinaram a ter perseverança durante este período de trabalho desta dissertação.

Aos meus pais que me ajudaram e se esforçaram de todas as maneiras, para eu sempre conseguir alcançar os meus objetivos, e que por mais que eu possa agradecer, ainda fico em débito.

Aos meus amigos do mestrado, que estiveram sempre prontos a ajudar nas resoluções dos problemas, nos momentos difíceis das aulas e pela excelente companhia em todo este progresso intelectual que foi este mestrado. Em especial ao Edson Kitani, por suas palavras sábias, que me ajudaram no meu progresso pessoal e ao Murilo Fernandes Martins, pela colaboração na finalização deste trabalho.

Aos professores da FEI e meus amigos da Escola Politécnica da Universidade de São Paulo, que sempre tinham uma palavra amiga nos momentos mais difíceis e me deram muito suporte e ensinamentos que me foram úteis para a elaboração desta dissertação.

À Esther Luna Colombini, Celeny Alves e Jackson Matsuura, do Instituto Tecnológico da Aeronáutica, que sempre estiveram prontos a me ajudar nos diversos problemas que apareceram durante esta dissertação.

À Valéria Montañola, por sua colaboração na finalização deste trabalho.

À Kelly Pagamonha, por seu carinho e por seu amor e pela sua paciência no decorrer deste trabalho.

Ao meu orientador, Reinaldo Augusto da Costa Bianchi, que sempre teve a paciência de me ajudar nas minhas horas de dúvidas e me ajudou de inúmeras maneiras no meu progresso intelectual e pessoal.

A performance de hoje é o produto do aprendizado do passado. A performance de amanhã é um produto do aprendizado de hoje.

Bob Guns

RESUMO

O Aprendizado por Reforço é uma técnica muita conhecida para a solução de problemas quando o agente precisa atuar com sucesso em um local desconhecido por meio de tentativa e erro. Porém, esta técnica não é eficiente o bastante para ser usada em aplicações com exigências do mundo real, devido ao tempo que o agente leva para aprender. Este trabalho apresenta o uso do Aprendizado por Reforço, acelerado por heurísticas, no domínio da robótica móvel, utilizando para testes a plataforma do *RoboCup* 2D simulação. Esta plataforma vem sendo usada cada dia mais no meio científico, a qual possibilita fazer inúmeros experimentos com jogadores virtuais, sem sofrer com problemas que comumente são encontrados em sistemas reais, além de manterem sempre as mesmas características de ambiente.

O principal problema abordado neste trabalho é o uso da aceleração por heurísticas no Aprendizado por Reforço. Porém esta aceleração só é possível se primeiro for resolvido o problema de como desenvolver um sistema com Aprendizado por Reforço no *RoboCup* 2D. Tal sistema apresenta diversos desafios, sendo o maior deles o tamanho do ambiente, o que gera grande dificuldade para um agente aprender uma política de decisões. Para solucionar este problema foram propostas formas de generalizar os estados, sem causar qualquer interferência no aprendizado.

As experiências realizadas foram feitas sem o uso das heurísticas e depois com o uso das heurísticas. Para a validação do trabalho, cada experimento foi repetido dez vezes, e seus resultados médios comparados através de uma análise estatística. Os resultados indicam algumas vantagens no uso das heurísticas, possibilitando a definição de algumas diretrizes importantes para a aplicação do uso de heurísticas no domínio do futebol de robôs simulado.

Palavras-chave: Inteligência Artificial, Robótica Inteligente, Robótica Móvel, Aprendizado por Reforço, RoboCup 2D.

ABSTRACT

The Reinforcement Learning is a very well known technique for the solution of problems when the agent needs to act with success in an unknown place through trial and error. However, this technique is not efficient enough to be used in applications with demands of the real world, due to the time that the agent takes to learn. This work presents the use of the heuristics accelerated reinforcement learning in the domain of the movable robotics, using for tests the platform of RoboCup 2D simulation. This platform has been used every day more in the scientific way, it makes possible to do countless experiments with virtual players, without suffering of problems that commonly are found in real systems, besides they always maintain the same caracteristics of the ambient.

The main problem approached in this work is the use of the heuristics accelerated reinforcement learning. However this acceleration is only possible if first the problem be solved of how to develop a system with Reinforcement Learning in RoboCup 2D. Such system presents several challenges, being the largest the size of the ambient generating great difficulty for an agent to learn a politics of decisions. To solve this problem forms they were proposed of generalizing the states, without causing any loss in the learning.

The accomplished experiences were made without the use of the heuristics and later with the use of the heuristics. For the validation of the work, each experiment was repeated ten times, and their medium results compared through a statistical analysis. The results indicate some advantages in the use of the heuristics, making possible the definition of some important guidelines for the application of the use of heuristics in the domain of the simulated soccer of robots.

Keywords: Artificial intelligence, Intelligent Robotics, Movable Robotics, Reinforcement Learning, RoboCup soccer 2D.

SUMÁRIO

1 INTRODUÇÃO 18

1.1 DOMÍNIO 20
1.2 OBJETIVO 21
1.3 ORGANIZAÇÃO DO TRABALHO 23

2 APRENDIZADO POR REFORÇO 24

2.1 PROCESSO DE DECISÃO DE MARKOV E PROGRAMAÇÃO DINÂMICA 25
2.2 ALGORITMO DIFERENÇA TEMPORAL – TD (0) 27
2.3 Q-LEARNING 28
2.4 SARSA 30
2.5 MINIMAX-Q 32
2.6 DISCUSSÕES 34

3 ACELERAÇÃO DO APRENDIZADO POR REFORÇO 35

3.1 ACELERAÇÃO POR GENERALIZAÇÃO 35
3.2 ACELERAÇÃO POR ABSTRAÇÃO 38
3.3 ACELERAÇÃO POR DISTRIBUIÇÃO 40
3.4 ACELERAÇÃO BASEADA EM CASOS 41
3.5 ACELERAÇÃO DO APRENDIZADO COM HEURÍSTICAS 42
3.6 DISCUSSÕES 47

4 ROBOCUP 48

4.1 ROBOCUP SIMULAÇÃO 2D 48
4.1.1 Cliente 50
4.1.2 Servidor 50
4.1.3 Monitor 50
4.1.4 Treinador 52
4.2 UVA TRILEARN 52
4.3 TRABALHOS CORRELATOS 55
4.4 DISCUSSÕES 61

5 PROPOSTA 63

5.1 DISCRETIZAÇÃO DOS ESTADOS 64
5.2 MACRO AÇÕES 66
5.3 REFORÇOS 67
5.4 HEURÍSTICAS 67
5.5 DISCUSSÕES 68

6 EXPERIMENTOS E RESULTADOS NO DOMÍNIO DO FUTEBOL DE ROBÔS 69

6.1 APRENDIZADO DE UM GOLEIRO 70
6.2 APRENDIZADO DE UM GOLEIRO E UM ZAGUEIRO 78
6.3 APRENDIZADO DE UM ATACANTE. 87
6.4 APRENDIZADO DE DOIS ATACANTES 95
6.5 DISCUSSÕES 103

7 CONCLUSÃO E TRABALHOS FUTUROS 105

REFERÊNCIAS BIBLIOGRÁFICAS 109

APÊNDICE 1 115

APÊNDICE 2 123

LISTA DE FIGURAS

Figura 2.1 - Aprendizado por Reforço. 24
Figura 3.1 – Propagação de recompensas com $\lambda = 0$ (PEGORARO, 2001). 37
Figura 3.2 – Propagação de recompensas com $\lambda = 1$ (PEGORARO, 2001). 37
Figura 3.3 – Camadas do CMAC. 39
Figura 3.4 – Diagrama em Forma de Blocos do CMAC. (SUTTON; BARTO, 1998). 40
Figura 3.5 - Descontinuidades da Função Valor (DRUMMOND, 2002). 41
Figura 3.6 – Estando no estado S e desejando ir para o estado Sd. 44
Figura 3.7 - Imagens dos Valores *V(s).* 45
Figura 3.8 – Resultado. 46
Figura 3.9 - Regiões Encontradas no Ambiente do Agente. 46
Figura 4.1 - Arquitetura do Sistema de Simulação *SoccerServer* (REIS, 2003). 48
Figura 4.2 – Campo virtual (MAO et al.,2003). 49
Figura 4.3 - Soccer monitor clássico, time FeiSim06. 51
Figura 4.4 - Arquitetura do UVA Trilearn (BOER; KOK, 2002 p. 45). 53
Figura 4.5 – UML do UVA Trilearn (BOER, KOK, 2002 p. 48). 53
Figura 4.6 – Área do jogo (STONE, SUTTON; KUHLMANN, 2005). 56
Figura 4.7 – Determinação dos estados (STONE; SUTTON; KUHLMANN, 2005). 57
Figura 4.8 - Variáveis de estado. 58
Figura 4.9 – Representação da localização do índice id4 da chave. 60
Figura 5.1 – Funcionamento do agente. 64
Figura 5.2 – Estados discretizados. 66
Figura 6.1 - Grade utilizada para o agente goleiro. 71
Figura 6.2a – Ações goleiro. 73
Figura 6.2b – Ações goleiro. 73
Figura 6.2c – Ações goleiro. 73
Figura 6.2d – Ações goleiro. 73
Figura 6.3 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para o agente goleiro. 75
Figura 6.4 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para o agente goleiro com barras de erro. 75
Figura 6.5 - Resultado do teste *t* de Student para os algoritmos *Q-Learning* e HAQL. 76

Figura 6.6 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente goleiro.. 77
Figura 6.7 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente goleiro com barras de erro. 77
Figura 6.8 – Grade utilizada pelos agentes goleiro e zagueiro. 79
Figura 6.9a – Ações Goleiro e Zagueiro 81
Figura 6.9b – Ações Goleiro e Zagueiro 81
Figura 6.9c – Ações Goleiro e Zagueiro 81
Figura 6.9d – Ações Goleiro e Zagueiro 81
Figura 6.10 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para o agente goleiro e zagueiro. 83
Figura 6.11 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para o agente goleiro e zagueiro, com barras de erro. 83
Figura 6.12 - Resultado do teste *t* de Student para os algoritmos *Q-Learning* e HAQL. 84
Figura 6.13 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente goleiro. 85
Figura 6.14 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente goleiro, com barras de erro. 85
Figura 6.15 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente zagueiro. 86
Figura 6.16 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente zagueiro, com barras de erro. 86
Figura 6.17 – Grade utilizada pelo agente atacante. 88
Figura 6.18a – Ação do atacante 90
Figura 6.18b – Ação do atacante 90
Figura 6.18c – Ação do atacante 90
Figura 6.18d – Ação do atacante 90
Figura 6.19 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para o agente atacante. 92
Figura 6.20 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para o agente atacante, com barras de erro. 92
Figura 6.21 - Resultado do teste t de Student para os algoritmos Q-Learning e HAQL. 93
Figura 6.22 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente atacante. 94

Figura 6.23 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente atacante, com barras de erro. 94
Figura 6.23a – Ações dos atacantes 97
Figura 6.23b – Ações dos atacantes 97
Figura 6.23c – Ações dos atacantes 97
Figura 6.23d – Ações dos atacantes 97
Figura 6.24 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para os atacantes. 99
Figura 6.25 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para os agentes atacantes., com barras de erro. 99
Figura 6.26 - Resultado do teste *t* de Student para os algoritmos *Q-Learning* e HAQL. 100
Figura 6.27 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente atacante 1 101
Figura 6.28 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente atacante 1, com barras de erro. 101
Figura 6.29 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente atacante 2 102
Figura 6.30 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente atacante 2, com barras de erro. 102
Figura 6.31 – Aprendizado de baixo nível. 106
Figura A1.1 - Zona 10. 115
Figura A1.2 - Ação Parado. 116
Figura A1.3 - Ação Interceptar Bola. 116
Figura A1.4 - Ação Conduzir Bola. 117
Figura A1.5 - Ação Passe. 117
Figura A1.6 - Ação Pegar Bola. 118
Figura A1.7 - Ação Posição Defesa. 118
Figura A1.8 - Zona 14. 119
Figura A1.9 - Ação Parado. 120
Figura A1.10 - Ação Interceptar Bola. 120
Figura A1.11 - Ação Conduzir Bola. 121
Figura A1.12 - Ação Passe. 121
Figura A1.13 - Ação Chutar Longe. 122
Figura A1.14 - Ação Marcar. 122

LISTA DE TABELAS

Tabela 4.1 – Reforços. 58
Tabela 4.2 – Dígitos da chave (MAUSBERG, 2005). 59
Tabela 4.3 – Representação das distâncias. 60
Tabela 4.4 – Representação dos ângulos. 61
Tabela 5.1 – Macro ações. 66
Tabela 5.2 – Reforços. 67
Tabela 5.3 – Heurísticas. 67
Tabela 6.1 - Discretização dos ângulos de orientação do agente. 71
Tabela 6.2 - Chave de descrição do estado para o agente goleiro. 71
Tabela 6.3 - Exemplos de chaves de descrição de estado para o agente goleiro. 71
Tabela 6.4 - Reforços para o agente goleiro. 72
Tabela 6.5 - Total de gols sofridos pelo agente goleiro (média e desvio padrão). 74
Tabela 6.6 - Discretização dos ângulos (em graus). 79
Tabela 6.7 - Chave de estado. 79
Tabela 6.8 - Reforços do zagueiro 80
Tabela 6.9 - Total de gols e desvio padrão. 82
Tabela 6.10 - Chave de estado para o atacante. 88
Tabela 6.11 - Discretização dos ângulos. 88
Tabela 6.12 - Discretização das distâncias. 88
Tabela 6.13 - Reforços para o atacante. 89
Tabela 6.14 - Total de gols feitos (média e desvio padrão). 91
Tabela 6.15 - Discretização das distâncias. 95
Tabela 6.16 - Chave de estado. 96
Tabela 6.17 – Reforços. 96
Tabela 6.18 - Total de gols feitos e desvio padrão. 98
Tabela 6.19 – Comparação dos Saldos de gols (média e desvio padrão). 103

LISTA DE ALGORITMOS

Algoritmo 2.1 - TD (0) (SUTTON,1998). 28
Algoritmo 2.2 - *Q-Learning* (WATKINS, 1992). 30
Algoritmo 2.3 - SARSA (SUTTON, 1996). 31
Algoritmo 2.4 - Minimax-Q (LITTMAN, 1994). 33
Algoritmo 3.1 - Q(λ) (WATKINS, 1989). 38
Algoritmo 3.2 - HAQL (BIANCHI, 2004). 45

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

H - Função heurística

π - Política

π^* - Política Ótima

Q - Função Valor-Ação

Q^* - Função Valor-Ação Ótima

V - Função Valor

V^* - Função Valor Ótima

LISTA ABREVIATURAS

MDP - *Markov Decision Process* – Processos de Decisão Markovianos

IA - Inteligência Artificial

HAL - *Heuristically Accelerated Learning* – Aprendizado Acelerado por Heurísticas

HAQL – *Heuristically Accelerated Q–Learning* – *Q–Learning* Acelerado por Heurísticas

AR - Aprendizado por Reforço

RNA - Redes Neurais Artificiais

SMA - Sistemas Multiagentes

1 INTRODUÇÃO

Inteligência Artificial pode ser definida como um estudo das computações com o objetivo de torná-las possíveis a perceber, a raciocinar e a agir (WINSTON, 1992). Sendo assim, um agente robótico inteligente pode ter a capacidade de aprender, se perceber, raciocinar e agir sobre o ambiente no qual se encontra. Existem inúmeras técnicas que propõem o aprendizado dos agentes inteligentes, conhecido como Aprendizado de Máquina.

Aprendizado de Máquina é o estudo da construção de programas de computador que aperfeiçoam seu desempenho na realização de alguma tarefa automaticamente, com a experiência (MITCHELL, 1997). Para a construção de um sistema de Aprendizado de Máquina é necessária a existência de um agente aprendiz. Segundo RUSSELL e NORVIG (2004, p. 613) "Um agente aprendiz é aquele que pode melhorar seu comportamento através do estudo diligente de suas próprias experiências". Assim, com um agente aprendiz, o sistema irá melhorar seu desempenho e também sua eficiência na resolução de uma determinada tarefa. O uso do Aprendizado de Máquina no meio cientifico, vêm se expandido cada vez mais, fato este que é percebido pela quantidade de artigos escritos a cada ano sobre o assunto. No último Simpósio Brasileiro de Inteligência Artificial, por exemplo, dos 62 artigos aceitos, 26 artigos (41.9%) foram sobre Aprendizado de Máquina.

Diversas áreas e teorias tiveram influência no aparecimento do Aprendizado de Máquina, dentre elas as mais importantes são (MITCHELL, 1997):

- Inteligência Artificial,
- Métodos bayseanos,
- Teoria de complexidade computacional,
- Teoria de controle,
- Teoria da informação,
- Filosofia,
- Psicologia e Neurologia,
- Estatística.

É possível classificar o Aprendizado de Máquina pelo modo que o agente recebe informações sobre o que irá aprender. Sendo assim, um aprendizado pode ser considerado como supervisionado ou não supervisionado. No aprendizado supervisionado, é possível determinar se o agente está ou não se comportando corretamente, validando ou não suas

ações, por meio de um supervisor que pode corrigir o agente nos momentos que ele não estiver agindo corretamente.

No aprendizado não supervisionado não existe maneira de saber *a priori* se o agente está agindo de maneira correta ou não, e por isto o agente precisa interagir inúmeras vezes e aprender as regras do ambiente, para poder dirigir suas ações a fim de chegar a um objetivo. Neste tipo de aprendizado não existe nenhum tipo de supervisor para corrigir o agente. Sendo assim, o agente aprende sem nenhum conhecimento do problema que deve resolver, nem conhecimento do domínio, que serão adquiridas automaticamente.

Quando se deseja solucionar uma variedade de problemas e quando não existem modelos disponíveis *a priori,* o Aprendizado por Reforço (AR) é uma técnica muito atraente para ser usada, pois o agente irá aprender a cumprir uma função de maneira correta em um ambiente desconhecido através de tentativa e erro.

No Aprendizado por Reforço, o agente aprende por meio da interação direta entre o agente e o ambiente, e das recompensas recebidas. Estas recompensas são dadas na forma de reforços positivos e negativos, que são usadas para sinalizar ao agente se ele está realizando as ações corretas ou não.

Uma desvantagem do Aprendizado por Reforço é o tempo que o agente leva para aprender; em muitos casos é necessário centenas de milhares de interações, ou episódios, para o agente poder aprender sobre o ambiente onde ele esta inserido. Devido a esta demora, é comum o uso de alguma forma de aceleração do aprendizado (BIANCHI, 2004).

Uma das maneiras de diminuir o tempo que agente leva para aprender no Aprendizado por Reforço é aplicar uma heurística (BIANCHI, 2004), isto é, uma ajuda que acelere o aprendizado. Utilizar uma heurística em um aprendizado por reforço, por exemplo, é fornecer uma ajuda ao agente para que ele possa conseguir chegar a um objetivo mais facilmente, utilizando um tempo menor para isto.

Uma heurística indica que uma ação deve ser executada em detrimento de outra. A principal vantagem no uso de heurísticas no Aprendizado por Reforço é que se ela estiver correta, irá causar uma aceleração no aprendizado, porém se estiver incorreta, causa uma atraso no sistema, mas mesmo assim o sistema irá aprender.

Neste trabalho as heurísticas devem influenciar um agente em quais ações ele deve realizar, dentre as diversas ações propostas e possíveis de serem realizadas. Um exemplo de uma heurística é quando o agente está na frente do gol adversário, a heurística deve influenciar o agente a chutar a bola no gol.

1.1 Domínio

Neste trabalho foi estudado o domínio de agentes robóticos inteligentes que irão atuar em um ambiente de futebol de robôs simulado denominado *RoboCup* 2D (NODA, 1995). Este sistema consegue simular um ambiente com 11 jogadores e também as condições encontradas em um jogo real.

Uma aplicação prática da utilização de agentes autônomos que possuem um comportamento inteligente pode ser visto em um projeto de um time de futebol simulado. Este comportamento inteligente engloba a utilização de cooperação entre os agentes, tendo em vista a realização de um tarefa que é computacionalmente complexa e com um grau de interação que depende de informações colhidas durante a execução de uma tarefa. (BIANCHI, 2001). Além disso, este domínio possui características bastante complexas, envolvendo necessidade de atuação em tempo real, tratamento de incertezas e ruídos das informações extraídas de ambientes dinâmicos, em geral imprecisas e incompletas.

O *RoboCup* 2D é uma das categorias das competições da *RoboCup*. A *RoboCup* (KITANO et al.,1995), foi inicialmente proposta para ser um meio de divulgação da robótica e da pesquisa em inteligência artificial, e fornecer meios para a avaliação de várias teorias, algoritmos e arquitetura, servindo também como uma ferramenta para a integração e estudos de como várias tecnologias podem trabalhar em conjunto (BOER; KOK, 2002).

Segundo KRAETZCHMAR (1998), a *RoboCup* deu a oportunidade dos pesquisadores trabalharem nas mais variadas áreas da inteligência artificial, como por exemplo em sistemas multiagentes, estratégia, aprendizado, visão, redes neurais, controle e em muitas outras, pois a criação de times para o *RoboCup* vai além da simples integração da Inteligência Artificial.

Cada dia mais é utilizado o *RoboCup* para estudos científicos, pois é uma ferramenta simples e de fácil acesso a todos e não necessita de robôs reais para testes. REIS (2003), em seu trabalho, descreve algumas futuras aplicações desenvolvidas no contexto do futebol de robôs simulado. Dentre as aplicações destaca-se (REIS, 2001; REIS, 2002; REIS, 2003, p. 290):

- *Routing* de pacotes de rede,
- Coordenação de pilotos de helicópteros e análise de cenários de guerra,
- Simulação de combate aéreo e de veículos (aéreos) inteligentes e autônomos,
- Coordenação automática de pessoas,

- Combate de catástrofes, tais como incêndios florestais ou no âmbito do *RoboCup Rescue*,
- Realidade virtual,
- Análise de jogos de futebol,
- Coordenação de AGVs (*Automated Guided Vehicles*),
- Robótica industrial e controle de processos industriais,
- Controle de satélites,
- Limpeza de minas e de lixo radioativo,
- Controle de robôs hospitalares,
- Controle inteligente de câmeras.

Outro fator que torna interessante o uso do *RoboCup* é a possibilidade de estudar como resolver um problema único que é composto pela existência de inúmeros sub-problemas (REIS, 2003), dando assim a possibilidade de poder resolver este problema por meio de diversas formas diferentes. Um exemplo é o uso do Aprendizado por Reforço para problema de como chutar uma bola (RIEDMILLER, 2000).

Atualmente, o interesse pelo estudo da robótica e inteligência artificial tem crescido em grande parte incentivados pelos campeonatos criados pelo *RoboCup*, dando assim também um propósito educacional ao evento, sendo mais fácil criar um envolvimento dos estudantes em investigações mais sérias com o uso do *RoboCup* (STONE et al.,1999).

1.2 Objetivo

A proposta principal deste trabalho é utilizar o Aprendizado por Reforço acelerado por heurísticas no domínio da robótica móvel, utilizando a plataforma do *RoboCup* 2D (NODA, 1995).

Para tanto, foi implementado um sistema onde agentes devem aprender a agir, apresentando diversos comportamentos relacionados ao jogo de futebol de robôs. As heurísticas foram criadas com base em regras bem conhecidas e utilizadas tradicionalmente no domínio do *RoboCup* 2D, como por exemplo, chute se a bola estiver perto, ir para bola, dentre outras. Foram realizadas comparações do funcionamento do sistema com e sem o uso de heurísticas. As principais dificuldades encontradas na realização deste trabalho, se deram

principalmente ao fato do tamanho da área que o agente precisava percorer, além da escolha de recompensas corretas e da criação de uma heurística correta.

Para este trabalho foi utilizado o algoritmo *Q-Learning*, esta escolha foi feita por ser um dos mais conhecidos algoritmos de Aprendizado por Reforço e por já ter sido aplicado em uma grande variedade de domínios (MITCHELL, 1997). Este algoritmo será aplicado ao agente, que deverá dentre as ações possíveis, aprender a escolher qual é a melhor ação, levando em conta o seu estado. Para a escolha do estado, foi realizado uma generalização de estados, devido ao agente precisar se deslocar em uma grande área, cujo tamanho traria problemas ao aprendizado. Nesta generalização de estados, foi criado uma chave que representava o estado, esta chave é formado por diversos valores, como por exemplo: posição do agente em campo, direção do agente, posição da bola no campo, dentre outras.

As experiências realizadas neste trabalho foram:

- Aprendizado de um goleiro: o goleiro deve aprender a defender a bola, contra um atacante. O atacante é constituído de um jogador, com uma programação de levar a bola para o gol e chutar.
- Aprendizado de um goleiro e um zagueiro: nesta experiência foi adicionado um zagueiro. Este zagueiro deve junto com o goleiro, tentar evitar que os dois atacantes façam gols.
- Aprendizado de um atacante: Situação inversa das experiências anteriores. Neste caso é o atacante que aprende a como fazer gols.
- Aprendizado de dois atacantes: Similar ao caso anterior, porém com dois atacantes trabalhando em conjunto para fazer gols.

Este trabalho foi influenciado a partir das idéias dos artigos de STONE, SUTTON e KUHLMANN (2005), KALYANAKRISHNAN, LIU e STONE (2006) e MAUSBERG (2005), que trabalharam com o *RoboCup*, desenvolvendo agentes que podiam dentro de um campo de futebol, aprender suas funções.

Este trabalho tem como contribuições:

- Aplicação de heurísticas e da aceleração do aprendizado para os agentes criados: tão importante quando poder fazer um agente aprender, é poder acelerar este aprendizado. Com o uso da aceleração por heurísticas, o agente pode estar mais preparado rapidamente para desenvolver sua função em campo.

- Desenvolvimento de técnicas para aprendizado em grandes espaços: grandes espaços de estados nunca foram bem vistos para o uso do aprendizado por reforço. Neste trabalho é mostrada uma maneira de como se pode aprender corretamente nestas situações.

1.3 Organização do Trabalho

Este trabalho está organizado da seguinte maneira: no capítulo 2 é realizada uma revisão sobre Aprendizado por Reforço, sobre os Processos de Decisão de Markov e são apresentados alguns dos algoritmos mais usados entre as técnicas de Aprendizado por Reforço. No capítulo 3 é visto o uso da aceleração do Aprendizado por Reforço e em seguida no capítulo 4 é feita uma explicação sobre *RoboCup* sobre as suas categorias e sobre alguns trabalhos nesta área.

O capítulo 5 apresenta a proposta principal deste trabalho, e no capítulo 6 são explicados os experimentos realizados e seus resultados. Finalmente, no capítulo 7, é apresentada uma conclusão e uma breve descrição dos possíveis trabalhos futuros.

2 APRENDIZADO POR REFORÇO

No Aprendizado por Reforço, um agente sem conhecimentos prévios aprende por meio de interações com o ambiente, recebendo recompensas por suas ações e assim descobrindo a política ótima para a resolução de um determinado problema. A suposição principal do Aprendizado por Reforço é a existência de um agente que pode aprender a escolher suas ações que resultarão em um melhor resultado futuro na realização de uma tarefa (PEGORARO, 2001).

Em uma grande variedade de domínios complexos, para os quais não se conhece a solução de um problema, uma opção é o uso do Aprendizado por Reforço (RUSSELL; NORVIG, 2004). O Aprendizado por Reforço é usado nas mais diversas áreas hoje em dia, como por exemplo, em jogos, aplicação de controle de robôs (RUSSELL; NORVIG, 2004), controle otimizado de elevadores (CRITE; BARTO, 1996), controle de tráfego veicular urbano (KRAUS, 2004), dentre outros.

O Aprendizado por Reforço é uma técnica de aprendizado não supervisionado devido a não existência de uma representação de pares de entrada e de saída. Para cada movimentação do agente não é fornecida nenhum tipo de informação externa que ajude seu deslocamento, tirando aquela que ele mesmo percebe da sua interação com o ambiente (KAELBLING; LITTMAN; MOORE, 1996).

O Aprendizado por Reforço funciona da seguinte maneira: em um ambiente, a cada intervalo de tempo o agente executa uma ação a_t. Esta ação é determinada pela política já aprendida e faz o agente ir para o estado s_{t+1} e tendo em vista a recompensa $r_{st,at}$ que irá ganhar. A recompensa pode ser dada por valores positivos ou negativos, indicando se o agente está seguindo corretamente para o objetivo ou não. A Figura 2.1 apresenta um esboço do funcionamento de um agente no Aprendizado por Reforço.

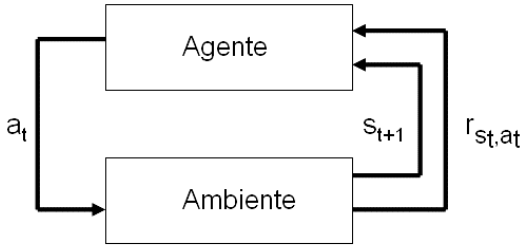


Figura 2.1 - Aprendizado por Reforço.

Seguindo este ciclo de funcionamento, o agente poderá aprender uma política ótima para o problema. Esta política tem o intuito de maximizar a soma das recompensas que foram recebidas durante o processo de aprendizado do agente.

2.1 Processo de Decisão de Markov e Programação Dinâmica

Utilizando conceitos do Processo de Decisão Markoviano (*Markov Decision Process* – MDP), é formalizado o Aprendizado por Reforço. Os MDPs constituem uma teoria bem estabelecida matematicamente e possui fundamentos sólidos, o que ajuda no estudo do Aprendizado por Reforço. Uma das principais características de um MDP é a condição de Markov (RIBEIRO, 2002).

É especificado na condição de Markov que o estado de um sistema em um instante $t+1$, será uma função que somente irá depender da observação do estado atual e da ação que o agente tomar neste estado, com desconto de perturbações aleatórias, sendo o sistema independente de sua história.

É possível definir formalmente um MDP (LITTMAN, 1994; KAELBLING, LITTMAN; MOORE, 1996) pela quádrupla (S, A, T, R) onde:

- S: é o conjunto finito de estados do ambiente,
- A: é o conjunto finito das ações que podem ser realizadas por um agente,
- T: $S \times A \rightarrow \Pi(S)$: é a função de transição de estado. Sendo $\Pi(S)$ uma distribuição de probabilidade para cada estado e ação,
- R: $S \times A \rightarrow R$: é a função de recompensa.

Uma política π é um conjunto de regras que irá determinar qual ou quais estados que o agente percorrerá, através das análises das recompensas. O objetivo do Aprendizado por Reforço é aprender qual é a melhor política para realizar as transições, a política ótima π^* (nem sempre encontrada). É por meio da política ótima que o agente maximiza a função de recompensa, conseguindo chegar melhor e mais facilmente ao resultado esperado (KAELBLING; LITTMAN; MOORE, 1996).

Dependendo da maneira que as transições ocorrem o MDP pode ser denominado como determinístico ou não determinístico. O MDP será determinístico caso possua apenas uma transição válida resultando em apenas uma ação resultante também válida, e será não determinístico se quando um agente executar uma transição resultar em uma coletânea de ações diferentes, não possíveis de serem verificadas antes da transição. Um bom exemplo de

não determinismo é o jogar de uma bola de basquete em um cesto, a bola pode entrar direto, bater na cesta e entrar, bater na cesta e não entrar ou simplesmente não entrar, porém o resultado só será verificado após a realização da ação.

Quando se segue uma determinada política π a partir de um estado inicial *s* é possível computar o custo esperado para aquela política. Esta função de custo é chamada de Função Valor Cumulativo Esperado e ela especifica um valor numérico para o estado. Este valor numérico compreende a soma dos reforços futuros esperados, desde o estado atual, até um estado terminal, seguindo uma política ótima.

As recompensas recebidas no Aprendizado por Reforço são amortizadas por um fator de desconto γ (onde $0 \leq \gamma \leq 1$), que determina o valor da recompensa imediata com relação as recompensas futuras. Assim sendo, é definido um modelo que considera a recompensa que pode ser recebida a longo tempo, que recebe o nome de Modelo de Horizonte Infinito com Desconto, sendo que o valor acumulativo com desconto é dado por (WATKINS, 1989):

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \qquad (2.1)$$

onde:

- r_{t+1} : é uma seqüência de reforços que são recebidos a partir do estado s_t e usando de forma repetida para selecionar as ações uma política π,
- γ: é um fator de desconto, sendo $0 \leq \gamma \leq 1$.

A política ótima π^* que maximiza $V^{\pi}(s)$ é (WATKINS, 1989):

$$\pi^* = argmax_{\pi} V^{\pi}(s), \forall s. \qquad (2.2)$$

Uma maneira bem conhecida de se encontrar a política ótima por um MDP é utilizando técnicas de Programação Dinâmica (PD) (BERTSEKAS, 1987). A coleção de técnicas que envolvem a Programação Dinâmica possui uma rigorosa base matemática, mas são conceitualmente simples, sendo um dos pontos fundamentais o Princípio da Otimalidade de Bellman (BELLMAN, 1957).

O Princípio da Otimalidade de Bellman diz que, se um agente segue uma política ótima que vai de um estado inicial até um estado final, passando por um estado particular intermediário, é equivalente a seguir a melhor política que se inicia do estado inicial e vai até o intermediário, seguida da melhor política que vai do estado intermediário até o estado final.

Com base no Princípio da Otimalidade de Bellman é possível afirmar que dado um agente em um estado s_t é possível definir a política de ações ótimas. Para isto é necessário

encontrar a ação que leva ao melhor estado s_{t+1} e neste estado seguir a política ótima até o último estado. Por meio da existência de inúmeros teoremas dentro da área de programação dinâmica, é garantido que, no momento em que o valor *V* não tem mais a possibilidade de melhorar, a política encontrada é ótima (BERTSEKAS, 1987; KAELBLING; LITTMAN; MOORE, 1996).

2.2 Algoritmo Diferença Temporal – TD (0)

Este algoritmo foi apresentado por SUTTON (1988), e sem ter a necessidade de um modelo do ambiente, calcula a soma dos reforços futuros esperados para cada estado *V(s)*. Os valores de *V(s)* são calculados a partir do estado sucessor e da recompensa recebida. Todos os estados percorridos pelo agente formam um caminho com uma seqüência de recompensas. O Algoritmo de Diferença Temporal trabalha com uma média ponderada com os valores das recompensas obtidas na trajetória realizada pelo agente. E desta forma, um estado é atualizado por esta média dos valores dos estados futuros. O TD (0) representa somente o primeiro estado futuro, e sua atualização usa a seguinte regra (SUTTON, 1998):

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1})\right] \tag{2.3}$$

Onde:

- s_t : é o estado atual,
- r_t : é o reforço ao se realizar uma ação em um determinado estado,
- s_{t+1} : é o próximo estado,
- α : é a taxa de aprendizado.

Para que tenha uma atualização de V(s) também para os outros valores de reforços realizadas em outros períodos no estado s, é utilizado o fator α, também chamado de taxa de aprendizado. O fator de α, onde $0 \leq \alpha \leq 1$, representa uma fração das recompensas futuras esperadas. Este fator α pode ter um valor constante para ambientes não estacionários ou um valor que pode ser reduzido lentamente, fazendo que a política convirja para um política ótima durante o aprendizado (JAAKKOLA et al., 1994). O algoritmo TD (0) que obtém a função valor ótima é apresentado no algoritmo 2.1

Algoritmo 2.1 - TD (0) (SUTTON,1998).

Inicialize $Q_t(s_t,a_t)$ arbitrariamente
Repita:
Visita o estado s_t.
Selecione uma ação a de acordo com a regra de transição de estados.
Execute a ação a_t.
Receba o reforço $r(s_t,a_t)$ e observe o próximo estado s_{t+1}.
Atualize os valores de $Q_t(s_t,a_t)$ de acordo com a regra de atualização:
$Q_{t+1}(s_t,a_t) \leftarrow Q_t(s_t,a_t) + \alpha [r(s_t,a_t) + \gamma max Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)]$
Atualize o estado $s_t \leftarrow s_{t+1}$.
Até que algum critério de parada seja atingido.

2.3 Q-Learning

Entre os diversos algoritmos de Aprendizado por Reforço existentes, o mais conhecido é o *Q-Learning* (WATKINS, 1989; WATKINS; DAYAN, 1992), considerado por vários autores um algoritmo de fácil implementação (KAELBLING; LITTMAN; MOORE, 1996) e por este motivo, utilizado em uma grande quantidade de domínios (MITCHELL, 1997). No *Q-Learning*, para cada ação realizada pelo agente, é computado sua recompensa e o valor esperado ao seguir a melhor política com um desconto. Esta política é aprendida por meio da interação com o ambiente e, assim, aprendidos quais as melhores ações para chegar a um objetivo. A informação da política é armazenada em uma matriz *Q(s,a)*, que guarda os valores estimados para cada par de estado,ação (UTHER; VELOSO, 2003).

O *Q-Learning* tem como principal característica a capacidade de aprender por meio da interação de uma política ótima π^*, quando não existe um modelo do sistema. Esta política ótima é encontrada escolhendo a ação que maximiza os valores de *Q,* para um determinado estado. O valor de custo de um estado (função valor *V(s)*) também é encontrado facilmente, sendo que o custo de um estado *V(s)* é o valor máximo de $Q(s_t,a_t)$ de um estado s_t, para todas as ações possíveis de serem executadas nesse estado. O algoritmo do *Q-Learning* é apresentado no algoritmo 2.2. Nesta técnica de aprendizado uma função de estimação *Q* que é calculada por meio da exploração on-line do agente. A função de estimação *Q* é computada por meio da equação (WATKINS, 1989):

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \right] \quad (2.4)$$

Onde:

- s_t : é o estado atual,
- a_t : é a ação que será realizada em s,
- $r(s_t,a_t)$: é o reforço ao se realizar uma ação em um determinado estado,
- s_{t+1} : é o próximo estado,
- a_{t+1} : é a ação a ser realizada no próximo estado,
- y : é o fator de desconto,
- α : é a taxa de aprendizado.

O parâmetro γ é compreendido entre os valores de 0 a 1. Caso o valor de γ seja muito perto de 0, o agente tende a considerar apenas valores imediatos de recompensa. Caso os valores sejam muito perto de 1, o agente considera as recompensas futuras com o maior peso.

Este algoritmo possui uma propriedade muito importante: pode ser usado qualquer tipo de estratégia de exploração durante um processo interativo de aproximação da função Q. Porém deve-se garantir que cada par de estados seja visitado inúmeras vezes, pra que a convergência de Q para Q^* seja garantida de maneira gradual e lenta (MITCHELL, 1997). Uma estratégia para a escolha das ações bastante utilizada em implementações do *Q-Learning* é a exploração aleatória ε-*Greedy*, na qual o agente executa a ação com maior valor Q com probabilidade $1 - \varepsilon$ e escolhe uma ação aleatória com probabilidade ε. Neste caso, a transição de estados é dada pela seguinte regra (WATKINS, 1989):

$$\pi(s_t) = \begin{cases} a_{random} & \text{se } q \leq \epsilon, \\ \arg\max_{a_t} Q_t(s_t, a_t) & \text{caso contrário,} \end{cases} \tag{2.5}$$

Onde:

- q : é um valor escolhido de maneira aleatória com distribuição de probabilidade uniforme sobre [0,1] e ε ($0 \leq \varepsilon \leq 1$) é o parâmetro que define a taxa de exploração: quando menor o valor de ε, menor a probabilidade de se fazer uma escolha aleatória,
- α_{random} : é uma ação aleatória selecionada entre ações possíveis de serem executadas no estado s_t.

Algoritmo 2.2 - *Q-Learning* (WATKINS, 1992).

Inicialize $Q_t(s_t,a_t)$ arbitrariamente
Repita:
Visita o estado s_t.
Selecione uma ação a de acordo com a regra de transição de estados.
Execute a ação a_t.
Receba o reforço $r(s_t,a_t)$ e observe o próximo estado s_{t+1}.
Atualize os valores de $Q_t(s_t,a_t)$ de acordo com a regra de atualização:
$Q_{t+1}(s_t,a_t) \leftarrow Q_t(s_t,a_t) + \alpha [r(s_t,a_t) + \gamma max Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)]$
Atualize o estado $s_t \leftarrow s_{t+1}$.
Até que algum critério de parada seja atingido.

O *Q-Learning* em ambientes determinístico pode ser utilizado com o fator de aprendizado α com valor fixo em 1, isto é, atualizar o valor de $Q_t(s_t,a_t)$ com o valor total da recompensa encontrada e do valor do valor futuro esperado descontado, "Isso ocorre porque, após a estabilização dos valores de $Q_t(s_t,a_t)$, uma ação em um estado terá sempre a mesma recompensa e o mesmo valor futuro esperado" (PEGORARO, 2001, p 31).

