



TRABALHO DE GRADUAÇÃO

Aprendizagem por Reforço aplicada a Seleção de Comportamento em Agentes Autônomos

Por,
Tiago Pimentel Martins da Silva

Brasília, dezembro de 2014



**ENGENHARIA
MECATRÔNICA**
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

Aprendizagem por Reforço aplicada a Seleção de Comportamento em Agentes Autônomos

Por,
Tiago Pimentel Martins da Silva

*Relatório submetido como requisito parcial de obtenção
de grau de Engenheiro de Controle e Automação*

Banca Examinadora

Profa. Carla Maria Chagas e Cavalcante Koike, CiC/UnB _____
Orientadora

Prof. Guilherme Novaes Ramos, CiC/UnB _____
Examinador interno

Prof. Alexandre Ricardo Soares Romariz, ENE/UnB _____
Examinador interno

Brasília, dezembro de 2014

FICHA CATALOGRÁFICA

SILVA, TIAGO PIMENTEL MARTINS DA

Aprendizagem por Reforço aplicada a Seleção de Comportamento em Agentes Autônomos,
[Distrito Federal] 2014.

vii, 62p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2014). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

- 1. Agentes autônomos
 - 2. Seleção de comportamento
 - 3. Aprendizagem por Reforço
 - 4. Redes bayesianas

I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

SILVA, T. P., (2014). Aprendizagem por Reforço aplicada a Seleção de Comportamento em Agentes Autônomos. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-nº015, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 62p.

CESSÃO DE DIREITOS

AUTOR: Tiago Pimentel Martins da Silva

TÍTULO DO TRABALHO DE GRADUAÇÃO: Aprendizagem por Reforço aplicada a Seleção de Comportamento em Agentes Autônomos.

GRAU: Engenheiro

ANO: 2014

É concedida à Universidade de Brasília permissão para reproduzir cópias deste Trabalho de Graduação e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte desse Trabalho de Graduação pode ser reproduzida sem autorização por escrito do autor.

Tiago Pimentel Martins da Silva
SQN 106 Bloco K - Asa Norte
70.742-110 Brasília, DF, Brasil

Dedicatória

À minha família (isso já te inclui há anos, amor!) e à minha família estendida.

Tiago Pimentel Martins da Silva

Agradecimentos

Finalmente, após (não tão) longos anos de graduação é chegada a hora de agradecer a todos aqueles que sempre me deram apoio, alegraram meus dias e me ajudaram a chegar aqui. Nada mais justo do que começar com os três que se tornaram meus melhores amigos durante o curso e que, sem sua companhia e amizade (e cadernos), não estaria escrevendo esse trabalho tão cedo. O Bruno, que, em seis anos, não encontrei uma vez em que não estivesse sorrindo e bem humorado. A Marcela, ex-concorrente que virou coleguinha e que virou uma das minhas melhores amigas. A pessoa que faz mais falta por estar em outro país. E o Guiga que ouve eu reclamar de “;” há seis anos e ainda entra na brincadeira. Que consegue, nessa brincadeira, convencer pessoas passantes de que a discussão do século está acontecendo. E com quem trabalhei no LARA, na UnBall, LARA de novo, além de outros projetos e que, sem sua presença, esses trabalhos seriam muito mais longos e tediosos.

Preciso agradecer também a todos os outros amigos que fiz na mecatrônica e em toda UnB. Todos os LARAbboys e LARAgirls, que ficam 24h por dia no laboratório comigo, em especial a Pam, sempre lá e sempre simpática, o Levy, sempre fingindo estar no Facebook, mas secretamente trabalhando, e todos do AMORA, com quem almoço toda semana. O Tcholas, que é um dos caras mais inteligentes que eu conheço e me ensinou muito sobre muita coisa. Todos os forrozeiros, grupo mais simpático da UnB, em especial Aninha, André, De Hong e Levy, pessoas de quem eu gosto e que admiro muito, embora eu nunca responda no whatsapp. Um agradecimento também ao George, que me ensinou ainda mais que o Tcholas.

Gostaria de fazer também um agradecimento a todos meus professores, em especial: Célius, Zaghetto, Lineu e Padilha e a minha orientadora, Profa. Carla Koike, que me ensinou quase tudo que sei de robótica.

Agradeço, agora, todos meus amigos do Sigma, mas em especial a todo o grupo do pão de queijo. Acho que eu nem preciso dizer o quanto eu aprecio todos os momentos que passamos juntos em qualquer lugar (no Outback). Maurício, Jessica, Filipe, Bernardo, Thiago e Ana Paula, eu amo vocês e não vou me estender nesse agradecimento porque não caberia aqui. Mari e Cainã, também amo muito vocês, como vocês sabem.

Por último, mas não menos importante, agradeço toda minha família. Meus pais, que me dão todo apoio e suporte há mais de 23 anos, meus irmãos, que roubam meu lugar na hora do almoço, e meus primos, sem os quais não consigo imaginar um natal (mesmo o Gabriel, que foge de SP quando vou pra lá, e a Carol, que só vem pra Brasília quando não estou aqui). E novamente agradeço você, Ana Paula, que já é parte da minha família.

Tiago Pimentel Martins da Silva

RESUMO

Este trabalho apresenta uma abordagem para uma escolha autônoma de ação, baseada em uma seleção de comportamento utilizando redes bayesianas e aprendizagem por reforço. Um agente consegue, por meio de recompensas recebidas do sistema, aprender uma política de escolha de comportamentos em um sistema parcialmente observável sem qualquer conhecimento prévio desses comportamentos. O modelo é dividido em duas fases: durante e após treinamento. Durante o treinamento existem cinco etapas para uma estimativa recursiva do estado atual e para a escolha da ação: Predição, Seleção de comportamento, Observação, Seleção de ação motora e Aprendizagem. Após o treino estar completo não se aplica mais a etapa de aprendizagem.

Palavras Chave: Aprendizagem de Máquina, Aprendizagem por Reforço, Redes Bayesianas, Seleção de Comportamento, Q-Learning, Robótica Móvel, Inteligência Artificial

ABSTRACT

This work describes an approach for autonomous action selection, based in a previous behaviour selection, using Bayesian networks and reinforcement learning. An agent can, by receiving rewards from the system, learn a behaviour selection policy in a partially observable system, without any previous knowledge about the behaviours. The model is divided in two parts: during and post training. During the training there are five steps for a recursive state estimation and action selection: Prediction, Behaviour selection, Observation, Action selection and Learning. After the training is complete, in the post training part, the learning step is no longer applied.

Keywords: Machine Learning, Reinforcement Learning, Bayesian Networks, Behaviour Selection, Q-Learning, Mobile Robotics, Artificial Intelligence

SUMÁRIO

1 INTRODUÇÃO	1
1.1 CONTEXTUALIZAÇÃO	1
1.2 DEFINIÇÃO DO PROBLEMA	2
1.3 OBJETIVOS DO PROJETO.....	2
1.4 RESULTADOS OBTIDOS	2
1.5 APRESENTAÇÃO DO MANUSCRITO.....	2
2 FUNDAMENTAÇÃO TEÓRICA.....	4
2.1 INTRODUÇÃO	4
2.2 REVISÃO BIBLIOGRÁFICA.....	4
2.3 FILTRO BAYESIANO.....	6
2.3.1 PREDIÇÃO	7
2.3.2 OBSERVAÇÃO.....	7
2.3.3 ESCOLHA DE AÇÃO MOTORA	7
2.4 PROCESSO DE DECISÃO DE MARKOV.....	8
2.5 DIFERENÇA TEMPORAL.....	10
2.6 Q-LEARNING.....	11
2.6.1 GENERALIZAÇÃO DOS PARES ESTADO-AÇÃO.....	12
2.6.2 ESCOLHA DE AÇÕES E EXPLORAÇÃO EPSILON GULOSA	13
3 DESENVOLVIMENTO	15
3.1 INTRODUÇÃO	15
3.2 MODELAGEM MATEMÁTICA	15
3.2.1 APRENDIZAGEM POR REFORÇO PARA SELEÇÃO DE COMPORTAMENTOS	15
3.2.2 SELEÇÃO DE COMPORTAMENTOS EM UM SISTEMA PARCIALMENTE OBSERVÁVEL	16
3.2.3 ABORDAGEM BAYESIANA COM A SELEÇÃO DE COMPORTAMENTOS	18
3.2.4 MODELO FINAL E COMPLETO	20
3.3 AMBIENTE DE TESTES E SIMULAÇÃO	23
3.4 TESTES REALIZADOS.....	25
3.4.1 3 COMPORTAMENTOS NO MAPA PEQUENO (TESTE 1)	25
3.4.2 3 COMPORTAMENTOS NO MAPA ORIGINAL (TESTE 2)	29
3.4.3 5 COMPORTAMENTOS NO MAPA PEQUENO (TESTE 3)	31
3.4.4 5 COMPORTAMENTOS NO MAPA ORIGINAL (TESTE 4)	35

4 ANÁLISE DE DADOS	38
4.1 INTRODUÇÃO	38
4.2 3 COMPORTAMENTOS NO MAPA PEQUENO (TESTE 1)	38
4.2.1 RESULTADOS E ANÁLISE	39
4.2.2 DISCUSSÃO	42
4.3 3 COMPORTAMENTOS NO MAPA ORIGINAL (TESTE 2)	42
4.3.1 RESULTADOS E ANÁLISE	43
4.3.2 DISCUSSÃO	45
4.4 5 COMPORTAMENTOS NO MAPA PEQUENO (TESTE 3)	46
4.4.1 RESULTADOS E ANÁLISE	47
4.4.2 DISCUSSÃO	51
4.5 5 COMPORTAMENTOS NO MAPA ORIGINAL (TESTE 4)	52
4.5.1 RESULTADOS E ANÁLISE	52
4.5.2 DISCUSSÃO	55
4.6 DISCUSSÃO GERAL	56
5 CONCLUSÕES	58
5.1 PERSPECTIVAS FUTURAS	58
REFERÊNCIAS BIBLIOGRÁFICAS	60
APÊNDICES	63
I DESCRIÇÃO DO CONTEÚDO DO CD	64