2.4 SARSA

Inicialmente denominado de *Q-Learning* modificado (RUMMERY; NIRANJAN, 1994), este algoritmo recebeu o nome de SARSA por SUTTON (1996) e tem como proposta o aprendizado da política no tempo de execução, podendo levar significativas melhoras em um aprendizado em casos onde as penalidades devem ser evitadas (PEGORARO, 2001). Neste algoritmo o valor da política é estimado ao interagir com o ambiente. O SARSA retira a maximação das ações existente no *Q-Learning*, sendo que a nova equação fica (SUTTON, 1996):

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \right] \quad (2.6)$$

Onde:

- s_t : é o estado atual,
- a_t : é a ação que será realizada em s_t,
- $r(s_t,a_t)$: é o reforço ao se realizar uma ação em um determinado estado,
- s_{t+1} : é o próximo estado,
- a_{t+1} :é a ação a ser realizada no próximo estado,
- y : é o fator de desconto,
- α : é o taxa de aprendizado.

O SARSA alcança um rendimento melhor que o *Q-Learning* quando o ambiente apresenta somente penalidades ou com penalidades distantes das recompensas, ou em ambientes com um grande conjunto de ações (SUTTON, 1996). O algoritmo SARSA é apresentado no algoritmo 2.3.

Algoritmo 2.3 - SARSA (SUTTON, 1996).

Inicialize $Q_t(s_t,a_t)$ arbitrariamente
Repita:
Visita o estado s_t.
Selecione uma ação a de acordo com a regra de transição de estados.
Execute a ação a_t.
Receba o reforço $r(s_t,a_t)$ e observe o próximo estado s_{t+1}.
Atualize os valores de $Q_t(s_t,a_t)$ de acordo com a regra de atualização:
$Q_{t+1}(s_t,a_t) \leftarrow Q_t(s_t,a_t) + \alpha [r(s_t,a_t) + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)]$
Atualize o estado $s_t \leftarrow s_{t+1}$.
Até que algum critério de parada seja atingido.

A escolha das ações a_{t+1} é realizada por meio de uma distribuição de probabilidades, um exemplo é a Exploração de Boltzmann, que calcula uma probabilidade de executar a ação *a* no estado *s*, utilizando a equação descrita em 2.7. Caso seja utilizada uma política gulosa para a escolha das ações a_{t+1}, o algoritmo terá o mesmo funcionamento do *Q-Learning*, neste caso: $a_t = max Q_t(s_{t+1},a_{t+1})$. No aprendizado on-line o valor de Q^π é estimado a cada iteração a partir de π e a distribuição de probabilidade que define as escolhas das próximas ações é modificada ao mesmo tempo.

$$P(s,a) = \frac{e^{Q(s,a)/T}}{\sum_u e^{Q(s,u)/T}} \tag{2.7}$$

Onde:

- *P(s,a)* : é a probabilidade de selecionar uma ação *a* do agente, quando este esta no estado *s*,
- *u* : uma das possíveis ações que podem ser executados quando o agente esta no estado *s*
- *T*: valor que determina a exploração/explotação. Valores altos favorecem a exploração e valores baixos a explotação.

2.5 Minimax-Q

O Minimax-Q (LITTMAN, 1994) foi desenvolvido para resolver problemas que seguem os parâmetros dos Jogos de Markov quando o problema envolve dois jogadores. Sendo uma extensão da teoria dos jogos para MDPs, os Jogos de Markov permitem modelar agentes que competem entre si para realizar suas tarefas. O Minimax-Q foi proposto para solucionar os Jogos de Markov com soma zero. Esta especialização de Jogo de Markov muito estudada, considera a existência de apenas dois jogadores, isto é, um agente e um oponente com objetivos opostos permitido assim, a existência de apenas uma função de recompensa. O desempenho do agente depende diretamente da escolha do oponente, fazendo com que a escolha da política ótima seja uma tarefa não trivial.

O Minimax-Q tem um funcionamento muito semelhante ao *Q-Learning*, sendo a sua função valor de uma ação *a*, em um determinado estado *s*, quando o oponente toma a ação *o*, dada por (LITTMAN, 1994):

$$Q(s_t, a_t, o_t) = r(s_t, a_t, o_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, a_t, o_t, s_{t+1}) V(s_{t+1}) \tag{2.8}$$

E o valor de um estado poder ser determinado pela equação (LITTMAN, 1994):

$$V(s_t) = \max_{a_t \in \Pi(\mathcal{A})} \min_{o_t \in \mathcal{O}} \sum_{a_t \in \mathcal{A}} Q(s_t, a_t, o_t) \pi_a \tag{2.9}$$

Porém, quando os jogadores tomam suas ações em turnos, o jogador tem o conhecimento das ações do adversário, e assim, a política se torna determinística e a equação valor de um estado pode ser simplificada (LITTMAN, 1994):

$$V(s_t) = \max_{a_t \in \mathcal{A}} \min_{o_t \in \mathcal{O}} Q(s_t, a_t, o_t) \tag{2.10}$$

O Minimax-Q é usado nos mais diferentes domínios como, por exemplo, em futebol de robôs (LITTMAN, 1994; BOWLING; VELOSO, 2001; SAVAGAONKAR, 2002), busca de informação da internet (KHOUSSAINOV; KUSHMERICK, 2003; KHOUSSAINOV, 2003) e no uso de alocação de canais em redes de computadores (SAVAGAONKAR, 2002), entre outros. O Minimax-Q pose ser expresso como apresentado pelo algoritmo 2.4.

Algoritmo 2.4 - Minimax-Q (LITTMAN, 1994).

Inicialize $Q(s_t,a_t,o_t)$, $V(s_t)$ e $\pi(s_t,a_t)$.
Repita
  Visite o estado s_t.
  Selecione uma ação a de acordo com a política $\pi(s_t,a_t)$.
  Execute a ação a_t.
  Observe a ação o do adversário, receba o reforço $r(s_t,a_t,o_t)$.
  Observe o próximo estado s_{t+1}
  Atualize os valores de $Q(s_t,a_t)$ de acordo com a regra de atualização:
  $Q(s_t,a_t,o_t) \leftarrow Q(s_t,a_t,o_t)+\alpha[r(s_t,a_t,o_t)+\gamma V(s_{t+1})-Q(s_t,a_t,o_t)]$
  Compute os valores de $\pi(s_t,a_t)$ utilizando programação linear
  Atualize os valores de $V(s_t)$ de acordo com:
  $Vt(s) \leftarrow min_o \Sigma a' Q(s,a_{t+1},o_{t+1})\pi(s,a_{t+1})$
  Atualize $s_t \leftarrow s_{t+1}$
Até que algum critério de parada seja atingido

2.6 Discussões

Foram descritos neste capítulo alguns algoritmos de Aprendizado por Reforço e um resumo do Processo de Decisão de Markov, sem o intuito de ser uma referência completa, devido à existência de uma grande variedade de literatura para este fim, como as obras de KAELBLING, LITTMAN e MOORE (1996), SUTTON e BARTO (1998) ou RIBEIRO (2002). Os algoritmos descritos neste capítulo tiveram uma importância no estudo das experiências, em principal destaca-se o *Q-Learning* por ser um dos primeiros a serem utilizados nas experiências e por conter uma vasta quantidade de artigos com o seu uso.

Cada vez mais é estudado como agilizar o aprendizado e obter resultados em um tempo menor. No capítulo seguinte é apresentado um resumo de algumas técnicas que tem como objetivo realizar o aprendizado em um período menor de tempo.

3 ACELERAÇÃO DO APRENDIZADO POR REFORÇO

Apesar de ter sido aplicado em centenas de problemas de maneira bem sucedida, uma desvantagem do Aprendizado por Reforço é o tempo que o agente leva para aprender. Em muitos casos são necessárias centenas de milhares de interações (ou episódios) para o agente poder aprender sobre o ambiente onde ele está ou sobre a tarefa que deve realizar. Devido a esta demora, é comum o uso de alguma forma de aceleração do aprendizado.

Existem diversas maneiras de acelerar o Aprendizado por Reforço. Dentre as maneiras as mais conhecidas, pode-se citar: generalizações temporais (resultados são distribuídos pelos estados anteriormente executados), espaciais ou das ações (resultados são distribuídos para os estados usando medidas de similaridade do espaço de estados). Além da possibilidade das generalizações, também é possível o uso de abstrações (aprendizado em um nível mais elevado). Dentre as formas de abstrações possíveis, podem ser destacadas o uso de abstração temporal por meio de macro-ações, opções, entre outras. Outras maneiras são: a aceleração do aprendizado por meio de uma abordagem distribuída, uma utilização de uma base de casos ou usando heurísticas.

3.1 Aceleração por Generalização

Entre os algoritmos que podem ser classificados como utilizando acelerações por generalização, encontra-se, a arquitetura Dyna, proposta por SUTTON (1990). Esta arquitetura foi proposta como um método de se encontrar uma política ótima por meio do uso de um aprendizado do modelo do ambiente. Sendo assim, com as execuções das ações, o algoritmo irá aprendendo, por meio de interações do modelo de transições de estados $T(s_t,a_t,s_{t+1})$ e das recompensas $R(s_t,a_t)$ e irá usar a experiência adquirida e o modelo aprendido para ajustar a política.

Apesar do algoritmo Dyna requerer um número de vezes superior de tempo para executar uma interação, comparado ao *Q-Learning*, ele irá requerer um número menor de passos para chegar em sua política ótima (KAELBLING; LITTMAN; MOORE, 1996). A principal diferença no Dyna é que, além de atualizar o $Q(s_t,a_t)$ atual, é realizada também a atualização de *Q's* adicionais em um número *k* de estados escolhidos aleatoriamente.

Outro algoritmo importante que utiliza aceleração por generalização é o Q(λ) (WATKINS, 1989). Este algoritmo combina a noção de elegibilidade e o *Q-Learning*. No algoritmo Q(λ) quando $\lambda > 0$, sendo $0 < \lambda < 1$, as recompensas futuras de um período são

incluídas na atualização de $Q(s_t,a_t)$ e é aplicado uma regra de atualização para todos os estados que participaram em um determinado intervalo de tempo. A elegibilidade é reduzida por um fator $\gamma\lambda$ a cada intervalo de tempo e sempre que o estado é visitado, este recebe um incremento unitário como mostrado na equação 3.1 (WATKINS, 1989):

$$e(u,v) = \begin{cases} \gamma\lambda e(u,v) + 1 & \text{se } u = s_t \text{ e } v = a_t \\ \gamma\lambda e(u,v) & \text{caso contrário} \end{cases} \qquad 3.1$$

Quanto maior o valor de λ, maior será a influência das recompensas nos estados anteriores. Em outras palavras ao usar λ=0 apenas um estado futuro irá influenciar o estado anterior. Um exemplo dado por PEGORARO (2001) é demonstrado nas Figuras 3.1 e 3.2. Neste exemplo são considerados quatro estados (1, 2, 3, 4) e as transições acontecem do estado 1 para o 2, do estado 2 para 3 e finalmente do 3 para 4, considerando para análise apenas a recompensa que acontece na primeira interação. Assim, com λ=0 a recompensa obtida em r_3 só irá chegar ao estado 1 depois de três interações, ocorrendo de forma similar para r_2 (duas interações) e r_1 (uma interação) conforme mostra a Figura 3.1.

Se adotado λ=1, será necessário apenas uma interação para ser obtido o mesmo resultado, como mostrado na Figura 3.2. Assim, é possível concluir que valores grandes resultam em uma grande influência nas recompensas futuras e proporcionalmente valores pequenos influenciam pouco em recompensas futuras.

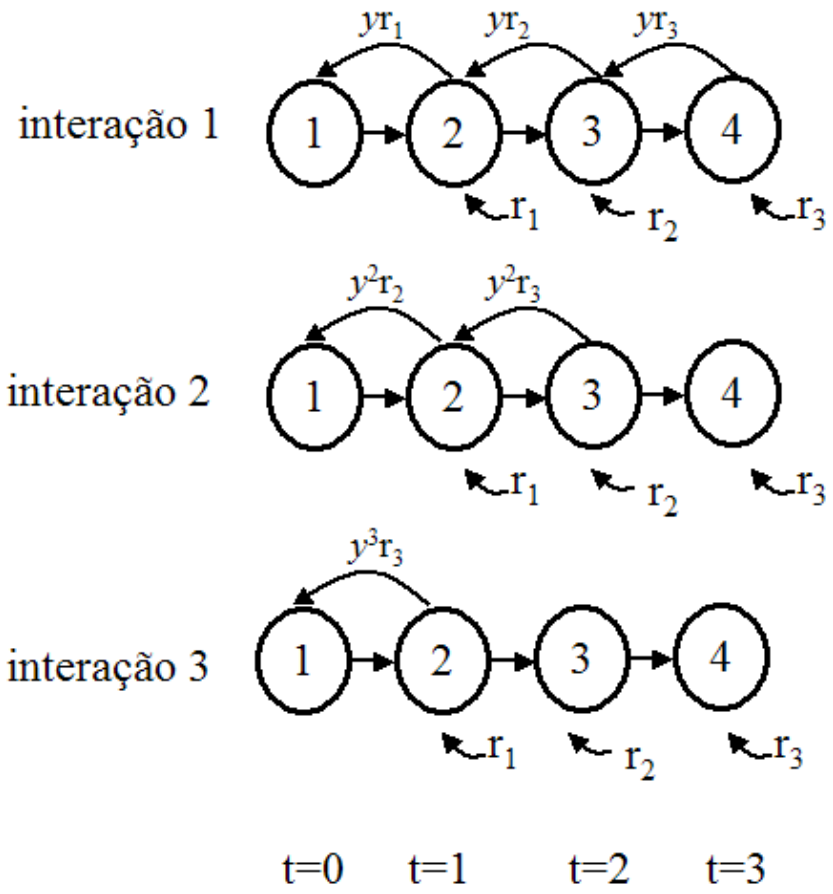


Figura 3.1 – Propagação de recompensas com $\lambda = 0$ (PEGORARO, 2001).

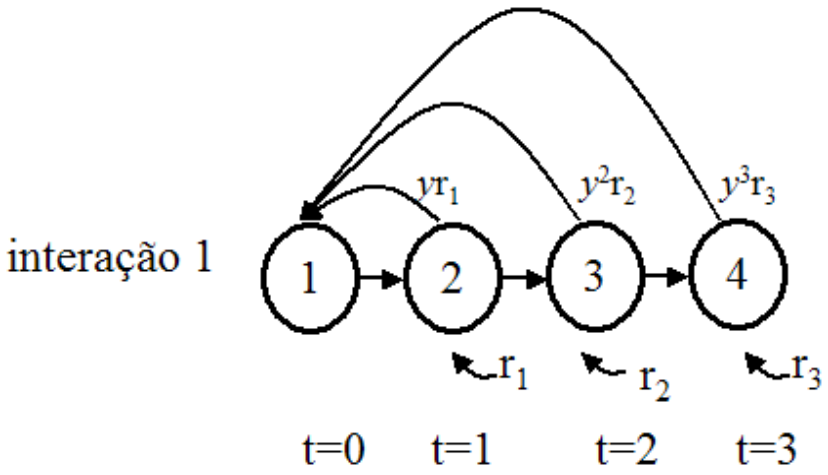


Figura 3.2 – Propagação de recompensas com $\lambda = 1$ (PEGORARO, 2001).

O algoritmo Q(λ) é mostrado no algoritmo 3.1. Uma boa escolha no valor do λ é de extrema importância, pois escolha de valores próximos de 1 ,por exemplo, podem causar um retardamento da convergência do algoritmo (PEGORARO, 2001).

Algoritmo 3.1 - Q(λ) (WATKINS, 1989).

Inicialize $Q(s_t,a_t)$ arbitrariamente e $e(s_t,a_t)=0$ para todos s_t,a_t
Repita para todos os episódios
Inicialize s_t,a_t
Repita:
Execute a ação a e observe r_t, s_{t+1}
Escolha a_{t+1} de s_{t+1} usando alguma política de Q (ex: ε-greedy)
$a^* \leftarrow argmaxQ(s_{t+1},a_{t+1})$
$\delta \leftarrow r+Q(s_{t+1},a^*)-Q(s_t,a_t)$
$e(s_t,a_t)=e(s_t,a_t)+1$
Para todo s, a:
$Q(s_t,a_t) \leftarrow Q(s_t,a_t)+\alpha.\delta.e(s_t,a_t)$
Caso $a_{t+1}=a^*$, então $e(s_t, a_t) \leftarrow \gamma\lambda e(s_t,a_t)$
Caso contrario $e(s_t,a_t) \leftarrow 0$
Até que algum critério de parada seja atingido.

Um outro tipo conhecido de aceleração por generalização é a generalização espacial, utilizando distribuição dos resultados de uma experiência para vários estados, por meio de alguma medida de similaridade. RIBEIRO (1998) propôs um algoritmo que realiza este espalhamento, denominado QS. Nele, o espalhamento espacial na função valor-ação é combinado com o *Q-Learning*. Neste sistema ao receber um reforço, pares de valor-ação que não estavam envolvidos são também atualizados. Outro algoritmo foi proposto por PEGORARO (2001), neste trabalho era utilizado o algoritmo QS que recebia informações prévias sobre o domínio, ajustando assim o espalhamento das atualizações segundo as similaridades entre estados e/ ou ações.

3.2 Aceleração por Abstração

Os algoritmos que utilizam, aceleração por abstração podem ser divididos em duas categorias: em abstração temporal e espacial. Ao usar abstrações temporais, tem-se como característica principal o aprendizado em um nível mais elevado, com funções mais complexas. Por exemplo, no domínio do futebol de robôs, ações de baixo nível como andar, girar e chutar podem ser substituídas por ações mais complexas como “chute a bola para o gol” ou “marcar o adversário”.

A abstração espacial é utilizada quando existe a necessidade da representação da função valor em espaços de estados muito grande. Um exemplo é um campo de futebol simulado. Neste caso existe um espaço de estados muito grande devido a grande quantidade

de possíveis estados que a bola e os jogadores podem ficar. Isto causa, entre outros problemas, um aumento de memória, proporcional ao tamanho de espaço usado.

Um dos procedimentos mais comuns é o uso de aproximadores de funções como redes neurais artificiais e CMAC (Cerebellar Model Articulation Controller). Este último, proposto por ALBUS (1975), tem como objetivo um modelo funcional do cerebelo e um método eficaz de aproximador de funções. Este modelo tem sido cada vez mais utilizado (GABRIELLI; RIBEIRO, 2003; SUTTON; STONE; KUHLMAN, 2004)

O CMAC é formado por um conjunto de entradas finitas de valores inteiros e apresenta na saída um vetor de saída com valores reais. Os valores de entrada irão "ativar" os conjuntos de entradas existentes que estão sobrepostos, formando o que se chama de telhas (tiles). Estas telhas estão colocadas de forma a existirem um deslocamento entre elas, com um espaço fixo, e o espaço de estados será particionado dependendo das dimensões das telhas.

A Figura 3.3 representa o funcionamento de um CMAC com 5 camadas de telhas. No exemplo da Figura um valor de entrada irá excitar as telhas que estão localizadas em diferentes camadas (cada telha excitada é representada por uma cor diferente). Na Figura 3.4 é mostrado um diagrama em forma de blocos do CMAC. O espaço de estados S é um conjunto de todos os possíveis vetores de entrada. Cada ponto do vetor S estará mapeado em um conjunto de pontos em C em uma memória virtual. O valor de saída correspondente a entrada, será a soma de todos os pontos em C, como mostrado no bloco A'.

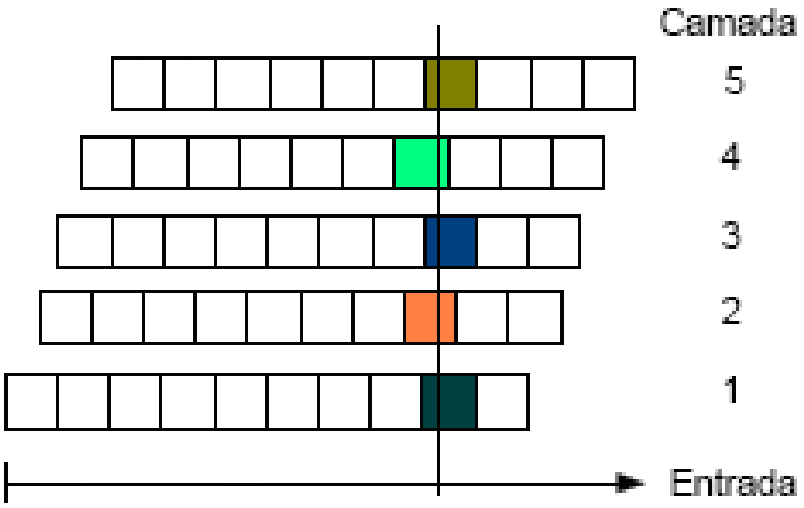


Figura 3.3 – Camadas do CMAC.

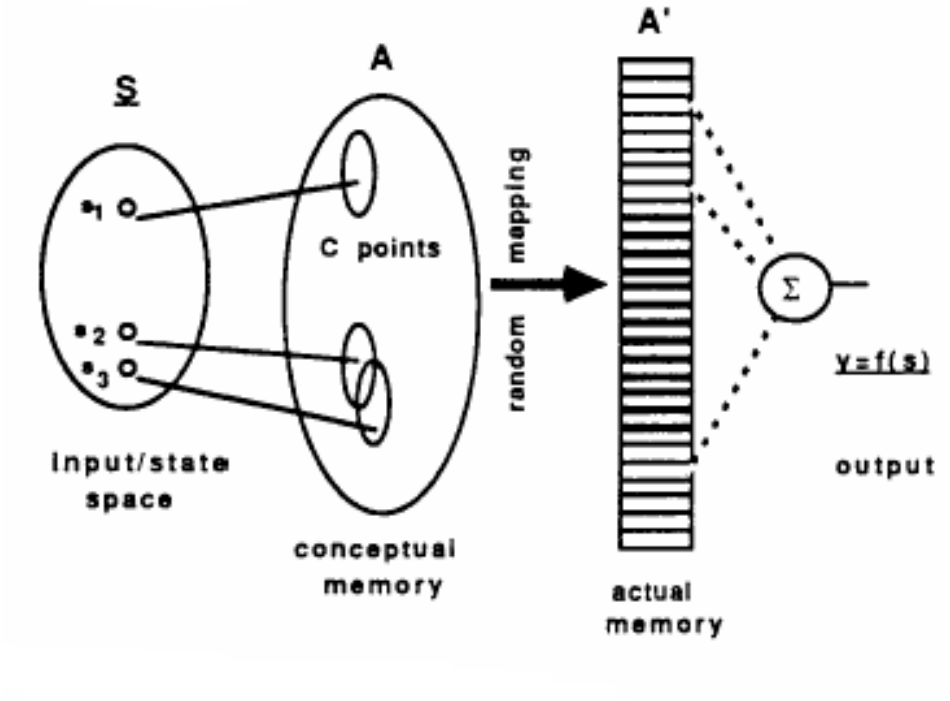


Figura 3.4 – Diagrama em Forma de Blocos do CMAC. (SUTTON; BARTO, 1998).

3.3 Aceleração por Distribuição

Por meio de abordagem distribuída é possível acelerar o Aprendizado por Reforço. Tentativas para este fim foram feitas por pesquisadores de Inteligência Artificial Distribuída (IAD) e de Sistemas Multiagentes (SMA), que começaram o usar vários agentes, em vez de apenas um.

Nestes sistemas é possível ter um aprendizado centralizado (um agente único) ou descentralizado (onde agentes se ajudam com um objetivo em comum). O laço comum nos SMA é a existência de algum nível de cooperação entre os agentes. Para isto pode existir uma comunicação com os agentes, distribuição do controle, entre outras.

Para sistemas que usam técnicas de aprendizado distribuídas foi definido o termo de Aprendizado por Reforço Multiagentes. Dentre as técnicas podemos citar o *Q-Learning* distribuído (MARIANO; MORALES, 2000, 2001), Dyna-Q Multiagente (WEIB, 2000) e otimização por colônia de formigas (DORIGO; CARO; GAMBARDELLA, 1999).

3.4 Aceleração Baseada em Casos

Um sistema que utiliza uma aceleração baseada em casos foi proposto por DRUMMOND (2002) e realiza uma aceleração do aprendizado por meio de partes de soluções previamente encontradas. Este método faz o uso das descontinuidades da função valor aprendidas, como mostrado na Figura 3.5, e conseguindo assim definir regiões correspondentes a sub-tarefas do sistema. Estas regiões são encontradas por meio de técnicas de contornos ativos (KASS; WITKIN; TERZOPOULOS, 1987) que são aplicados ao gradiente da função valor e particionando a função valor nos pontos de maior descontinuidade e assim definindo regiões que correspondem as sub-tarefas. Ao encontrar estas sub-tarefas o aprendizado é acelerado, usando informações armazenadas em uma base de casos.

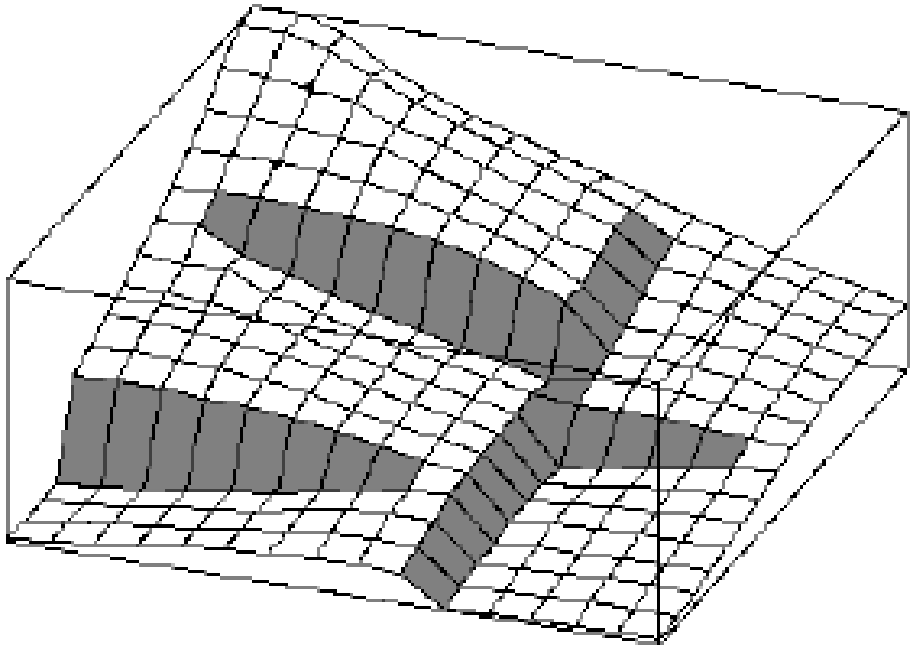


Figura 3.5 - Descontinuidades da Função Valor (DRUMMOND, 2002).

A base de casos é construída de maneira simples, usando o algoritmo *Q-Learning* para a solução do problema. Depois do uso do *Q-Learning* para aprender sobre o ambiente, o problema é dividido em sub-tarefas e então as soluções das sub-tarefas (funções valores) são armazenadas juntamente com os grafos que servirão como um índice da base de casos.

3.5 Aceleração do aprendizado com Heurísticas

Heurísticas são métodos ou principio de decisões, que indicam as melhores alternativas de ações que levem a solução de um problema mais facilmente (PEARL, 1984). Utilizar uma heurística em um Aprendizado por Reforço é fornecer uma ajuda ao agente para que ele possa conseguir chegar a um objetivo mais facilmente.

> A principal vantagem do uso da heurística é que ela limita o espaço de busca que o agente aprendiz explora. Neste sentido, a heurística funciona, no aprendizado, de maneira similar aos algoritmos de busca informada: indica o melhor caminho a seguir, reduzindo a busca (BIANCHI, 2004, p 142).

A classe de algoritmos de Aprendizado por Reforço que permite o uso de heurística para abordar o problema da aceleração do aprendizado é denominado aqui de "Aprendizado Acelerado por heurísticas" (*Heuristically Accelerated Learning – HAL*)

> De maneira mais formal um algoritmo da classe HAL pode ser definido como um modo de solucionar um problema modelável por um MDP que utiliza explicitamente a função heurística $H:S\times A\rightarrow R$ para influenciar o agente na escolha de suas ações, durante o aprendizado. $H(s_t,a_t)$ define a heurística que indica a importância de se executar a ação a estando no estado s. (BIANCHI,2004, página 57).

A equação 3.2, exemplifica a citação anterior, o valor de H representa a heurística que irá influenciar o agente em sua escolha de ações.

$$\pi(s_t) = \begin{cases} a_{random} & \text{se } q \leq \epsilon, \\ \arg\max_{a_t} \left[Q(s_t, a_t) + \ H_t(s_t, a_t) \right] & \text{caso contrário} \end{cases} \qquad 3.2$$

A política é determinada pela função heurística que está fortemente ligada a ela. Na heurística, a tomada de uma ação é realizado em detrimento de outra e por meio disto é obtido o que pode chamar de uma "Política Heurística", definida por BIANCHI (2004) como sendo uma política para a aceleração do aprendizado.

> Uma outra característica importante dos sistemas HAL é que a função heurística pode ser modificada a cada interação. Isto permite que a aceleração seja utilizada em um estágio precoce do aprendizado e modificada toda vez que mais informações se tornem disponíveis. (BIANCHI, 2004, p. 57).

Outras duas hipóteses levantadas por BIANCHI (2004) são que, conhecendo as informações existentes no domínio ou em estágios iniciais, pode ser definida uma heurística que poderá ser usada para acelerar o aprendizado. Devido a possível existência de uma grande quantidade de métodos que podem ser usados para extrair a função heurística, pode ser denominado através de uma maneira genérica como métodos de aprendizados por "Heurística

a partir de X" (*"Heuristic from X"*), em outras palavras, heurística a partir de qualquer informação. Outro ponto importante levantado por BIANCHI (2004) é que utilizar uma heurística adequada causa uma aceleração no Aprendizado por Reforço, porém o uso de uma heurística inadequada pode causar um atraso no sistema, mas não impede o algoritmo de convergir para uma política ótima.

Bianchi (2004) em seu trabalho, abordou alguns algoritmos acelerados por heurísticas, porém, o mais importante foi um algoritmo da classe HAL, denominado de, *Q-Learning* acelerado por heurísticas (*Heuristically Accelerated Q-Learning* - HAQL). O HAQL utiliza uma modificação da regra ε-*Greedy* para a regra de transição de estados, e incorpora a função heurística como uma somatória simples ao valor da função valor-ação. A regra de transição de estados é dada por (BIANCHI, 2004):

$$\pi(s_t) = \begin{cases} a_{random} & \text{se } q \leq \epsilon, \\ \arg\max_{a_t} \left[Q(s_t, a_t) + \xi H_t(s_t, a_t) \right] & \text{caso contrário.} \end{cases} \tag{3.2}$$

Onde:

- q : é um valor escolhido de maneira aleatória com distribuição de probabilidade uniforme sobre [0,1] e ε ($0 \leq \varepsilon \leq 1$) é o parâmetro que define a taxa de exploração: quando menor o valor de ε, menor a probabilidade de se fazer uma escolha aleatória,
- α_{random} : é uma ação aleatória selecionada entre ações possíveis de serem executadas no estado s_t.
- ξ: variável usada para ponderar a influência da função heurística.

E a heurística usada é definida como (BIANCHI, 2004):

$$H(s_t, a_t) = \begin{cases} \max_a Q(s_t, a) - Q(s_t, a_t) + \eta & \text{se } a_t = \pi^H(s_t), \\ 0 & \text{caso contrário.} \end{cases} \tag{3.3}$$

Onde:

- $a_t=\pi^H(s_t)$: heurística para a política a partir de um método "heurística a partir de X",
- η: valor pequeno.

Um exemplo dado por Bianchi (2004), descreve um estado com 4 possíveis ações e com os respectivos valores calculados [1,0 1,1 1,2 0,9], desejando-se que a ação selecionada seja a primeira, pode-se usar η=0,01, resultando em (1,2 – 1,0 + 0,01) $H(s_t,1)$ = 0,21 e igual a zero para as outras ações (a Figura 3.6 mostra este exemplo).

As únicas modificações realizadas no algoritmo HAQL em comparação ao *Q-Learning* é a modificação ao uso da função heurística para a escolha da ação a ser executada e a existência de um passo para atualização da função $H_t(s_t,a_t)$. O algoritmo do HAQL é mostrado no algoritmo 3.2 (BIANCHI, 2004).

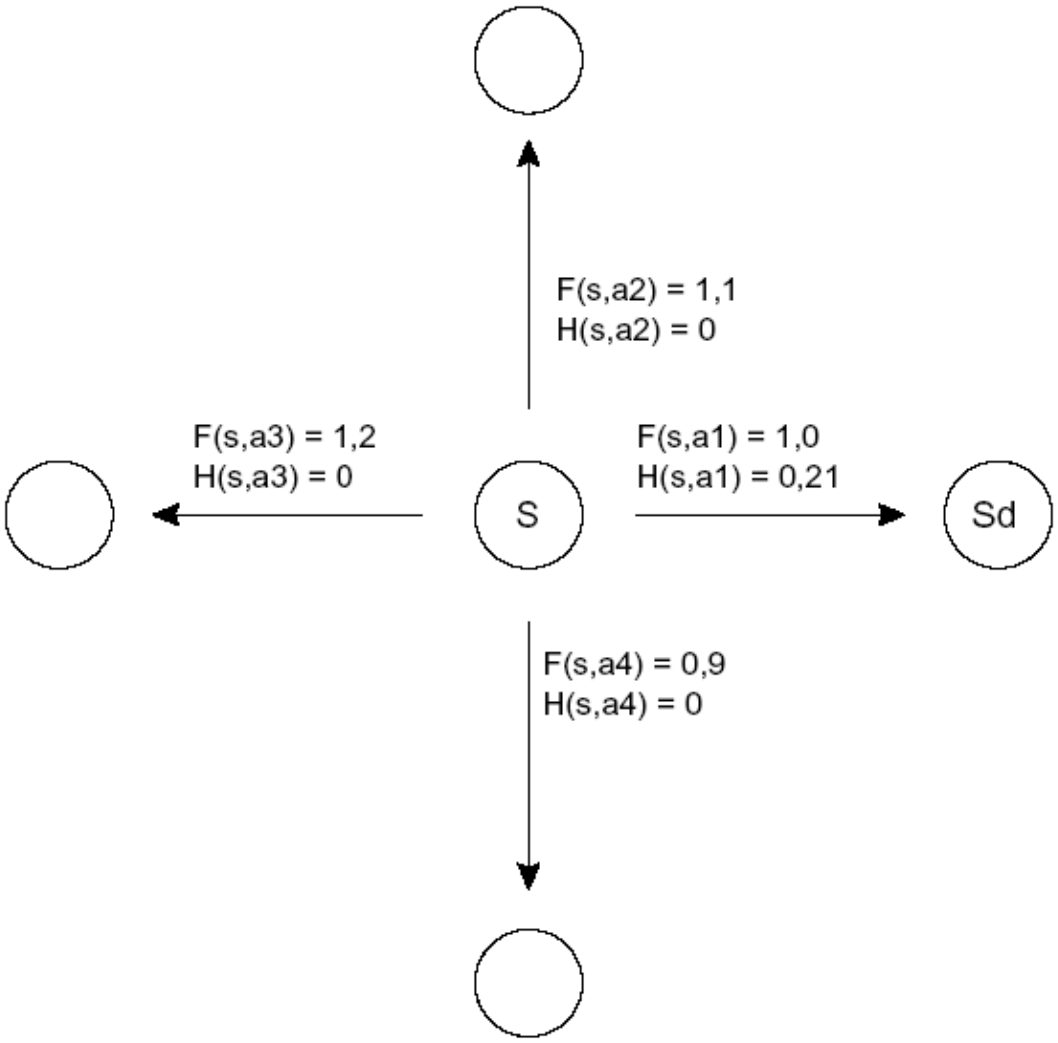


Figura 3.6 – Estando no estado S e desejando ir para o estado Sd.

Algoritmo 3.2 - HAQL (BIANCHI, 2004).

Inicialize $Q_t(s_t,a_t)$ arbitrariamente
Repita:
  Visita o estado s_t.
  Selecione uma ação a de acordo com a regra de transição de estados.
  Execute a ação a_t.
  Receba o reforço $r(s_t,a_t)$ e observe o próximo estado s_{t+1}.
  Atualize os valores de $H_t(s_t,a_t)$ utilizando um método "H a partir de X".
  Atualize os valores de $Q_t(s_t,a_t)$ de acordo com a regra de atualização:
  $Q_{t+1}(s_t,a_t) \leftarrow Q_t(s_t,a_t) + \alpha\ [\ r(s_t,a_t) + \gamma max Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)\ ]$
  Atualize o estado $s_t \leftarrow s_{t+1}$.
Até que algum critério de parada seja atingido.

Um método que permite definir as heurísticas a serem usadas, parecido com aceleração baseada em casos, foi proposto por Celiberto e Bianchi (2006). Este método utiliza técnicas de visão computacional para, a partir da imagem da tabela V, definir uma heurística. Para isto o algoritmo utiliza como entrada a tabela dos valores *V(s)* computada pelo algoritmo *Q-Learning*, a partir da tabela de valores *Q*. Esta tabela é uma imagem cujos valores dos pixel correspondem aos valores *V(s)*. Um exemplo da imagem criada é mostrado na Figura 3.7. Sobre esta imagem é aplicado um detector de bordas bem conhecido e usado na área de processamento de Imagens e Visão Computacional, o filtro Canny (FORSYTH; PONCE, 2003).

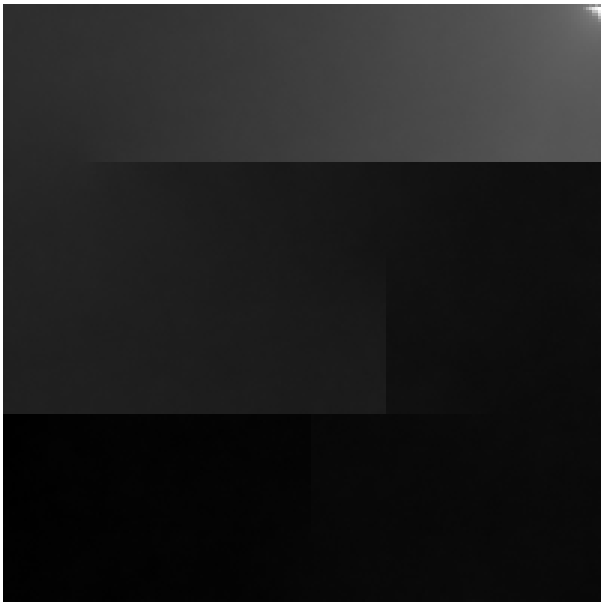


Figura 3.7 - Imagens dos Valores *V(s)*.

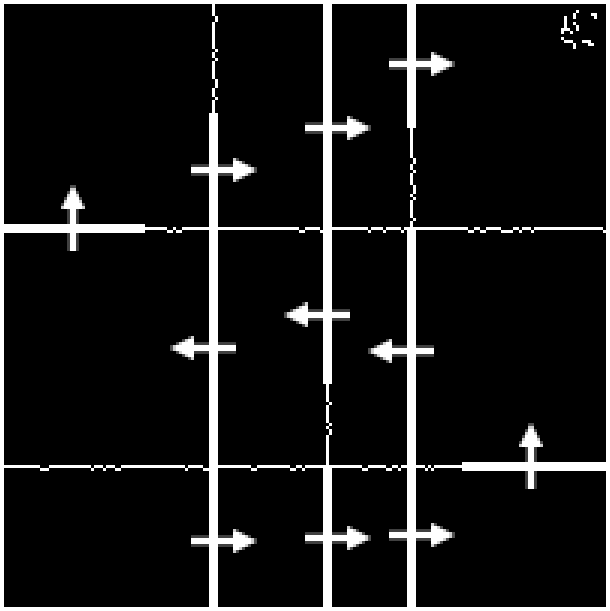


Figura 3.8 – Resultado.

É utilizado um algoritmo para reconhecimento da imagem, ou seja, retirar as informações da imagem e determinar a heurística para o melhor caminho para o agente. O resultado é mostrado na Figura 3.8. Com a ajuda de um algoritmo de rotulação de regiões (BALLARD; BROWN, 1982) a imagem é dividida em zonas, podendo ser usada para ajudar o agente que, conhecendo suas coordenadas saberá qual caminho ele deve tomar, como mostrado na Figura 3.9. Toda esta informação volta ao agente, ajudando na decisão e acelerando o aprendizado**.**

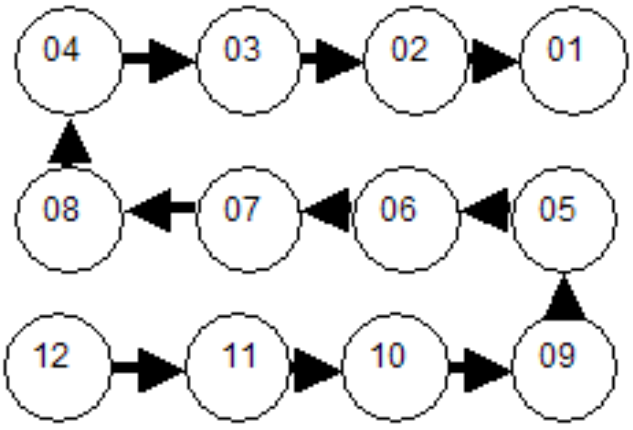


Figura 3.9 - Regiões Encontradas no Ambiente do Agente.

3.6 Discussões

O principal objetivo deste capítulo é fornecer um resumo sobre Aceleração do Aprendizado por Reforço. Atualmente, existem diversas outras formas além das descritas neste trabalho para acelerar o Aprendizado por Reforço. Os trabalhos aqui descritos foram os que incentivaram e deram oportunidade para a criação desta dissertação. Entre eles, pode-se notar que o Aprendizado por Reforço acelerado por heurísticas, que é o trabalho precursor desta dissertação, é o que apresenta como principal fator positivo a oportunidade de acelerar uma grande quantidade de algoritmos de AR, somente atuando na escolha das ações. No capítulo seguinte será discutido o uso do *Robocup* como a ferramenta de estudo para a realização desta dissertação.

4 ROBOCUP

Com os adventos dos campeonatos criados, surgiram diferentes tipos de categorias, cada uma tentando focar diferentes tipos de problemas. Dentre as categorias, podemos citar como as principais: *Small Size, Middle Size*, Simulação 2D, Simulação 3D, entre outras. A simulação 2D será dada mais importância neste trabalho.

4.1 *RoboCup* Simulação 2D

A categoria *RoboCup* 2D simulação foi originalmente desenvolvida pelo Dr. Itsuki Noda em 1993 (BOER; KOK, 2002). O *RoboCup* Simulação 2D é constituído por 11 jogadores sintéticos, criados por meio de softwares, que fazem uma partida de futebol simulado em um servidor designado por *SoccerServer* (NODA, 1995). 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 e de sistemas robóticos, com a adição de "ruídos" dependendo da distância entre jogador-bola ou jogador-jogador e de uma energia limitada (NODA; STONE, 2002). Esta energia limitada é reduzida com o passar do tempo em maior ou menor grau dependendo das ações do agente dentro do campo e restabelecida quando o agente pode "descansar". Basicamente é possivel dividir o *RoboCup* 2D em cliente, servidor (*SoccerServer*) e monitor (*SoccerMonitor*), como demonstrado na Figura 4.1.

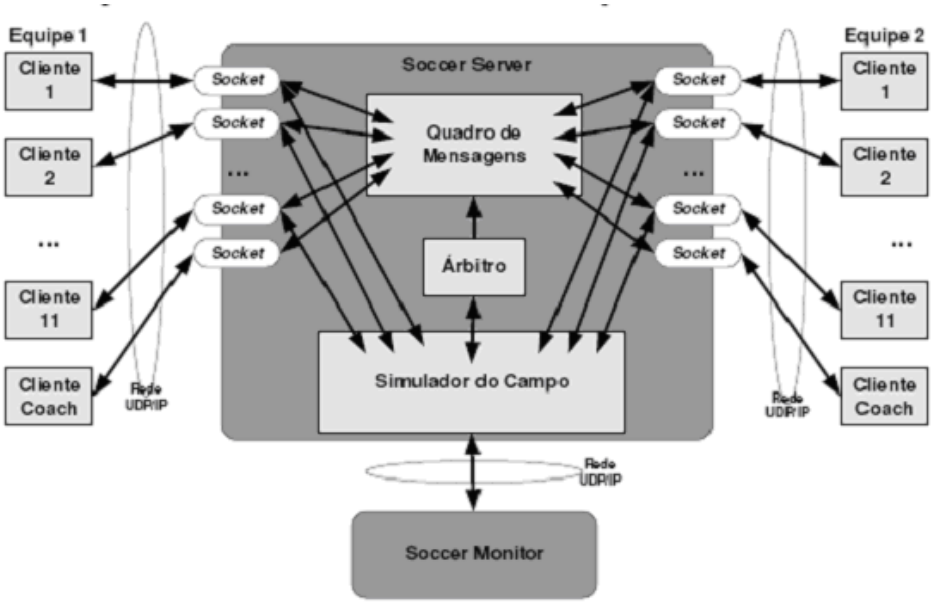


Figura 4.1 - Arquitetura do Sistema de Simulação *SoccerServer* (REIS, 2003).

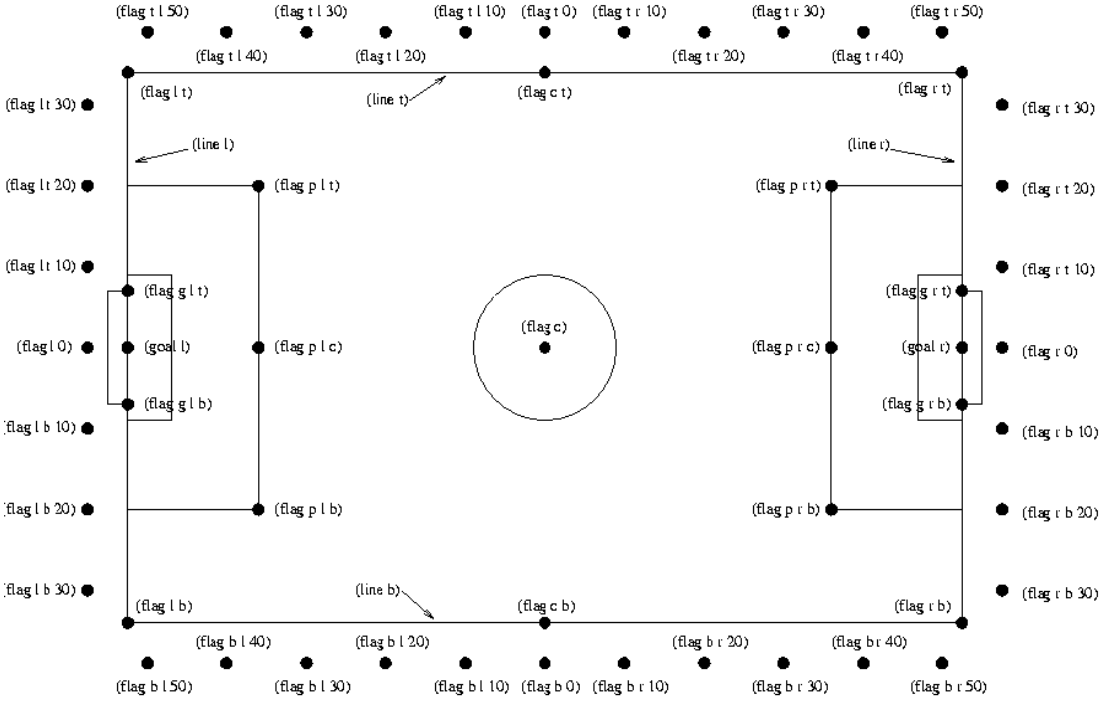


Figura 4.2 – Campo virtual (MAO et al.,2003).

Os jogos são realizados em um campo virtual de 105 por 68 metros. O campo contém linhas e bandeiras usadas para ajudar os jogadores a se localizarem. Na Figura 4.2 é mostrada uma representação deste campo com todas as bandeiras e linhas existentes.

A bandeira (fc), por exemplo, é uma bandeira localizada no meio do campo. As bandeiras (goal r) e (goal l) são bandeiras localizadas no centro do gol do lado esquerdo e do lado direito do campo. Existem bandeiras localizadas dentro do campo e fora do campo. A bandeira (flag bl20), por exemplo, é localizado a 5 metros de distância do limite inferior do campo e 20 metros à esquerda do centro do campo. As linhas (line t), (line b), (line r) e (line l) têm a função de limitar o campo, respectivamente, em acima, embaixo, à direita e à esquerda.

Uma partida realizada no simulador tem uma duração de aproximadamente 10 minutos. Este valor corresponde a 6.000 ciclos de simulação. Como em uma partida de futebol real, este valor da partida é dividido em dois tempos de 5 minutos. Quando ocorrem os empates, mais um tempo adicional de 3.000 ciclos é utilizado. Quando mesmo assim a partida continua em empate, é possível definir o jogo utilizando gol de ouro, vencendo o primeiro time que fizer um gol.

4.1.1 Cliente

O *RoboCup* 2D permite a competição entre os dois times de jogadores virtuais, sendo cada jogador controlado como sendo um cliente único autônomo, que irá reagir dependendo da situação na qual ele se encontra. Cada cliente envia informações ao servidor, que processa a informação recebida deste e de todos os outros clientes dos dois times. Toda comunicação entre o servidor e cliente é feito através de *sockets* TCP/IP.

O cliente irá atuar no campo de futebol pelas informações que são recebidas a ele. Estas informações são: auditiva (*hear*), visual (*see*) e sensorial (*sense_body*). Por meio destas informações recebidas o agente pode saber se está em uma posição favorável para fazer um gol ou em uma situação de desvantagem, e assim tentar evitar que receba um gol ou saber se têm energia para chegar no adversário ou receber alguma informação de outro jogador. Por outro lado, o cliente, ao processar a informação, necessita retornar ao servidor o que ele deseja fazer. O servidor então irá receber estas informações e atualizar o estado da partida. Esta troca de informações acontece deste modo até o final do jogo.

4.1.2 Servidor

O Servidor é o responsável por processar todas as ações requisitadas pelos clientes. Ele processa e atualiza o estado da partida e devolve aos clientes estas informações atualizadas. Todo o sistema que envolve o simulador trabalha em tempo real, com intervalos de tempos discretos, denominado de ciclo. Cada ciclo tem duração de aproximadamente 100ms e é neste intervalo de tempo que o servidor processa a informação recebida e devolve ao cliente a nova informação.

Cada cliente necessita mandar uma mensagem para porta 6000 do servidor ao começo de uma partida. Este responde aceitando a mensagem e fornece ao cliente uma porta especifica de comunicação que será usada entre servidor e o cliente por todo o jogo.

4.1.3 Monitor

Toda a partida é observada através do Soccer Monitor. Este programa é uma interface gráfica, do *RoboCup* 2D, usada para observar o que está acontecendo na simulação depois de uma atualização do ambiente pelo servidor. Na Figura 4.3 é apresentada a versão de monitor denominada clássica. Esta versão somente continha informação sobre o tempo de jogo e o

placar. Posteriormente foi criada uma nova versão de monitor que dispunha de uma quantidade maior de informações visíveis. Esta nova versão recebe o nome de *FrameView* (MERKE, 2002) e foi desenvolvida na Alemanha e permite fazer zoom no campo e saber de informações tais como energia (stamina) dos jogadores e visão.

Existe também a possibilidade de se conectar vários monitores ao servidor e assim é possível acompanhar o jogo de computadores diferentes. Além disto pode-se conectar equipamentos com sistemas operacionais diferentes ao servidor e acompanhar os jogos nos monitores desenvolvidos para este sistema, como é o caso dos monitores para Windows que podem muito bem visualizar jogos que estão sendo executados em servidores Linux. Porém não existe a necessidade de um monitor para ocorrer uma partida, pois as mesmas podem ser executadas sem a existência de um monitor.

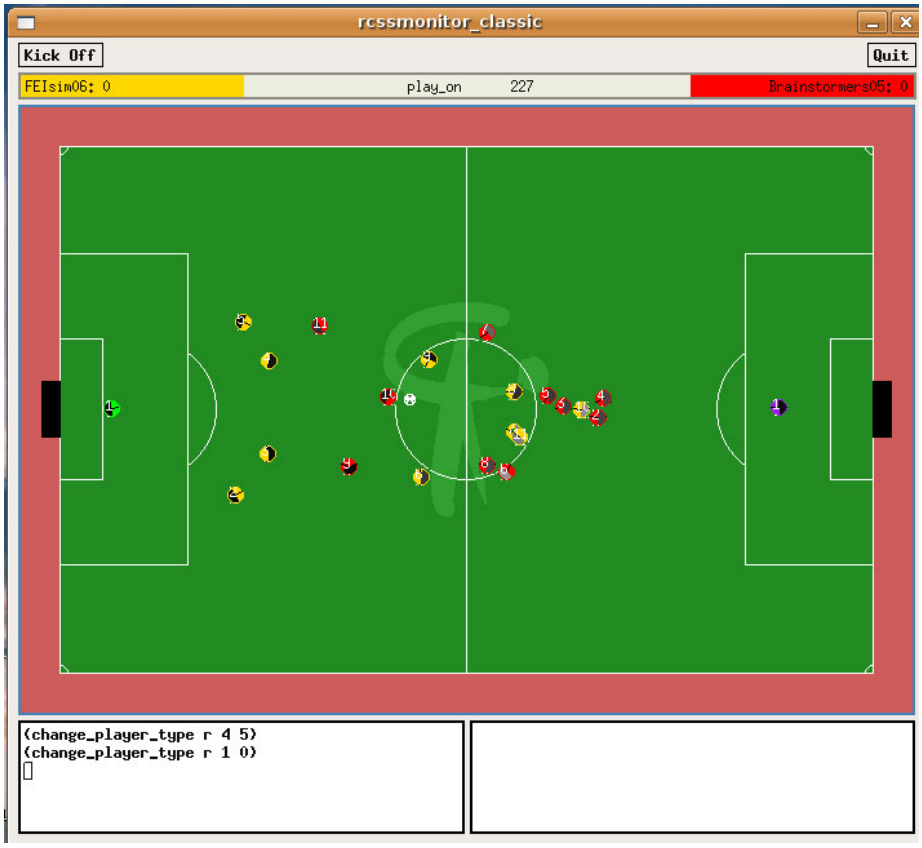


Figura 4.3 - Soccer monitor clássico, time FeiSim06.

4.1.4 Treinador

No *RoboCup* 2D dois tipos possíveis de treinadores podem ser aplicados ao time: off-line e on-line. No modo off-line, o treinador tem um controle quase total nos jogadores, podendo orientar de diversas maneiras, mudar a posição da bola para treinar o time, mudar velocidades de jogadores e de orientação. Sendo assim, neste modo a objetivo é treinar o time com diversas táticas e situações diferentes, como em um treino real. No modo on-line, o treinador não possui todas as facilidades como no modo off-line, sendo que neste modo o treinador só pode fornecer informações adicionais ao jogadores em campo. A comunicação do treinador com os jogadores é possivel, porém as mensagens chegam com relativo atraso aos jogadores.

O treinador pode receber as informações globais sem ruído e pode assim criar estratégias durante o jogo, se comunicar com os jogadores e passar a eles dicas para melhorar o jogo.

4.2 UVA Trilearn

O UVA Trilearn (BOER; KOK, 2002) foi um trabalho de mestrado que compunha-se da criação de um time de futebol para o *RoboCup* 2D, formado por uma arquitetura denominada de three-layer, que são classificadas como:

- Camada de interação: destinada a trabalhar diretamente com o servidor da *RoboCup*, recebe a informação vinda dos sensores (sensores aqui definidos como todas as percepsões recebidas pelo agente) e envia aos atuadores o que deve se feito (andar, girar,etc).
- Camada de competência : Constrói com base nas informações recebidas da camada de interação um modelo de ambiente e implementa inúmeras habilidades para os agentes.
- Camada de controle: escolhe entre as habilidades, qual é a melhor a ser utilizada, dependendo do que se necessita fazer e de sua localização no campo.

As camadas do UVA Trilearn trabalham em paralelo para minimizar o problema de recebimento e de transmisão de informação, fazendo com que os agentes gastem menos tempo nas tarefas. Os agentes do UVA Trilearn tem a capacidade de perceber, decidir e agir. A Figura 4.4 mostra uma descrição das camadas.

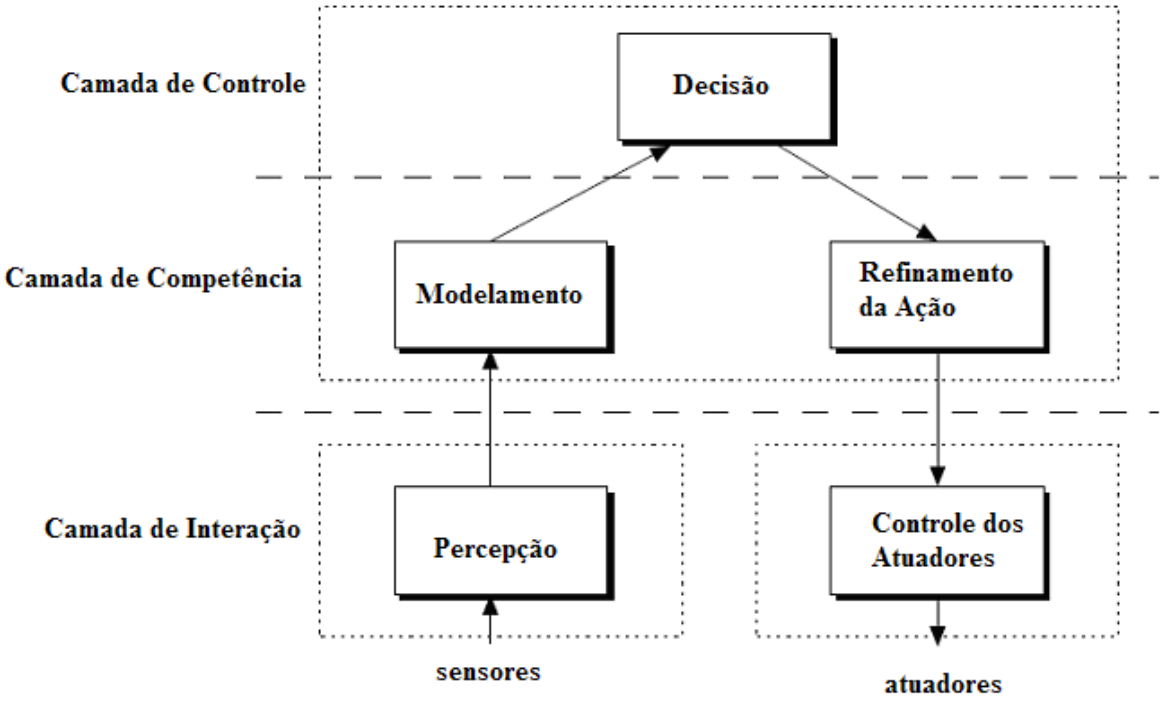


Figura 4.4 - Arquitetura do UVA Trilearn (BOER; KOK, 2002 p. 45).

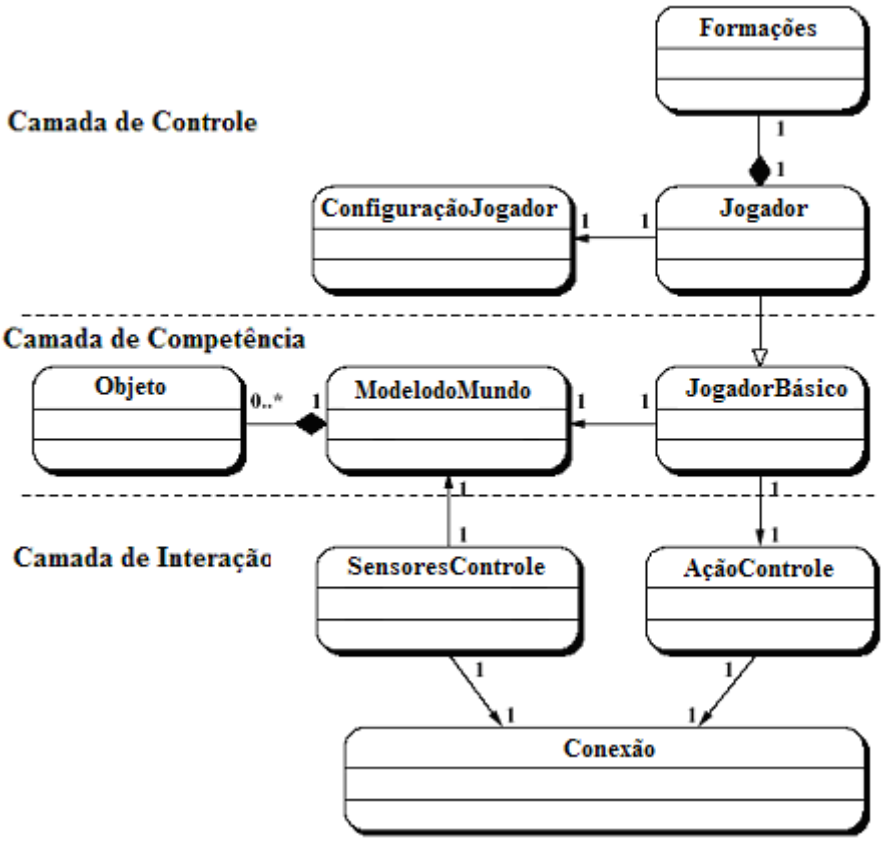


Figura 4.5 – UML do UVA Trilearn (BOER, KOK, 2002 p. 48).

A organização das camadas do UVA Trilearn pode ser melhor vista no diagrama UML, apresentado na Figura 4.5. Nesta Figura são encontrados os vários objetos existentes em cada camada e também como eles se interagem entre si. A interação entre os diversos objetos do UVA Trilearn tem como principal objetivo manter o tempo de resposta as ações dos agentes o menor possível. Os objetos que fazem parte da camada de interação são: SensoresControle, AçãoControle e Conexão. Na a camada de competência são encontrados os objetos: Objeto, ModeloMundo e JogadorBásico e finalmente para a última camada que forma o UVA Trilearn são encontrados os objetos Formação, ConFiguraraçãoJogador e Jogador.

Abaixo é fornecida uma explicação mais detalhada de cada objeto:

- Conexão: cria a conexão com o servidor e fornece todos o suporte para o recebimento e o envio de toda a informação.
- SensoresControle: recebe as informações e separa pelo tipo de informação recebida (auditiva (*hear*), visual (*see*) ou sensorial (*sense_body*)) e envia esta informação para o ModeloMundo.
- ModeloMundo: responsável para criar uma representação do que está acontecendo no jogo, recebe as informações e determina a posição dos agente, bola e suas respectivas velocidades. É responsável também por determinar as informações das posições futuras dos agentes, atualizar todas as informações sobre o jogo e determinar como, por exemplo, qual é o jogador mais rápido.
- Objetos: contém a informação de tudo que está no campo, como os jogadores, a bola, as bandeiras e a posição do gol.
- Formações: responsável pelas formações do time, determina posições estratégicas e a localização do jogador quando começa uma partida, etc. As formações ficam armazenadas em um arquivo que é lido e interpretado e depois passado ao agente no começo de cada jogo.
- ConFiguraçãoJogador: guarda os parâmetros dos jogadores, como por exemplo qual é o máximo valor de stamina, velocidade máxima, etc.
- Jogador: contém métodos para a decisão da melhor ação a ser tomada pelo jogador, determinado pela sua posição e de todas as ações possíveis a serem empregadas.
- JogadorBásico: todas as informações básicas sobre as competências necessárias para um jogador, então em JogadorBásico. Determinado o que o jogador deve fazer no Jogador e contendo uma informação atualizada do ModeloMundo, o

JogadorBásico determina como fazer um drible, chutar para o gol ou interceptar uma bola.

- AçãoControle: este objeto executa os comandos do que um jogador deve fazer, como por exemplo, correr, chutar e virar o pescoço.

O UVA Trilearn aprende a determinar qual é o melhor ponto para chutar a bola no gol usando uma estimação de probabilidades. O problema de chutar a bola no gol é dividido em dois sub-problemas. O primeiro é em como fazer o melhor chute no gol em um determinado ponto do gol e o segundo como fazer um chute que o goleiro não consiga interceptar a bola. Por meio da experiência do jogador no campo em outros jogos, dando chutes em direção ao meio do gol em diferentes distâncias, é possível calcular as probabilidades que resolvem o primeiro sub-problema. O segundo sub-problema é resolvido dando chutes na direção do gol com o goleiro estando em posições diferentes. O resultado é uma política de probabilidades, que fornece a informação da melhor maneira de se fazer um chute ao gol. O agente sempre tentará chutar ao gol, ao verificar que está em uma situação em que a probabilidade de se fazer um gol é muito alta, nos outros casos, ele irá decidir o que fazer com a bola, isto é, se deve fazer um passe ou um drible, por exemplo.

O UVA Trilearn joga na formação 4-3-3 e sua estratégia geral é manter a bola com o time, passando de uma jogador ao outro o mais rápido possível e levando a bola para o gol adversário, tentando passar entre os zagueiro do time adversário e desorganizar o seu oponente e explora as características de cada jogador, como por exemplo, mantendo os jogadores mais rápidos nas laterais e seu campo é divido em áreas e as ações dos jogadores mudam dependendo em que área a bola está.

Neste trabalho foi utilizado o UVA Trilearn Basic (KOK; VLASSIS; GROEN, 2002), que é um versão que se encontra a disposição para uso dos pesquisadores e que contém toda a infra estrutura para a construção de um time, faltando apenas implementar os comportamentos dos jogadores durante um jogo.

4.3 Trabalhos Correlatos

Alguns trabalhos realizados com o simulador 2D serviram como base para a realização desta dissertação. Estes trabalhos são os de STONE, SUTTON e KUHLMANN (2005), KALYANAKRISHNAN, LIU e STONE (2006) e MAUSBERG (2005).

STONE, SUTTON e KUHLMANN (2005) montaram um sistema de aprendizado usando o estilo de jogo chamado *Keepaway*. Neste tipo de jogo, o time com os jogadores que estão com a posse da bola, denominados de *keepers*, tentam ficar com esta posse o máximo tempo possível, passando a bola entre eles, até que um jogador adversário, denominado de *taker*, a "roube". Quando a bola é "roubada" e os *takers* ficam com a posse da bola por um período de tempo ou quando a bola vai para fora de uma área determinada, o jogo recomeça e a bola volta para a posse dos *keepers*. O jogo é realizado dentro de uma área de 20x20m que contém 3 *keepers* e 2 *takers*. A Figura 4.6 descreve o jogo em uma área de 20 x 20 m.



Figura 4.6 – Área do jogo (STONE, SUTTON; KUHLMANN, 2005).

As ações possíveis a serem executadas pelos agentes são:

- HoldBall( ) : segura a bola o mais longe possível do adversário,
- PassBall(k): passar a bola para um jogador k,
- GetOpen( ): vai para um posição livre de oponentes,
- GotoBall( ): vai para bola,
- BlockPass(k): marca um jogador adversário k, ficando entre o jogador e a bola.

Neste trabalho, a criação dos estados do agente foi desenvolvida baseada na posição do agente no campo. São usados 13 variáveis de estados para os *keepers* e 18 para os *takers*. Estas variáveis são, por exemplo, a distâncias entre os *keepers* e os *takers* e a distância entre o jogador e o centro do campo. A Figura 4.7 mostra as informações que são levadas em conta

para determinar estas variáveis de estado, sendo K1, K2 e K3 os *keepers*, T1, T2 os *takers* e C o centro do campo.

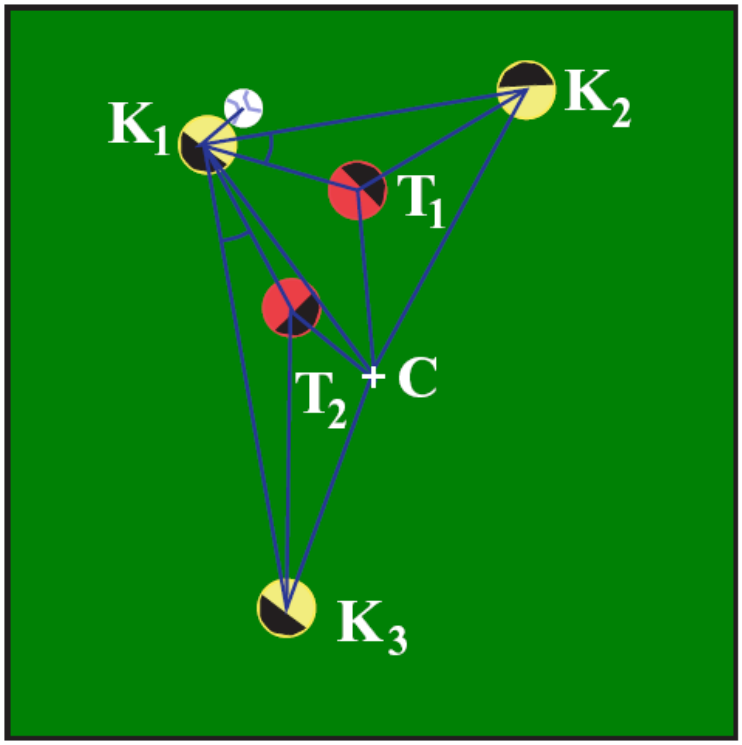


Figura 4.7 – Determinação dos estados (STONE; SUTTON; KUHLMANN, 2005).

O funcionamento se dá da seguinte forma: *keepers* quando estão com a posse da bola, têm a opção de tomar duas atitudes. A primeira é aprender o que fazer com a bola, caso ela esteja com o jogador, ou se não estiver, o jogador procura se posicionar em um local bom para receber um passe de outro jogador. Caso eles não tenham a possa da bola, vão tentar pegar ela de volta. Vale lembrar que o aprendizado só ocorre com a escolha da ação quando o jogador estiver com a bola. Ele deve aprender se deve segurar a bola ou fazer um passe para um outro jogador, sendo o reforço definido como o tempo que a bola estiver com os *keepers*. O *takers* por outro lado, devem tomar posse da bola, e quando este fato ocorre, eles seguram a bola. O trabalho somente apresentou o aprendizado dos *keepers*, e posteriormente deve ser estendido para os *takers*.

Um outro trabalho que segue a mesma linha do anterior foi apresentado por KALYANAKRISHNAN, LIU e STONE (2006). Muito parecido com o *Keepaway*, este trabalho leva em conta metade do campo de futebol e um pequeno grupo de atacantes que aprendem, contra alguns jogadores de defesa e um goleiro que seguem uma estratégia fixa. Outro ponto importante foi o uso de comunicação pelos agentes, informando parâmetros como o estado, ação e reforço recebido. Neste sistema, todo o aprendizado é feito somente

utilizando a informação recebida pela comunicação. A tabela 4.1 apresenta os valores de reforços para o uso do aprendizado. Os estados são representados de forma parecida ao *Keepaway*, e a Figura 4.8 mostra as informações que são levadas em conta para determinar as variáveis de estado, sendo O1,O2,O3 e O4 os jogadores, D1,D2,D3,D4 os adversários, Dg o goleiro e GL o meio do gol.

Tabela 4.1 – Reforços.

Ações/Cenário	**Reforço**
Gol	1.0
Bola com jogador do time	0
Bola pega pelo goleiro	-0.1
Bola fora dá área	-0.1
Bola com jogador adversário.	-0.2

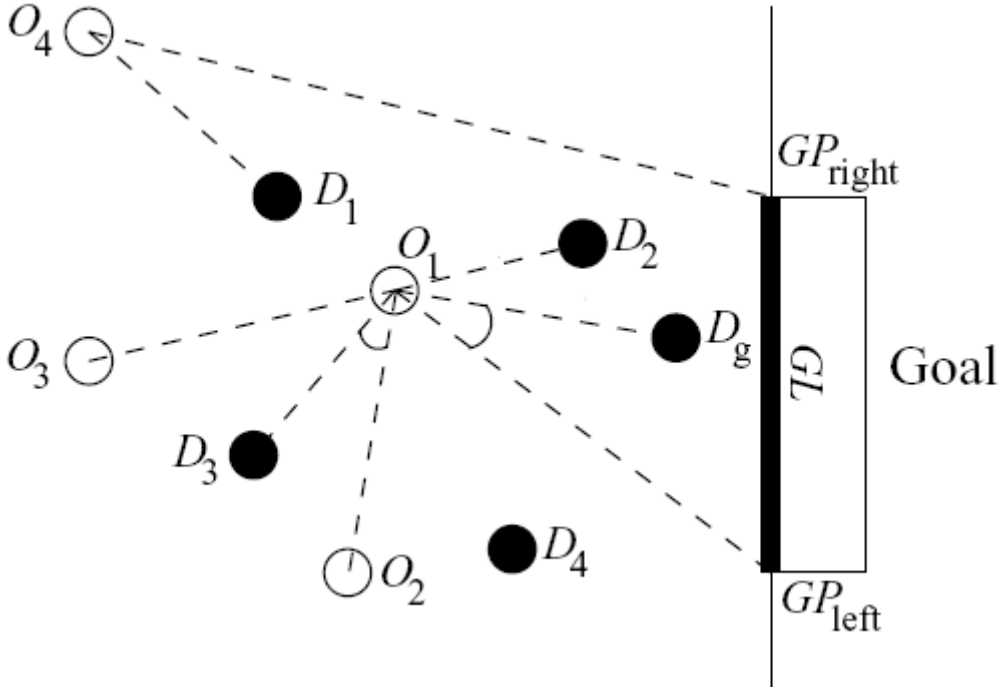


Figura 4.8 - Variáveis de estado.

As ações possíveis a serem executadas pelos agentes são:

- PassK( ): o jogador com a bola pode passar para o outro jogador do mesmo time com a bola,
- Drible( ): é criado um cone virtual na frente da visão do jogador, que dá pequenos chutes na bola sem que a mesma saia do cone, mas que maximize a distância entre a bola e um jogador adversário que está dentro deste cone,
- Shoot( ): o jogador que estiver com a bola chuta para o gol,
- GetOpen( ): vai para um posição livre de oponentes,
- GetBall( ): vai para bola.

O funcionamento se dá da seguinte forma: o jogador com a posse da bola executa uma ação aprendida (Passe, Drible ou Chutar para o gol), e o jogador do mesmo time que estiver mais perto ficará em uma posição para receber um passe. Caso a bola seja perdida, o jogador que estiver mais perto do adversário, tentará pegar a posse da bola de volta. O aprendizado só ocorre com o jogador que tiver com a bola.

Por último, o trabalho de MAUSBERG (2005), trata de uma forma diferente dos anteriores a criação de estados e o uso de reforços. Neste trabalho foi criada uma chave de estados que contém informações da localização do agente, de sua orientação, da distância do agente até a bola, da distância do agente mais perto e por final da distância do adversário mais perto. Estes valores são adicionados a chave por uma representação qualitativa, variando de zero a cinco, como por exemplo, demonstrada na tabela 4.3. A tabela 4.2 descreve como é formada a chave.

Tabela 4.2 – Dígitos da chave (MAUSBERG, 2005).

i4	**i3**	**i2**	**i1**	**i0**
[0; 5]	**[0; 3]**	**[0; 5]**	**[0; 5]**	**[0; 5]**
Posição no campo	Orientação do agente	Distância do agente a bola	Distância do agente ao jogador do mesmo time mais próximo	Distância do agente ao oponente.

A posição do campo (i4) é informada com o valor dado como 0 se o agente está localizado na área centra da defesa, com 1 se o agente está em qualquer outra posição dentro

da área da defesa, com 2 se ele está localizado na área central do meio de campo, com 3 se está localizado em outra posição dentro do meio do campo, com 4 se estiver localizado na área central do ataque e com 5 se estiver em outra posição dentro da área de ataque. A Figura 4.9 exemplifica o que foi descrito.

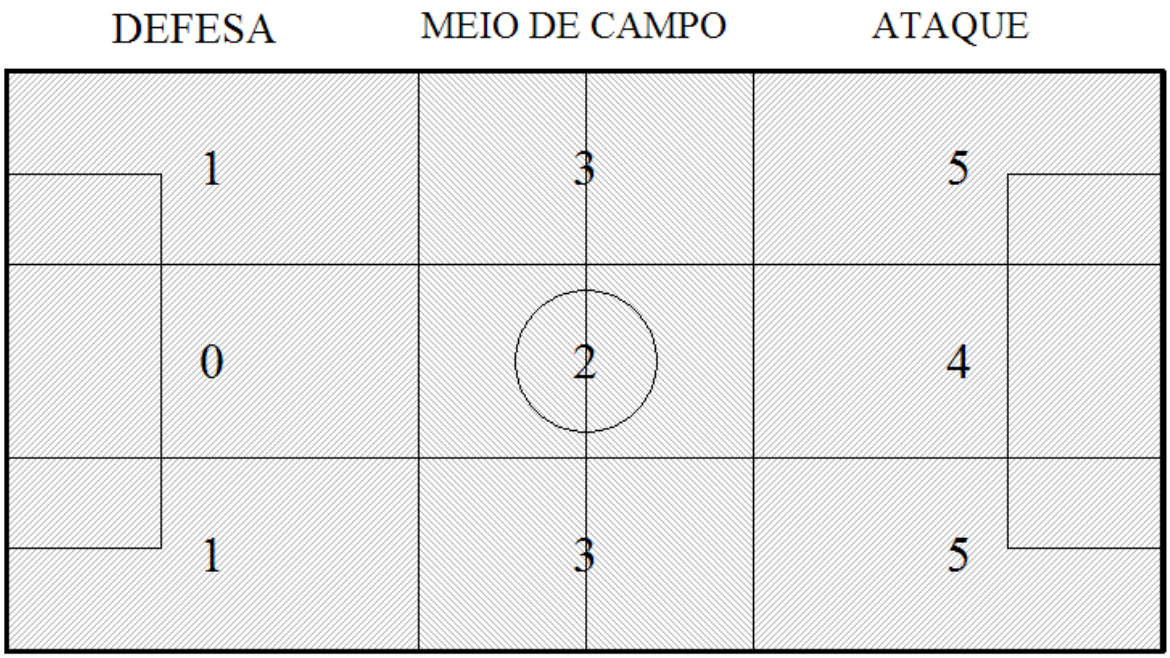


Figura 4.9 – Representação da localização do índice id4 da chave.

As distâncias de i2,i1 e i0, são representadas na tabela 4.3 e os valores do angulo de i3 são representados na tabela 4.4.

Tabela 4.3 – Representação das distâncias.

Distância	**Representação**
[0.0 ; 5.26851]	0
[5.26851 ; 10.5]	1
[10.5 ; 26.3]	2
[26.3 ; 50.0]	3
[50.0 ; 85.6]	4
[85.6 ; 139.0]	5

Tabela 4.4 – Representação dos ângulos.

Ângulo (radianos)	**Representação**
[-1/4π; 1/4 π]	0 (para frente)
[1/4 π; ¾ π]	1 (direita)
[3/4 π; π]	2 (para trás)
[-3/4 π ; - π]	2 (trás)
[-1/4 π;-3/4 π]	3 (esquerda)

A recompensa é obtida por meio de uma multiplicação de vários fatores de recompensas, assim o valor da recompensa final *r*, é obtido por: $r = f1*f2*f3*f4*f5$ e este valor obtido nunca deverá ser maior que 1. Entre as inúmeras possibilidades de recompensas, pode-se citar por exemplo:

- Gol feito: $f1 = 1.0$,
- Bola no ataque e o agente no ataque: $f2 = 0.75$,
- Bola no campo do adversário e movendo na direção do gol: $f3 = 1.0$,
- Bola no ataque e podendo chutar a bola: $f4 = 0.75$,
- Bola perto do agente: $f5 = 1.0$.

Para a situação descrita acima no exemplo, o valor da recompensa pelo agente estar no ataque, com a bola perto do agente e tendo feito um chute na direção do gol (e ter realizado o gol) foi de $r = 1.0 * 0.75 * 1.0 * 0.75 * 1.0 = 0.5625$.

Neste trabalho, só existem duas possibilidades de ações. Uma denominada deliberativa, que tenta descobrir qual é a melhor ação a ser executada dentre de inumeras possibilidades criadas e tendo em vista a recompensa esperada acumulada e uma reativa que gera uma ação rápida do que deve ser feito pelo agente com base no atual ambiente do jogo.

4.4 Discussões

Este capítulo descreveu de forma simplificada o funcionamento do *RoboCup* 2D, a fim de fornecer a informação básica de como o agente recebe e manda as informações para que ocorra uma partida, além de descrever o funcionamento do UVA Trilearn e de trabalhos na mesma linha desta dissertação.

Entre alguns outros trabalhos desenvolvidos no simulador do *RoboCup*, podemos citar: aprendizado para otimizar habilidades de baixo nível (STONE, 1998; RIEDMILLER et al., 2000), técnicas de cooperação (REIS; LAU, 2003), formalizações de estratégias em

equipes complexas (PROKOPENKO; BUTTLE, 1999), planejamento flexível (STONE; VELOSO, 1999), sistemas de comunicação inteligente (STONE; VELOSO, 1999), entre outros. Times como CMUnited (STONE et al., 2000), Brainstormers (RIEDMILLER, 2000) trabalham com o Aprendizado por Reforço no auxilio do desenvolvimento de suas competências (*skills*) básicas.