LISTA DE FIGURAS

2.1	Filtro Bayesiano Recursivo.....	7
2.2	Mapa de Probabilidades.....	9
3.1	Filtro bayesiano utilizando <i>Q-Learning</i> para seleção de comportamento. Durante treinamento.....	21
3.2	Filtro bayesiano utilizando <i>Q-Learning</i> para seleção de comportamento. Após treinamento completo.	21
3.3	Plataforma do jogo Pac-Man desenvolvida em Berkeley para aulas de IA.....	23
3.4	Mapa pequeno do jogo Pacman. Usado nos testes 1 e 3.....	25
3.5	Mapa clássico do jogo Pacman. Usado nos testes 2 e 4.....	30
4.1	Evolução dos pesos ω_1 e ω_2	39
4.2	Evolução do peso ω_3	39
4.3	Escolha de comportamentos por partida.....	40
4.4	Polinômios referentes à escolha de comportamentos por partida.....	41
4.5	Pontuação por partida.....	42
4.6	Evolução dos pesos ω_1 e ω_2	43
4.7	Evolução do peso ω_3	43
4.8	Escolha de comportamentos por partida.....	44
4.9	Polinômios referentes à escolha de comportamentos por partida.....	45
4.10	Pontuação por partida.....	46
4.11	Evolução dos pesos ω_1 e ω_2	47
4.12	Evolução dos pesos ω_3 e ω_4	48
4.13	Evolução dos pesos ω_5 e ω_6	48
4.14	Evolução do peso ω_7	48
4.15	Escolha de comportamentos por partida.....	49
4.16	Polinômios referentes à escolha de comportamentos por partida.....	50
4.17	Pontuação por partida.....	51
4.18	Evolução dos pesos ω_1 e ω_2	53
4.19	Evolução dos pesos ω_3 e ω_4	53
4.20	Evolução dos pesos ω_5 e ω_6	54
4.22	Polinômios referentes à escolha de comportamentos por partida.....	54
4.21	Escolha de comportamentos por partida.....	55
4.23	Pontuação por partida.....	56

LISTA DE TABELAS

4.1	Média da escolha dos comportamentos (Teste 1).....	41
4.2	Média da escolha dos comportamentos (Teste 2).....	45
4.3	Média da escolha dos comportamentos (Teste 3).....	51
4.4	Média da escolha dos comportamentos (Teste 4).....	55

LISTA DE SÍMBOLOS

Símbolos Latinos

m	Ação motora
M	Conjunto das ações motoras
z	Medida sensorial
Z	Conjunto das medidas sensoriais
s	Estado do sistema
S	Conjunto dos estados do sistema
t	Instante de tempo
u	Ação genérica
U	Conjunto das ações genéricas
b	Comportamento
B	Conjunto dos comportamentos
a	Distribuição de probabilidade dos estados
A	Conjunto das distribuições de probabilidade dos estados
s_b	Variável de estado relativa ao comportamento
S_b	Conjunto das variáveis de estado relativas ao comportamento
x	Posição no eixo horizontal
y	Posição no eixo vertical
d_x	Distância no eixo horizontal
d_y	Distância no eixo vertical
n	Contador

Símbolos Gregos

π_f	Conhecimento inerente que se tem do mundo
ω_i	Peso de uma característica i
γ	Fator de desconto
α	Fator de aprendizagem
β	Fator de exploração
μ	Média
σ	Desvio padrão
ν	Fator de acerto para atuação

Funções

P	Probabilidade
π	Política
r	Recompensa
V	Valor esperado do estado
Q	Valor esperado do par estado-ação (ou estado-comportamento)
f_i	Característica i do par estado-ação (ou estado-comportamento)
$mean$	Média aritmética
δ	Gera número randômico com probabilidade baseada em gaussiana

Subscritos

$\#(número)$	Índice
π_{td}	Relativo à política fixa do algoritmo TD
b	Relativo à comportamento

Sobrescritos

$'$	No instante seguinte de tempo ($t + 1$)
t	No instante de tempo t
$0 : t$	Valores para todos instantes de tempo entre $[0, t]$
$*$	Valor ótimo ou exato
$+$	Variável com a definição expandida

Capítulo 1

Introdução

“Do what you can, with what you have, where you are.” – Theodore Roosevelt

1.1 Contextualização

Uma pergunta que nasceu junto com a própria consciência humana e que ainda não possui uma resposta concreta é proposta por Carla Koike como a pergunta universal em sua tese de Doutorado [1] “What can I do next, knowing what I have seen and what I have done?”, ou, em português, “O que posso fazer a seguir, sabendo o que já vi e o que fiz?”.

Outra questão complexa é definir o que é um robô. Isso é difícil pois essa palavra é aplicada a uma diversidade de máquinas que tem poucos pontos em comum. O dicionário Webster, por exemplo, para levar em conta a diversidade de aplicações dessa palavra, apresenta uma definição tripla pra esse termo, sendo uma delas a seguinte: “um mecanismo guiado por um controle automático” [2, 3]. A definição deixa clara essa que é das principais características de um sistema robótico, a autonomia.

Tendo esse conceito e voltando agora a pergunta anterior, modificando-a ligeiramente para “O que devo fazer a seguir, sabendo o que já vi e o que fiz?”, vê-se porque ela é tão importante no campo da robótica. Quando ela é respondida, ou seja, quando se sabe o que fazer a cada momento, define-se o controle autônomo do robô.

Em todas as áreas da robótica muitos esforços têm sido aplicados para se obter uma resposta a essa questão e estratégias diferentes são utilizadas em aplicações diferentes. Para alguns braços robóticos, por exemplo, devido a sua precisão de movimento, o planejamento de movimento é feito deterministicamente. Outras aplicações devem levar em conta as limitações dos atuadores e sensores.

Robótica probabilística é uma área da robótica que leva em conta a incerteza das percepções (sensores) e do controle (atuadores) de um robô. Baseada no campo de estatística, essa estratégia provê o robô com algoritmos mais robustos para lidar com situações do mundo real [4]. Esse nível de robustez é essencial quando se tem como plataforma um robô móvel, onde sensores e atuadores apresentam incertezas que, se não tratadas, levam a performances piores que as esperadas.

Trazendo essa pergunta para o campo de robótica probabilística obtém-se uma equação que a resume

$$P(M^t | z^{0:t} m^{0:t-1} \pi_f), \quad (1.1)$$

ou seja, a probabilidade de se executar uma ação $m \in M$ no tempo t , sendo que foram observados $z^{0:t}$ nos tempos de 0 a t e foram realizadas as ações $m^{0:t-1}$ nos tempos 0 a $t - 1$. π_f representa o conhecimento inerente que se tem do mundo.

Levando esse problema a um novo nível, pode-se fazer uma nova pergunta: “como fazer um robô aprender dando-se recompensas a ele por certas ações e, com isso, aumentar sua eficiência com o passar do tempo?”.

1.2 Definição do problema

Neste trabalho é abordado o problema de planejamento de tarefas para um agente autônomo, utilizando aprendizagem por reforço e redes bayesianas. Este problema envolve a escolha de um comportamento e uma ação para serem tomados pelo agente a cada momento tendo como base os dados dos sensores e as ações anteriores tomadas pelo agente, além das recompensas recebidas por ele para cada uma dessas ações executadas. Para isso é preciso integrar as teorias (seleção de comportamento e rede bayesiana) sem comprometer a precisão matemática de ambas, utilizando a seleção de comportamento como uma tomada de decisão a alto nível e a rede bayesiana para decidir que ação de baixo nível tomar.

1.3 Objetivos do projeto

O objetivo principal desse trabalho é obter um algoritmo de tomada de decisões hierárquico (seleção de comportamento a alto nível e rede bayesiana a baixo) que consiga aprender com experiências reais, aplicado em um agente autônomo. Para isso é necessária a integração entre ambos os algoritmos utilizando-se métodos matemáticos confiáveis e a implementação desses algoritmos em uma plataforma, com erros sensoriais e de atuação, que será utilizada para testar o sistema.

1.4 Resultados obtidos

O agente conseguiu aprender a escolher os comportamentos de forma satisfatória, aumentando seu desempenho ao longo de sua execução. Ele conseguiu também detectar particularidades do sistema onde era executado, tendo sua aprendizagem condicionada ao ambiente em que é utilizado.

1.5 Apresentação do manuscrito

Esse trabalho foi dividido em quatro capítulos, além deste: Fundamentação Teórica, Desenvolvimento, Análise de Dados e Conclusão.

- Fundamentação Teórica — Introduz as teorias matemáticas em que esse trabalho se baseia;
- Desenvolvimento — Mostra a modelagem matemática desenvolvida e utilizada nesse trabalho. Também descreve os testes realizados;
- Análise de Dados — Contém os resultados dos testes realizados e sua análise;
- Conclusão — Conclui o trabalho e apresenta perspectivas de trabalhos futuros que podem ser realizados para expandir a teoria descrita aqui.

Capítulo 2

Fundamentação Teórica

“If I have seen further it is by standing on the shoulders of giants.” – Sir Isaac Newton

2.1 Introdução

Este capítulo apresenta algumas soluções para o problema de tomada de decisões em sistemas de robótica móvel descritas em trabalhos anteriores. Ele aborda, também, o problema de aprendizagem por reforço e algumas alternativas relatadas na literatura. Por último, esse capítulo introduz a teoria matemática dos algoritmos utilizados nesse projeto.

2.2 Revisão Bibliográfica

Aqui são descritas diferentes abordagens encontradas na literatura para o problema de tomada de decisões aplicado a sistemas de robótica móvel. A tomada de decisões é um problema de amplo espectro para o qual existem várias estratégias diferentes de solução.

Analizando a literatura recente, é possível ver a preferência crescente por algoritmos probabilísticos nessa área, no lugar dos determinísticos. Para uma modelagem realista do ambiente do robô, é preciso considerar as incertezas nele presentes.