5 PROPOSTA

A proposta deste trabalho é utilizar o Aprendizado por Reforço Acelerado por heurísticas no domínio da robótica móvel, utilizando a plataforma do *RoboCup* 2D para fazer o aprendizado e avaliação de seu funcionamento. Para tanto, foram implementados dois sistemas dotados de capacidade de aprender: um utilizando um algoritmo de AR normal e o outro utilizando um algoritmo acelerado por heurísticas. As heurísticas utilizadas visam melhorar o aprendizado no agente e foram escolhidas de modo a determinar a melhor ação em um determinado estado.

No desenvolvimento deste trabalho, foram encontrados três grandes desafios. O primeiro se refere ao aprendizado: desenvolver um agente que aprenda é fundamental para a realização do trabalho, mas o problema nesta situação é poder determinar qual é o melhor reforço para que o agente aprenda corretamente. Além deste problema, o espaço de estados é enorme, o que dificulta muito o aprendizado. O terceiro problema é determinar qual heurística pode ser utilizada para melhorar o desempenho do agente.

Para solucionar estes desafios, propõe-se usar no sistema de aprendizado uma discretização do espaço de estados e fornecer diversos reforços diferentes dependendo da situação em que o agente se encontra. No caso da heurística, está é criada com base em possíveis ações, dando mais ênfase ao movimento do agente e deixando o agente aprender o que fazer em outras situações, como por exemplo, no controle da bola.

O aprendizado é realizado a partir das informações recebidas pelo agente. A Figura 5.1 descreve o funcionamento de um agente no campo. A informação enviada pelo servidor é recebida pelo agente, analisada e atualizada no Modelo do Mundo deste agente. Em outras palavras, o agente decodifica os dados que está recebendo e encontra sua posição em coordenadas globais e a posição de outros jogadores, da bola e de distâncias de qualquer objeto no campo de que se tenha necessidade. Com esta informação é possível extrair os estados do agente e determinar as futuras ações deste agente no campo de futebol. Esta informação do que o agente precisa fazer é então transmitida ao servidor, que irá executar a ação escolhida e fornecer os novos estados atualizados do agente.

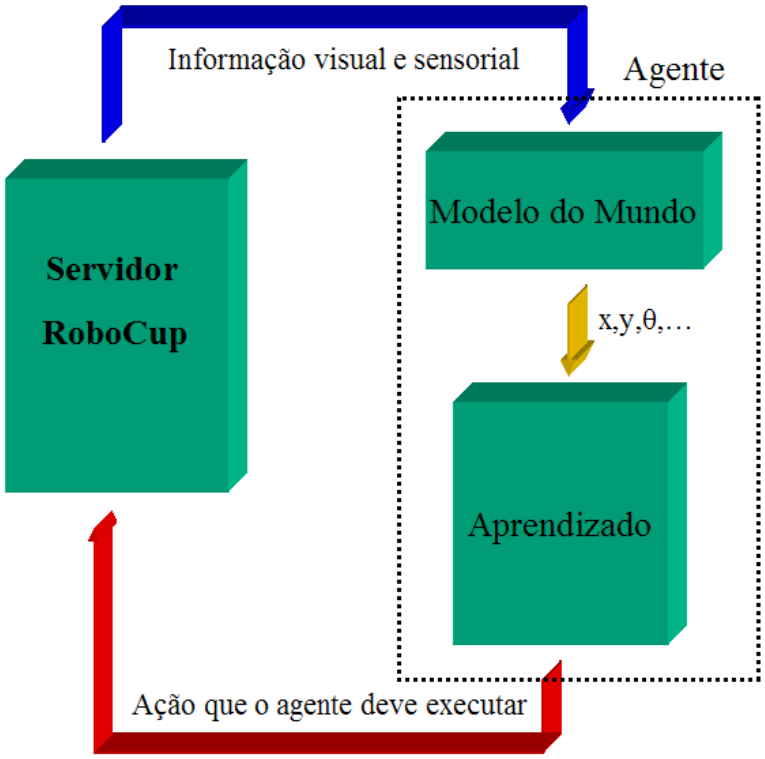


Figura 5.1 – Funcionamento do agente.

O aprendizado usando o *Q-Learning* funciona da seguinte forma: com a informação recebida pelo modelo do mundo, é determinado o estado do agente s_t e a ação a_t que ele deve executar (sendo $a_t = argmax\ (Q(s_{t+1},a_{t+1}))$. Após aplicado a ação é possível determinar o reforço r por ter escolhido a ação e o próximo estado s_{t+1} que o agente se encontra. Estas informações são usadas para atualizar a tabela Q do aprendizado. De forma similar, é o funcionamento do aprendizado acelerado por heurísticas. Neste tipo de sistema é adotado um valor H que irá influenciar a escolha das ações.

5.1 Discretização dos Estados

Devido a grande quantidade de estados, é necessário aplicar alguma forma de abstração para se poder determinar o estado em que se encontra um agente. Este problema pode ser percebido ao se determinar o estado utilizando as localizações possíveis em X e Y do agente e da bola no campo e pelo ângulo do agente (0 a 360º). Para este exemplo o espaço de estados é calculado (levando em conta que o agente só deve se locomover pela metade do campo e que o agente se encontra em uma grade de 1 metro x 1 metro) pela seguinte multiplicação: (X do agente) * (Y do agente) * (ângulo do agente) * (X da bola) * (Y da bola) = 52 * 68 * 360 * 52 *68 = 4.501.186.560 estados. Este valor encontrado é muito

grande e problemático para o aprendizado, pois além de exigir um enorme tempo de aprendizado, também traz problemas físicos de armazenamento de tabela em memória.

Tendo em vista um melhor uso da tabela em memória e diminuir o número de estados, foi criada uma discretização dos estados. Esta discretização compunha-se de zonas numeradas que não precisam ter necessariamente tamanhos fixos (podendo ser mais discretizadas em partes do ambiente com pouco interesse e menos discretizadas em partes do ambiente com muito interesse), e uma simplificação dos ângulos do agente, determinando quatro posições para aonde o agente pode girar e numerando estas posições e assim deste modo diminuir a quantidade de estados existentes. Nestas zonas, ficam tanto o agente, quanto a bola, e o estado é formado pela posição conhecida dentro da grade do agente e da bola. Este sistema satisfaz os casos em que o agente pode esperar a bola ir para dentro da zona criada, por exemplo, um goleiro.

Nos casos em que o agente precisava mover a bola (quando a bola não vai na direção do agente), o sistema descrito anteriormente não foi de total sucesso, e precisou ser ligeiramente modificado. Além do motivo que o agente precisava aprender a mover a bola, o tamanho utilizado da área do campo para o agente foi muito maior e o tamanho de estados começou a crescer muito e somente as generalizações utilizadas anteriormente não satisfaziam mais na resolução do problema. No caso da existência de somente um agente, a localização da bola foi substituída por uma tabela de representação numérica da distância entre a bola e o agente. Ao se colocar dois agentes, além da representação numérica da distância da bola, foi utilizado uma outra representação numérica da distância entre os dois agentes.

Na Figura 5.2 é apresentado um exemplo de discretização de estados utilizados para o aprendizado do zagueiro. Neste exemplo foram criadas 36 zonas possíveis que o agente e a bola podem estar e os ângulos possíveis do agente foram simplificados e numerados dependendo para onde ele esta olhando. Para este exemplo, o número de estados é calculado pela seguinte multiplicação: Zona do Agente * Zona da bola * Ângulo do agente = 36*36*4 = 5.184 estados. Este valor de estados, muito inferior ao anterior valor calculado no primeiro exemplo, apresenta melhores resultados no uso do aprendizado.

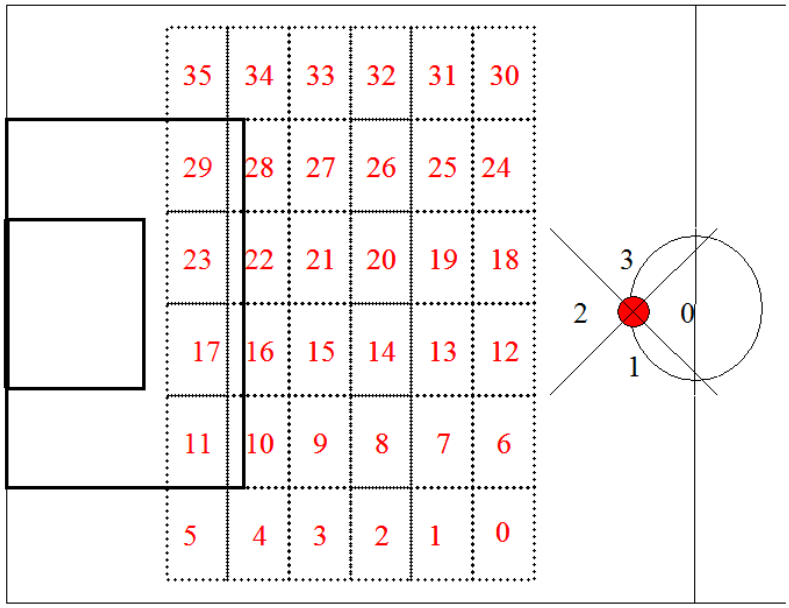


Figura 5.2 – Estados discretizados.

5.2 Macro Ações

As macro ações representam aqui ações de alto nível e de maior complexidade de execução das ações básicas do jogador. As macro ações ajudam a diminuir o número total de ações necessárias para a execução de uma tarefa. Alguns exemplos de macro ações utilizadas neste trabalho são apresentadas na tabela 5.1.

Tabela 5.1 – Macro ações.

Macro ação	**Descrição**	**Agentes que utilizam**
Parado	Não fazer nenhum movimento	Goleiro, zagueiro, atacante
Interceptar a bola	Determina a posição e velocidade de deslocamento para poder interceptar a bola	Goleiro, zagueiro, atacante
Pegar a bola	O agente pode pegar a bola	Goleiro.
Passe para o goleiro	Chuta a bola na direção do goleiro	Zagueiro
Chutar para o gol	Chutar a bola na direção do gol	Atacante
Drible	Faz um drible com a bola, passando ela pelo adversário	Atacante
Segurar a bola	Parar o movimento da bola	Atacante

5.3 Reforços

Os reforços, como já comentado no começo deste capítulo, são fornecidos dependendo da situação em se encontra o agente. Deste modo são estipulados diferentes reforços para diferentes tipos de cenários. As escolhas dos reforços foram feitas com o cuidado de não fornecer dicas, mas sim recompensas por ações corretas. A tabela 5.2 fornece um exemplo de reforços que um goleiro pode receber durante um aprendizado.

Tabela 5.2 – Reforços.

Situação	**Reforço**
Chutar a bola quando estiver a menos que 0.7 metros	+ 25
Pegar a bola quando estiver a menos que 1 metro	+ 25
Conduzir a bola quando estiver a menos que 0.7 metros	+ 25
Gol tomado	- 100
Adversário com a posse da bola e o agente não se aproximando da bola	- 25

5.4 Heurísticas

As heurísticas fornecem informações ajudando na escolha das ações, visando obter uma aceleração no aprendizado. Escolher qual é a melhor heurística é uma tarefa difícil: para este trabalho a heurística será criada para fornecer uma ajuda para o deslocamento do agente no campo, deixando o agente aprender a escolher os outros tipos de ações, como controle de bola, chutar para o gol, dentre outras. A tabela 5.3 descreve as heurísticas utilizadas.

Tabela 5.3 – Heurísticas.

Situação	**Heurística**
Bola distante do jogador	Correr para a bola
Bola perto do jogador	Executar alguma ação, menos não fazer nada.

5.5 Discussões

Este capítulo descreveu a proposta do mestrado e propõem como resolver alguns problemas com o tamanho do espaço de estados. No próximo capítulo são apresentados os experimentos usando o *Q-Learning* e o *Q-Learning* acelerado por heurísticas (HAQL).

6 EXPERIMENTOS E RESULTADOS NO DOMÍNIO DO FUTEBOL DE ROBÔS

Neste capítulo são apresentados os estudos realizados no domínio do futebol de robôs utilizando AR e AR acelerado por heurísticas, e os resultados obtidos. Foram realizados 4 experimentos, comparando o aprendizado utilizando o algoritmo *Q-Learning* (sem heurísticas) com o aprendizado utilizando o algoritmo HAQL utilizando uma heurística pré-definida. Os experimentos realizados implementarão os seguintes agentes:

- Um goleiro que aprende contra um atacante sem aprendizado,
- Um atacante com aprendizado, contra um goleiro sem aprendizado,
- Um goleiro e um zagueiro, ambos com aprendizado, contra dois atacantes sem aprendizado,
- Dois atacantes com aprendizado, contra um goleiro e um zagueiro sem aprendizado.

Todos os experimentos foram realizados utilizando o algoritmo *Q-Learning*, com os seguintes parâmetros: $\gamma = 0.9$, $\alpha = 0.125$, e taxa de exploração/explotação igual a 0.05. Os parâmetros γ e exploração/explotação são os mesmos utilizados por diversos autores (PEGORARO, 2001; BIANCHI, 2004; MAUSBERG, 2005, dentre outros). Para o α foi escolhido o mesmo valor usado por STONE, SUTTON e KUHLMANN (2005) que afirmam ter encontrado bons resultados com um $\alpha = 2^{-3}$. A escolha dos reforços e a quantidade de reforços necessários, foram determinados por meio de testes empíricos. A princípio o reforço deveria ser dado apenas quando um gol foi feito ou sofrido, mas os resultados não foram satisfatórios. Para resolver estes problemas, os reforços foram determinados pelas ações positivas ou negativas realizadas pelo agente, levando em conta a situação do momento da ação, com base nos trabalhos de KALYANAKRISHNAN, LIU e STONE (2006) e MAUSBERG (2005). Todos os valores de reforços foram escolhidos para serem o menor valor possível de reforço que possa ajudar no aprendizado de um agente, ajudando assim na orientação correta do agente, sem fornecer fortes dicas para o aprendizado.

Para verificar se apenas o uso das heurísticas em um agente produz bons resultados, foram realizados testes com agentes que utilizaram apenas as heurísticas, para os quatro

experimentos. Para finalizar, cada episódio foi formado de 3.000 ciclos (aproximadamente 5 minutos) com todos os jogos dos agentes aprendizes começando do lado esquerdo do campo.

Os experimentos realizados foram codificados em linguagem C++ e executados em um Microcomputador Pentium 4 2.8 Ghz HT, com 1GB de memória RAM e sistema operacional Linux e em um Microcomputador AMD Athlon 64 3500+, com 1GB de memória RAM e sistema operacional Linux. Cada uma das quatro experiências propostas foi realizada sempre no mesmo computador, permitindo que os resultados para os algoritmos com heurística e sem heurísticas possam ser comparados, pois foram gerados exatamente com a mesma conFiguração.

6.1 Aprendizado de um Goleiro

Neste cenário um agente precisa aprender a defender o gol contra um outro jogador. O agente só irá defender o gol quando a bola estiver dentro da área de defesa do goleiro; em outras situações, o agente ficará esperando a bola entrar na sua área. Esta área de defesa será a área que o agente aprenderá a defender. O time adversário é formado por um jogador do UVA Trilearn Basic (KOK; VLASSIS; GROEN, 2002), sendo que suas ações compreendem apenas ir para a bola e chutar a bola para o gol.

Para a realização do aprendizado foi implementada uma grade de 4 x 4 células que será a área de defesa do goleiro, e que compreende a grande e pequena área. Cada célula tem 4 x 10 metros. O agente sabe a todo tempo sua posição e também a localização da bola dentro de um plano cartesiano, podendo estimar em que célula ele e a bola se encontram, quando ela estiver dentro da área de defesa do agente.

O estado é descrito por uma chave formada pela posição do agente na grade, da posição da bola na grade e de uma discretização dos ângulos do agente em 4 orientações. A tabela 6.1 mostra a discretização dos ângulos em que o agente pode estar orientado. A tabela 6.2 mostra como é formada a chave que descreve o estado do agente. Finalmente, a tabela 6.3 mostra alguns exemplos de chaves de descrição dos estados. A Figura 6.1 mostra a grade criada para o agente e a bola, bem como a discretização dos ângulos nos quais o agente pode se encontrar.

Tabela 6.1 - Discretização dos ângulos de orientação do agente.

Ângulo (radianos)	**Representaçao**
[-1/4π; 1/4 π]	0 (para frente)
[1/4 π; ¾ π]	1 (direita)
[3/4 π; π]	2 (para trás)
[-3/4 π ; - π]	2 (para trás)
[-1/4 π;-3/4 π]	3 (esquerda)

Tabela 6.2 - Chave de descrição do estado para o agente goleiro.

Ângulo do agente	**Posição do agente**	**Posição da bola**
[0; 3]	[0; 15]	[0; 15]

Tabela 6.3 - Exemplos de chaves de descrição de estado para o agente goleiro.

chave	**Descrição do estado**
01011	Jogador virado para frente e dentro da grade 10 e bola dentro da grade 11
11412	Jogador virado para esquerda, dentro da grade 14 e bola dentro da grade 12
00012	Jogador virado para frente, dentro da grade 0 e bola dentro da grade 12
20000	Agente virado para trás e dentro da grade 0 e bola dentro da grade 0

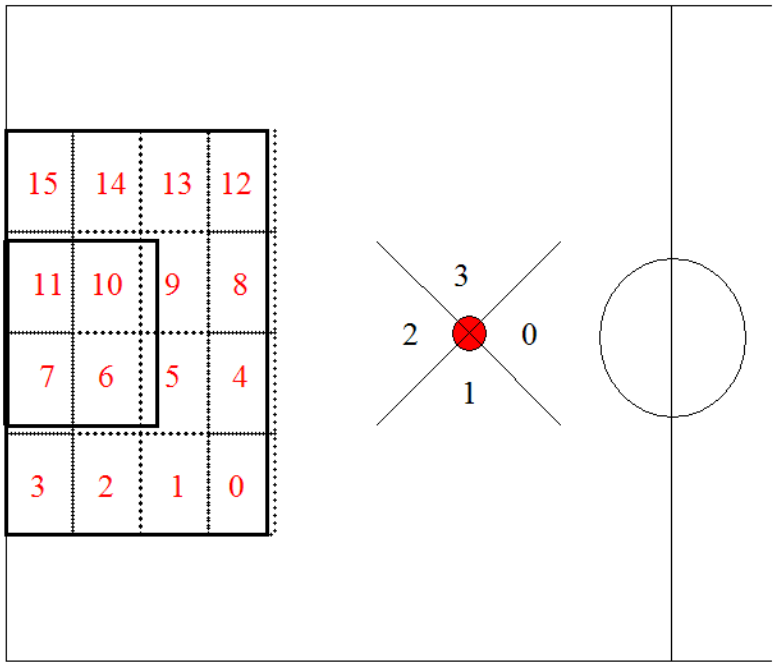


Figura 6.1 - Grade utilizada para o agente goleiro.

As escolhas das ações foram determinadas por dois fatores. Primeiro o número de ações não poderia ser grande, pois isto causaria uma demora no aprendizado e segundo, deveria ter um número de ações suficiente para ajudar o agente na sua função. Tento em vista estes dois fatores, as ações escolhidas para este experimento são:

- Parado: não executa nenhum movimento,
- Interceptar a bola: determina a melhor posição e velocidade de deslocamento para poder interceptar a bola,
- Conduzir a bola: levar a bola para frente, dando chutes fracos,
- Chutar a bola: faz um chute forte na bola, fazendo com que ela vá para uma área longe do adversário.
- Pegar a bola: o agente pode pegar a bola,
- Posição de defesa: determina uma posição que se a bola for chutada para o gol, é possível defender o gol.

Os valores de reforços utilizados para esta experiência são mostrados na tabela 6.4.

Tabela 6.4 - Reforços para o agente goleiro.

Situação	**Reforço**
Chutar a bola quando estiver a menos que 0.7 metros.	25
Pegar a bola quando estiver a menos que 1 metro.	25
Conduzir a bola quando estiver a menos que 0.7 metros.	25
Gol tomado.	-100
Adversário com a posse da bola e o agente não se aproximando da bola.	-25

Os valores das heurísticas usados neste experimento foram definidos como + 500 para o momento no qual se deve influenciar alguma ação positivamente (ou seja, deseja-se que esta ação seja executada) e – 500 no caso contrário. As regras usadas para definir quando é interessante executar uma ação foram:

- Se o agente e a bola se encontram em zonas diferentes: então corra em direção a bola,
- Se o agente e bola se encontram na mesma zona: então não ficar parado.

Na Figura 6.2 são demonstradas as ações realizadas pelo goleiro que se prepara para defender o gol. Nas Figuras (6.2a, 6.2b, 6.2c e 6.2d), são mostradas as ações que levam a defesa da bola. Percebe-se que o goleiro vai ao encontro da bola (interceptar a bola) somente após a bola entrar na grande área (Figura 6.2c), e após a defesa (chutar a bola) e com a bola se afastando do gol, o goleiro fica parado, esperando uma nova oportunidade para defender a bola (Figura 6.2d). No começo do aprendizado, estás ações a serem tomadas, ainda não foram bem aprendidas, somente após inúmeros episódios e por meio de tentativa e erro o agente goleiro aprende qual a melhor ação a se tomar.

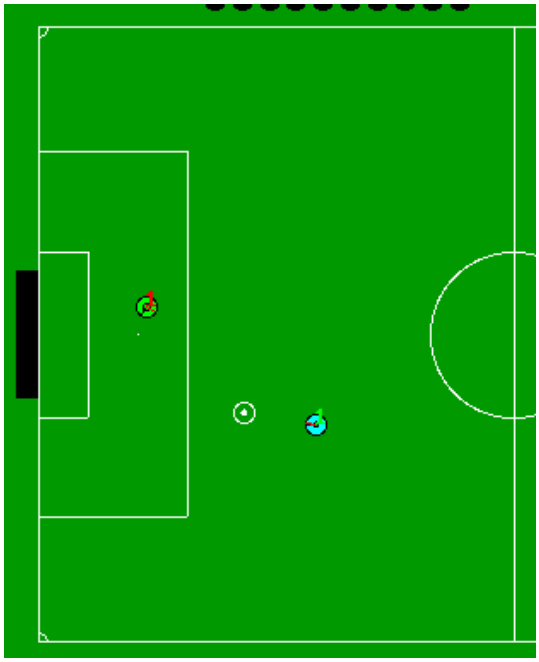


Figura 6.2a – Ações goleiro.

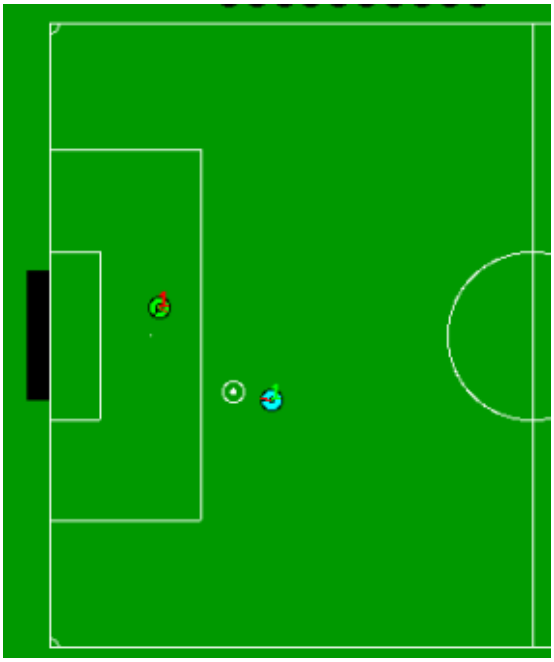


Figura 6.2b – Ações goleiro.

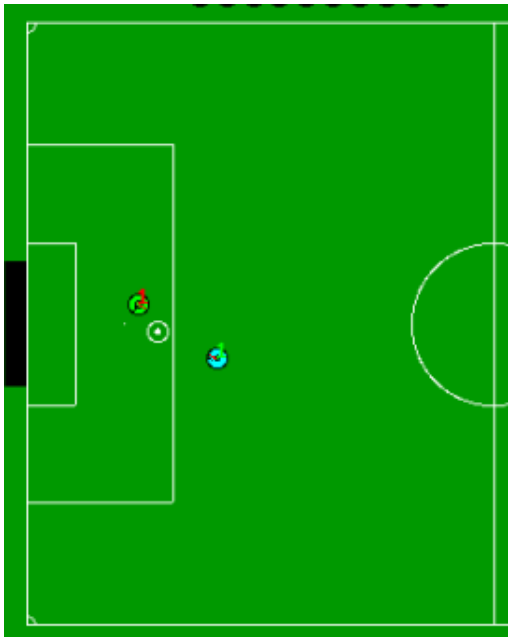


Figura 6.2c – Ações goleiro.

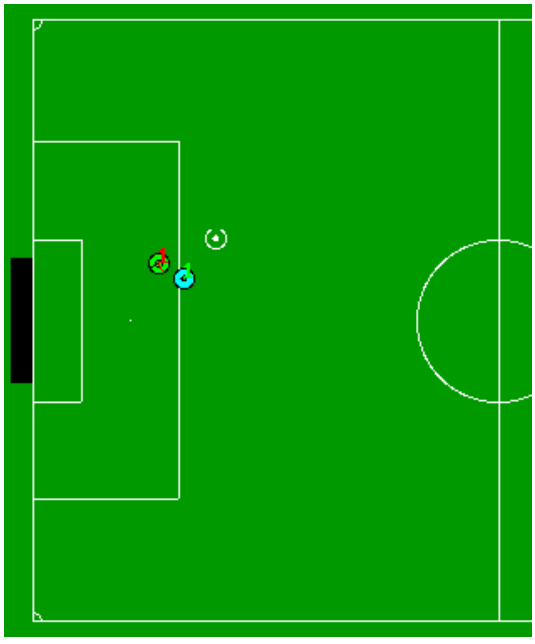


Figura 6.2d – Ações goleiro.

Os resultados para a execução dos algoritmos *Q-Learning* e HAQL (média de 10 jogos) são apresentados nas Figura 6.3 e Figura 6.4 (com barras de erro). Pode-se perceber que utilizando o algoritmo *Q-Learning*, o goleiro inicia o aprendizado sofrendo uma média de 13 gols por jogo e no episódio número 100 este valor cai para aproximadamente 8 gols por jogo. Comparando com o algoritmo *HAQL*, pode-se ver que o goleiro já começa sofrendo um valor menor de gols (10 gols por jogo, em média) e que no episódio número 40 este valor fica em aproximadamente 7. Também se pode observar que o uso das heurísticas em um agente sem aprendizado não evolui com o tempo, e que não produz um goleiro eficiente, sofrendo uma média de 21 gols por partida (valor da média de uma partida). Com isso, pode-se concluir que as heurísticas aceleram aprendizado, mas não são tão fortes que eliminam a necessidade do aprendizado.

Foi utilizado o teste *t* de Student (SPIEGEL, 1984) (que se encontra explicado no apêndice 2), para verificar se a hipótese de que o uso do algoritmo do *HAQL* acelera o aprendizado em relação ao algoritmo *Q-Learning* é válida. O resultado para o teste *t* de Student computado utilizando os dados da Figura 6.3, é apresentado na Figura 6.5. Nesta Figura é possível ver que entre o décimo e o centésimo episódio, os algoritmos são significativamente diferentes, com nível de confiança maior que 5%. Isto confirma que as heurísticas usadas tornam o algoritmo HAQL mais rápido que o *Q-Learning*.

A soma dos gols sofridos pelo agente goleiro em 150 episódios (média de 10 treinamentos) utilizando o algoritmo *Q-Learning* e o HAQL é apresentada na tabela 6.5. Pode-se notar que o agente que utiliza o algoritmo HAQL sofreu 425 gols a menos que o agente que utiliza o *Q-Learning*. Isto ocorre, pois, ao receber menos gols no início do treinamento, o HAQL tem uma vantagem que o *Q-Learning* não consegue reduzir, mesmo quando sua atuação se torna similar a do HAQL.

Tabela 6.5 - Total de gols sofridos pelo agente goleiro (média e desvio padrão).

Algoritmo	**Gols**
Q-Learning.	1451 ± 21
HAQL	1026 ± 7

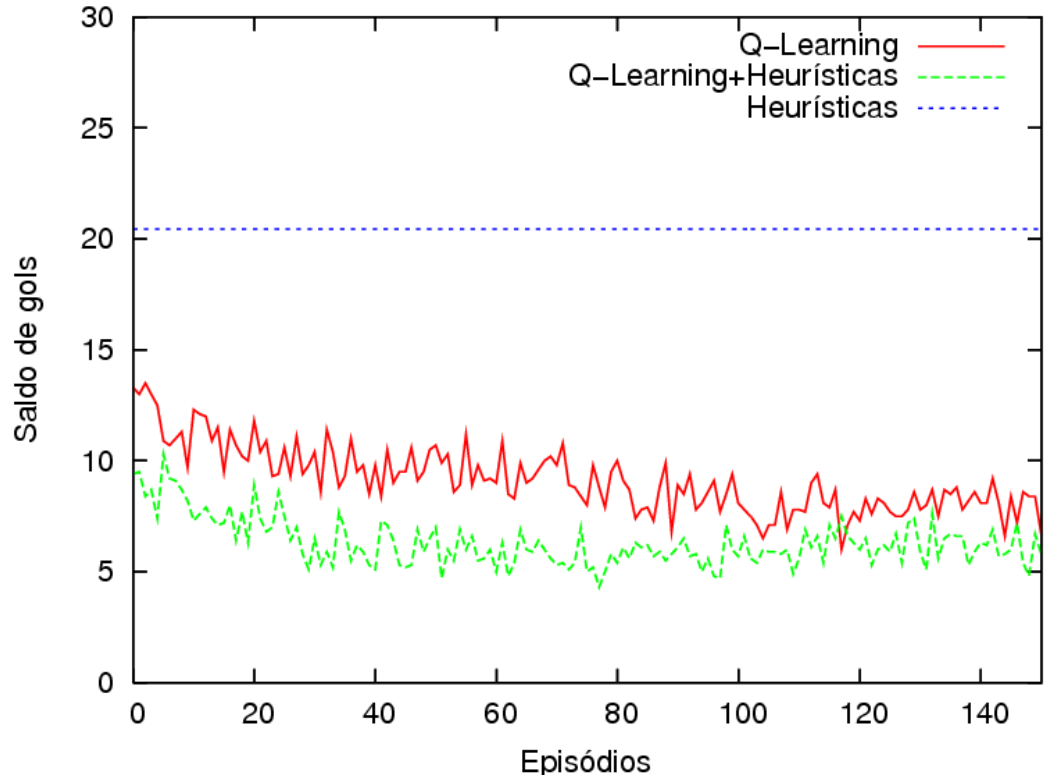


Figura 6.3 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para o agente goleiro.

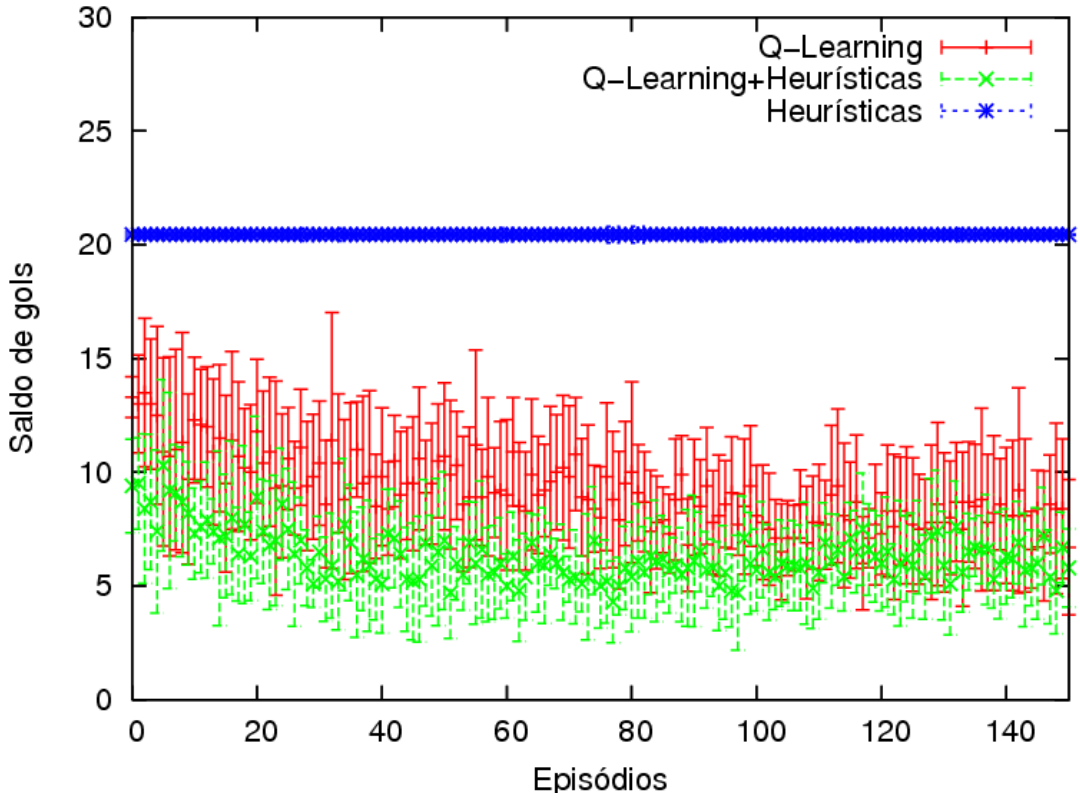


Figura 6.4 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para o agente goleiro com barras de erro.

Foi analisada a evolução dos valores da tabela Q, para os dois algoritmos, durante o aprendizado. Este valor é composto da diferença dos valores de Q(s,a), para todos os estados e ações, entre dois instantes de tempo. Este valor é dado pela fórmula:

$$Q(t-1) - Q(t) = \sum_{s \in S, a \in A} Q_t(s,a) - Q_{t-1}(s,a) \qquad (6.1)$$

O valor encontrado permite verificar a convergência do algoritmo: no começo do aprendizado a diferença encontrada entre duas tabelas mostra que o agente ainda está aprendendo. Porém, depois de um certo tempo, a diferença é mínima. A Figura 6.6 mostra os resultados obtidos pela subtração da tabela Q a cada 3.000 ciclos (de uma média de 10 jogos), na Figura 6.7 são apresentados os resultados com barras de erro. Percebe-se que o tempo de convergência do algoritmo HAQL é menor que o do algoritmo *Q-Learning*, similarmente ao analisado na Figura 6.3.

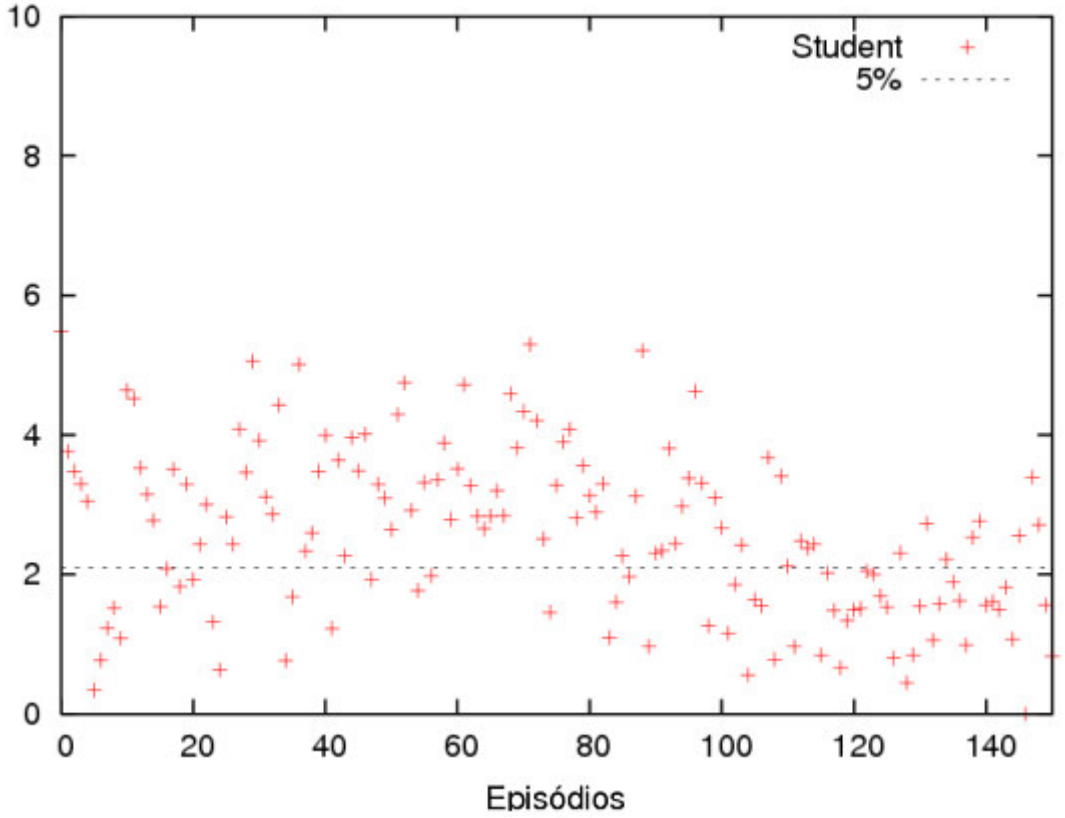


Figura 6.5 - Resultado do teste *t* de Student para os algoritmos *Q-Learning* e HAQL.

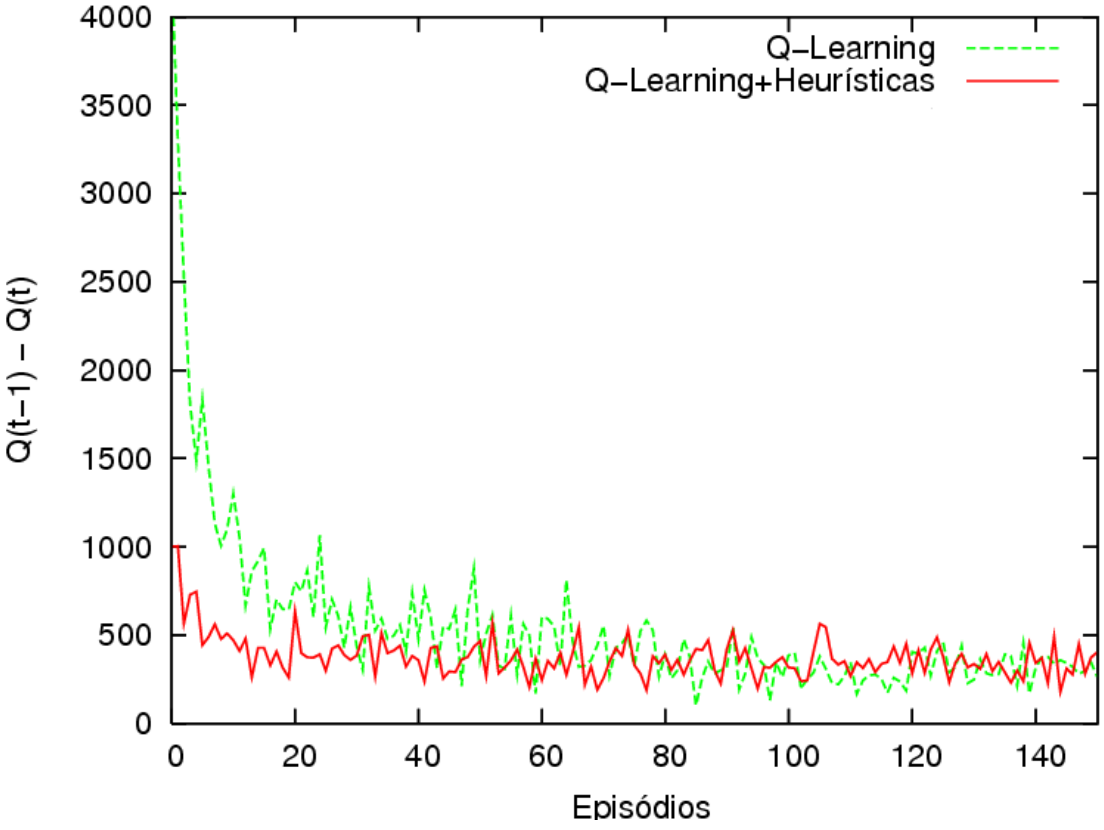


Figura 6.6 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente goleiro.

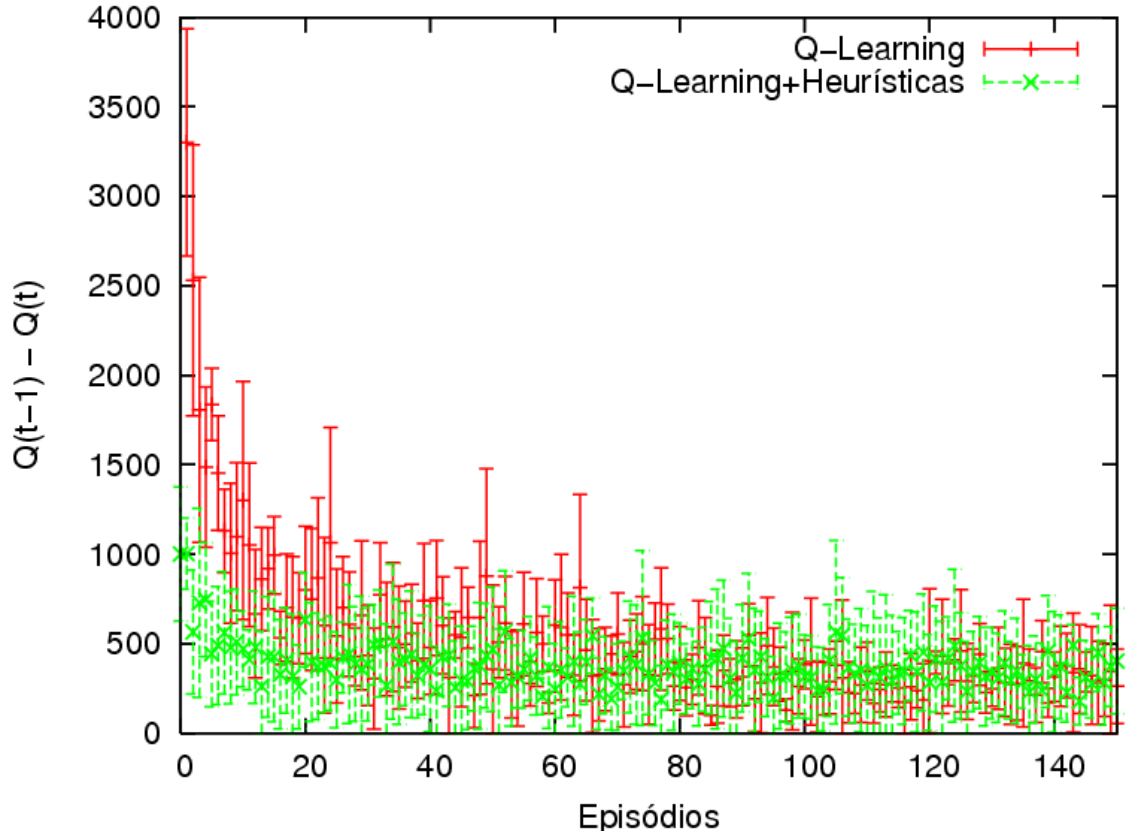


Figura 6.7 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente goleiro com barras de erro.

Devido a dúvidas sobre o funcionamento do aprendizado em uma única célula da grade, foi estudado o caso em que o agente e a bola estão em mesma área. Este estudo visa demonstrar que apesar de cada zona ser formada por uma área de 4 x 10 m, o aprendizado ocorre, recompensando as melhores ações para aquele estado. Os resultados e os comentários pertinentes a este assunto estão no apêndice 1.

6.2 Aprendizado de um Goleiro e um Zagueiro

Este experimento é uma extensão do anterior, que continha somente um agente goleiro. Para ambos os casos (goleiro e zagueiro) o agente irá defender esta bola somente na sua área de defesa. Quando a bola estiver fora desta área, ele não executará nenhuma ação até o momento da bola voltar para esta área. O time adversário é formado por dois jogadores atacantes do UVA Trilearn Basic (KOK; VLASSIS; GROEN, 2002), e suas ações compreendem apenas ir para a bola e chutar a bola para o gol.

O goleiro utilizado neste experimento funciona de maneira semelhante ao descrito em 6.1, com a única diferença na mudança de uma ação: a ação conduzir a bola encontrada anteriormente, foi substituída por uma ação de passe. Nesta nova ação o agente goleiro deve fazer um chute na direção do agente com a função de zagueiro. Todos os reforços possuem o mesmo valor já descrito anteriormente (seção 6.1) para o goleiro, a ação de passe, recebe o mesmo reforço dado ação de conduzir a bola no experimento anterior.

A principal mudança está no acréscimo de um novo agente com a função de zagueiro. Para a realização do aprendizado foi implementada uma grade de 6 x 6 células que será a área de defesa do zagueiro, cada célula tem aproximadamente 5 x 10 metros. O agente sabe a todo tempo sua posição e também a localização da bola dentro de um plano cartesiano, podendo estimar em que célula ele e a bola se encontram, quando ela estiver dentro da área de defesa do agente.

O estado é descrito por uma chave formada pela posição do agente na grade, da posição da bola na grade e de uma discretização dos ângulos do agente em 4 orientações. A tabela 6.6 descreve a discretização dos ângulos e a Figura 6.8 mostra a grade criada para o agente e a bola, bem como a discretização dos ângulos nos quais o agente pode se encontrar, além da grade também utilizada pelo goleiro. A tabela 6.7 mostra como é formada a chave que descreve o estado do agente, e por final a tabela 6.8 mostra os reforços fornecidos ao agente.

Tabela 6.6 - Discretização dos ângulos (em graus).

Ângulo (radianos)	Representaçao
[-1/4π; 1/4 π]	0 (para frente)
[1/4 π; ¾ π]	1 (direita)
[3/4 π; π]	2 (para trás)
[-3/4 π ; - π]	2 (para trás)
[-1/4 π;-3/4 π]	3 (esquerda)

Tabela 6.7 - Chave de estado.

Ângulo do agente	Posição do agente	Posição da bola
[0;3]	[0;35]	[0;35]

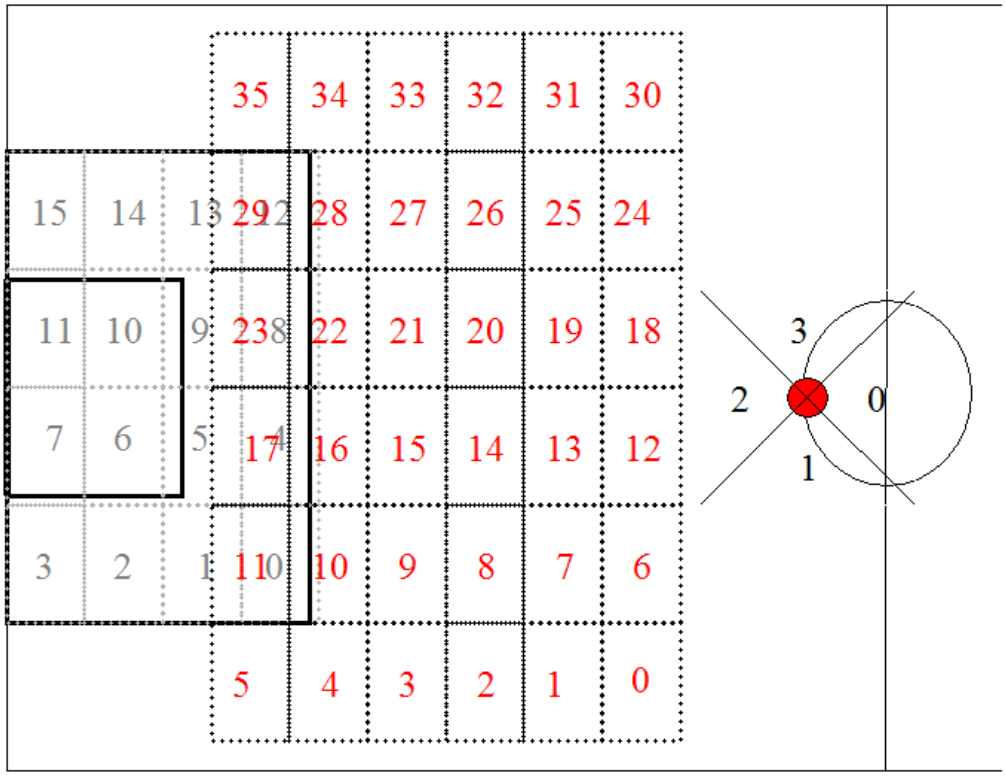


Figura 6.8 – Grade utilizada pelos agentes goleiro e zagueiro.

As ações que o zagueiro pode realizar são:

- Parado: não executa nenhum movimento,
- Interceptar a bola: determina a melhor posição e velocidade de deslocamento para poder interceptar a bola,
- Conduzir a bola: levar a bola para frente, dando chutes fracos,
- Passe para o goleiro: chuta a bola na direção do goleiro.
- Chutar a bola: faz um chute forte na bola, fazendo com que ela vá para uma área longe do adversário,
- Marcar: procura o adversário mais perto, fica próximo a ele e no caminho da bola.

Tabela 6.8 - Reforços do zagueiro

Situação	**Valor do reforço**
Chutar a bola quando estiver a uma distância menor ou igual a 0.7 metros.	15
Fazer um passe quando estiver a uma distância menor ou igual a 0.7 metros.	15
Gol tomado.	-15
Adversário com a posse da bola e o agente não se aproximando da bola.	-10

As heurísticas foram aplicadas ao goleiro e ao zagueiro de forma igual e são similares as usadas em 6.1. O valor das heurísticas foram: + 200 para quando deveria influenciar alguma ação positivamente e – 200 no caso contrario. As regras usadas para definir quando é interessante executar uma ação foram:

- Se o agente e a bola se encontram em zonas diferentes: então corra em direção a bola,
- Se o agente e bola se encontram na mesma zona: então não ficar parado.

Na Figura 6.9 são demonstradas as ações realizadas por um goleiro e um zagueiro que se preparam para defender o gol. Nas Figuras (6.9a, 6.9b, 6.9c e 6.9d), são mostradas as ações que levam a defesa da bola. Percebe-se que tanto o goleiro quando o zagueiro vão ao encontro da bola. Pelas ações do atacante, a bola já passou o zagueiro e está na área que ambos o goleiro e o zagueiro podem defender. Para este caso o goleiro adiantou-se a um dos atacantes (interceptar a bola, Figura 6.9c) e chutou a bola levando a mesma para longe da grande área.

Ao se afastar da grande área, a bola ainda está na área de defesa do zagueiro, este ao perceber que a bola está em sua área de defesa, tenta chegar a bola para mandar a bola para longe de sua área. No começo do aprendizado, estás ações a serem tomadas, ainda não foram bem aprendidas, somente após inúmeros episódios e por meio de tentativa e erro os agentes aprendem qual a melhor ação a se tomar.

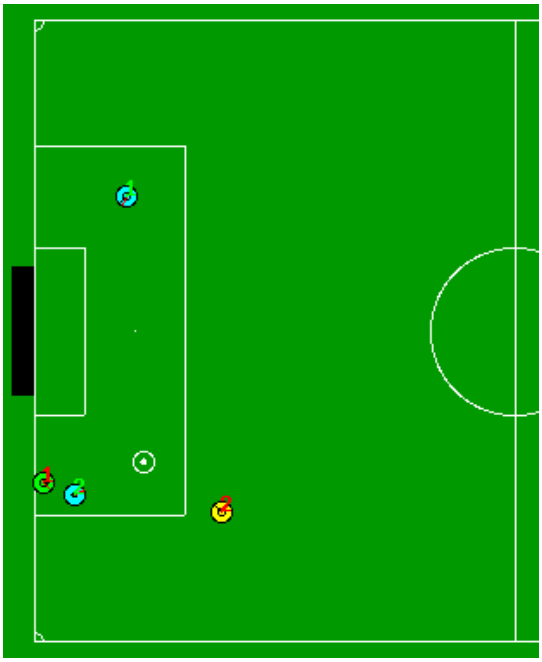


Figura 6.9a – Ações Goleiro e Zagueiro

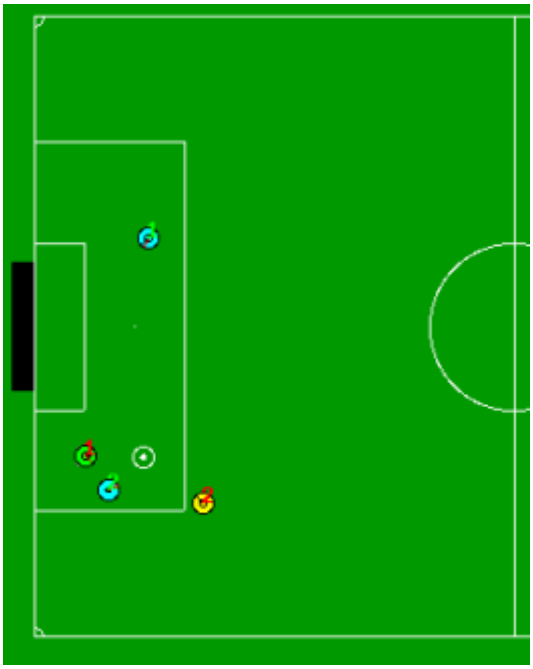


Figura 6.9b – Ações Goleiro e Zagueiro

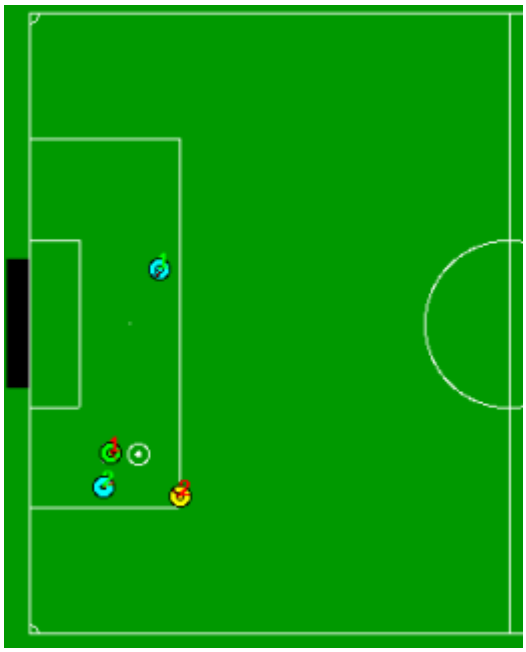


Figura 6.9c – Ações Goleiro e Zagueiro

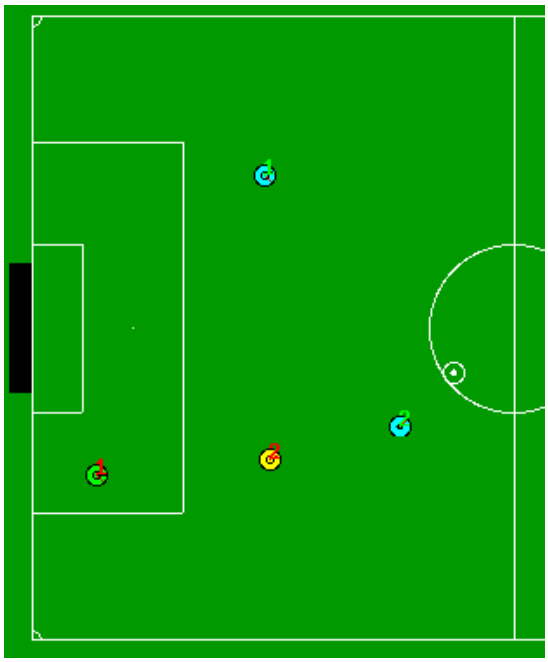


Figura 6.9d – Ações Goleiro e Zagueiro

Os resultados para a execução dos algoritmos *Q-Learning* e HAQL de uma média de 10 jogos, são apresentados na Figura 6.10 e na Figura 6.11 (com barras de erro). Pode-se perceber que utilizando o algoritmo *Q-Learning*, o goleiro e o zagueiro iniciam o aprendizado sofrendo uma média de 16 gols e no episódio 80 este valor fica em 10 gols. Comparando com o algoritmo HAQL, o jogo já começa com um valor de 13 gols e ao final do episódio 40 este valor fica em aproximadamente 7 gols. Pode-se observar que o uso das heurísticas em um agente sem aprendizado não evolui com o tempo, e que não produz um goleiro e nem um zagueiro eficiente, sofrendo uma média de 24 gols por partida (valor da média de uma partida). Com isso, pode-se concluir que as heurísticas aceleram aprendizado, mas não são tão fortes que eliminam a necessidade do aprendizado.

Foi utilizado o teste *t* de Student (SPIEGEL, 1984), para verificar se a hipótese de que o uso do algoritmo do HAQL acelera o aprendizado em relação ao algoritmo *Q-Learning* é válida. O resultado para o teste *t* de Student computado utilizando os dados da Figura 6.10 é apresentado na Figura 6.12. Nesta Figura é possível ver que os algoritmos são significativamente diferentes, com nível de confiança maior que 5%. Isto confirma que a heurísticas usadas tornam o algoritmo HAQL mais rápido que o *Q-Learning*.

A soma dos gols sofridos pelo agente goleiro em 100 episódios (média de 10 treinamentos) utilizando o algoritmo *Q-Learning* e o HAQL é apresentada na tabela 6.9. Pode-se notar que o agente que utiliza o algoritmo HAQL sofreu 341 gols a menos que o agente que utiliza o *Q-Learning*. Isto ocorre, pois, ao receber menos gols no início do treinamento, o HAQL tem uma vantagem que o *Q-Learning* não consegue reduzir, mesmo quando sua atuação se torna similar a do HAQL.

Tabela 6.9 - Total de gols e desvio padrão.

Algoritmo	**Gols**
Q-Learning.	1177 ± 51
HAQL	836 ± 10

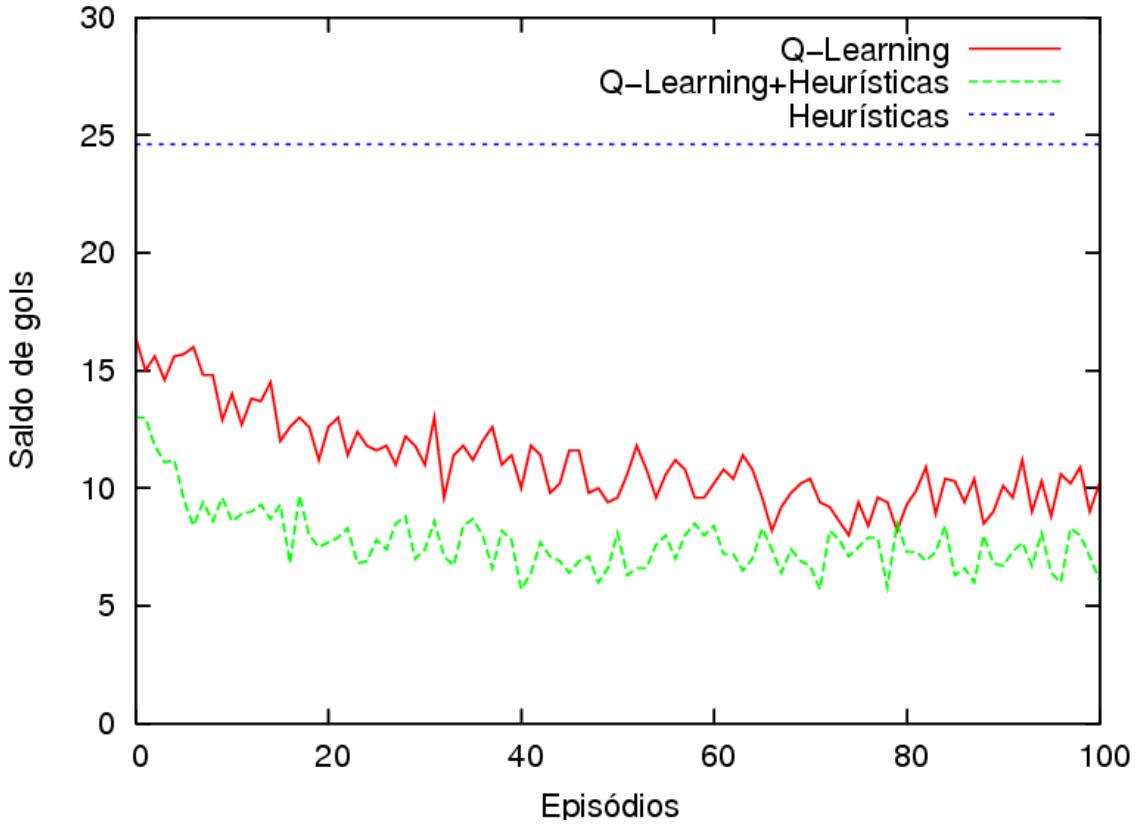


Figura 6.10 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para o agente goleiro e zagueiro.

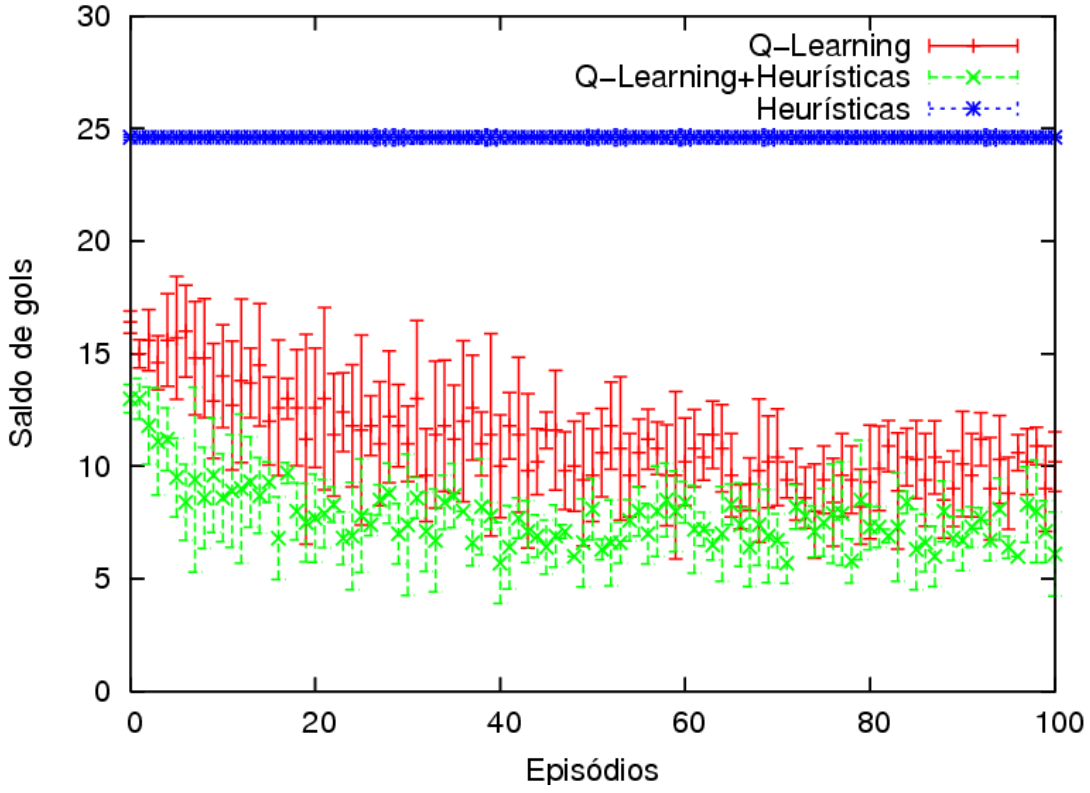


Figura 6.11 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para o agente goleiro e zagueiro, com barras de erro.

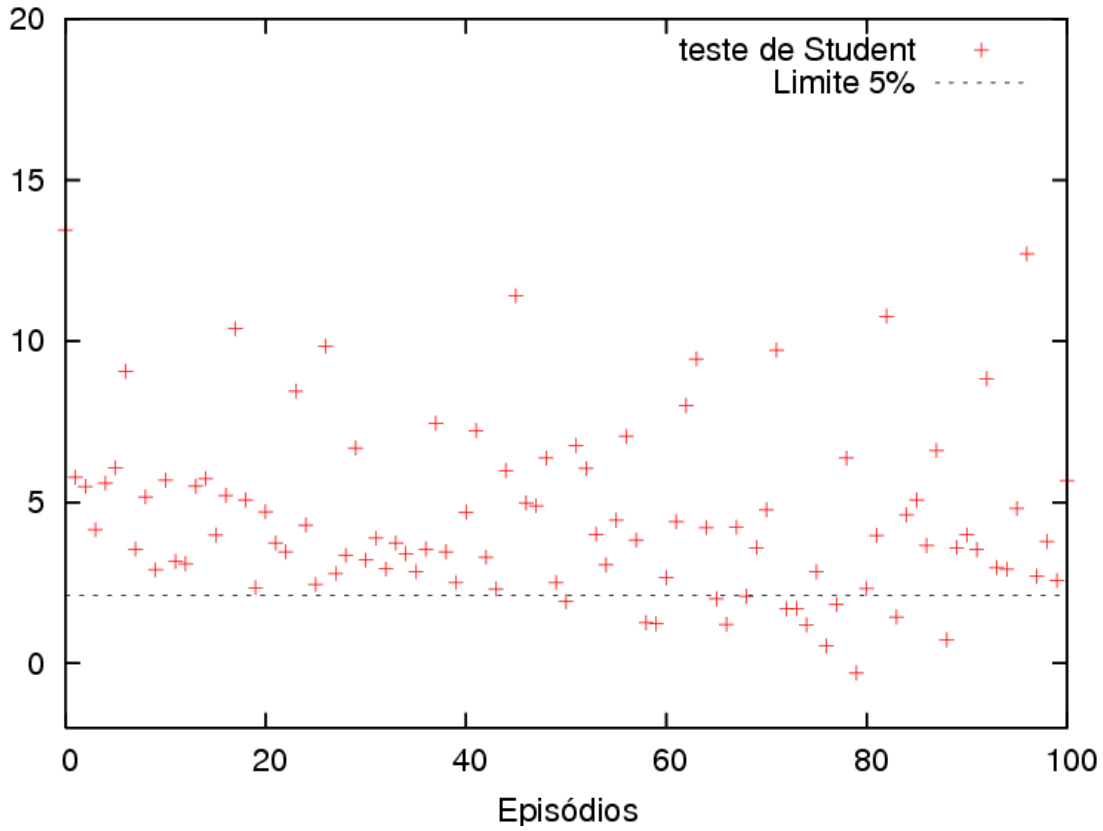


Figura 6.12 - Resultado do teste *t* de Student para os algoritmos *Q-Learning* e HAQL.

Foi analisada a evolução dos valores da tabela Q, para os dois algoritmos, durante o aprendizado. Este valor é composto da diferença dos valores de Q(s, a), para todos os estados e ações, entre dois instantes de tempo. O valor encontrado permite verificar a convergência do algoritmo: no começo do aprendizado a diferença encontrada entre duas tabelas mostra que o agente ainda está aprendendo. Porém, depois de um certo tempo, a diferença é mínima. A Figura 6.13 e a Figura 6.15 mostram os resultados obtidos pela subtração da tabela Q a cada 3.000 ciclos (de uma média de 10 jogos), na Figura 6.14 e na Figura 6.16 6 são apresentados os resultados com barras de erro. Percebe-se que o tempo de convergência do algoritmo HAQL é menor que o do algoritmo *Q-Learning*, similarmente ao analisado na Figura 6.8.

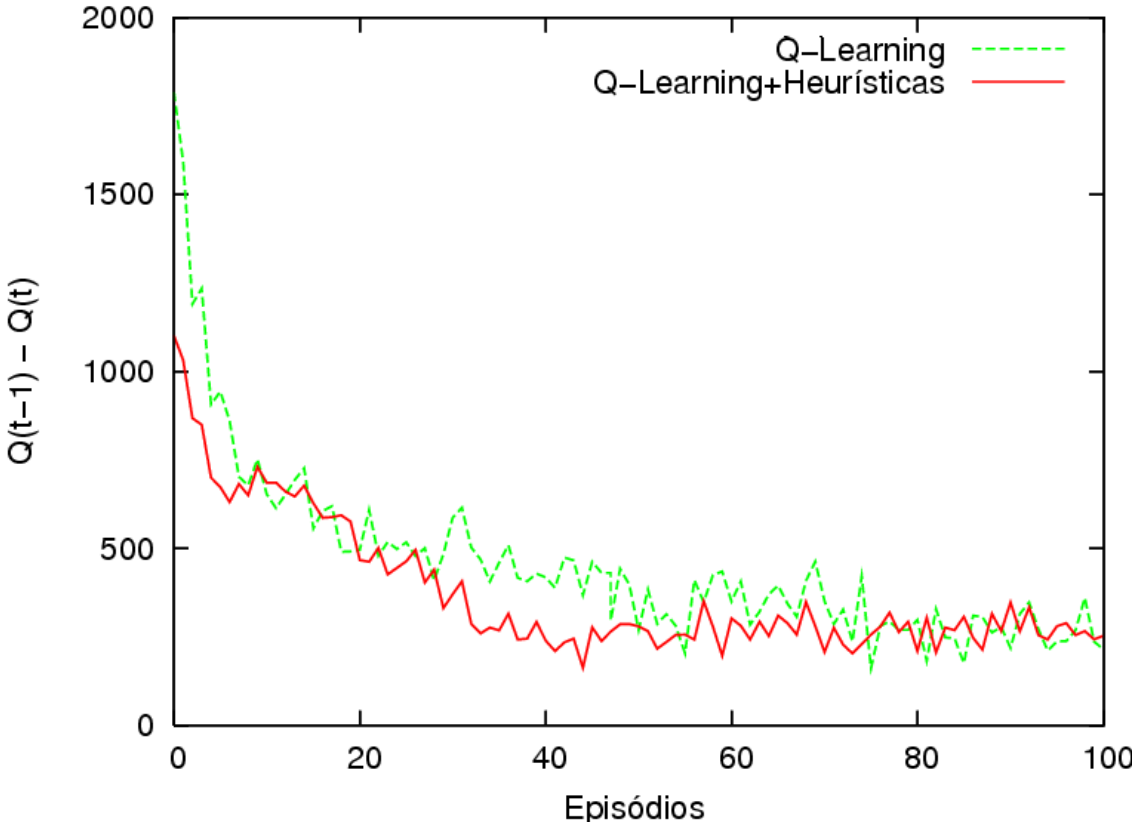


Figura 6.13 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente goleiro.

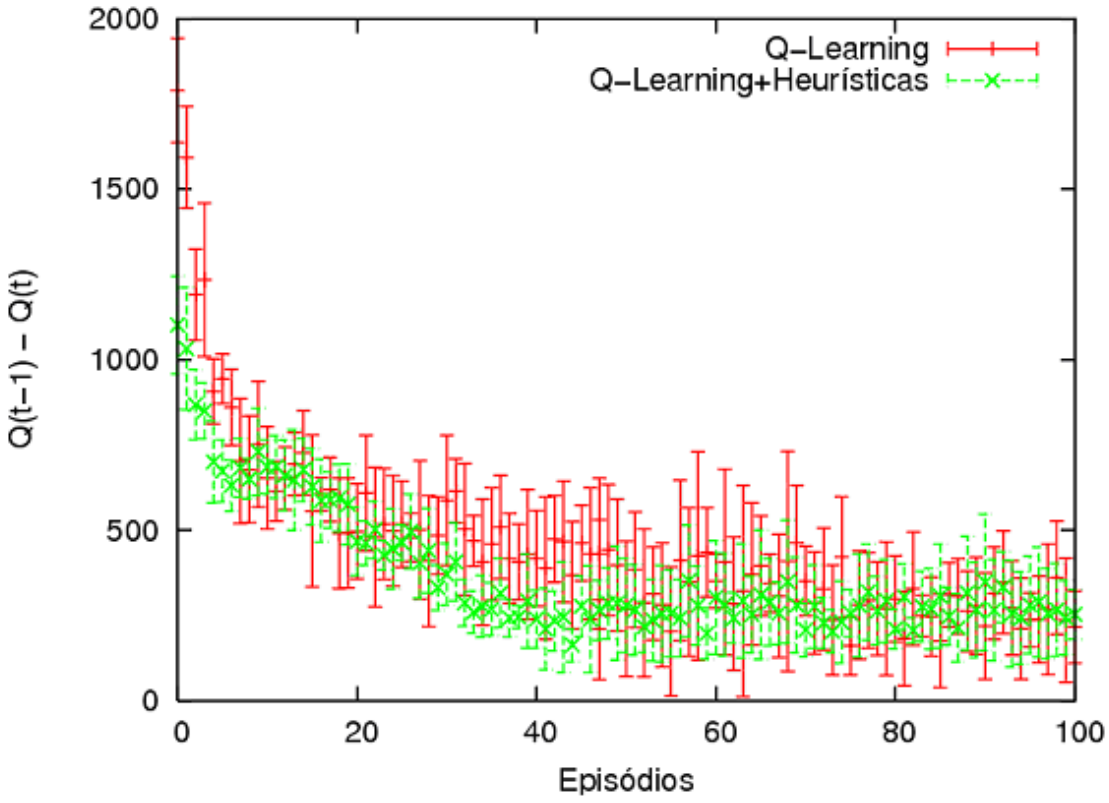


Figura 6.14 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente goleiro, com barras de erro.

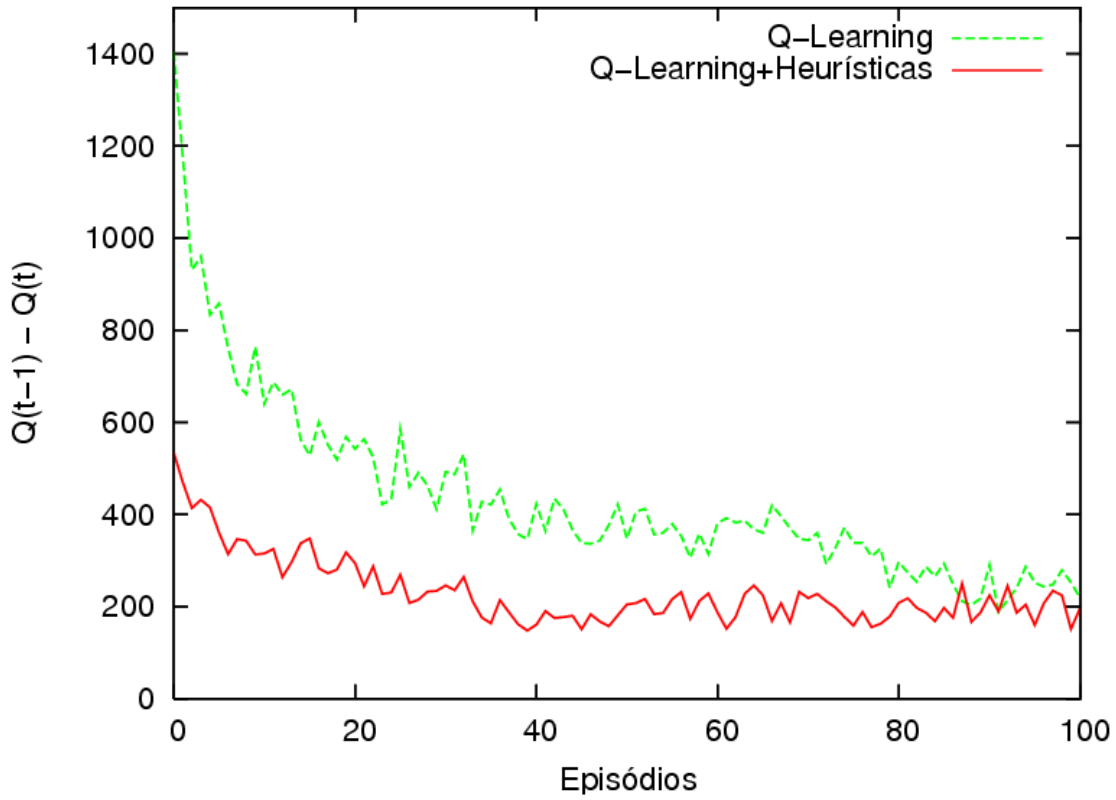


Figura 6.15 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente zagueiro.

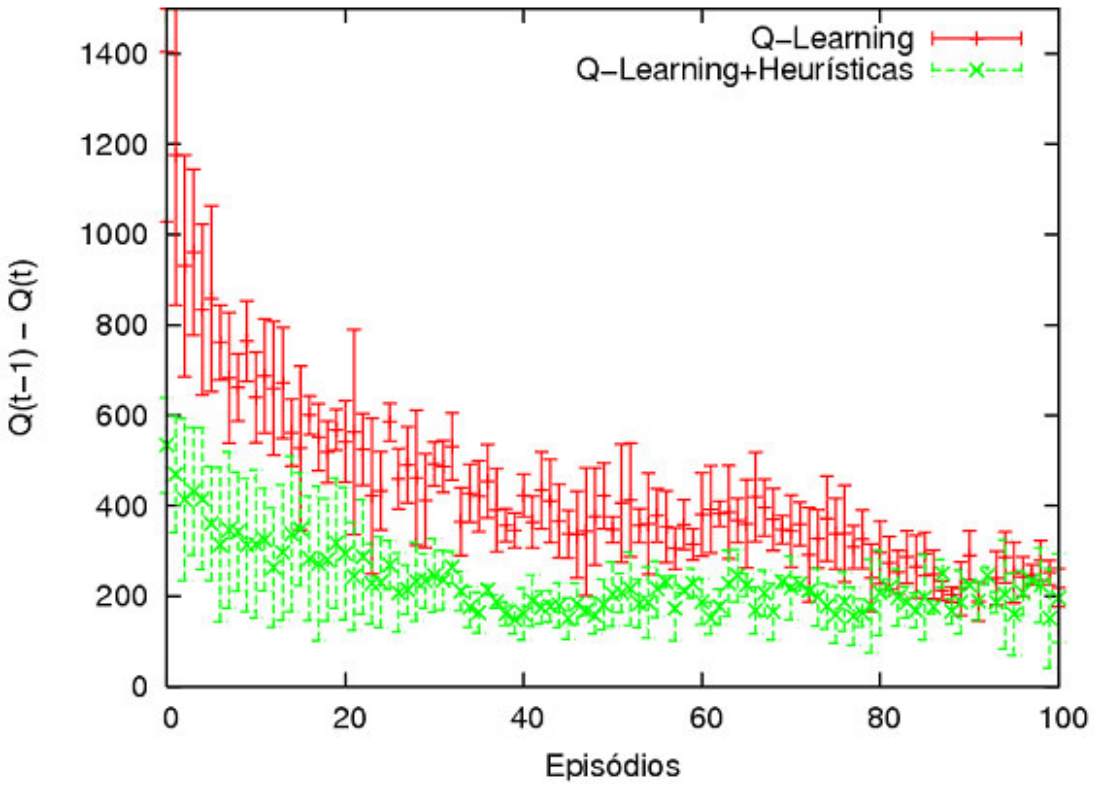


Figura 6.16 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente zagueiro, com barras de erro.

Como no caso do goleiro, é necessário estudar o caso quando o agente zagueiro e a bola estão na mesma área e assim demonstrar que apesar de cada zona ser formada por 5 x 10

m, o aprendizado ocorre, recompensando as melhores ações para aquele estado, este estudo encontra-se comentado no apêndice 1.

6.3 Aprendizado de um Atacante.

Neste cenário um agente precisa aprender a fazer gols contra um outro jogador. A área que o agente terá para trabalhar com a bola, corresponderá ao meio de campo do adversário, mais uma pequena área dentro do seu campo, que corresponderá ao que o agente precisa para sua posição inicial. O agente deve aprender neste cenário, quando deve chutar a bola para o gol. Foram desenvolvidas algumas ações para este fim e foram feitas algumas modificações em relação aos experimentos anteriores no que se diz respeito à criação do estado do agente.

Foi utilizado o goleiro do UVA Trilearn Basic. Este goleiro recebeu algumas restrições, que são: O goleiro só pode chutar a bola, ele deve se locomover somente pela grande e pequena área do gol e seu chute tem potência para ir até ¾ do lado do campo que o goleiro estiver.

Forem feitas tentativas de se construir um estado semelhante aos descritos anteriormente, porém os resultados não foram satisfatórios, o principal motivo foi a existência de um espaço de estados muito maior que os anteriores, visto que o atacante deve no mínimo explorar todo o campo adversário. Para a solução deste problema, foi necessário trabalhar de uma forma melhor a criação dos estados.

Foi criada uma grade virtual sobre todo o campo adversário, mas utilizando uma divisão de zonas por importância, para uma distância inferior a 30 metros do gol adversário, as células desta grade possuem o tamanho de 5x5 metros, para distâncias superiores a 30 metros as células adquirem um tamanho de 10x5 metros, esta mudança foi adotada porque aprender bem o que fazer perto do gol é mais importante do que longe do gol, então é possível ter uma discretização diferente. Foi realizada também, utilizando a noção da importância uma discretização para as distâncias da bola ao agente (mais importante aprender bem quando a bola está perto, do que quando está longe), além da discretização dos ângulos como já feito anteriormente. Sendo assim a chave que representa o estado é mostrada na tabela 6.10, a discretização dos ângulos é apresentada na tabela 6.11 e das distâncias na tabela 6.12, a Figura 6.17 mostra a grade criada para o agente e por final a tabela 6.13 mostra os reforços fornecidos ao agente.