Muitos algoritmos de planejamento de tarefas são específicos para as tarefas que os robôs desempenham. Alguns exemplos são na área de exploração, como o uso de algoritmos de exploração de fronteiras [5, 6, 7], ou na área de braços robóticos, para manipulação de objetos [8].

Neste trabalho, é desejado um algoritmo que não seja específico para um único tipo de aplicação. Ao analisar a literatura recente, vê-se que muitos algoritmos nessa área de tomada de decisões são baseados em cadeias de Markov. Essa teoria assume que um ambiente pode ser modelado por um certo número de variáveis que representam os estados e que cada estado depende apenas do estado anterior, ou seja, o sistema não tem memória de longo prazo [9].

$$P(S^t | S^{t-1}S^{t-2}) = P(S^t | S^{t-1}) \quad (2.1)$$

Um algoritmo frequentemente utilizado baseado em cadeias de Markov é o Processos de De-

cisão de Markov (Markov Decision Processes, MDP) [4]. Embora o MDP leve em conta que as ações executadas possam ter um resultado imprevisível, ele não incorpora a incerteza provinda dos sensores. Em outras palavras, ele considera que os estados do sistema podem ser completamente observados a todo instante, ou seja, que dadas as medidas dos sensores, sabe-se perfeitamente o estado atual do sistema. Isso, em robótica, geralmente não é o caso.

Um algoritmo de extensão do MDP que corrige esse problema é o Processos de Decisão de Markov Parcialmente Observável (Partially Observable Markov Decision Processes, POMDP) [4]. Esse algoritmo é utilizado no caso completamente probabilístico, em que tanto sensores quanto atuadores têm funcionamento imprevisível e com certa curva conhecida de probabilidade. O problema com esse algoritmo é a sua demanda computacional, mesmo podendo ser calculado *offline*, ele muitas vezes não pode ser computado em um tempo aceitável.

Pineau e Thrun [10] utilizam uma estratégia de vários POMDPs hierarquizados para reduzir o problema de esforço computacional que um único POMDP combinado iria precisar. Embora a demanda computacional seja reduzida com essa estratégia, dependendo do tamanho do espaço de estados, ela ainda não é tolerável.

Outra estratégia para o planejamento de tarefas é a abordagem bayesiana para seleção de comportamentos presente em [1, 11]. Essa é uma estratégia interessante por ser computacionalmente eficiente, pois, ao selecionar somente um comportamento e posteriormente escolher uma ação para ele, consegue reduzir consideravelmente as opções possíveis (existem menos comportamentos do que ações possíveis). Essa estratégia foi expandida posteriormente em [12], onde são aprendidos os modelos probabilísticos de seleção de comportamentos a partir de entradas humanas.

Na área de aprendizagem de modelos ou de políticas para execução de ações, existem vários algoritmos baseados no MDP citado acima. Alguns deles focam apenas na aprendizagem do modelo [13, 14] e têm várias aplicações, sendo análise de mercados uma delas [13]. Outro exemplo de algoritmo de aprendizagem é o de diferença temporal, que é utilizado para avaliar uma dada política [15, 16, 17]. Uma política pode ser vista como uma função $\pi(S)$, a qual, dado um estado, retorna uma ação possível de ser executada nele.

Outras estratégias de aprendizagem por reforço permitem aprender uma política ótima de ações para um dado agente. Uma delas é o *Q-Learning* [17, 18], que é um algoritmo simples, mas eficiente, que aprende valores de ganho para cada par estado-ação (S, U) , e com isso consegue alcançar uma política ótima. Outro algoritmo relevante é o ator crítico [17, 19, 20, 21], que aprende uma política ótima ao separar um ator, que escolhe e executa essa política, e um crítico, que avalia a política e provê feedback para o ator.

Nesse trabalho, uma abordagem é apresentada, baseada em [1, 11], para fazer a seleção de comportamento e de ação de um agente móvel com sensores probabilísticos. Além disso, um algoritmo de aprendizagem por reforço, *Q-Learning*, é utilizado para aprender uma política ótima de escolha de comportamento, presente em [17].

2.3 Filtro Bayesiano

Um filtro bayesiano é uma abordagem probabilística [1]. Ele estima uma função de densidade de probabilidade desconhecida de forma recursiva no tempo, utilizando ações e observações, além de um modelo matemático do processo.

No filtro bayesiano, é desejado estimar a distribuição conjunta de probabilidade:

$$P(M^{0:t} S^{0:t} Z^{0:t} | \pi_f), \quad (2.2)$$

sendo M^i o conjunto de ações, S^i o estado do sistema, Z^i os dados dos sensores, todos nesse instante i , e t o instante de tempo para o qual se deseja obter o valor dessa distribuição. Essa função de probabilidade representa a probabilidade de uma sequência de ações, observações e estados acontecerem dado o que se sabe sobre o mundo (π_f). Para obter essa distribuição assume -se que esse sistema pode ser modelado como na distribuição conjunta presente na equação 2.3.

$$P(M^{0:t} S^{0:t} Z^{0:t} | \pi_f) = P(M^0 S^0 Z^0 | \pi_f) \cdot \prod_{j=1}^t \begin{pmatrix} P(S^j | S^{j-1} M^{j-1} \pi_f) \\ \times P(Z^j | S^j \pi_f) \\ \times P(M^j | S^j M^{j-1} \pi_f) \end{pmatrix}. \quad (2.3)$$

Essa equação explicita quatro itens:

- $P(M^0 S^0 Z^0 | \pi_f)$: As probabilidades do estado inicial do sistema são necessárias para se obter seus futuros valores;
- $P(S^j | S^{j-1} M^{j-1} \pi_f)$: A probabilidade de cada estado depende das probabilidades do estado anterior e da ação tomada no tempo anterior (modelo de movimento);
- $P(Z^j | S^j \pi_f)$: É necessário um modelo probabilístico do sensor. Esse termo representa a probabilidade de se ter recebido uma certa medida dos sensores, dado que o estado atual realmente seja S^j (modelo de observação);
- $P(M^j | S^j M^{j-1} \pi_f)$: A escolha da próxima ação depende de qual o estado do sistema e qual foi a ação anterior (modelo de ação motora).

A partir dessas densidades probabilísticas, pode-se decidir por uma estratégia de ação em qualquer instante de tempo utilizando a ação $M^t = m$ que maximize a probabilidade descrita na Equação 2.3. Pode-se também descobrir o estado mais provável do sistema em um instante de tempo j , calculando essa distribuição (Equação 2.3) e vendo qual estado $s^j \in S^j$ possui maior probabilidade.

Esse modelo pode, ainda, ser reescrito para cada instante de tempo, sendo descrito pela Equação 2.4.

$$P(M^{0:t} S^{0:t} Z^{0:t} | \pi_f) = P(M^{0:t-1} S^{0:t-1} Z^{0:t-1} | \pi_f) \cdot \begin{pmatrix} P(S^t | S^{t-1} M^{t-1} \pi_f) \\ \times P(Z^t | S^t \pi_f) \\ \times P(M^t | S^t M^{t-1} \pi_f) \end{pmatrix}. \quad (2.4)$$

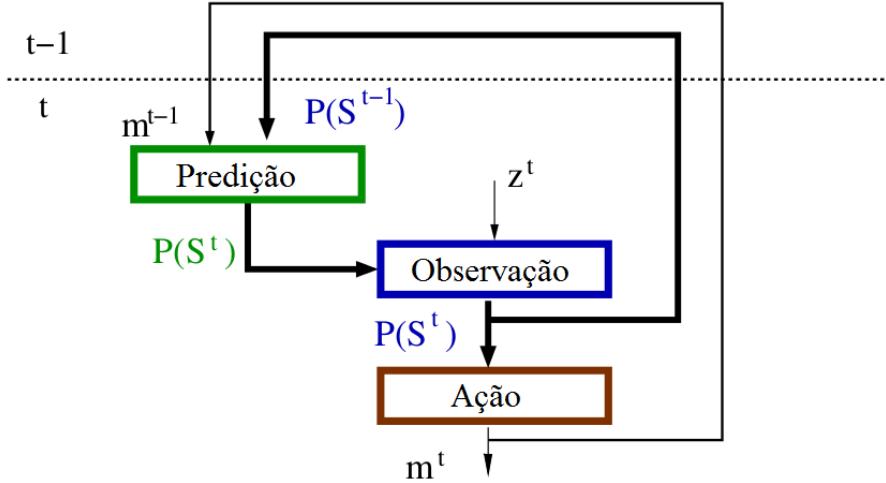


Figura 2.1: Filtro Bayesiano Recursivo.

Com isso tem-se um modelo recursivo que pode ser utilizado a cada instante de tempo para atualizar o modelo de probabilidades. Essa atualização pode ainda ser dividida em três partes distintas: predição, observação e escolha de ação motora, como descrito a seguir.

2.3.1 Predição

Nessa etapa, a partir da ação executada no período anterior de tempo (instante de tempo $t-1$), é realizada uma estimativa de qual será o estado após sua execução (instante de tempo t).

$$P(S^t | z^{0:t-1} m^{0:t-1} \pi_f) \propto \sum_{S^{t-1}} \begin{pmatrix} P(S^t | S^{t-1} m^{t-1} \pi_f) \\ \times P(m^{t-1} | S^{t-1} m^{t-2} \pi_f) \\ \times P(S^{t-1} | z^{0:t-1} m^{0:t-2} \pi_f) \end{pmatrix}. \quad (2.5)$$

2.3.2 Observação

Então, o estado probabilístico do agente é atualizado a partir dos sensores presentes no robô.

$$P(S^t | z^{0:t} m^{0:t-1} \pi_f) \propto \begin{pmatrix} P(z^t | S^t \pi_f) \\ \times P(S^t | z^{0:t-1} m^{0:t-1} \pi_f) \end{pmatrix}. \quad (2.6)$$

2.3.3 Escolha de ação motora

Por último, a seleção de uma ação a ser executada pelo robô é realizada. Calcula-se a distribuição de probabilidade de cada ação $m^t \in M^t$ e aquela com maior valor de probabilidade é escolhida.

$$P(M^t | z^{0:t} m^{0:t-1} \pi_f) \propto \sum_{S^t} \left(\frac{P(M^t | S^t m^{t-1} \pi_f)}{\times P(S^t | z^{0:t} m^{0:t-1} \pi_f)} \right). \quad (2.7)$$

2.4 Processo de Decisão de Markov

Esse processo de decisão foi nomeado a partir de Andrey Markov e provê uma base matemática para o modelamento de tomada de decisões. Os MDPs são uma extensão de cadeias de Markov, tendo como diferença a adição de ações e recompensas (escolha e motivação).