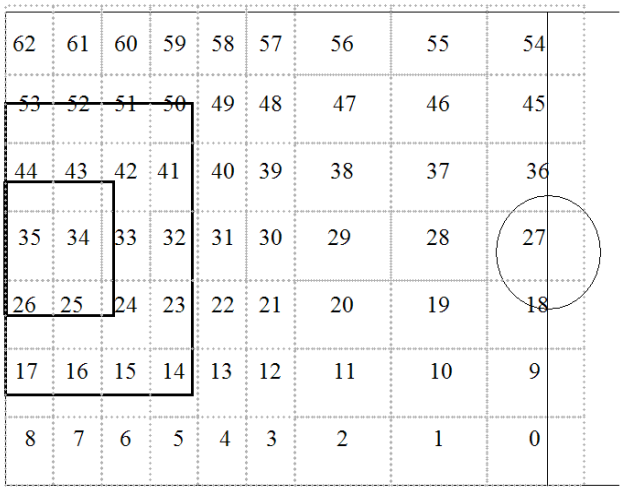


Figura 6.17 – Grade utilizada pelo agente atacante.

Tabela 6.10 - Chave de estado para o atacante.

Ângulo do agente	**Posição do agente**	**Distância bola-agente**
[0; 3]	[0; 62]	[0; 9]

Tabela 6.11 - Discretização dos ângulos.

Ângulo (radianos)	**Representaçao**
[-1/4π; 1/4 π]	0 (para frente)
[1/4 π; ¾ π]	1 (direita)
[3/4 π; π]	2 (para trás)
[-3/4 π ; - π]	2 (para trás)
[-1/4 π;-3/4 π]	3 (esquerda)

Tabela 6.12 - Discretização das distâncias.

Distância bola – agente (metros)	**Representação**
0; 1	0
1; 2	1
2; 3	2
3; 4	3
4; 5	4
5; 15	5
15; 25	6
25; 35	7
35; 45	8
45; 00	9

As ações que o atacante pode realizar são:

- Parado: não executa nenhum movimento,
- Interceptar a bola: determina a melhor posição e velocidade de deslocamento para poder interceptar a bola,
- Chutar para o gol: chutar a bola na direção do gol,
- Chutar a bola para o ataque: chuta a bola, levando ela para a grade área,
- Drible: faz um drible com a bola, passando ela pelo adversário,
- Segurar a bola: parar o movimento da bola,

Tabela 6.13 - Reforços para o atacante.

Situação	**reforço**
Chutar a bola quando estiver a uma distância menor ou igual a 0.7 m.	3
Chutar a bola para o ataque quando estiver a uma distância menor ou igual a 0.7 m.	3
Adversário com a posse da bola e o agente não se aproximando da bola.	-4
Fazer Gol	100

Os valores das heurísticas foram: + 200 para quando deveria influenciar alguma ação positivamente e – 200 no caso contrario. As regras usadas para definir quando é interessante executar uma ação foram:

- A bola a uma distância menor que 1 metro: então não ficar parado,
- A bola a uma distância de 1 metro ou mais: então corra em direção a bola.

Na Figura 6.18 são demonstradas as ações realizadas por um atacante que se prepara para fazer um gol. Nas Figuras (6.18a, 6.18b, 6.18c e 6.18d), são mostradas as ações que levam ao gol. Percebe-se que o atacante vai até a bola e trabalha com ela, levando a bola até o gol. O atacante depois de um certo tempo de aprendizado, pode escolher qual é a ação que deve ser feita naquela ocasião, neste caso ele chuta a bola para o gol. O goleiro do UVA tentará evitar o gol, e para este caso demonstrado na Figura 6.18, ele tem grandes chances de evitar este gol. Caso o gol não seja realizado, o atacante irá aprender que naquela posição não é o melhor local para se fazer o gol, e irá por meio de tentativa e erro, procurar novas posições para chutes.

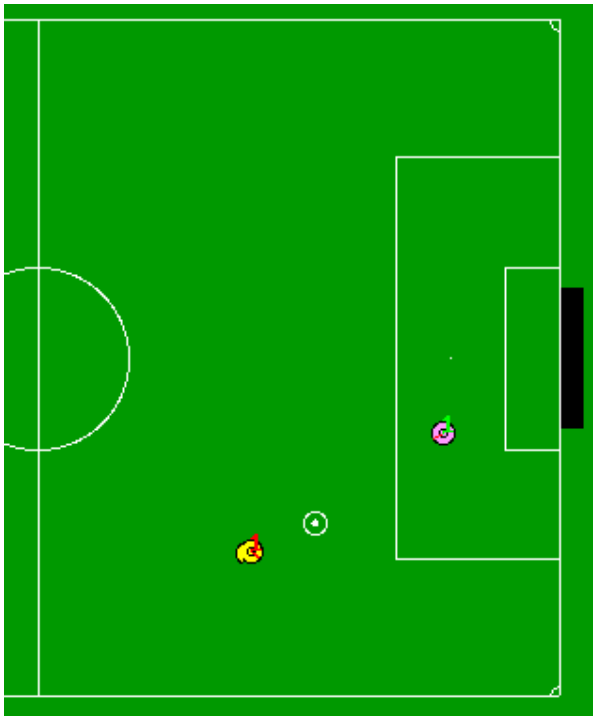


Figura 6.18a – Ação do atacante

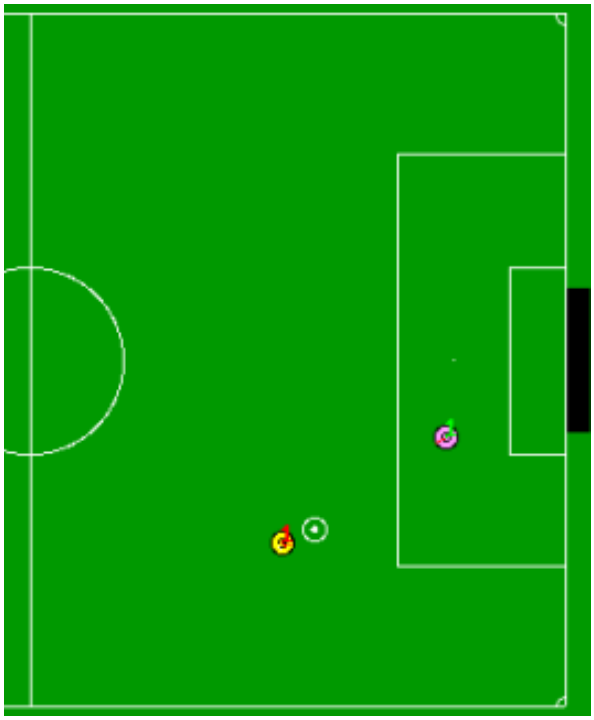


Figura 6.18b – Ação do atacante

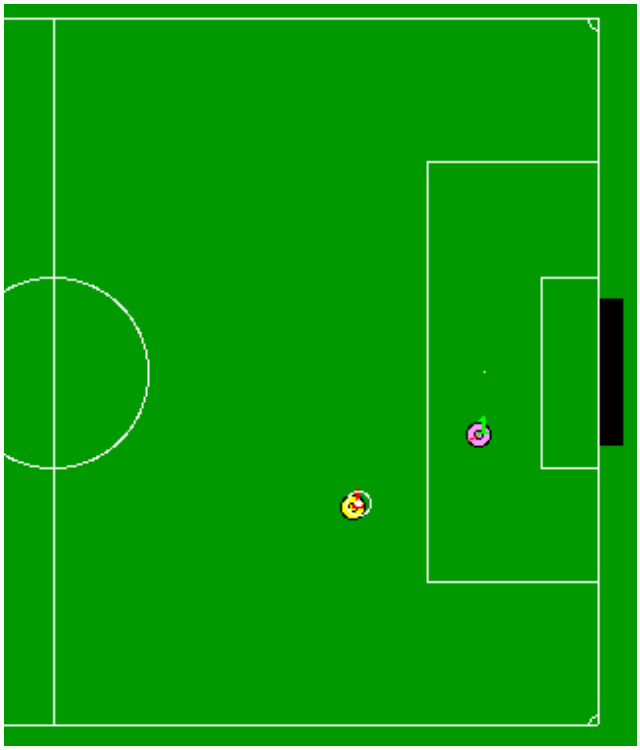


Figura 6.18c – Ação do atacante

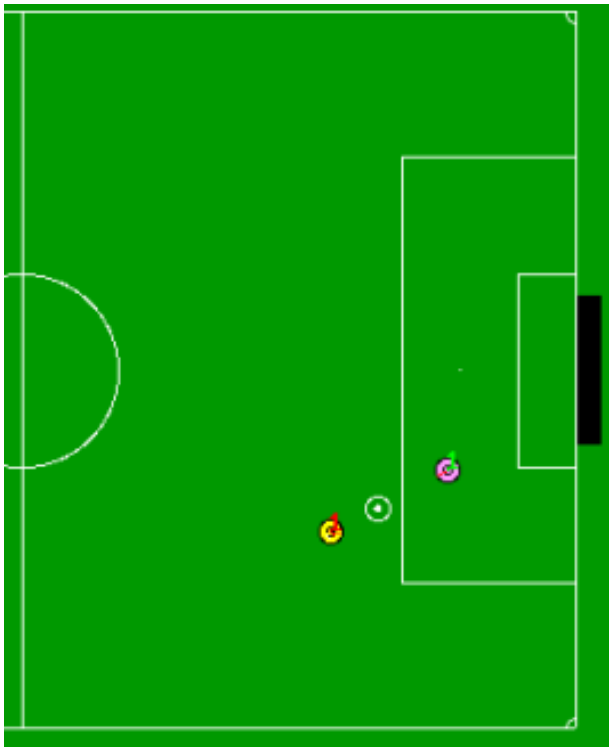


Figura 6.18d – Ação do atacante

Os resultados para a execução dos algoritmos *Q-Learning* e HAQL (média de 10 jogos) são apresentados na Figura 6.19 e na Figura 6.20 (com barras de erro). O uso de somente o *Q-Learning*, o atacante começa fazendo uma média de 7 gols e no episódio número 100 este valor fica em aproximadamente 18 gols. Comparando com o algoritmo *HAQL*, pode-se ver que o atacante começa com um valor de aproximadamente 12 gols e no final faz um média de 18 gols no goleiro do UVA Trilearn. Também se pode observar que o uso das heurísticas em um agente sem aprendizado não evolui com o tempo, e que não produz um atacante eficiente, não fazendo praticamente gols. Com isso, pode-se concluir que as heurísticas aceleram aprendizado, mas não são tão fortes que eliminam a necessidade do aprendizado.

Foi utilizado o teste *t* de Student (SPIEGEL, 1984). O resultado para o teste *t* de Student computado utilizando os dados da Figura 6.19, é apresentado na Figura 6.21. Nesta Figura é possível ver que o começo do aprendizado ocorreu mais rápido, porém depois do episódio número 18 as diferenças dos gráficos não são grandes. Para este caso, o inicio do gráfico mostra que os algoritmos são significativamente diferentes, com nível de confiança maior que 5% até o episódio número 18. Isto confirma que as heurísticas usadas tornam o algoritmo HAQL mais rápido que o *Q-Learning*. Os valores totais da soma de gols feitos pelo atacante, são apresentados na tabela 6.14, as diferenças de gols feitos com o uso de heurísticas foram de 162 gols a mais.

Tabela 6.14 - Total de gols feitos (média e desvio padrão).

Algoritmo	**Gols**
Q-Learning.	1742 ± 9
HAQL	1904 ± 89

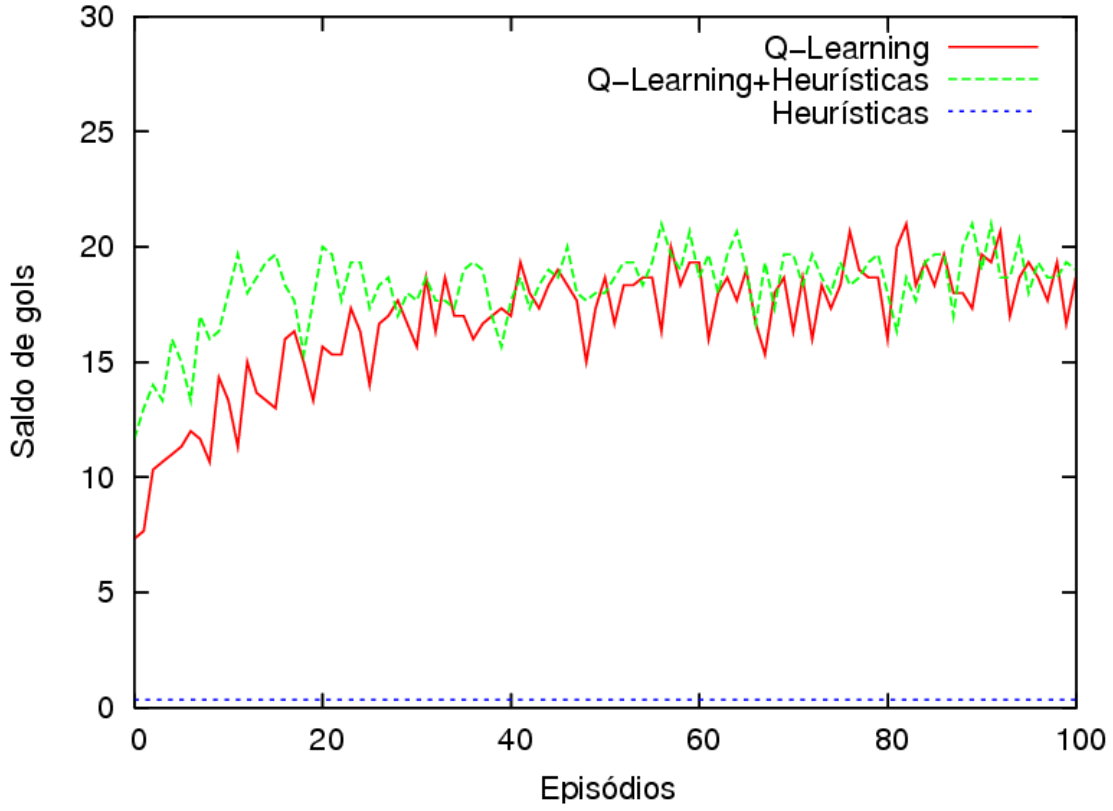


Figura 6.19 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para o agente atacante.

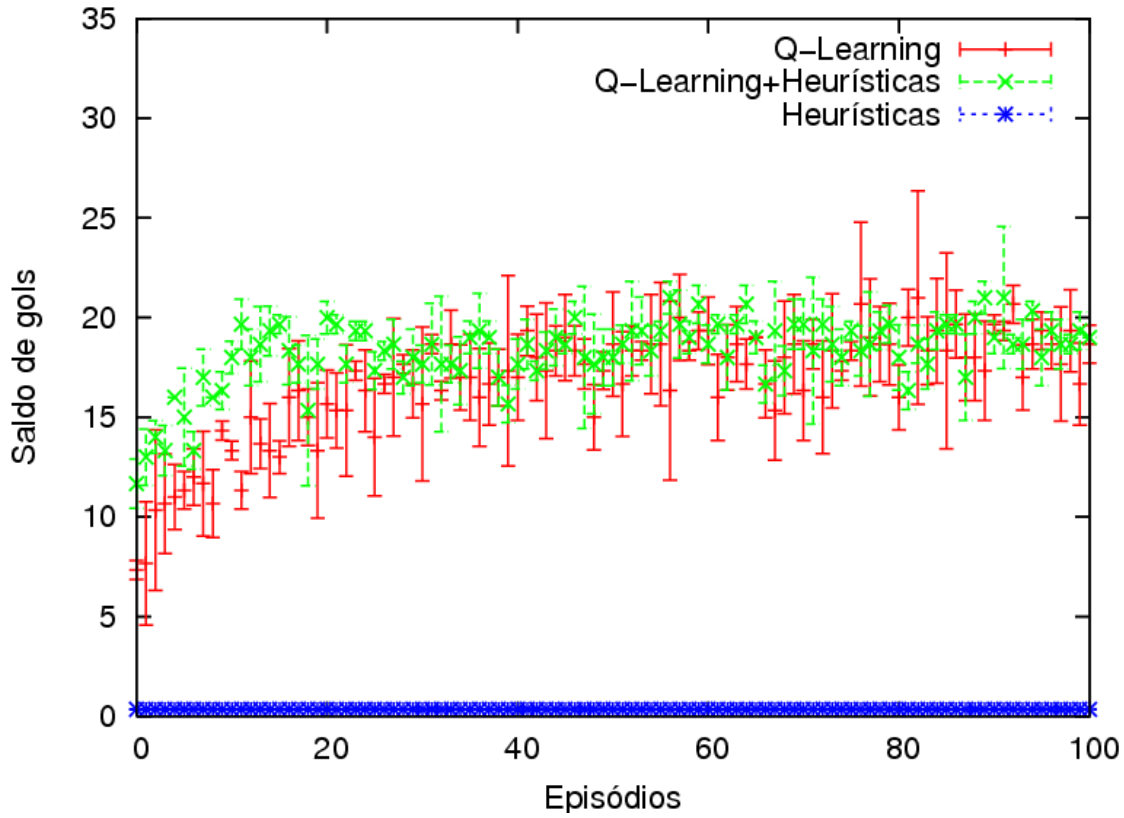


Figura 6.20 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para o agente atacante, com barras de erro.

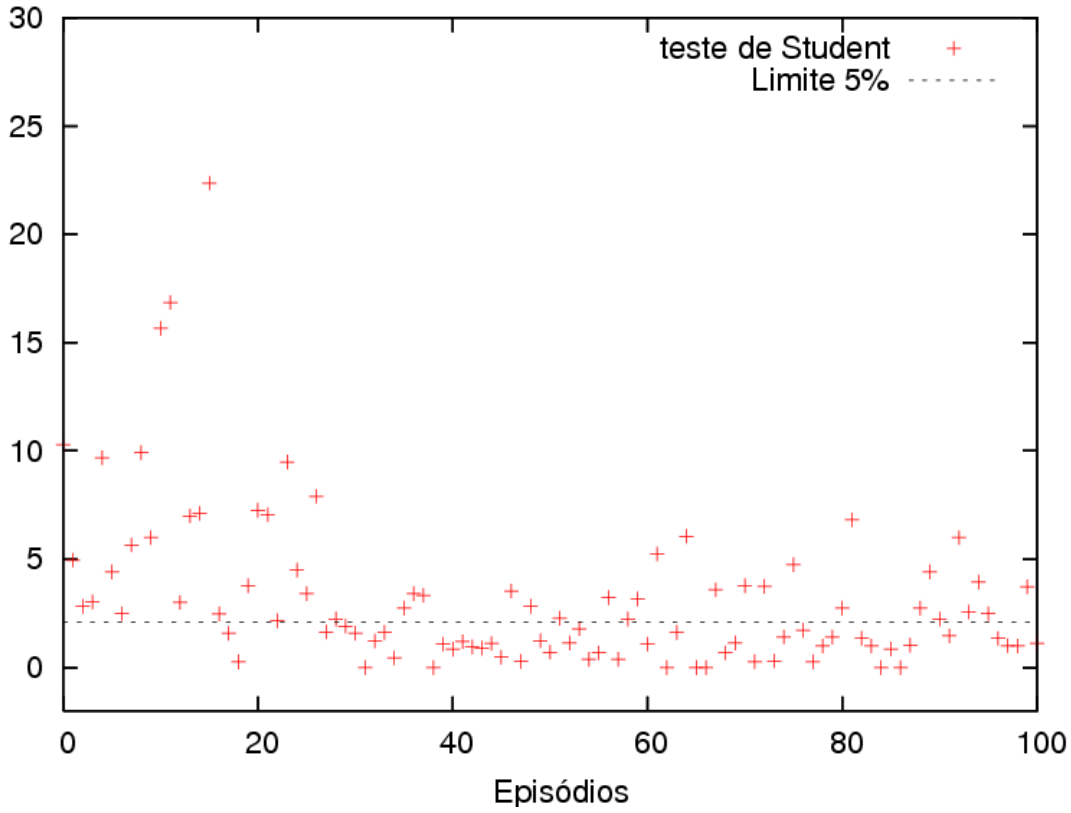


Figura 6.21 - Resultado do teste t de Student para os algoritmos Q-Learning e HAQL.

Foi analisada a evolução dos valores da tabela Q, para os dois algoritmos, durante o aprendizado. Este valor é composto da diferença dos valores de Q(s, a), para todos os estados e ações. O valor encontrado permite verificar a convergência do algoritmo: no começo do aprendizado a diferença encontrada entre duas tabelas mostra que o agente ainda está aprendendo. Porém, depois de um certo tempo, a diferença é mínima. A Figura 6.22 mostra os resultados obtidos pela subtração da tabela Q a cada 3.000 ciclos (de uma média de 10 jogos), na Figura 6.23 são apresentados os resultados com barras de erro. Percebe-se que o tempo de convergência do algoritmo HAQL é menor que o do algoritmo *Q-Learning*, similarmente ao analisado na Figura 6.19.

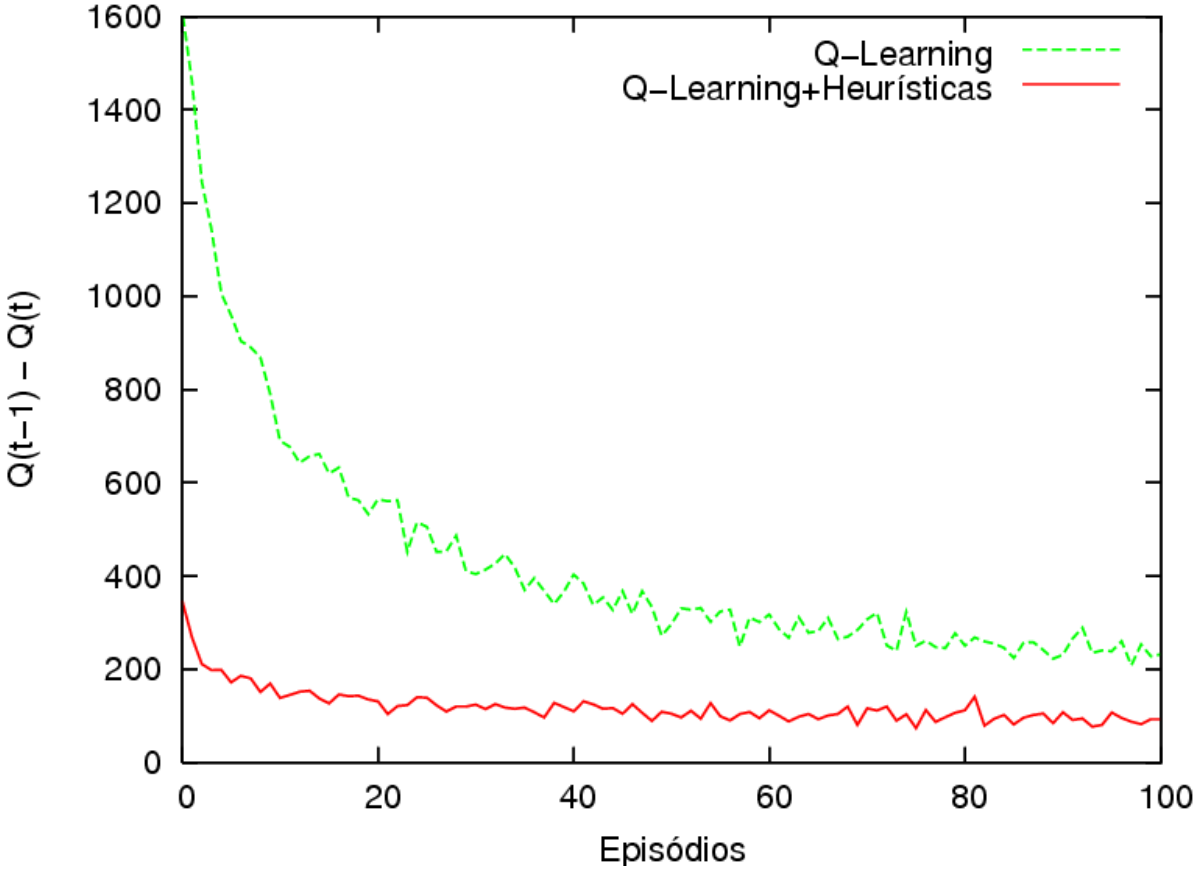


Figura 6.22 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente atacante.

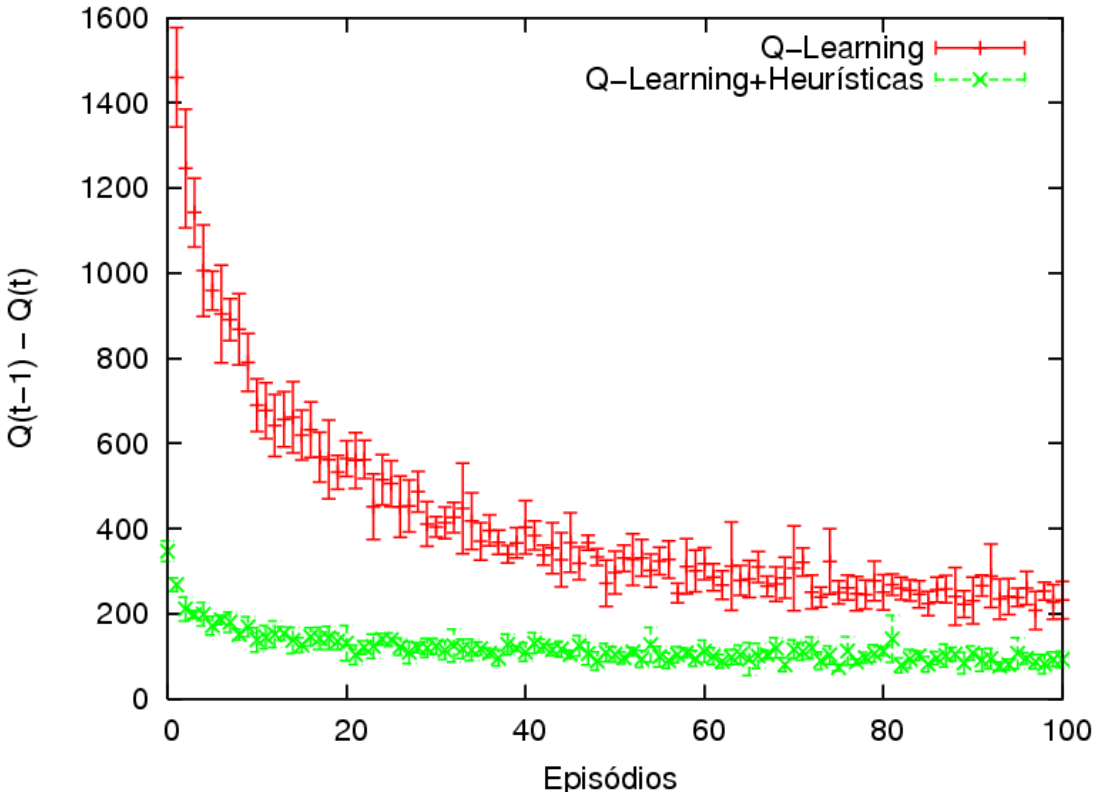


Figura 6.23 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente atacante, com barras de erro.

6.4 Aprendizado de Dois Atacantes

Muito parecido com a experiência anterior, porém agora com a adição de mais um atacante. Os agentes precisam aprender, neste cenário, quando é a melhor oportunidade para chutar a bola para o gol, ou se a melhor escolha é passar a bola ao adversário. Foi utilizado o goleiro do UVA Trilearn Basic e um outro jogador do UVA Trilearn Basic como zagueiro. Estes dois jogadores receberam algumas restrições, que são: O goleiro só pode chutar a bola, ele deve se locomover somente pela grande e pequena área do gol e seu chute tem potência para ir até ¾ do lado do campo que o goleiro estiver. Para o zagueiro o chute deve ir ao máximo até ¾ do seu campo e o zagueiro pode se locomover até ¾ do campo do lado que estiver.

A principal mudança realizada entre este experimento e o anterior (6.3) foi à adição de mais um item na chave de estado do agente: o valor da distância entre os agentes. O objetivo desta adição na chave é fornecer informação de quando é melhor ou pior momento, para fazer um passe o outro agente, em detrimento da distância entre eles. A tabela 6.15 mostra a discretização das distâncias e a tabela 6.16 mostra como fica a nova chave formada. A grade utilizada é exatamente igual ao citado no experimento anterior (Figura 6.17), bem como a discretização dos ângulos do agente (tabela 6.11). Por final a tabela 6.17 mostra os reforços utilizados.

As ações que os atacantes podem realizar são praticamente a mesma, exceção de uma ação entre eles. O atacante 1 (o primeiro jogador que entrar em campo), possui a ação de marcar um outro jogador adversário, porém o atacante 2 (segundo jogador que entrar em campo) não possui esta ação e em vez dela, possui a ação de segurar a bola. Esta mudança foi adotada para fornecer uma pequena característica diferente aos jogadores.

Tabela 6.15 - Discretização das distâncias.

Distância agente – agente (metros)	**Representação**
0; 10	0
10; 20	1
20; 30	2
30; 40	3
40; 50	4
50; 00	5

Tabela 6.16 - Chave de estado.

Ângulo do agente	Posição do agente	Distância da bola	Distância agente -agente
[0; 3]	[0; 62]	[0; 9]	[0; 5]

Tabela 6.17 – Reforços.

Situação	Reforço
Chutar a bola quando estiver a uma distância menor ou igual a 0.7 m.	3
Chutar a bola para o ataque quando estiver a uma distância menor ou igual a 0.7 m.	3
Adversário com a posse da bola e o agente não se aproximando da bola.	-4
Fazer Gol	100
Outro jogador fez o gol	25

As ações possíveis então de serem realizadas são:

- Parado: não executa nenhum movimento,
- Interceptar a bola: determina a melhor posição e velocidade de deslocamento para poder interceptar a bola,
- Chutar para o gol: chutar a bola na direção do gol,
- Drible: faz um drible com a bola, passando ela pelo adversário,
- Marcar: ficar perto de um jogador adversário e interceptar a bola quando ela for à direção deste adversário (ação feita somente pelo atacante 1),
- Segurar a bola: para o movimento da bola adversário (ação feita somente pelo atacante 2),
- Passe: passar a bola ao outro jogador.

As heurísticas que foram aplicadas aos atacantes são similares as utilizadas nos experimentos anteriores. Os valores das heurísticas foram: + 200 para quando deveria influenciar alguma ação positivamente e – 200 no caso contrario. As regras usadas para definir quando é interessante executar uma ação foram:

- A bola a uma distância menor que 1 metro: então não ficar parado,
- A bola a uma distância de 1 metro ou mais: então corra em direção a bola.

Na Figura 6.23 são demonstradas as ações realizadas por dois atacantes. Nas Figuras (6.23a, 6.23b, 6.23c e 6.23d), são mostradas estas ações feitas por dois atacantes, dando ênfase na marcação. Um dos atacantes faz a saída da bola, sendo está interceptada por outro jogador atacante, neste intervalo de tempo, o jogador atacante que fez a saída da bola, escolhe marcar o zagueiro adversário, e assim dificultando o recebimento da bola por este jogador. O atacante com a posse da bola, tentará levar esta até o gol adversário, e tentará fazer o gol, sendo ajudado por seu companheiro que poderá ficar marcando o adversário, para pegar alguma sobra na defesa da bola feita pelo goleiro.

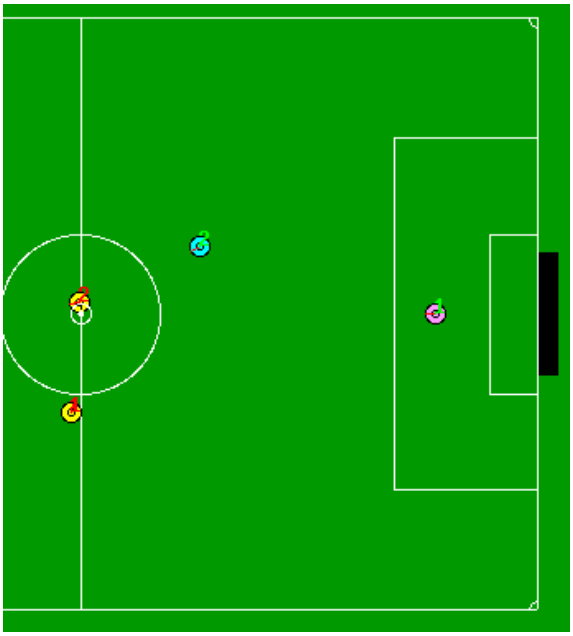


Figura 6.23a – Ações dos atacantes

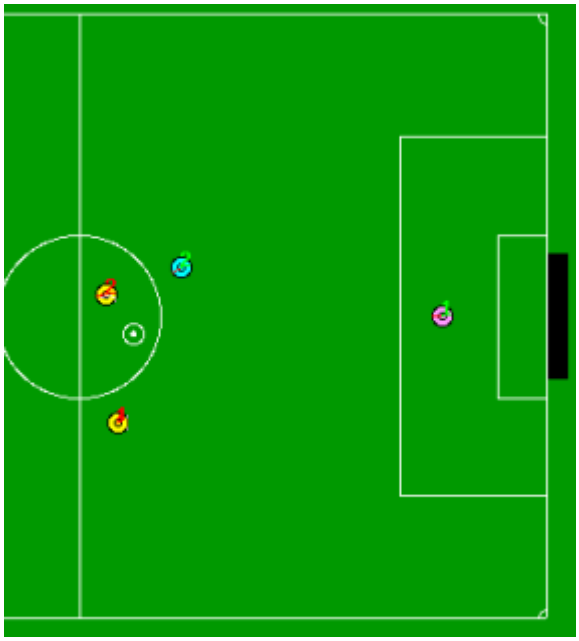


Figura 6.23b – Ações dos atacantes

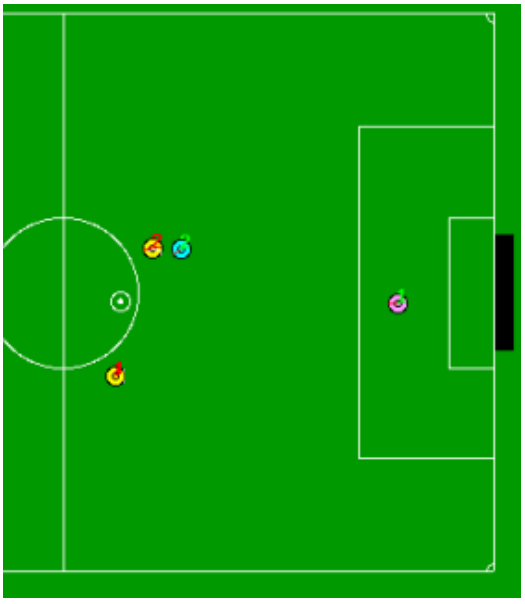


Figura 6.23c – Ações dos atacantes

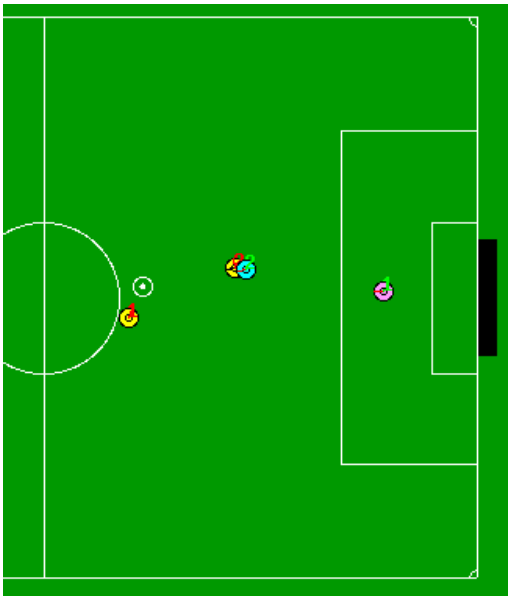


Figura 6.23d – Ações dos atacantes

Os resultados para a execução dos algoritmos *Q-Learning* e HAQL (média de 10 jogos) são apresentados na Figura 6.24 e na Figura 6.25 (com barras de erro). Nos gráficos da Figura 6.24 mostram que com o uso de somente o *Q-Learning*, o atacante começa com uma média de 3 gols feitos e depois de algum tempo, este valor sobe para uma média de 16 gols no episódio número 40. Comparando com o algoritmo *HAQL* pode-se ver que os atacantes começam com um valor de aproximadamente 9 gols feitos e no episódio número 12 faz um média de 16 gols no goleiro do UVA Trilearn. Como em todos os outros experimentos foi observado o uso de somente as heurísticas, estas apesar de ajudar no aprendizado, estas não são suficientemente fortes para serem usadas sozinhas para influenciar os atacantes para fazerem gols.

Foi utilizado o teste *t* de Student (SPIEGEL, 1984). O resultado para o teste *t* de Student computado utilizando os dados da Figura 6.24, é apresentado na Figura 6.26. Nesta Figura é possível ver que no começo do aprendizado (até episódio 18), os algoritmos são significativamente diferentes (bem evidente até o episódio número 12), com nível de confiança maior que 5%. Isto confirma que as heurísticas usadas tornam o algoritmo HAQL mais rápido que o *Q-Learning*.

A soma dos gols feitos pelos agentes atacantes em 100 episódios (média de 10 treinamentos) utilizando o algoritmo *Q-Learning* e o HAQL é apresentada na tabela 6.18. Pode-se notar que o agente que utiliza o algoritmo HAQL fez 80 gols a mais que o agente que utiliza o *Q-Learning*.

Tabela 6.18 - Total de gols feitos e desvio padrão.

Algoritmo	**Gols**
Q-Learning	1484 ± 2
HAQL	1564 ± 55

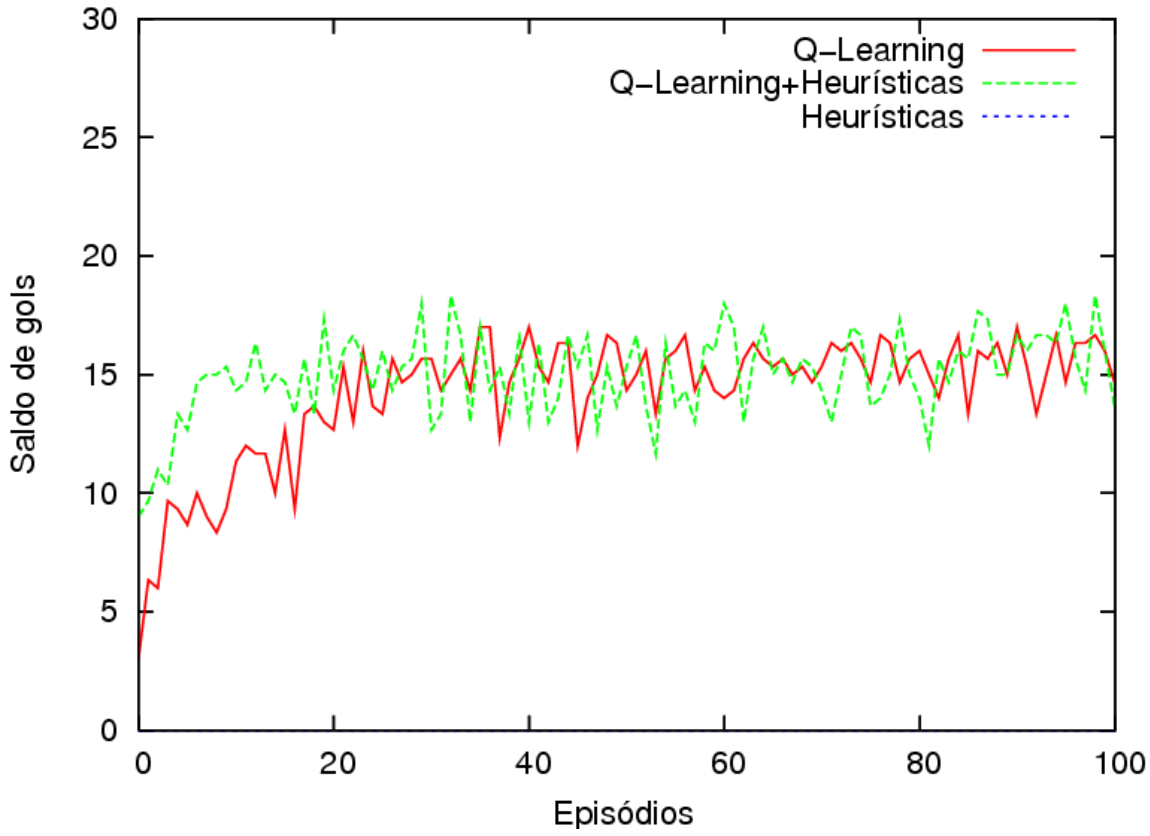


Figura 6.24 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para os atacantes.

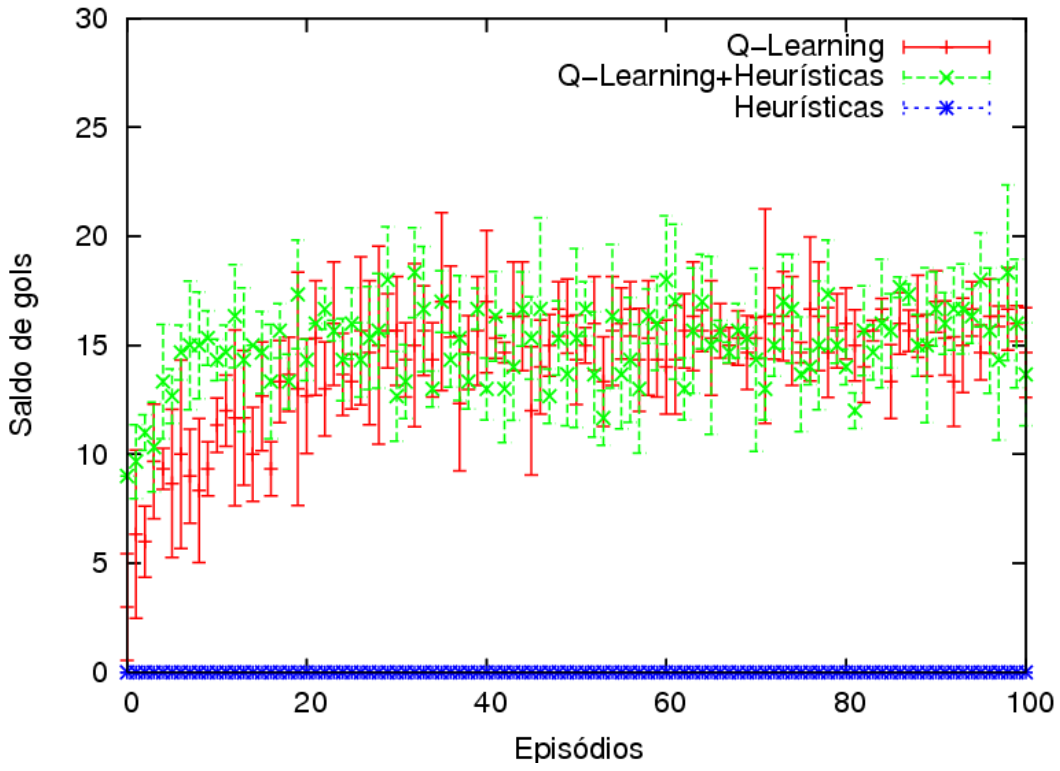


Figura 6.25 - Evolução do saldo de gols para os algoritmos *Q-Learning*, HAQL e somente heurísticas para os agentes atacantes., com barras de erro.

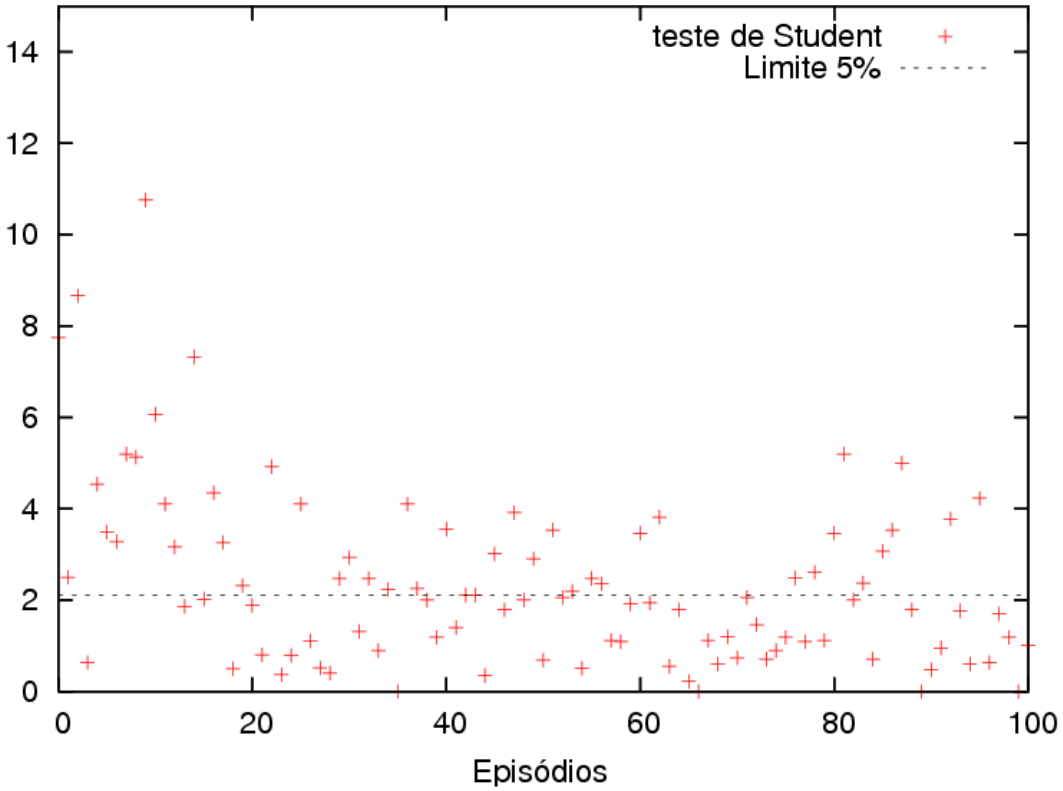


Figura 6.26 - Resultado do teste *t* de Student para os algoritmos *Q-Learning* e HAQL.

Foi analisada a evolução dos valores da tabela Q, para os dois algoritmos, durante o aprendizado. O valor encontrado permite verificar a convergência do algoritmo: no começo do aprendizado a diferença encontrada entre duas tabelas mostra que o agente ainda está aprendendo. Porém, depois de um certo tempo, a diferença é mínima. A Figura 6.27 e Figura 6.29 mostram os resultados obtidos pela subtração da tabela Q a cada 3.000 ciclos (de uma média de 10 jogos) dos dois atacantes, na Figura 6.28 e na Figura 6.30 são apresentados os resultados com barras de erro. Percebe-se que o tempo de convergência do algoritmo HAQL é menor que o do algoritmo *Q-Learning*, similarmente ao analisado na Figura 6.21.

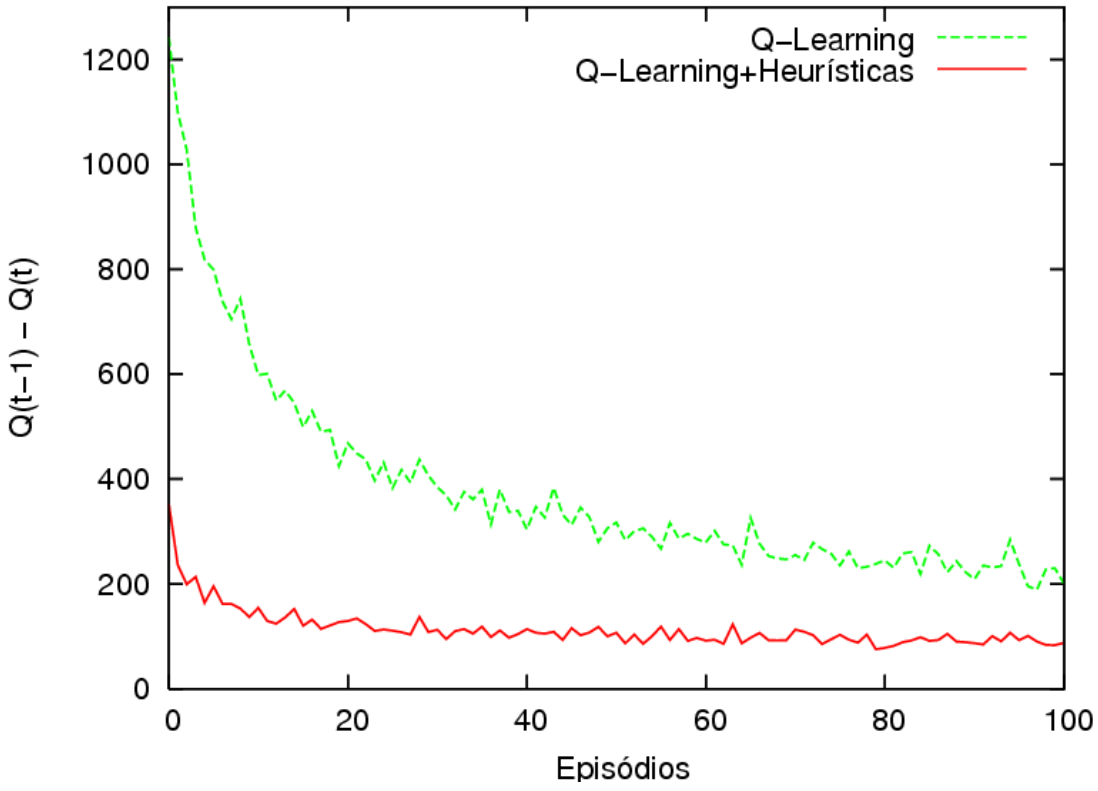


Figura 6.27 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente atacante 1.

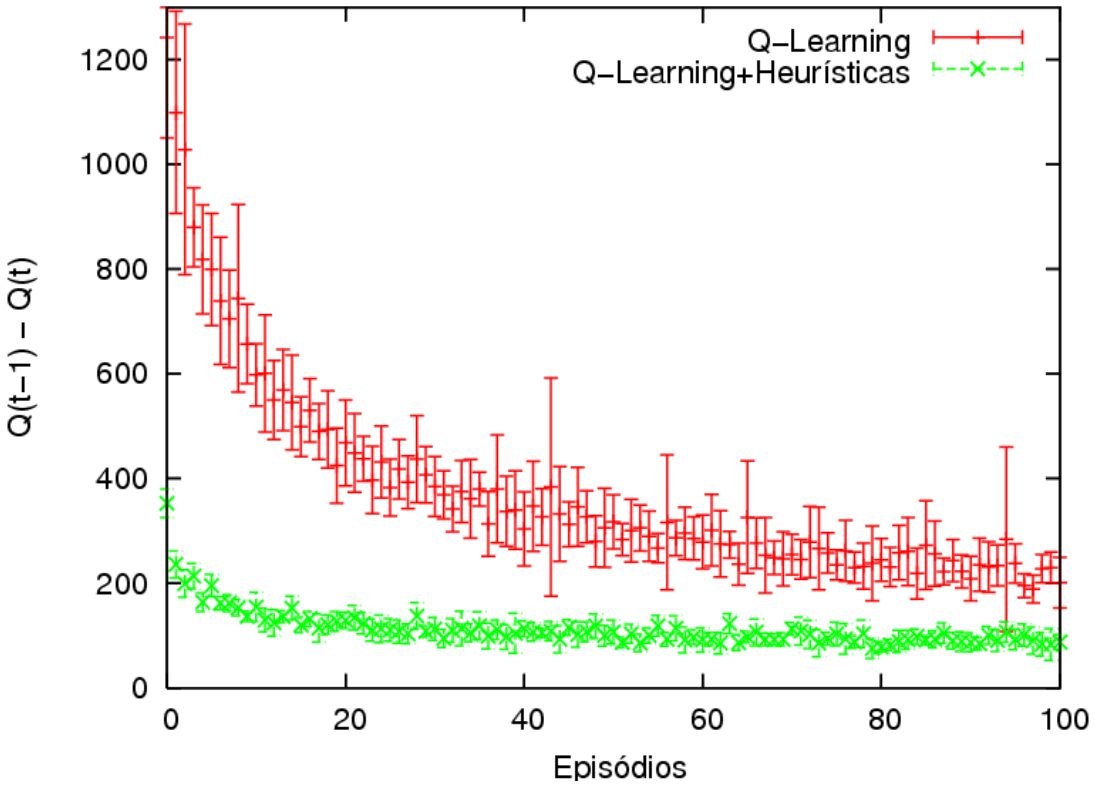


Figura 6.28 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente atacante 1, com barras de erro.

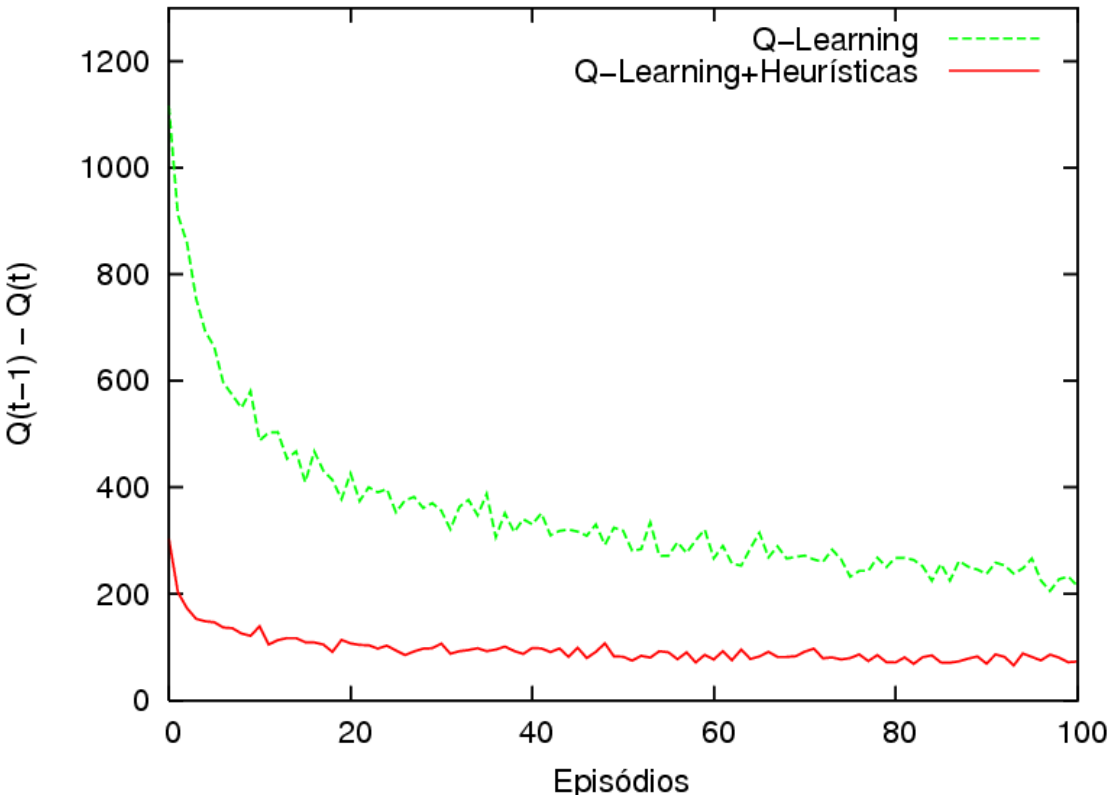


Figura 6.29 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente atacante 2.

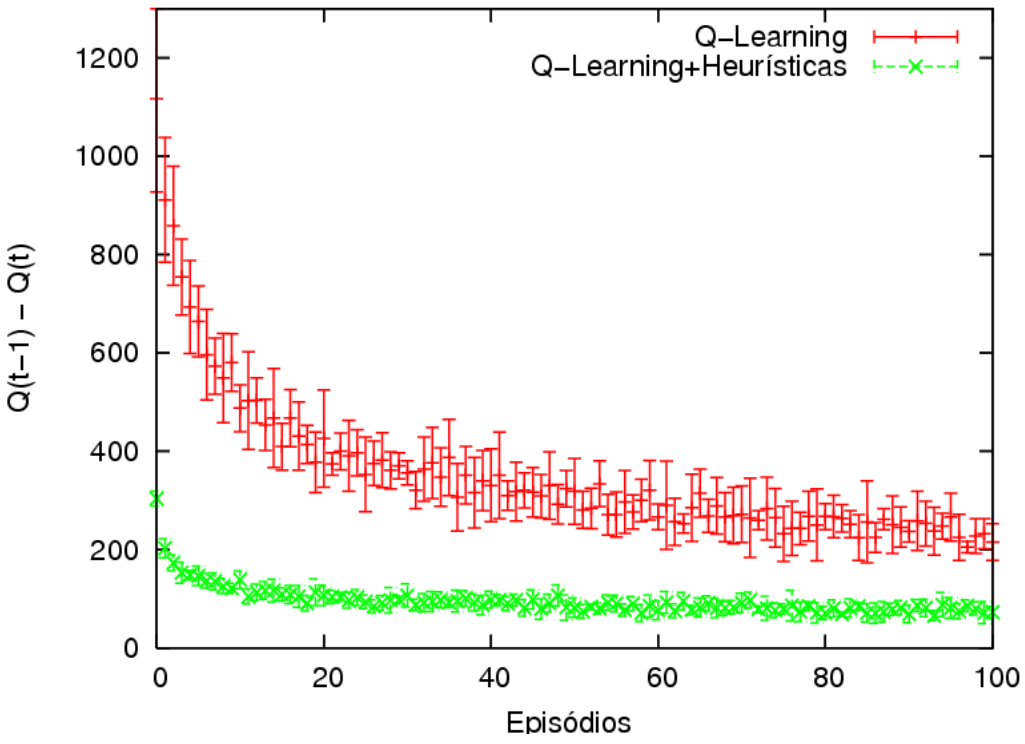


Figura 6.30 - Evolução da tabela Q para os algoritmos *Q-Learning* e HAQL, agente atacante 2, com barras de erro.

6.5 Discussões

Este capítulo mostrou que com uso de heurísticas para a aceleração do aprendizado para o domínio do futebol de robôs pode apresentar bons resultados. Foram realizados quatro experimentos e a tabela 6.19 mostra os saldos de gols (feitos ou tomados) dos experimentos.

Tabela 6.19 – Comparação dos Saldos de gols (média e desvio padrão).

Algoritmos	**Experimento**			
	1	**2**	**3**	**4**
Q-Learning	1451 ± 21	1177 ± 51	1742 ± 9	1484 ± 2
HAQL	1026 ± 7	836 ± 10	1904 ± 89	1564 ± 55
	- 425	- 341	+162	+ 80

Pode-se perceber que dos experimentos realizados, o que teve melhor rendimento (diferença de gols) com o uso do HAQL, foi o experimento número 1 (goleiro). Porém este necessitou de um valor maior de recompensas para poder aprender. Por outro lado o que teve o menor rendimento, foi o experimento 4, que precisou de um menor valor de reforço. Nota-se que o valor do reforço, está diretamente ligado no desempenho global do agente. Isto se deve ao fato, que um reforço muito alto faz com que o agente aprenda mais rapidamente quais são as melhores ações a serem executadas e quais não devem ser.

Neste trabalho, os valores de reforços foram adotados por meio de experimentos empíricos, que determinaram quais os valores mínimos necessários de reforços para um agente aprender, outro fator que se tornou determinante foi uma janela de tempo de observação, foi estipulado um valor de no máximo 150 episódios para o aprendizado. Este tempo compreende aproximadamente 12.5 horas de treinamento, e o seu valor foi escolhido com base na quantidade de experimentos necessários a se fazer e o tempo total para término deste trabalho. Apesar de todos os experimentos terem sido com 150 episódios, o único que realmente usou todo este tempo foi o experimento 1, nos outros a partir do episódio número 100 já era visível uma estabilidade no gráfico do aprendizado.

Outro fator importante a se levantar, é que o primeiro experimento precisou de mais reforços, devido ao tamanho da área de defesa do agente. Assim, é possível observar nos experimentos que quanto maior a área, menor é o valor de reforço necessário. O motivo para este fato é que quanto menor é a área que o agente atuará no *RoboCup*, menor será seu contato

com os agentes adversários e assim um valor maior de reforço é necessário para que neste breve tempo de contato com o adversário, o agente aprenda o que deve fazer.

Todos os experimentos receberam heurísticas muito similares: em todos os casos as heurísticas tinham a função de informar ao agente que quando a bola está muito longe, a melhor ação a ser realizada é ir até ela e quando a bola está perto é melhor realizar uma ação qualquer do que ficar parado.

Outro ponto de análise é a criação das áreas de trabalho dos agentes (zonas). Nas experiências usando agente que defendiam o gol (experiência 2) os agentes trabalhavam em zonas diferentes e nas experiências que os agentes tinham que fazer gol (experiência 4), estes trabalhavam na mesma zona. Quando se tem um goleiro e um zagueiro, é fácil perceber que estes têm uma área de trabalho, pois o zagueiro deve trabalhar para a bola não chegar até o gol, e se caso ele não tiver sucesso e a bola passar por ele, quem deve ter o maior trabalho é o goleiro, porém quando os atacantes devem levar a bola para gols, e eles não têm zonas diferentes, ambos devem trabalhar na mesma zona e de preferência com alguma interação. Ficou evidente nos resultados, que defender uma bola, é mais fácil que fazer um gol.

Tanto na experiência 3 como na 4, a criação das zonas são diferentes que nas duas primeiras experiências, visto que o espaço de estados começou a ficar muito grande, quando se criava o estado da forma dos experimentos anteriores. Assim, foi necessária uma nova maneira de descrever o espaço de estados dos agentes. O trabalho desenvolvido por MAUSBERG (2005) e do conceito do CMAC (ALBUS, 1975), permitiram a criação de um espaço de estados generalizado em zonas de diferentes formas. Em outras palavras, pode-se aprender mais ou menos sobre determinado assunto, dependendo de sua importância. Ao aplicar estes conceitos, foi obtido um espaço de estados de menor tamanho e assim foi possível desenvolver um sistema de aprendizado para um ou dois atacantes.

O próximo capítulo apresenta as conclusões deste trabalho e indica alguns trabalhos futuros nesta mesma linha de estudo.

7 CONCLUSÃO E TRABALHOS FUTUROS

Os quatro experimentos realizados neste trabalho: um goleiro, um zagueiro com um goleiro ,um atacante e por final dois atacantes, visam explorar da melhor forma o uso da heurística na aceleração do aprendizado dos agentes aqui utilizados.

A vantagem do uso das heurísticas é que estas limitam o espaço de busca realizada pelos agentes e indica o melhor caminho a seguir, reduzindo assim a busca. Outra vantagem ao se utilizar o aprendizado com o uso de heurísticas é a possibilidade dos agentes poderem aprender a superar as heurísticas ruins e encontrar o melhor caminho.

Para a realização deste trabalho, diversas tentativas de se criar agentes inteligentes foram feitas, porém algumas não obtiveram sucesso. Um dos primeiros experimentos foi à criação de um time, utilizando uma única tabela de aprendizado, foi-se dado como reforço somente ao ver a bola e gol e também por fazer gol. Para movimento dos agentes e informar o que deveria ser feito, foram utilizadas ações de baixo nível, como, “andar”, “chutar” e “virar”. O resultado deste experimento foi insatisfatório, com todos os jogadores no final do aprendizado olhando para a bola e alguns olhando para a bola e para o gol, porém sem tomar nenhuma atitude para fazer o gol, como mostrado Figura 6.31. Possíveis explicações para este resultado são:

- Falta de uma correta distribuição de reforços, e assim, os reforços não forneceriam a correta recompensa pelas ações dos agentes e deste modo, eles não aprenderiam corretamente,
- Grande quantidade de ações para se realizar, usando ações de baixo nível, uma simples movimentação de ir até a bola, usa uma grande quantidade de ações, tornando mais difícil o aprendizado,
- Espaço de estados muito grande, se o agente precisa explorar uma grande área, ele irá demorar muito tempo para fazer isto e quando maior este espaço, muito maior será o tempo que o agente vai usar para aprender,
- O uso de uma tabela só, fez agentes aprenderem e desaprenderem ações, isto acontece quando um agente sobrescrevia uma parte da tabela, que já tinha sido aprendida por outro agente.

Estes resultados demonstraram a necessidade de uma atenção maior nos reforços e de se optar por usar ações de alto nível e tabelas separadas.



Figura 6.31 – Aprendizado de baixo nível.

Seguindo o mesmo principio da experiência anterior, outro time foi criado, usando ações de alto nível, tabelas independentes para cada jogador e como reforço somente fazer o gol. O resultado não foi conclusivo, o gráfico não apresentava bom desempenho dos agentes e nem mesmo podia se ter certeza que o agente aprendia ou se os gols feitos eram determinados pelo aprendizado. Neste experimento ficou claro que, era necessário utilizar mais reforços que somente fazer o gol.

Todos estes experimentos demonstram as seguintes conclusões, durante este trabalho:

- Escolha das ações dos jogadores: As ações devem ser escolhidas com cautela, para proporcionar de forma efetiva um bom desempenho dos agentes, e na quantidade suficiente para proporcionar um aprendizado de todas as ações.
- Os reforços precisam ser distribuídos pelas ações, pois fazer o gol é o melhor reforço para o agente, mas para conseguir fazer o gol, é necessário aprender quando chutar a bola, quando chegar nela, entre outras. Estas informações são melhores aprendidas por um agente ao receber mais reforços por ações do que simplesmente fazer o gol.
- Necessidade de uma melhor forma de representar os estados. Espaço de estados muito grande, além de exigir um enorme tempo de aprendizado, também traz problemas físicos de armazenamento de tabela em memória.

A partir do uso de uma discretização dos estados e uma melhor forma de trabalhar com eles, além de uma melhor distribuição dos reforços, foi possível realizar o aprendizado e compara-lo ao uso do HAQL. Como principais vantagens, ao realizar este trabalho, destacam-se:

- Agentes que podem se adaptar ao jogo. Os jogadores com aprendizado podem mudar suas características durante o jogo, caso o time adversário jogue diferente, ou se o time adversário manter as mesmas características, os agentes podem se tornar um jogador melhor com o passar do tempo.
- Menor espaço de busca para o agente aprendiz, com uma heurística correta o agente não precisava ficar passando por todos os estados para aprender o que fazer.
- Aceleração do algoritmo com HAQL em relação ao uso do *Q-Learning* sem as heurísticas, isto se deve ao fato, principalmente pelo menor espaço de busca, o agente aprende mais rápido. O uso das heurísticas faz com que os agentes recebam mais rapidamente os reforços por suas ações, aprendendo mais rápido.
- Formas de discretização, trabalhando com uma discretização que ajudou no aprendizado em grandes espaços, sendo este um dos piores problemas dentro desta área. Estas generalizações podem ser usadas em outros ambientes e também utilizadas por outros algoritmos de aprendizado por reforço, como por exemplo o SARSA.
- Novas formas de se criar estados usando uma chave de estados. Além de poder ser de fácil alteração para outros tipos de situações, agentes e ambientes diferentes.

Como principais desvantagens encontradas, durante este trabalho, destacam-se:

- Mesmo com o uso do HAQL, o tempo que o agente precisa para aprender é ainda é muito alto para ser usado em campeonatos oficiais da *RoboCup*.
- O jogador aprende com as experiências a melhorar suas habilidades, mas ainda não se torna um jogador de destaque. Sendo assim, por exemplo, o goleiro, aprende com o passar do tempo, a pegar as bolas que são chutadas na direção do gol, mas ainda não se torna o melhor goleiro em campo. Novas formas de se melhorar isto, precisam ainda ser estudadas.
- Times que alteram seu formato de jogar, tendem a atrapalhar o aprendizado. Ao aprender como o time joga, os agentes têm um comportamento, este comportamento pode mudar se o time adversário mudar também, mas esta

mudança é gradual. Para times que mudam muito rapidamente, os agentes com aprendizado podem ter grandes dificuldades em aprender.

Para resolver algumas das desvantagens e ainda para uma melhora nos resultados do trabalho, propõe-se para os trabalhos futuros:

- Estudar melhor as generalizações, com base nos resultados do último experimento. Aplicando uma discretização melhor das partes do ambientes que precisam de mais atenção (por exemplo, área do gol), e uma menor para os que não precisam (por exemplo, longe do gol). Além de aplicar esta forma de discretização aos experimentos, usando o goleiro e o zagueiro, e comparando com os resultados anteriores.
- Trabalhar com outros algoritmos de Aprendizado por Reforço como, por exemplo, o Dyna-Q e o Minimax-Q e Minimax-Q(λ).
- Utilizar mais jogadores no aprendizado, principalmente no aprendizado com o atacante, podendo haver comunicação entre os agentes.
- Utilizar as novas formas de se reduzir o tamanho de estados e assim pode levar os experimentos para o domínio dos multiagentes, fazendo trocas de aprendizado. O agente que aprender mais rápido ou que tiver a melhor eficiência em campo, ajuda os outros jogadores, passando o seu aprendizado com heurísticas.
- Usar o Aprendizado por Reforço acelerado por heurísticas no treinador, para um aprendizado de escolha de diversas táticas de jogos, para se aplicado durante a partida e de um aprendizado de melhores substituições de jogadores.

REFERÊNCIAS BIBLIOGRÁFICAS

ALBUS, J. S. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). **Trans. of the ASME, J. Dynamic Systems, Measurement, and Control**, v. 97, n. 3, p. 220–227, 1975.

BALLARD, D. H.; BROWN, C. M. **Computer Vision. Englewood Cliffs**, Prentice-Hall, 1982.

BELLMAN, R. E. Dynamic programming. New Jersey: Princeton University Press, 1957.

BERTSEKAS, D. P. **Dynamic Programming: Deterministic and Stochastic Models**. Upper Saddle River, NJ: Prentice-Hall, 1987.

BIANCHI, R. A. C. **Uso de Heurística para a Aceleração do Aprendizado por Reforço**. Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo, 2004.

BIANCHI, R. A. C. **Visão Computacional Aplicada ao Controle de Micros-Robôs** , Boletim Interno, No. OS/5886. Faculdade de Engenharia Industrial - FEI, 2001.

BOER; KOK, **The Incremental Development of a Synthetic Multi-Agente System: The UvA Trilearn 2001 RoboCup Soccer Simulation Team**, Tese (Mestrado) , 2002.

BOWLING, M. H.; VELOSO, M. M. Rational and Convergent Learning in Stochastic Games. In: **International Joint Conferences on Artificial Intelligence**, p. 1021–1026, 2001.

CELIBERTO,L, A; BIANCHI, R. A. C. Extração de Estruturas do Ambiente para Aceleração do Aprendizado por Reforço em uma Aplicação de Robótica Móvel, **Anais** do XVI Congresso Brasileiro de Automática, Salvador v. 1. p. 2387-2392, 2006.

CRITE,R.H. ; BARTO,A.G. Improving Elevator Performance Using Reinforcement Learning. In **Advances in Neural Information Processing Systems** 8, MIT Press, Cambridge, MA, 1996.

DORIGO, M.; CARO, G. D.; GAMBARDELLA, L. M. Ant Algorithms for Discrete Optimization. **Artificial Life**, v. 5, n. 3, p. 137 – 172, 1999.

DRUMMOND, C. Accelerating Reinforcement Learning by Composing Solutions of Automatically Identified Subtasks. **Journal of Artificial Intelligence Research**, v. 16, p. 59 –104, 2002.

FORSYTH; PONCE, **Computer Vision: A Modern Approach**, Prentice Hall, 2003.

GABRIELLI, L. H.; RIBEIRO, C. H. C. Discretização CMAC para Aprendizagem por Reforço em Times de Robôs. In: **6° Simpósio Brasileiro de Automação Inteligente Bauru**, SP: SBA, p. 434 – 438, 2003.

JAAKKOLA el al. On the Convergence of Stochastic Interative Dynamic Programming Algorith. **Neural Computation**, v.6, n.6, p. 1185 – 1201, 1994

KAELBLING, L.P.; LITTMAN M. L.; MOORE, W.A. Reinforcement Learning: A Survey, **Journal of Artificial Intelligence Research,** p.237 – 285. 4, May 1996.

KASS, M.; WITKIN, A.; TERZOPOULOS, D. Snake: active contour model. Int. J. **Computer Vision**, v. 1, p. 321–331, 1987.

KHOUSSAINOV, R. Playing the Web Search Game. **IEEE Intelligent Systems**, 2003.

KHOUSSAINOV, R.; KUSHMERICK, N. Automated Index Management for Distributed Web search. In: **Proceedings of the Eleventh ACM International Conference on Information and Knowledge Management** (CIKM 2003). New Orleans, LA, USA: ACM Press, New York, USA, p. 386–393, 2003.

KITANO, H.; TAMBE, M.; STONE, P.; VELOSO, M.; NODA, I.; OSAWA, E.; ASADA, M., The RoboCup Synthetic Agents' Challenge. **Proceedings of the International Joint Conference on Artificial Intelligence** (IJCAI), 1997.

KOK, J.; VLASSIS, N.; GROEN, F. UvA Trilearn 2003 Team Description. In Proceedings **CD RoboCup 2003**, Springer-Verlag, Padua, Italy, July 2003. Código fonte para o Basic disponível em < http://staff.science.uva.nl/~jellekok/robocup/2003/index_en.html >. Acesso em 1 out, 2006.

KRAETZCHMAR, G. et al. "The ULM Sparrows: Research into Sensorimotor Integration,Agency, Learning, and Multiagent Cooperation". In: **RoboCup Workshop**, 2, Paris.Proceedings. FIRA,. p. 459 - 465, 1998.

KRAUS; WERNER J., **Aplicações de Aprendizagem por Reforço em Controle de Tráfego Veicular Urbano**. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal de Santa Catarina, UFSC, Brasil, 2004.

LITTMAN, M. L. Markov Games as a Framework for Multi-agent Reinforcement Learning. In: **Proceedings of the 11th International Conference on Machine Learning** (ICML-94). New Brunswick, NJ: Morgan Kaufmann,. p. 157 – 163, 1994

MAO et al.,2003 , RoboCup Soccer Server Manual for Soccer Server version 7.07 or Latest, Disponível em <http://sourceforge.net/projects/sserver>. Acesso em 10 nov, 2006.

MARIANO, C. E.; Morales, E. A New Distributed Reinforcement Learning Algorithm for Multiple Objective Optimization Problems. **Lecture Notes in Artificial Intelligence**, v. 1952, p. 290 – 299, 2000.

MARIANO, C. E.; Morales. DQL: A New Updating Strategy for Reinforcement Learning Based on Q–Learning. In: RAEDT, L. D.; FLACH, P. A. (Ed.). EMCL 2001, **12th European Conference on Machine Learning**, Freiburg, Germany, September 5 – 7, 2001.

MERKE, **FrameView** Disponível em <http://ls1-www.cs.unidortmund. de/~merke/software/frameview/index.html>.Acesso em 10 nov. 2006.

MITCHELL, T. **Machine Learning**. New York: McGraw Hill, 1997.

NEHMZOW, U. **Mobile Robotics: A Practical Introduction**. Berlin, Heidelberg: Springer-Verlag, 2000.

NODA,I, Soccer Server: A Simulator of Robocup, Proceedings of AI symposium ’95 **Japanese Society for Artificial Intelligence**, pp. 29 - 34, 1995.

NODA,I; STONE, P, The RoboCup Soccer Server and CMUnited Clients: Implemented Infrastructure for MAS Research, **Autonomous Agents and Multi-Agent Systems**, 2002.

PEARL, J., Heuristics - Intelligent Search Strategies for Computer Problem Solving, **Addison-Wesley Publishing Company**, 1984.

PEGORARO, RENÊ, EPUSP/PCS. **Agilizando Aprendizagem por Reforço em Robótica Móvel Através do Uso de Conhecimento Sobre o Domínio**. Tese (Doutorado) , Setembro 2001.

PROKOPENKO, M. e BUTLER, M., Tactical Reasoning in Synthetic Multi-Agent Systems: a Case Study. Proceedings of the IJCAI-99 **Workshop on Nonmonotonic Reasoning, Action and Change**, pp. 57-64, Estocolomo, 1999.

REIS, L. P. **Coordenação em Sistemas Multi-Agente: Aplicações na Gestão Universitária e Futebol Robótico**, Tese (Doutorado), Faculdade de Engenharia da Universidade do Porto, Junho, 2003.

REIS, L. P. Estratégias e Tácticas no Futebol Robótico, Instituto Politécnico de Bragança, Escola Superior de Tecnologia e Gestão, **6ª Semana das Engenharias, Dia da Engenharia Eletrotécnica**, Auditório Eng. Alcínio Miguel, 15 de Abril de 2002.

REIS, L. P. RoboCup Tutorial – The RoboCup International Initiative: AI Meets Robotics to CreateRoboSoccer Teams, editado por Reis, Luís Paulo, integrado no EPIA2001 – **10th Portuguese Conference on Artificial Intelligence**, Seminário de Vilar, Porto, 20 de Dezembro de 2001.

REIS, L. P.; LAU, N., FC Portugal 2002 Team Description: Flexible Coordination Techniques em Gal Kaminka, Pedro Lima e Raul Rojas editors, RoboCup-2002: Robot Soccer World Cup V, Springer **Verlag Lecture Notes in Artificial Intelligence**, Berlin, 2003.

RIBEIRO, C. H. C. Embedding a Priori Knowledge in **Reinforcement Learning. Journal of Intelligent and Robotic Systems**, v. 21, p. 51 – 71, 1998.

RIBEIRO, C. H. C. Reinforcement Learning Agents. **Artificial Intelligence Review**, v. 17, p. 223 – 250, 2002.

RIEDMILLER. M., et al., Karlsruhe Brainstormers 2000 - A Reinforcement Learning Approach to Robotic Soccer. **Proceedings of the Fourth International Workshop on RoboCup. Melbourne**, Agosto de 2000.

RUMMERY, G.; NIRANJAN, M. On-line Q-learning Using Connectionist Systems. **Technical Report CUED/F-INFENG/TR 166**. Cambridge University, Engineering Department, 1994.

RUSSELL, S.e NORVIG, P. **Inteligência Artificial**. 2. ed. Rio de Janeiro: Elsevier, 2004.

SAVAGAONKAR, U. **Network Pricing Using Sampling Techniques for Markov Games**. Tese (Doutorado) , Purdue University, West Lafayette, Indiana, 2002.

SPIEGEL, M. R. **Estatística.** 2. ed. São Paulo: McGraw-Hill, 1984.

STONE et al, CMUnited-99 Source Code, 1999. Disponível em < http://www.cs.utexas.edu/~pstone/RoboCup/CMUnited99-sim.html >. Acesso em 10 de jan. 2000

STONE, P, **Layered Learning in Multi-Agent System**, Tese (Doutorado, School of Computer Science, Carnegie Mellon University, Dezembro de 1998.

STONE, P.; RILEY, P., VELOSO, M., The CMUnited-99 Champion Simulator Team. **Journal Title: AI Magazine**, v: 21, p. 33 – 40,2000

STONE, P.; VELOSO, M., Task Decomposition, Dynamic Role Assignment and Low-Bandwidth Communication for Real-Time Strategic Teamwork. **Artificial Intelligence**, Vol. 110 (2), pp. 241 - 273, Junho de 1999.

STONE, SUTTON; KUHLMANN - Reinforcement Learning for RoboCup-Soccer Keepaway. **Adaptive Behavior**, v13,p.165 – 188, 2005.

SUTTON, R. S. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In: TOURETZKY, D. S.; MOZER, M. C.; HASSELMO, M. E. (Ed.). **Advances in Neural Information Processing Systems**. The MIT Press, v. 8, p. 1038 –1044, 1996.

SUTTON, R. S. Integrated Architectures for Learning, Planning and Reacting Based on Approximating Dynamic Programming. In: **Proceedings of the 7th International Conference on Machine Learning. Austin**, TX: Morgan Kaufmann, 1990.

SUTTON, R. S. Learning to Predict by the Methods of Temporal Differences. **Machine Learning**, v. 3, n. 1, 1988.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. Cambridge, MA: MIT Press,ed1, 1998.

UTHER, W; VELOSO, M. **Adversial Reinforcement Learning**, Carnegie Mellon University, 2003.

VELOSO, M.; PAGELLO, E; KITANO, H. editores. RoboCup-99: Robot Soccer World Cup III, Springer, **Lecture Notes in Artificial Intelligence**, 2000.

WATKINS, C. J. C. H. **Learning from Delayed Rewards**. Tese (Doutorado) University of Cambridge, 1989.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. **Machine Learning**, v. 8, p. 279 – 292, 1992.

WEIß, G. A multiagent variant of Dyna-Q. In: Proceedings of the Fourth International Conference on Multi-Agent Systems – ICMAS. Los Alamitos: **IEEE Computer Society**, p. 461 – 462. 2000.

WINSTON, P. H. **Artificial Intelligence**. Third Addition. Addison-Wesley, 1992.

APÊNDICE 1

ESTUDO DA GRADE DO APRENDIZADO DE UM GOLEIRO

Para a experiência descrita em 6.1, foi realizado um estudo para demonstrar o funcionamento do aprendizado quando bola e agente estão em uma mesma zona. Este estudo tem o objetivo de comprovar que apesar da célula ser relativamente grande, ainda sim existe o aprendizado.