MDP é, mais precisamente, um processo de controle estocástico de tempo discreto [4]. Isso significa que ele é um processo de controle probabilístico, baseado em passos. Em cada passo o robô se encontra em um estado S perfeitamente conhecido em que se pode executar qualquer ação genérica u disponível para esse estado. O modelo de percepção do robô, $p(Z | S)$ é uma equação determinística e bijetora, o robô só pode estar em um estado $s \in S$ para uma percepção $z \in Z$ do ambiente.

Após executada essa ação u , o robô se encontra em um novo estado $s' \in S$ com probabilidade $p(s' | u, s)$, essa equação é conhecida como modelo de atuação. O modelo de atuação não é, em geral, determinístico, ou seja, uma ação pode ter várias consequências com diferentes probabilidades. Uma consequência disso é que planejar uma única sequência de ações não é o suficiente, o planejador deve decidir por uma ação para cada possível estado $s \in S$, em que o robô possa se encontrar.

No grafo da figura 2.2, para uma ação a ou b , tem-se as probabilidades $p(s' | u, s)$ através do percentual indicado nas setas saindo do estado s para s' . Pode-se ver que, para esse sistema, $P(s2 | a, s1) = 0,2$ e $P(s4 | a, s1) = 0,8$. Se $s1$ for o estado inicial e $s6$ for o objetivo, não adianta obter uma sequência de ações $a \rightarrow a \rightarrow b$, por exemplo, pois seria possível que o robô se encontrasse em vários possíveis estados ao final dessa sequência: $s4, s2$ ou $s6$.

É necessário mapear uma ação para cada estado possível, gerando o que é chamado de política de controle. Essa política de controle tem formato $\pi : S^t \rightarrow U^t$, ou seja, a política π recebe um estado s^t e retorna a ação u^t planejada para ele. Portanto, com a política $\pi(S)$, não há mais o problema anterior, pois o planejamento da ação a cada instante é baseado no estado em que o robô se encontra.

Calcular essa política não é trivial, devido, principalmente, à existência de infinitos caminhos para se chegar de um estado $s1$ a $s6$. Um possível caminho é $s1 \rightarrow s2 \rightarrow s3 \rightarrow s6$, por exemplo, outro caminho seria $s1 \rightarrow s2 \rightarrow s5 \rightarrow s2 \rightarrow s3 \rightarrow s6$, e outro ainda é $s1 \rightarrow s2 \rightarrow s5 \rightarrow s2 \rightarrow s5 \rightarrow s2 \rightarrow s3 \rightarrow s6$.

Antes de obter um algoritmo definitivo, é necessário modelar o objetivo do problema. Para isso é utilizada uma função de recompensa $r(S, U, S')$ que retorna um número real. Ela indica quão desejado é se chegar a um estado $s' \in S'$ a partir de $s \in S$ executando a ação $u \in U$. Para sistemas cujo único objetivo é chegar a estados desejados, é possível simplificar essa função para $r(s')$, por exemplo.

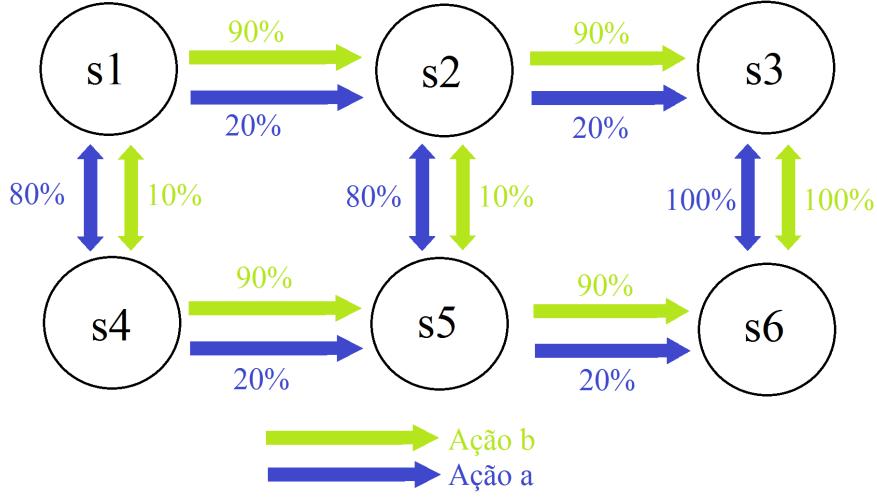


Figura 2.2: Mapa de Probabilidades.

Sabendo que é desejado encontrar uma política que maximize o ganho de recompensas, é possível gerar um plano de políticas ótimo pensando-se apenas na próxima ação, maximizando uma recompensa imediata. Caso as recompensas sejam dadas apenas pela escolha de uma ação baseada no estado atual $r(S, U)$, sem considerar o estado alcançado, uma possível escolha de política seria a dada pela Equação 2.8.

$$\pi_1(s) = \underset{u}{\operatorname{argmax}}(r(s, u)). \quad (2.8)$$

Como a recompensa depende também do estado alcançado, é necessário ponderar cada recompensa $r(S, U, S')$ pela probabilidade de se alcançar um estado $s' \in S'$:

$$\pi_1(s) = \underset{u}{\operatorname{argmax}} \left(\int r(s, u, s') \cdot P(s' | u, s) ds' \right). \quad (2.9)$$

É possível então gerar, para essa política, uma função de valor que representa a recompensa esperada para cada estado s seguindo a política π :

$$V_1(s) = \underset{u}{\operatorname{max}} \left(\int r(s, u, s') \cdot P(s' | u, s) ds' \right). \quad (2.10)$$

Ao pensar dois passos a frente, pode-se utilizar esse novo valor $V_1(S)$ para calcular a política $\pi_2(S)$, planejando agora um passo à frente de antes. Depois generaliza-se esse pensamento para qualquer caso futuro, pensando j passos a frente.

$$\pi_j(s) = \underset{u}{\operatorname{argmax}} \left(\int (r(s, u, s') + \gamma \cdot V_{j-1}(s')) \cdot P(s' | u, s) ds' \right). \quad (2.11)$$

Onde γ é uma constante de desconto, usada para incentivar o ganho de recompensas imediatas em oposição às tardias. Essa função indica que a política ótima é a que gera uma maior recom-

pensa imediata, somada à somatória de todas as recompensas futuras esperadas, ponderadas pelas probabilidades de serem recebidas. Gera-se, novamente, uma função de valor para essa política, presente na Equação 2.12.

$$V_j(s) = \max_u \left(\int (r(s, u, s') + \gamma \cdot V_{j-1}(s')) \cdot P(s' | u, s) ds' \right). \quad (2.12)$$

Ao pensar $j \rightarrow \infty$ passos à frente, a política geralmente converge e uma política ótima para esse sistema é encontrada.

$$\pi^*(s) = \pi_\infty(s) = \operatorname{argmax}_u \left(\int (r(s, u, s') + \gamma \cdot V_\infty(s')) \cdot P(s' | u, s) ds' \right). \quad (2.13)$$

$$V^*(s) = V_\infty(s) = \max_u \left(\int (r(s, u, s') + \gamma \cdot V_\infty(s')) \cdot P(s' | u, s) ds' \right). \quad (2.14)$$

2.5 Diferença Temporal

Caso não se conheçam os valores das recompensas $r(S, U, S')$ e nem resultado de uma ação genérica $P(s' | u, s)$ pode-se utilizar, como solução, a aprendizagem de reforço. O algoritmo de diferença temporal visa, a partir de uma política fixa $\pi(S)$, aprender os valores finais de $V(S)$ através da execução de ações em diferentes estados, sem ter um conhecimento prévio das recompensas recebidas ou do modelo probabilístico do resultado de suas ações.

Esse algoritmo baseia-se na Equação 2.14 obtida no seção 2.4 e parte da premissa que uma política π_{td} já foi escolhida. Ele então atualiza o valor de $V(S)$ a cada experiência qualquer (s, u, s', r) , ou seja, sempre que um estado $s' \in S$ é alcançado a partir do estado $s \in S$, com uma ação $u \in U$ e recebendo uma recompensa $r \in \mathbb{R}$. Dessas experiências, pode-se obter um valor *amostra* tal que:

$$\text{amostra}^t = r(s, u, s') + \gamma \cdot V_{\pi_{td}}^t(s'). \quad (2.15)$$

Agora, escolhendo-se uma constante de aprendizagem α , pode-se atualizar o valor de $V_{\pi_{td}}(s)$ tal que:

$$V_{\pi_{td}}^t(s) = (1 - \alpha) \cdot V_{\pi_{td}}^{t-1}(s) + \alpha \cdot \text{amostra}^t. \quad (2.16)$$

É possível observar que, utilizando valores suficientemente pequenos de α , os valores de $V_{\pi_{td}}(s)$ convergem com o tempo e, principalmente, convergem para os mesmos valores que o algoritmo MDP, considerando que a política π_{td} utilizada é ótima [17].

A limitação da diferença temporal é que ela apenas avalia uma política fixa já existente. Caso deseje-se mudar a política, ou gerar uma utilizando este algoritmo, surge um problema, pois todos os valores de $V(S)$ dependem somente do estado e não da ação executada. Caso a política π_{td} executada seja alterada, é necessário recalcular todos os valores $V(S)$.