Foi estudado o caso particular, quando a bola e agente estão na zona 10 e o agente esta olhando na direção do meio do campo. Os resultados são demonstrados nos gráficos das Figuras A1.1. Um detalhamento melhor para analise das ações foi realizada, mostrando nos gráficos das Figuras, A1.2, A1.3, A1.4, A1.5, A1.6, A1.7 o valor separado de cada ação. Uma analise destes gráficos demonstram que a ação que teve melhor reforço foi a de pegar a bola, isto se deve ao fato que ao goleiro pegar uma bola, faz parar o ataque do outro jogador, e este não pode tirar a bola do goleiro até o mesmo chutar a bola ou ficar segurando a bola além do tempo permitido. Sendo assim, é possível determinar que dentro da zona 4x10m o aprendizado ocorre sem ser prejudicado.

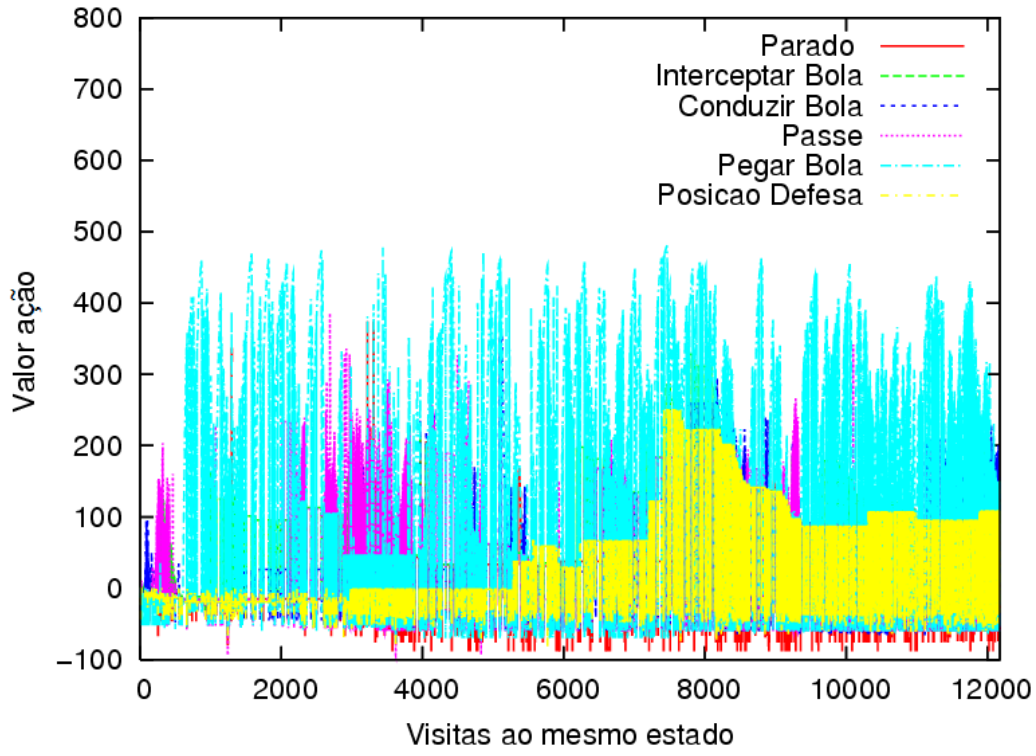


Figura A1.1 - Zona 10.

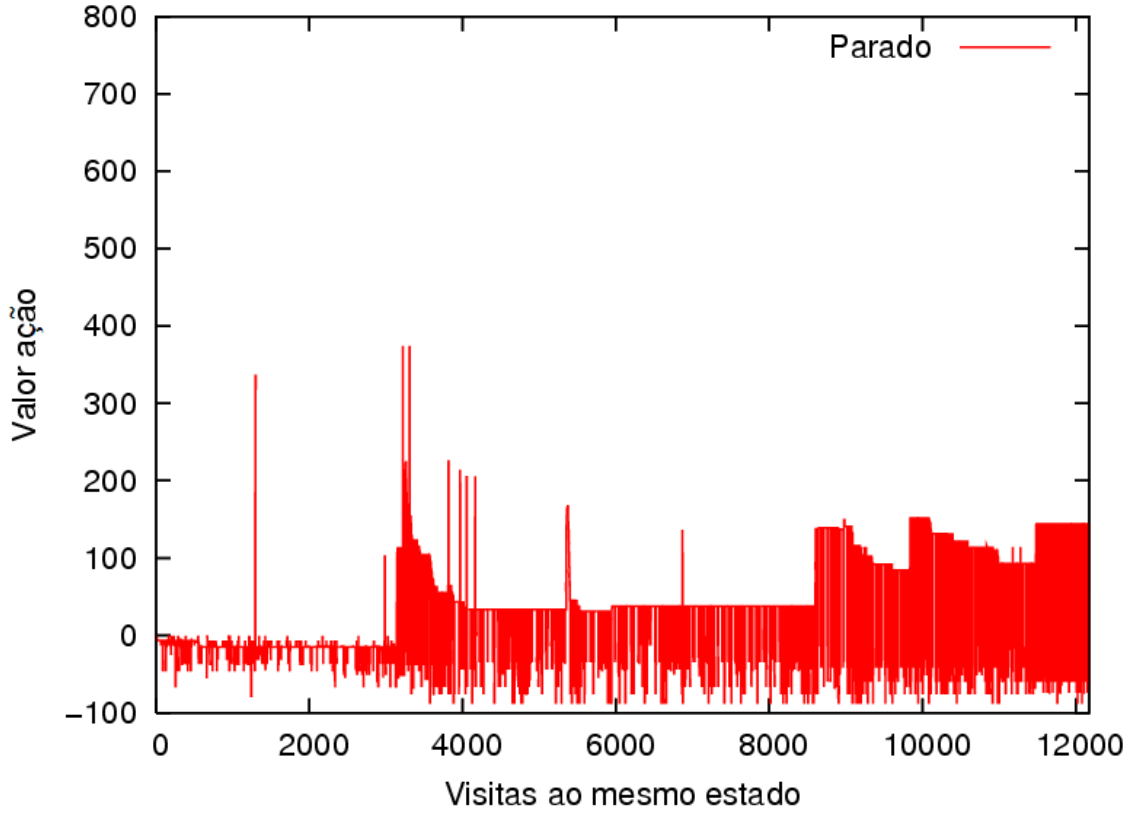


Figura A1.2 - Ação Parado.

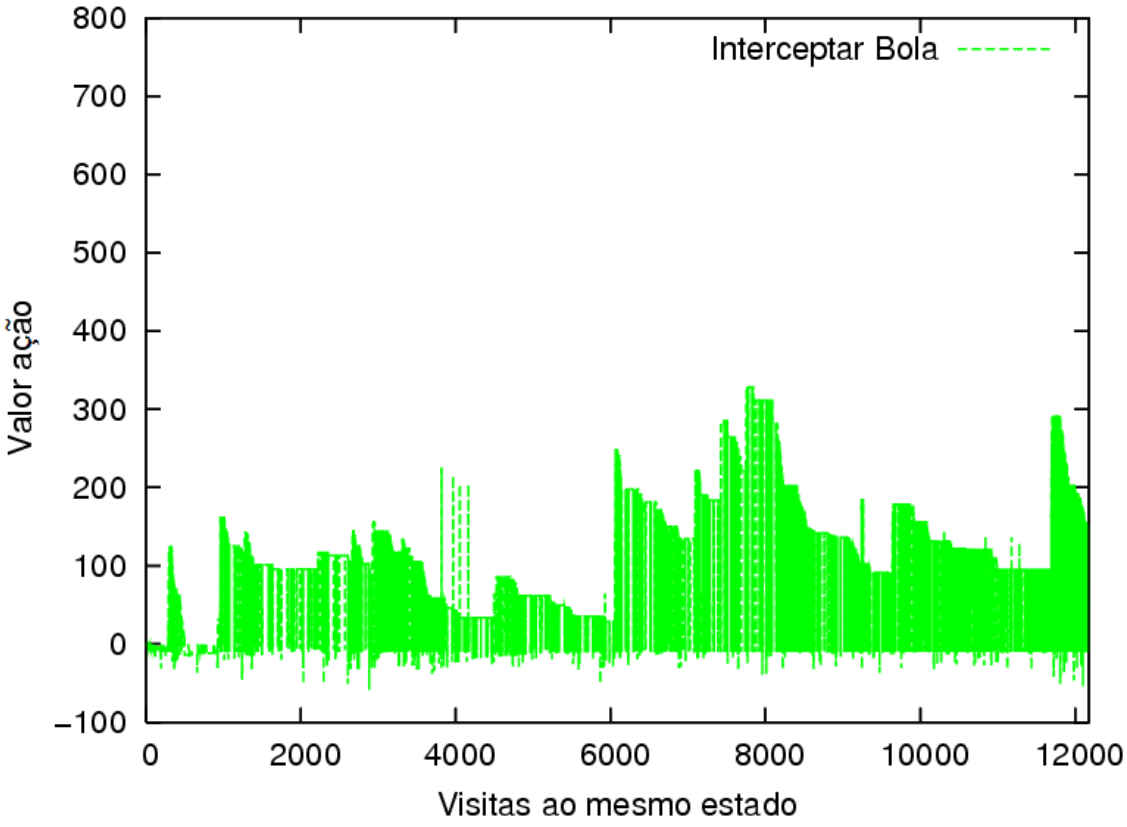


Figura A1.3 - Ação Interceptar Bola.

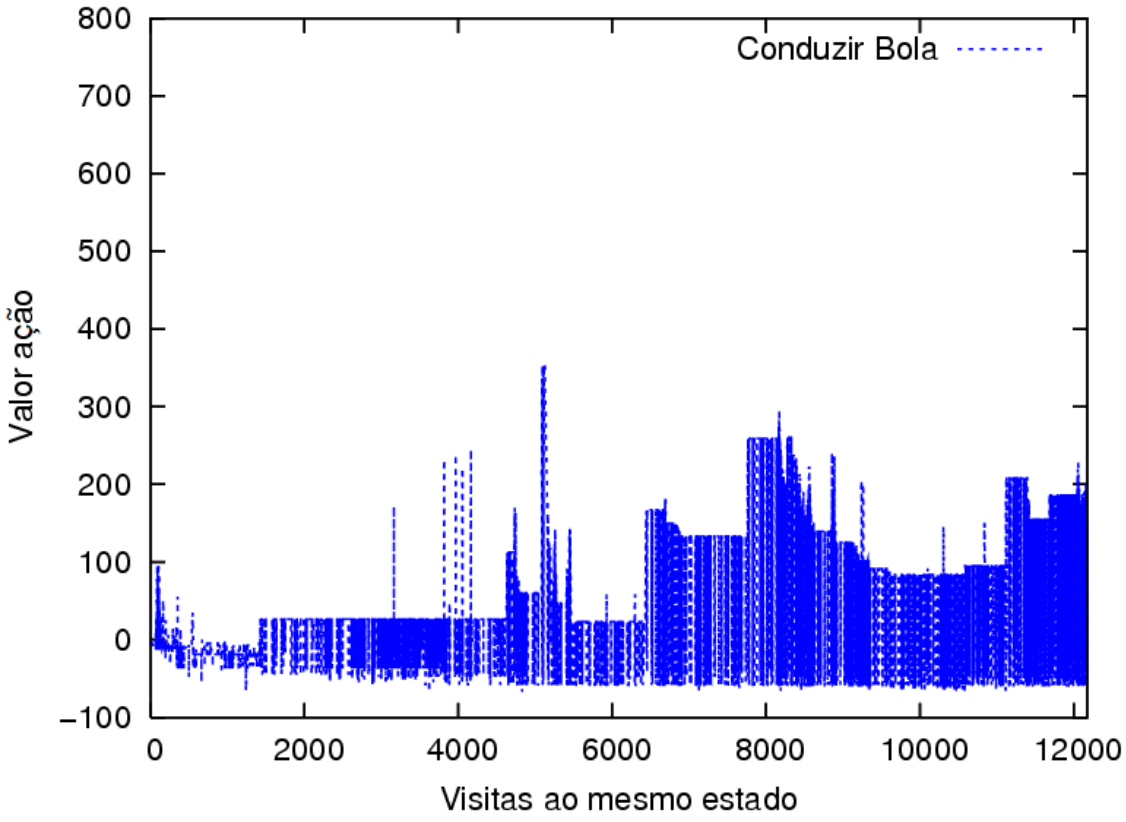


Figura A1.4 - Ação Conduzir Bola.

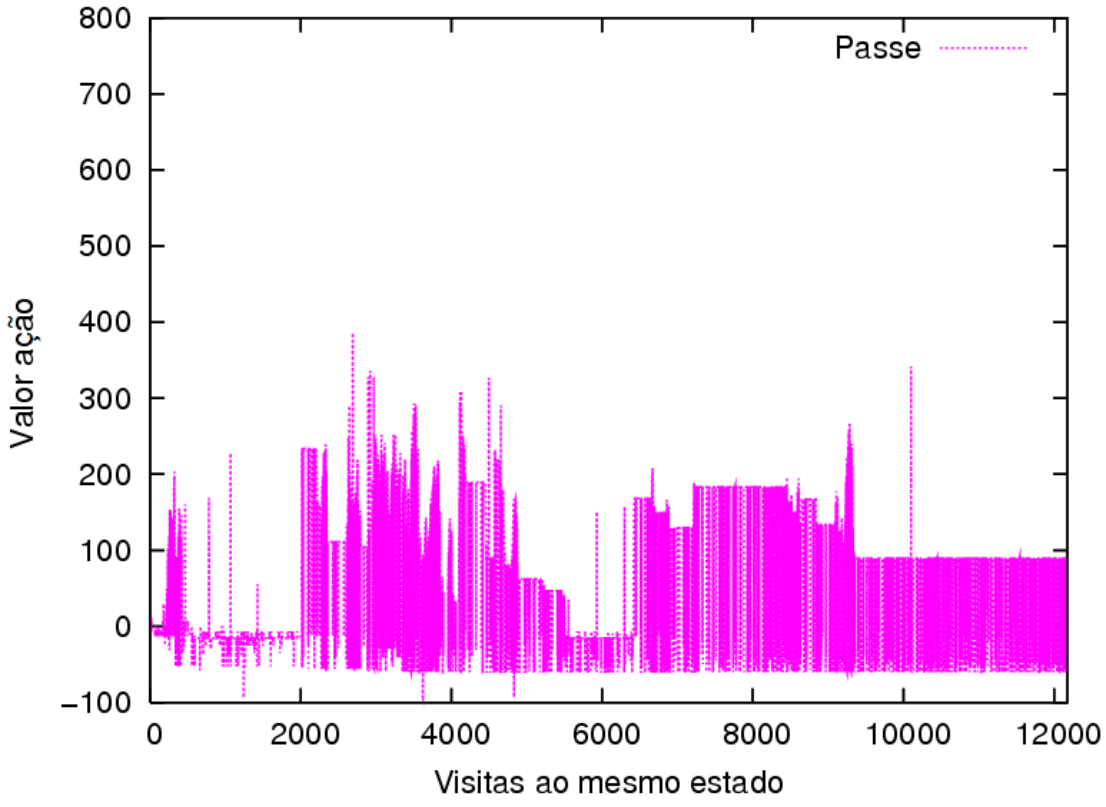


Figura A1.5 - Ação Passe.

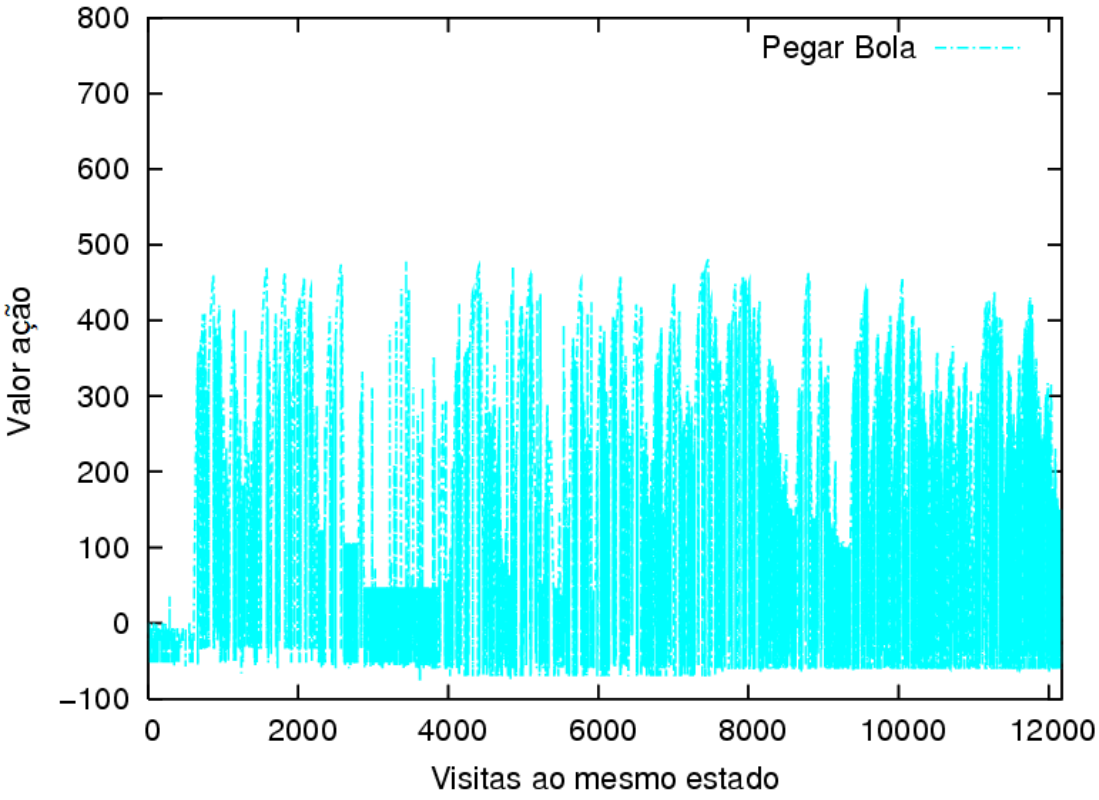


Figura A1.6 - Ação Pegar Bola.

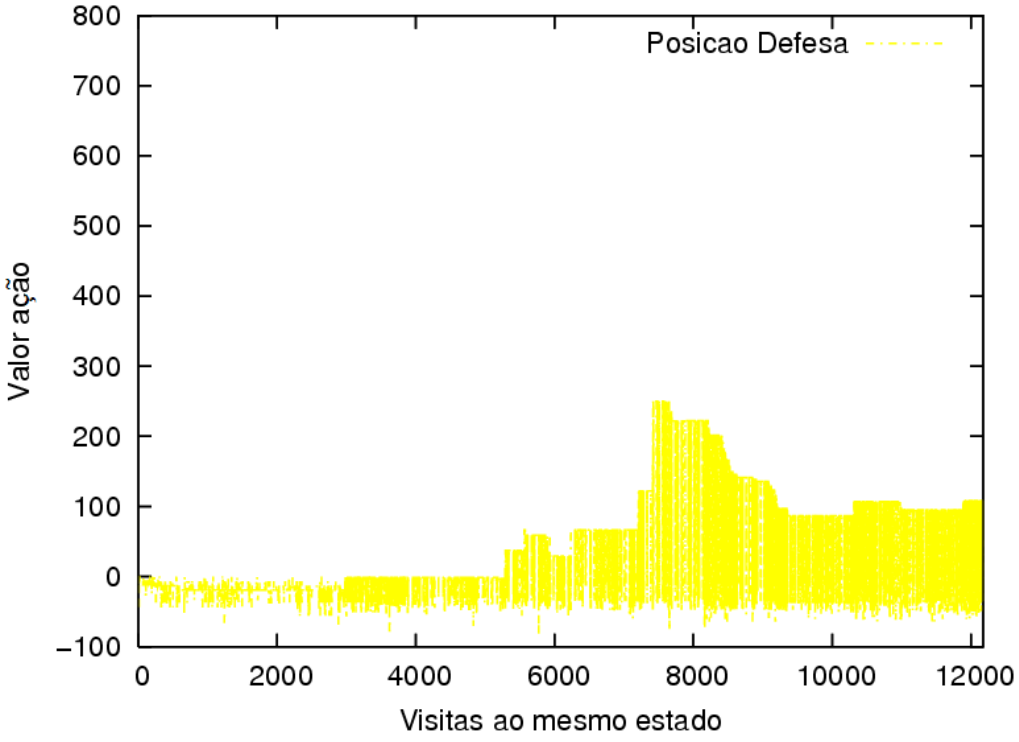


Figura A1.7 - Ação Posição Defesa.

ESTUDO DA GRADE DO APRENDIZADO DE UM GOLEIRO E UM ZAGUEIRO

Para a experiência descrita em 6.2, foi realizado um outro estudo para demonstrar o funcionamento do aprendizado quando bola e agente estão em uma mesma zona. Como o anterior,este estudo tem o objetivo de comprovar que apesar da célula ser relativamente grande, ainda sim existe o aprendizado.

Foi estudado o caso, quando a bola e agente estão na zona 14 e o agente esta olhando na direção do meio de campo. Os resultados são demonstrados nos gráficos das Figuras A1.8. Um detalhamento melhor para analise das ações foi realizada, mostrando nos gráficos das Figuras, A1.9, A1.10, A1.11, A1.12, A1.13, A1.14 o valor separado de cada ação.

Uma analise destes gráficos demonstram que a ação que teve melhor reforço foi a de chutar a bola longe, isto se deve ao fato que ao zagueiro chutar a bola, atrapalha o ataque do adversário, fazendo o adversário ter que correr atrás da bola de novo e preparar outro ataque e a quantidade de gols toados se tornam menor a proporção que o zagueiro consegue atrapalhar o ataque. Sendo assim, é possível determinar que dentro da zona 5x10m o aprendizado ocorre sem ser prejudicado.

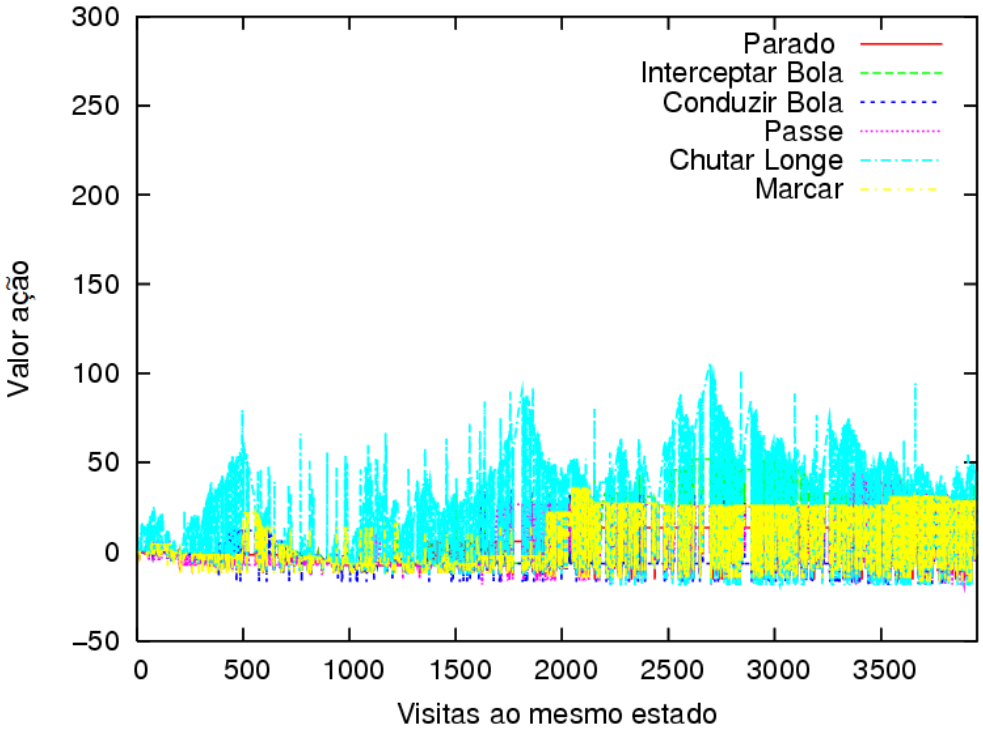


Figura A1.8 - Zona 14.

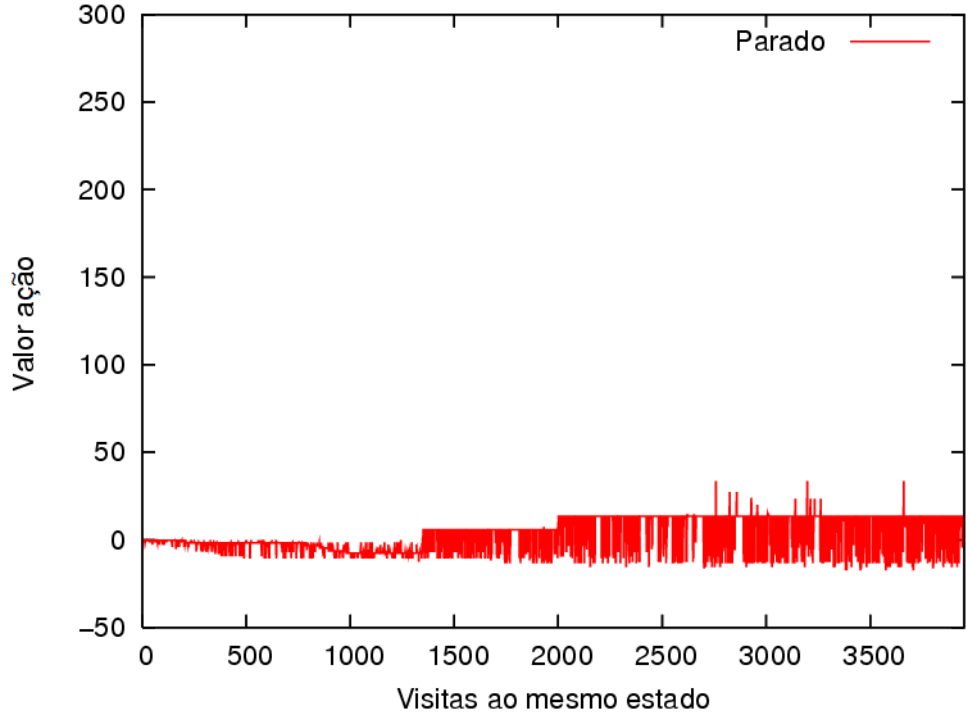


Figura A1.9 - Ação Parado.

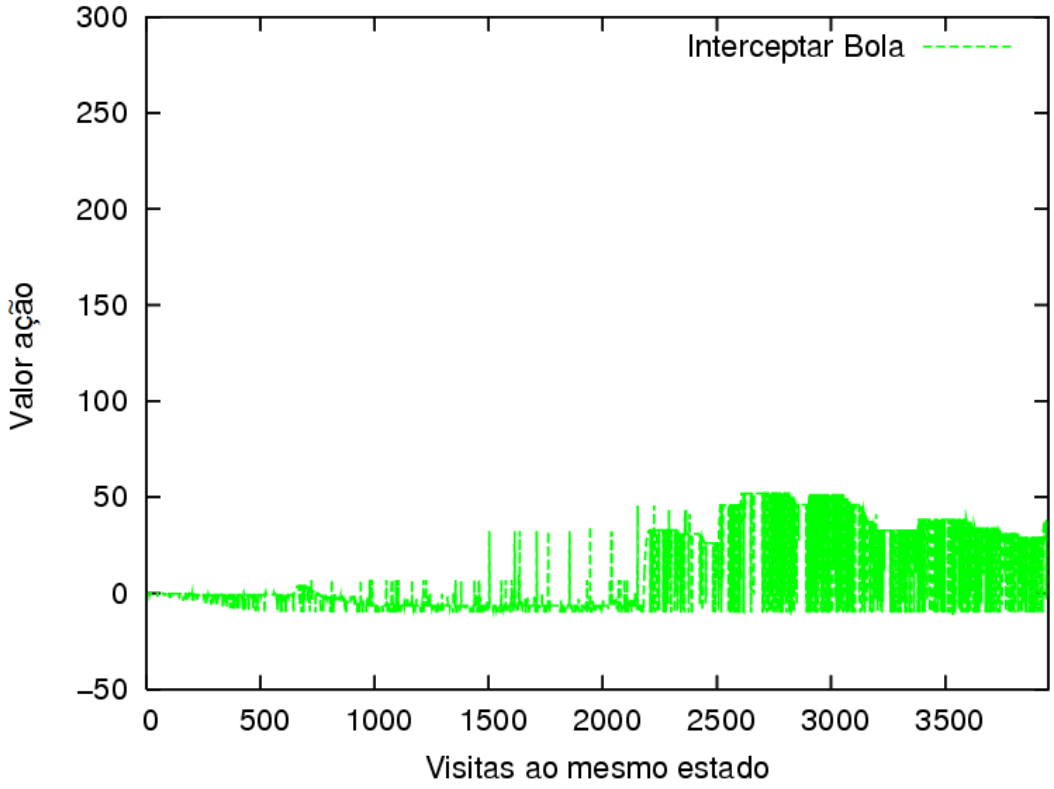


Figura A1.10 - Ação Interceptar Bola.

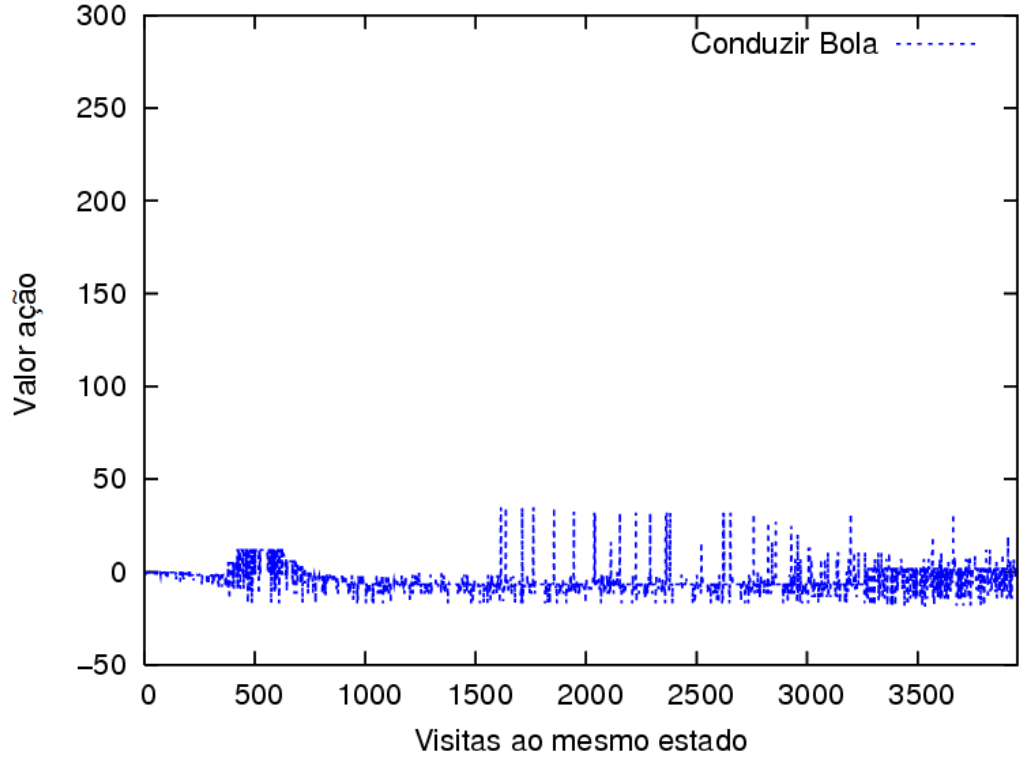


Figura A1.11 - Ação Conduzir Bola.

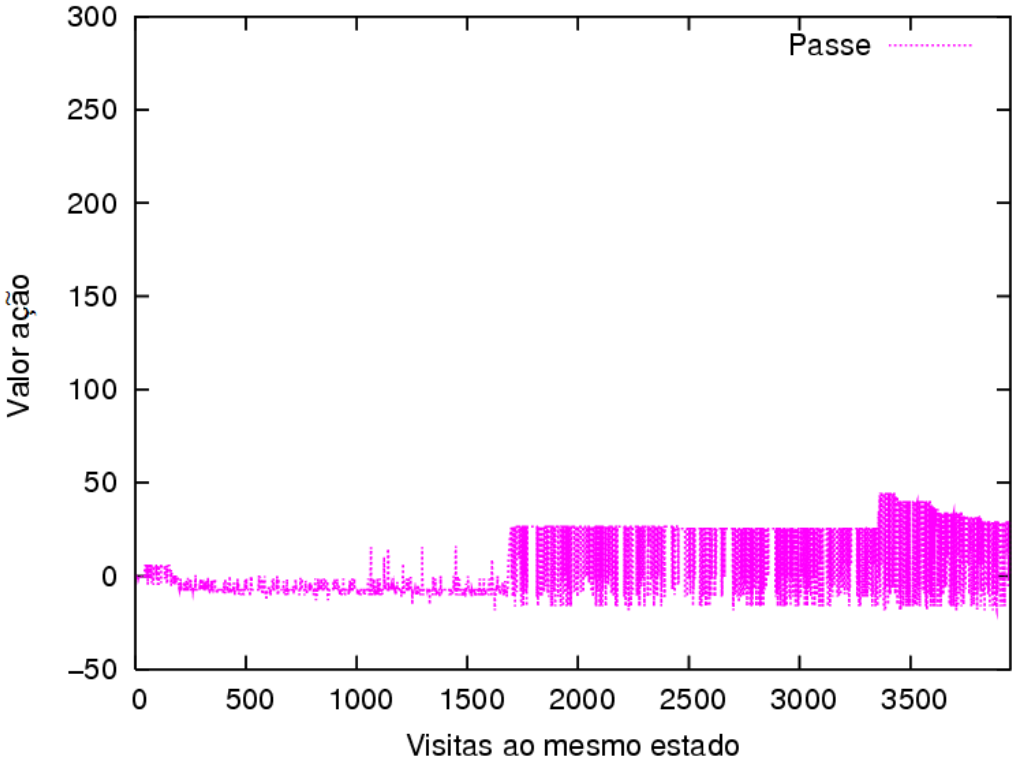


Figura A1.12 - Ação Passe.

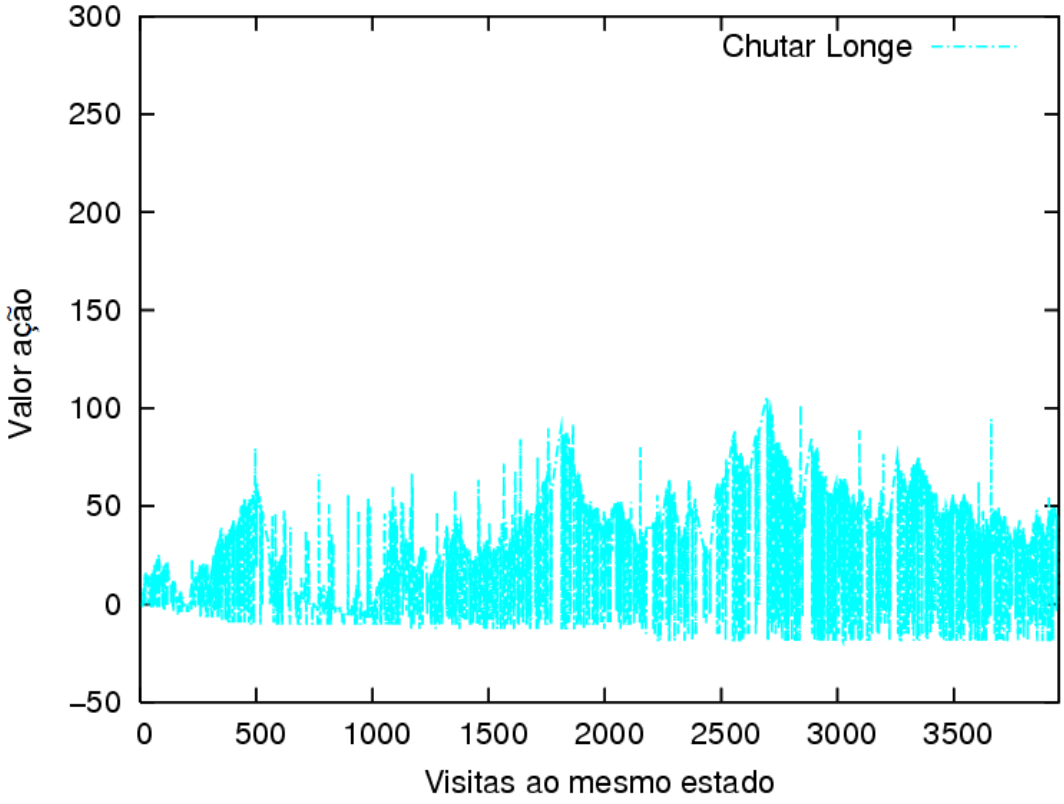


Figura A1.13 - Ação Chutar Longe.

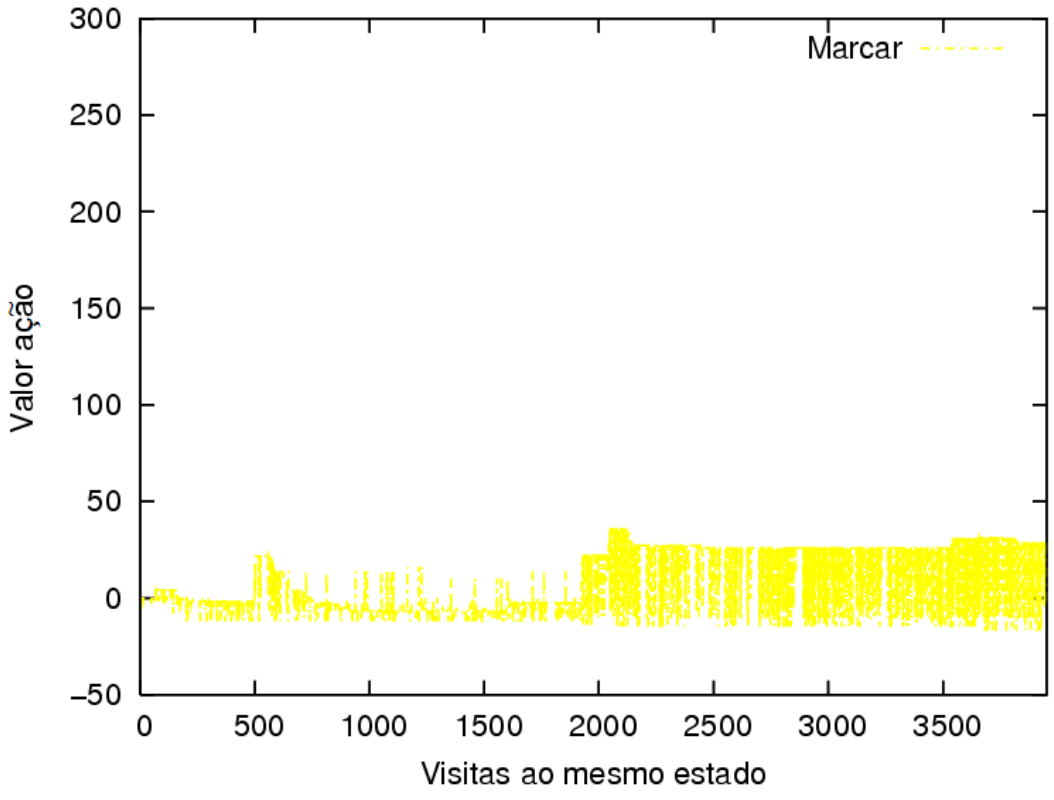


Figura A1.14 - Ação Marcar.

APÊNDICE 2

O TEST T DE STUDENT

Conhecido também como *T-Test*, o teste *t* de Student (SPIEGEL, 1984; NEHMZOW, 2000), fornece a ferramenta para poder verificar se os resultados de duas experiências são significativamente iguais ou diferentes. Esta ferramenta é muito utilizada em todas as ciências e permite decidir se um algoritmo exibe um desempenho superior em comparação a outro.

Para poder estabelecer entre dois valores médios chamados de μx e μy, se são diferentes, o valor T é calculado segundo a equação A2.1 (SPIEGEL, 1984):

$$T = \frac{\mu_x - \mu_y}{\sqrt{(n_x - 1)\sigma_x^2 + (n_y - 1)\sigma_y^2}} \sqrt{\frac{n_x n_y (n_x + n_y - 2)}{n_x + n_y}} \qquad \text{(A2.1)}$$

Onde, dos pontos no experimento x,y

- η_x e η_y são os números de pontos,
- μ_x e μ_y são as médias,
- σ_x e σ_y são o desvios padrões.

Se o valor do módulo de *T* for maior que um dada constante tα, a hipótese H0 que μx = μy é rejeitada e os experimentos são diferentes significativamente. O valor t_α é um constante que define a probabilidade de um teste estar errado, chamado de nível de confiança. Este valor é na maioria dos casos definido como 5%. O valor de t_α depende do número de experimentos realizado e pode ser facilmente encontrado na maioria dos livros da área de estatística.