2.6 Q-Learning

Q-Learning é um dos mais conhecidos e mais utilizados algoritmos de aprendizagem por reforço, sendo utilizado para controle de robôs [22], tratamento de problemas em processamento de imagens [23] e criação de sistemas financeiros [24]. Esse é um algoritmo de aprendizagem ativa, ou seja, a escolha da política e das ações é feita juntamente com a aprendizagem e é possível alcançar uma nova política ótima.

Ao invés de utilizar o valor de $V(S)$, na equação 2.14, o *Q-Learning* utiliza um valor $Q(S, U)$, tal que:

$$\pi^t(s) = \underset{u}{\operatorname{argmax}}(Q^t(s, u)) \quad (2.17)$$

e

$$V^t(s) = \max_u(Q^t(s, u)). \quad (2.18)$$

Ou seja:

$$Q^t(s, u) = \int (r(s, u, s') + \gamma \cdot V^{t-1}(s')) \cdot P(s' | u, s) \, ds', \quad (2.19)$$

que pode ser reescrito como:

$$Q^t(s, u) = \int \left(r(s, u, s') + \gamma \cdot \max_{u'}(Q^{t-1}(s', u')) \right) \cdot P(s' | u, s) \, ds'. \quad (2.20)$$

Ao aprender o valor de $Q^t(S, U)$, ao invés do de $V^t(S)$, é possível modificar a política sem ter de reaprender todos os seus valores. Analogamente às equações 2.15 e 2.16 em TD, aprende-se através de amostras obtidas a partir de experiências (s, u, s', r) :

$$\text{amostra}^t = r(s, u, s') + \gamma \cdot \max_u(Q^{t-1}(s', u')) \quad (2.21)$$

e

$$Q^t(s, u) = (1 - \alpha) \cdot Q^{t-1}(s, u) + \alpha \cdot \text{amostra}^t. \quad (2.22)$$

Com um número suficiente de iterações e um decaimento apropriado do parâmetro de aprendizagem α , $Q^t(S, U)$ tende a um valor ótimo com uma probabilidade 1 (100%) [25, 26, 27] e a política escolhida com base na função 2.17 também é ótima.

Algumas limitações com esse algoritmo são:

- podem existir muitos estados para se visitar: em um espaço contínuo, por exemplo, seriam infinitos;
- podem existir muitas ações possíveis para cada estado: para o acionamento analógico de um motor, por exemplo, seriam infinitas;
- se houver muitos pares ação-estado (s, u) , mesmo que seja possível aprender por um tempo muito grande, é necessário armazenar o valor de $Q^t(s, u)$ para cada ação que for possível de ser executada em cada estado;
- o algoritmo não consegue aplicar o que aprendeu em um estado para outros estados com características similares.

2.6.1 Generalização dos Pares Estado-Ação

Uma solução para esses problemas é generalizar a informação de um par estado-ação para outros similares. Para isso utiliza-se um vetor de características f para descrever o estado. Essas características podem ser:

- distância para uma posição que oferece uma recompensa positiva;
- distância para uma posição que oferece uma recompensa negativa;
- número de alvos a serem capturados;
- estado do robô (com ou sem defeito).

Elas também podem descrever a ação a ser executada, como:

- ficar parado;
- mover uma garra;
- economizar energia.

Essas características podem, também, descrever um estado futuro s' (possivelmente utilizando alguma das características descritas acima) que tenha grande chance de ser alcançado por esse par estado-ação.

Utiliza-se esse vetor f de características $f_j(S, U)$ para obter um valor $Q(S, U)$ tal como descrito na equação 2.23.

$$Q^t(S, U) = \omega_1^t \cdot f_1(S, U) + \omega_2^t \cdot f_2(S, U) + \cdots + \omega_n^t \cdot f_n(S, U). \quad (2.23)$$

Sendo ω_j o peso utilizado para ponderar o valor de cada característica f_j . Com essa equação se consegue um valor próximo de $Q(S, U)$ para estados distintos, mas que possuam características $f_j(S, U)$ com valores próximos. A desvantagem de utilizar essa generalização é que deve-se escolher esses valores/características $f_i(S, U)$ com cuidado para obter uma boa representação do par estado-ação a partir deles, ou pode-se obter estados-ações com valores de $Q(S, U)$ próximos, mas que possuem ganhos distintos na aplicação.

Com esse novo modelo, aprende-se valores a partir de cada experiência, utilizando o erro atual do modelo, descrito pela equação a seguir:

$$\text{erro} = r(s, u, s') + \gamma \cdot \max_u (Q^{t-1}(s', u')) - Q^{t-1}(s, u). \quad (2.24)$$

Esse erro é calculado pela diferença entre a recompensa recebida, somada com o valor de ganho esperado para o novo estado alcançado e o valor atual esperado para o par estado-ação

executada. Utiliza-se o erro acima para atualizar cada um dos pesos usados para obter $Q_t(S, U)$, como explicitado na equação a seguir.

$$\omega_i^t = \omega_i^{t-1} + \alpha \cdot erro \cdot f_i(s, u). \quad (2.25)$$

Assim, caso haja um *erro* grande para um dado estado, os valores de todos os estados com características similares àquele são atualizados. É importante notar também que, caso esse estado tenha um valor maior para uma característica f_j , o peso ω_j dela terá maior alteração que os das outras características.

A função utilizada para calcular $Q(S, U)$ (2.23) é um perceptron, sendo um perceptron um algoritmo de apredizagem supervisionada que permite classificar uma entrada em uma de várias opções. Ele é um classificador linear, ou seja, ele utiliza uma série de pesos, combinados com um vetor de características, para fazer essa classificação.

Esse tipo de função (perceptron) possui limitações, sendo a principal delas a capacidade de aprender apenas problemas linearmente separáveis [28, 29]. Por isso, os parâmetros $f_i(S, U)$ devem ser bem escolhidos: não só para caracterizar bem um estado e permitir uma boa diferenciação entre dois deles, mas também para permitir essa aprendizagem.

2.6.2 Escolha de Ações e Exploração Epsilon Gulosa

Utilizando a Equação 2.17, em conjunto com a 2.23, é possível criar uma política π para escolha de ações. Essa política, caso o valor de $Q(S, U)$ já tenha sido aprendido com sucesso, será ótima.

Mas, se desde o começo do aprendizado esse método for utilizado para escolher a ação executada, é possível ficar preso em máximos locais¹, sem explorar outras opções de ação. Um meio de evitar esse problema é utilizar outros métodos de exploração durante o treinamento, que podem ser desativados quando o treinamento acabar.

Um desses métodos é chamado de exploração epsilon gulosa (*epsilon greedy*) [17]. É um método em que, antes de se selecionar uma ação, se “arremessa uma moeda”: dependendo do resultado, ou uma ação randômica é executada, ou uma ação baseada nos valores de $Q^t(S, U)$ atuais é escolhida, sendo escolhida a ação que maximize o valor dessa função Q^t .

Algorithm 1 Exploração Epsilon Gulosa

```

1: procedure PEGAR_ACAO
2:   numero_randomico  $\leftarrow$  random numero
3:   if numero_randomico < FATOR_DE_EXPLORACAO then
4:     return acao_randomica
5:   else
6:     return acao_aprendida

```

¹Um estado que é melhor do que outros com pequenas variações, mas, globalmente, existem outros estados melhores.

Esse parâmetro $\beta = FATOR_DE_EXPLORACAO$ é importante e deve ser bem escolhido. Um valor muito baixo limita a exploração, enquanto um muito alto atrasa a aprendizagem e pode impedir a aprendizagem de uma política que execute uma sequência de ações mais complexa [17].

Capítulo 3

Desenvolvimento

“Because the people who are crazy enough to think they can change the world, are the ones who do.”
– Apple Inc.

3.1 Introdução

Este capítulo apresenta as soluções adotadas para os problemas de planejamento de tarefas e de tomada de decisões, bem como sua modelagem matemática. Nele também são descritos os ambientes de teste em simulação e a modelagem utilizada para descrever o agente móvel. Por último, é apresentada uma descrição dos testes realizados.

3.2 Modelagem Matemática

Nesta seção são descritas as mudanças realizadas nos modelos matemáticos utilizados nesse projeto. Também é apresentada uma explicação sobre o modelo final do projeto e sobre como as várias teorias se encaixam.

3.2.1 Aprendizagem por Reforço para Seleção de Comportamentos

Um comportamento pode ser descrito como um conjunto de reações a estímulos externos, ou seja, ele é um modelo que um agente, ou sistema, utiliza para fazer a escolha de suas ações. Um agente, em um mesmo estado do sistema, pode ter duas escolhas de ação diferentes, se estiver em comportamentos diferentes.

Neste trabalho, a aprendizagem por reforço é utilizada para aprender uma política de seleção de comportamento e não de ações motoras. Para isso, é necessário alterar as equações do algoritmo *Q-Learning* para que, ao invés de escolher uma ação $u \in U$, escolher um comportamento $b \in B$. Reescrevendo as principais equações desse algoritmo elas são:

$$Q^t(s, b) = \int (r(s, b, s') + \gamma \cdot V^{t-1}(s')) \cdot P(s' | b, s) \, ds', \quad (3.1)$$

$$\pi^t(s) = \underset{b}{\operatorname{argmax}}(Q^t(s, b)) \quad (3.2)$$

e

$$V^t(s) = \max_b(Q^t(s, b)). \quad (3.3)$$

Além disso, as características f_j agora serão baseadas em pares estado-comportamento (S, B) e não mais em pares estado-ação.

$$Q(S, B) = \omega_1 \cdot f_1(S, B) + \omega_2 \cdot f_2(S, B) + \cdots + \omega_n \cdot f_n(S, B), \quad (3.4)$$

$$\text{erro} = r(s, b, s') + \gamma \cdot \max_b(Q^{t-1}(s', b')) - Q^{t-1}(s, b) \quad (3.5)$$

e

$$\omega_i^t = \omega_i^{t-1} + \alpha \cdot \text{erro} \cdot f_i(s, b). \quad (3.6)$$

3.2.2 Seleção de Comportamentos em um Sistema Parcialmente Observável

O algoritmo *Q Learning* foi criado baseado no MDP, que, como citado na seção 2.4, é utilizado para sistemas completamente observáveis¹. É necessário, então, expandir sua definição para aplicá-lo em um sistema parcialmente observável².

Para isso, deve-se primeiramente introduzir uma nova variável $a \in A$ que representa a distribuição de probabilidades em todos os possíveis estados S . Considere, por exemplo, um caso em que o conjunto de estados seja composto por duas variáveis $S = \binom{S_1}{S_2}$, que podem ter valores $S_1 \in \{1, 2, 3\}$ e $S_2 \in \{\text{verdadeiro}, \text{falso}\}$. Uma variável $a \in A$, nesse caso, seria dada por cinco variáveis, representando as probabilidades de S ter cada um dos possíveis valores.

$$a = \begin{pmatrix} P(S_1 = 1) & P(S_2 = \text{verdadeiro}) \\ P(S_1 = 2) & P(S_2 = \text{falso}) \\ P(S_1 = 3) \end{pmatrix}$$

A partir dessa definição, utilizando como inspiração o algoritmo POMDP [4], o estado do sistema $s \in S$ é substituído, na equação 2.19, por uma variável $a \in A$, que represente essa distribuição de probabilidades nos possíveis estados S .

$$Q^t(a, b) = \int (r(a, b, a') + \gamma \cdot V^{t-1}(a')) \cdot P(a' | b, a) da'. \quad (3.7)$$

¹Sistemas em que existe uma função $f(Z)$ que, dado uma medida dos sensores, retorna o estado atual do sistema. Em outras palavras, é um sistema em que, dado uma medida $z \in Z$ dos sensores se sabe, sem dúvidas, qual o estado presente do sistema $s \in S$

²Sistema que não é completamente observável, ou seja, que para uma medida $z \in Z$ dos sensores só se pode estimar qual a probabilidade desse sistema se encontrar num estado $s \in S$

Tem-se, agora, uma seleção de política baseada na distribuição de probabilidades nos estados $s \in S$:

$$\pi^t(a) = \underset{b}{\operatorname{argmax}}(Q^t(a, b)), \quad (3.8)$$

$$V^t(a) = \max_b(Q^t(a, b)). \quad (3.9)$$

Analogamente às equações 2.21 e 2.22 no *Q-Learning* original, aprende-se através de amostras obtidas a partir de experiências (a, b, a', r) .

$$\text{amostra} = r(a, b, a') + \gamma \cdot \max_b(Q^{t-1}(a', b')), \quad (3.10)$$

$$Q^t(a, b) = (1 - \alpha) \cdot Q^{t-1}(a, b) + \alpha \cdot \text{amostra}. \quad (3.11)$$

O problema da equação 3.11 é que, ao utilizar uma distribuição $a \in A$ de probabilidades de todos os estados $s \in S$ e não próprio estado, não é possível simplesmente aprender os valores para cada item. Isso acontece pois, mesmo para espaços de estados discretos, há infinitos valores de $a \in A$ possíveis.

A teoria vista no tópico 2.6.1 é utilizada, então, para criar uma generalização desses estados através de características deles. Essas características, ao contrário das vistas anteriormente, devem ser baseadas nas probabilidades de se estar/alcançar um estado. Alguns exemplos são:

- distância da posição mais provável do robô, para uma posição que ofereça um produto (probabilidade de receber uma recompensa * recompensa) alto;
- distância da posição mais provável do robô, para uma posição que ofereça um produto (probabilidade de receber uma recompensa * recompensa) negativo;
- probabilidade de existir algum perigo (recompensa negativa) mais próximo do robô;
- probabilidade de capturar um alvo;
- porcentagem de chance do robô ter um erro.

Essas características podem, como no caso anterior, descrever também o comportamento a ser executado, como:

- tentar capturar um alvo;
- tentar fugir de um perigo;
- economizar energia.

Com isso um valor $Q(A, B)$ é obtido, tal como descrito na equação a seguir:

$$Q(A, B) = \omega_1 \cdot f_1(A, B) + \omega_2 \cdot f_2(A, B) + \cdots + \omega_n \cdot f_n(A, B). \quad (3.12)$$

Com esse novo modelo, é possível aprender os valores dos pesos, a partir de cada experiência, utilizando o erro atual do sistema:

$$\text{erro} = r(a, b, a') + \gamma \cdot \max_b (Q^{t-1}(a', b')) - Q^{t-1}(a, b), \quad (3.13)$$

$$\omega_i^t = \omega_i^{t-1} + \alpha \cdot \text{erro} \cdot f_i(a, b). \quad (3.14)$$

Assim, caso exista um erro grande para uma dada distribuição de probabilidades, são atualizados os valores de todos os pares (a, b) que possuam distribuição de probabilidades dos estados com características similares àquele. E caso esse estado tenha um valor maior para uma das características, o peso dela será mais modificado que as outras.

Então, caso for utilizada como característica, por exemplo, f_i = “probabilidade de existir algum perigo (recompensa negativa) a menos que uma certa distância”. Se for recebida uma recompensa negativa em um estado no qual o valor de f_i é alto, estados com essa característica serão considerados piores. Se for recebida uma recompensa positiva, os estados com valor alto para essa característica serão considerados melhores.

3.2.3 Abordagem Bayesiana com a Seleção de Comportamentos

Partindo dos modelos obtidos na seção 2.3, pode-se agora utilizar a seleção de comportamento, vista no tópico 3.2.2, como uma nova medida de sensor, fazendo:

$$Z^+ = \begin{pmatrix} Z \\ Z_b \end{pmatrix} = \begin{pmatrix} Z \\ B \end{pmatrix}. \quad (3.15)$$

Sendo Z as medidas sensorias e $B = Z_b$ o comportamento obtido utilizando a aprendizagem por reforço. É necessário expandir também o modelo dos estados para:

$$S^+ = \begin{pmatrix} S \\ S_b \end{pmatrix}. \quad (3.16)$$

Sendo, S o modelo do estado atual usual e S_b uma variável de estado que representa o comportamento atual do agente.

O filtro bayesiano é, então, reescrito como:

$$P(M^{0:t}S^{0:t}S_b^{0:t}Z^{0:t}B^{0:t} | \pi_f) = P(M^0S^0S_b^0Z^0B^0 | \pi_f) \cdot \prod_{j=1}^t \begin{pmatrix} P(S^jS_b^j | S^{j-1}S_b^{j-1}M^{j-1}\pi_f) \\ \times P(Z^jB^j | S^jS_b^{j-1}\pi_f) \\ \times P(M^j | S^jS_b^jM^{j-1}\pi_f) \end{pmatrix}. \quad (3.17)$$

Existem três etapas para atualizar o estado S^+ recursivamente. Para esse novo filtro elas são:

- Predição:

$$P(S^tS_b^t | z^{0:t-1}b^{0:t-1}m^{0:t-1}\pi_f) \propto \sum_{S^{t-1}S_b^{t-1}} \begin{pmatrix} P(S^tS_b^t | S^{t-1}S_b^{t-1}m^{t-1}\pi_f) \\ \times P(m^{t-1} | S^{t-1}S_b^{t-1}m^{t-2}\pi_f) \\ \times P(S^{t-1}S_b^{t-1} | z^{0:t-1}b^{0:t-1}m^{0:t-2}\pi_f) \end{pmatrix}; \quad (3.18)$$

- Observação:

$$P(S^tS_b^t | z^{0:t}b^{0:t}m^{0:t-1}\pi_f) \propto \begin{pmatrix} P(z^tb^t | S^tS_b^t\pi_f) \\ \times P(S^tS_b^t | z^{0:t-1}b^{0:t-1}m^{0:t-1}\pi_f) \end{pmatrix}; \quad (3.19)$$

- Seleção de ação motora:

$$P(M^t | z^{0:t}b^{0:t}m^{0:t-1}\pi_f) \propto \sum_{S_i^{t-1}S_b^{t-1}} \begin{pmatrix} P(M^t | S^tS_b^tm^{t-1}\pi_f) \\ \times P(S^tS_b^t | z^{0:t}b^{0:t}m^{0:t-1}\pi_f) \end{pmatrix}. \quad (3.20)$$

Assume-se que o estado S^t é independente do comportamento sendo executado no mesmo momento S_b^t e que esse comportamento atual independe de tempos anteriores. Assume-se, também, que os dados sensoriais Z^t são independentes do comportamento escolhido pelo *Q-Learning* (B^t) em um mesmo momento. Com isso, é possível simplificar o filtro para:

$$\begin{aligned} P(M^{0:t}S^{0:t}S_b^{0:t}Z^{0:t}B^{0:t} | \pi_f) &= \\ P(M^0S^0S_b^0Z^0B^0 | \pi_f) \cdot \prod_{j=1}^t &\begin{pmatrix} P(S^j | S^{j-1}M^{j-1}\pi_f) \times P(S_b^j | \pi_f) \\ \times P(Z^j | S^j\pi_f) \times P(B^j | S_b^{j-1}\pi_f) \\ \times P(M^j | S^jS_b^jM^{j-1}\pi_f) \end{pmatrix}. \end{aligned} \quad (3.21)$$

Pode-se perceber que o modelo de seleção de ação motora:

$$P(M^t | S^tS_b^tM^{t-1}\pi_f), \quad (3.22)$$

agora depende do comportamento sendo utilizado.

$$P(M^t | S^tS_b^tM^{t-1}\pi_f) = \begin{cases} P(M^t | S^t [S_b^t = b_1] M^{t-1}\pi) \\ P(M^t | S^t [S_b^t = b_2] M^{t-1}\pi) \\ \dots \\ P(M^t | S^t [S_b^t = b_{N_b}] M^{t-1}\pi) \end{cases}. \quad (3.23)$$

A vantagem é que se pode ter, para cada comportamento, um modelo de ação diferente. Esse modelo pode ainda ser escrito em sua forma recursiva, para um dado instante de tempo t , ficando como na equação a seguir:

$$P(M^{0:t}S^{0:t}S_b^{0:t}Z^{0:t}B^{0:t} | \pi_f) = \\ P(M^{0:t-1}S^{0:t-1}S_b^{0:t-1}Z^{0:t-1}B^{0:t-1} | \pi_f) \cdot \begin{pmatrix} P(S^t | S^{t-1}M^{t-1}\pi_f) \times P(S_b^t | \pi_f) \\ \times P(Z^t | S^t\pi_f) \times P(B^t | S_b^{t-1}\pi_f) \\ \times P(M^t | S^tS_b^tM^{t-1}\pi_f) \end{pmatrix}. \quad (3.24)$$

Esse filtro simplificado pode, como o da seção 2.3, ser escrito na sua forma recursiva e separado em 3 etapas.

- Predição:

$$P(S^tS_b^t | z^{0:t-1}b^{0:t-1}m^{0:t-1}\pi_f) \propto \sum_{S^{t-1}S_b^{t-1}} \begin{pmatrix} P(S^t | S^{t-1}m^{t-1}\pi_f) \times P(S_b^t | \pi_f) \\ \times P(m^{t-1} | S^{t-1}S_b^{t-1}m^{t-2}\pi_f) \\ \times P(S^{t-1}S_b^{t-1} | z^{0:t-1}b^{0:t-1}m^{0:t-2}\pi_f) \end{pmatrix}. \quad (3.25)$$

- Observação:

$$P(S^tS_b^t | z^{0:t}b^{0:t}m^{0:t-1}\pi_f) \propto \begin{pmatrix} P(z^t | S^t\pi_f) \times P(b^t | S_b^t\pi_f) \\ \times P(S^tS_b^t | z^{0:t-1}b^{0:t-1}m^{0:t-1}\pi_f) \end{pmatrix}. \quad (3.26)$$

- Seleção de ação motora:

$$P(M^t | z^{0:t}b^{0:t}m^{0:t-1}\pi_f) \propto \sum_{S^{t-1}S_b^{t-1}} \begin{pmatrix} P(M^t | S^tS_b^tm^{t-1}\pi_f) \\ \times P(S^tS_b^t | z^{0:t}b^{0:t}m^{0:t-1}\pi_f) \end{pmatrix}. \quad (3.27)$$

3.2.4 Modelo Final e Completo

Em posse da modelagem matemática introduzida no capítulo 2 e explorada nos tópicos 3.2.1, 3.2.2 e 3.2.3, é possível agora apresentar o modelamento completo utilizado neste trabalho.

É possível separar esse modelo em duas partes: uma de treinamento e outra em que a aprendizagem já foi concluída.

Durante a fase de treinamento, então, um modelo com cinco etapas é utilizado. As já conhecidas: predição; observação; seleção de ação motora. Integradas com a seleção de comportamento e aprendizagem. Esse modelo está representado na Figura 3.1.

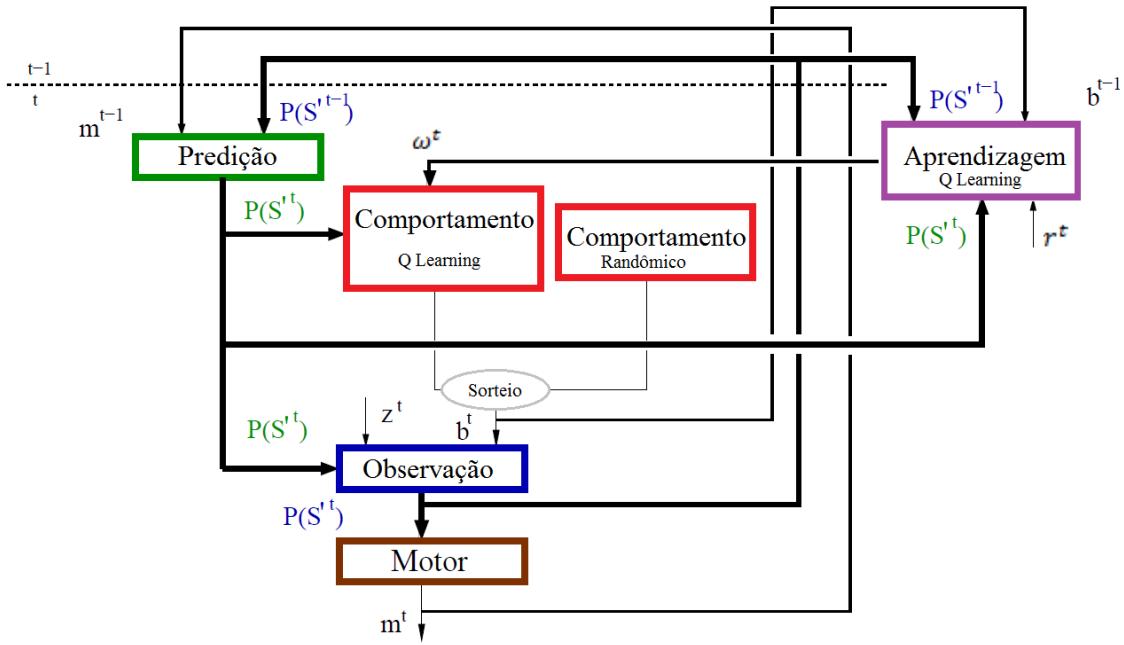


Figura 3.1: Filtro bayesiano utilizando *Q-Learning* para seleção de comportamento. Durante treinamento.

Após acabado o treinamento, não se tem mais necessidade da etapa de aprendizagem, ficando então um modelo com apenas 4 etapas. Esse modelo pode ser representado pela Figura 3.2.

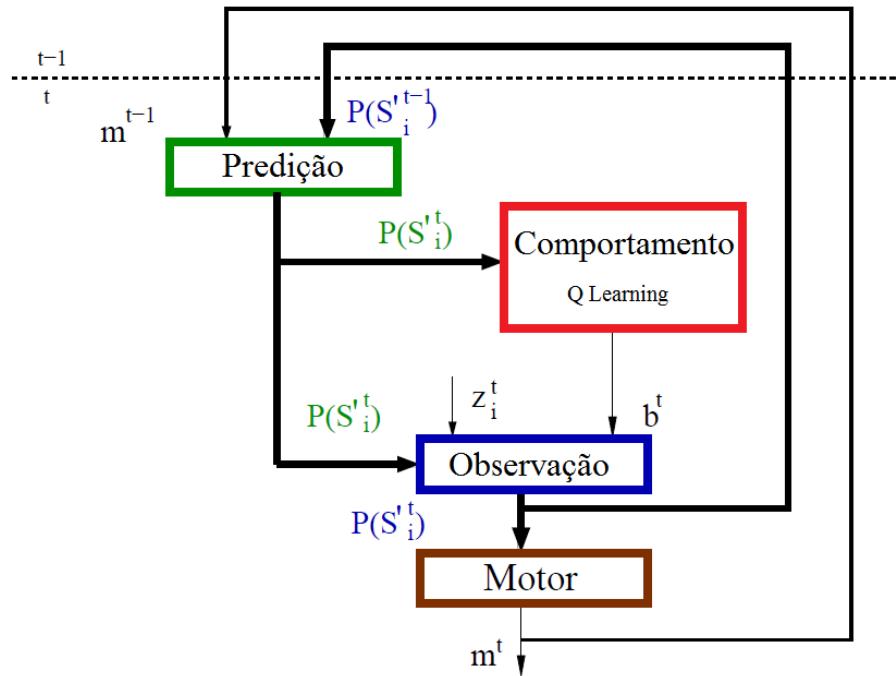


Figura 3.2: Filtro bayesiano utilizando *Q-Learning* para seleção de comportamento. Após treinamento completo.

3.2.4.1 Predição

Nessa etapa, a partir da ação escolhida no período anterior de tempo (instante de tempo $t - 1$), faz-se uma estimativa de qual será o estado após sua execução (instante de tempo t).

$$P(S^t S_b^t | z^{0:t-1} b^{0:t-1} m^{0:t-1} \pi_f) \propto \sum_{S^{t-1} S_b^{t-1}} \begin{pmatrix} P(S^t | S^{t-1} m^{t-1} \pi_f) \times P(S_b^t | \pi_f) \\ \times P(m^{t-1} | S^{t-1} S_b^{t-1} m^{t-2} \pi_f) \\ \times P(S^{t-1} S_b^{t-1} | z^{0:t-1} b^{0:t-1} m^{0:t-2} \pi_f) \end{pmatrix}. \quad (3.28)$$

3.2.4.2 Escolha de comportamento

Essa etapa é onde a escolha do comportamento acontece. Ela pode ter dois métodos distintos, um durante a fase de treinamento e outro após acabar a aprendizagem.

Durante a aprendizagem utiliza-se uma exploração gulosa, descrita no tópico 2.6.2, para fazer essa escolha do comportamento. Nela, primeiro sorteia-se um número randômico $x \in [0, 1]$. Se esse número for menor que um fator de exploração β , escolhe-se um comportamento randômico entre todos os possíveis. Caso contrário, escolhe-se um comportamento da mesma forma que após a aprendizagem ter acabado, descrito a seguir.

Para a escolha de comportamento após terminado o treinamento, primeiro calcula-se a função 3.29, a seguir, para cada comportamento $b \in B^t$ e para o conjunto de probabilidades³ $a^t \in A^t$ de se encontrar em cada estado S^t .

$$Q(a^t, B^t) = \omega_1 \cdot f_1(a^t, B^t) + \omega_2 \cdot f_2(a^t, B^t) + \dots + \omega_n \cdot f_n(a^t, B^t). \quad (3.29)$$

Depois, escolhe-se um comportamento a partir desses valores calculados, sendo escolhido o que maximiza essa função.

3.2.4.3 Observação

A partir dos sensores presentes no robô e do valor $b \in B$, obtido com o algoritmo de aprendizado, atualiza-se o estado probabilístico do agente.

$$P(S^t S_b^t | z^{0:t} b^{0:t} m^{0:t-1} \pi_f) \propto \begin{pmatrix} P(z^t | S^t \pi_f) \times P(b^t | S_b^t \pi_f) \\ \times P(S^t S_b^t | z^{0:t-1} b^{0:t-1} m^{0:t-1} \pi_f) \end{pmatrix}. \quad (3.30)$$

3.2.4.4 Escolha de ação motora

Por último se faz a seleção de uma ação a ser executada pelo robô. Para isso, calcula-se a distribuição de probabilidade de se executar cada ação $m^t \in M^t$ e escolhe-se a ação que possui

³É importante ressaltar que a^t equivale ao conjunto de probabilidades $P(S^t S_b^t | z^{0:t-1} b^{0:t-1} m^{0:t-1} \pi_f)$, obtido em 3.2.4.1.

maior valor nessa distribuição.

$$P(M^t | z^{0:t} b^{0:t} m^{0:t-1} \pi_f) \propto \sum_{S^t S_b^t} \left(\begin{array}{c} P(M^t | S^t S_b^t m^{t-1} \pi_f) \\ \times P(S^t S_b^t | z^{0:t} b^{0:t} m^{0:t-1} \pi_f) \end{array} \right). \quad (3.31)$$

3.2.4.5 Aprendizagem

Nessa etapa são atualizados os valores do modelo de seleção de comportamento $Q(A, B)$ a partir de uma recompensa r recebida. Tendo posse dessa recompensa, da distribuição de probabilidades do estado anterior $a^{t-1} \in A^{t-1}$ e atual $a^t \in A^t$ do sistema e de qual o comportamento $b^{t-1} \in B^{t-1}$ escolhido no tempo anterior pelo algoritmo de aprendizagem, é possível atualizar os pesos ω_i do sistema de aprendizagem, como descrito nas equações 3.13 e 3.14.

3.3 Ambiente de Testes e Simulação

Para poder testar as teorias aqui descritas foi utilizado uma plataforma de Pac-Man⁴, criada em Berkeley para suas aulas de inteligência artificial (IA).

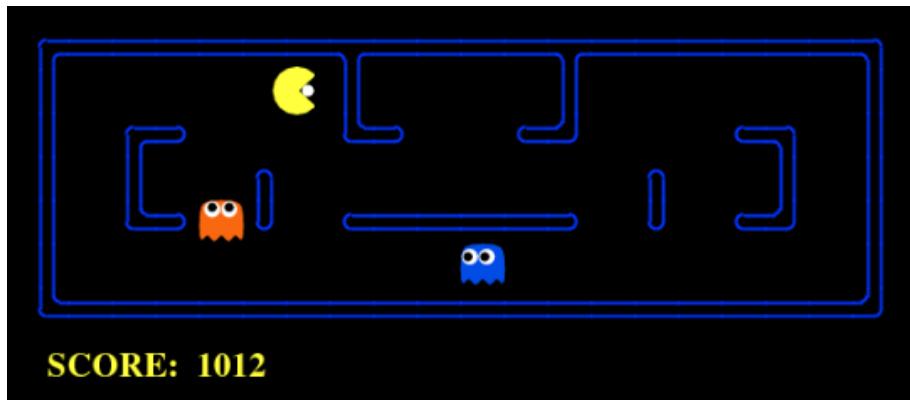


Figura 3.3: Plataforma do jogo Pac-Man desenvolvida em Berkeley para aulas de IA.

A plataforma provê um sistema com informação completa⁵ e sem erros na movimentação⁶. Essa plataforma foi programada em Python e integrada com ROS⁷ [30]. Utilizando o ROS, essa plataforma foi conectada com um programa em C++, no qual foi programado o modelo de aprendizagem e seleção de comportamento proposto neste trabalho.

A comunicação entre a plataforma em Python e o programa em C++ dá-se a partir de quatro mensagens diferentes, as quais simulam os sensores, atuadores e recompensas do sistema.

⁴Essa plataforma pode ser encontrada em http://ai.berkeley.edu/project_overview.html.

⁵Todas as propriedades do jogo são conhecidas a todo momento, como por exemplo: posição do Pacman, posição dos fantasmas e localizações com ou se comida.

⁶Uma ação executada em um dado estado do sistema terá sempre o mesmo resultado.

⁷Ferramenta com funções que simulam um sistema operacional utilizada para facilitar a integração e reutilização de códigos em robótica.

- Posição do agente (Pacman) — $(x, y) \in \mathbb{R}^2$.
- Distância para os fantasmas — $(d_x, d_y) = (\Delta x, \Delta y) \in \mathbb{R}^2$.
- Ação a ser executada — $m \in \{Norte, Sul, Leste, Oeste, Parar\}$.
- Recompensa recebida do ambiente — $r \in \mathbb{R}$.

Para simular erros de sensoriamento, comuns em aplicações de robótica móvel, antes de enviar essas informações são inseridos erros Gaussianos tanto na posição do agente (Pacman), quanto nas distâncias para os fantasmas.

$$(x, y) = (x^*, y^*) + (\delta(0, \sigma_{pacman}), \delta(0, \sigma_{pacman})),$$

$$(x, y) = (\delta(x^*, \sigma_{pacman}), \delta(y^*, \sigma_{pacman})). \quad (3.32)$$

Sendo:

- $\delta(\mu, \sigma)$ uma função que gera um número aleatório baseado em uma gaussiana com média μ e desvio padrão σ ;
- σ_{pacman} o desvio padrão do erro inserido na posição do agente;
- (x^*, y^*) a posição exata do agente.

Analogamente a distância percebida para os fantasmas é:

$$(d_x, d_y) = (\delta(d_x^*, \sigma_{dist_fant}), \delta(d_y^*, \sigma_{dist_fant})). \quad (3.33)$$

A atuação tem um conjunto de valores possíveis tal que $m \in M = \{Norte, Sul, Leste, Oeste, Parar\}$. Por ela possuir um valor discreto e não numérico, não se insere um erro gaussiano gaussiano nela. Para simular erros de atuação, quando uma ação é recebida pela plataforma de simulação ela é executada de acordo com a função presente no algoritmo 2.

Algorithm 2 Erro na Atuação

```

1: procedure EXECUTAR_AÇÃO(ação_escolhida)
2:   numero_randomico  $\leftarrow$  random numero
3:   if numero_randomico  $>$  FATOR_DE_ACERTO then
4:     acao_randomica  $\leftarrow$  random ação  $\in$  Ações  $- \{açao\_escolhida\}
5:     return acao_randomica
6:   else
7:     return ação_escolhida$ 
```

Ou seja, a atuação tem uma chance $\nu_{atuaacao} = FATOR_DE_ACERTO$ de ser executada como esperado. Existe também uma probabilidade $1 - \nu_{atuaacao}$ de ser selecionada uma ação randômica diferente da escolhida.

A recompensa r é, por sua vez, enviada da plataforma de simulação para o sistema de aprendizagem. Nela não é inserida qualquer forma de erro e ela representa a diferença de pontuação entre a etapa atual e a anterior no jogo, ou seja, a pontuação recebida pela última ação executada.

3.4 Testes Realizados

Nessa seção é apresentado a configuração de cada teste realizado e analisado no capítulo 4, além dos parâmetros utilizados.

Foram realizados 4 testes diferentes, sendo eles em 2 mapas diferentes e com 2 configurações de comportamentos diferentes. Os objetivos por trás da escolha desses teste é ver como a aprendizagem é realizada para um número menor de comportamentos (mais simples) em mapas diferentes, um menor e um maior, e observar essas mesmas características para uma aprendizagem mais complexa, com mais comportamentos.

3.4.1 3 Comportamentos no mapa pequeno (Teste 1)

Esse primeiro teste foi feito no mapa pequeno, mostrado na figura 3.4, e foram utilizados apenas três comportamentos:

- parar;
- comer;
- fugir.

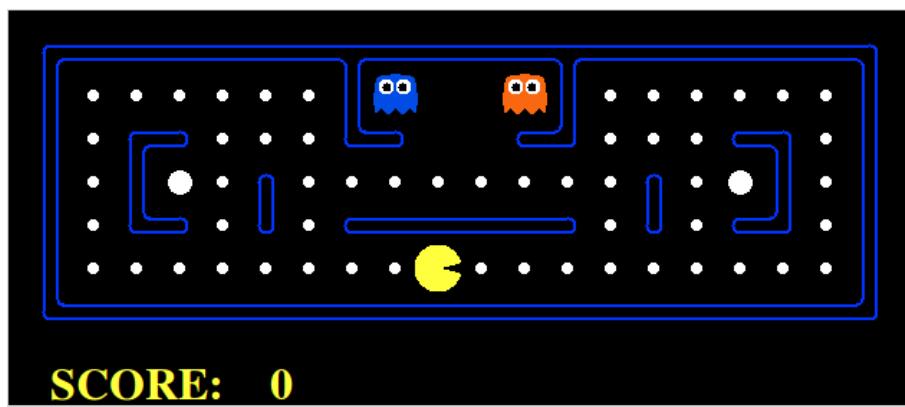


Figura 3.4: Mapa pequeno do jogo Pacman. Usado nos testes 1 e 3.

3.4.1.1 Parâmetros Utilizados

$$\begin{aligned}
\text{Num. partidas}^8 &= 3000; & \gamma^{13} &= 0.99; \\
\text{Num. treinos}^9 &= 2700; & \sigma_{pacman}^{14} &= 0.5; \\
\text{Num. Greedy}^{10} &= 1500; & \sigma_{dist_fant}^{15} &= 0.5; \\
\beta^{11} &= 1 - \left(\frac{n}{\text{Num.Greedy}} \right); & \nu_{atuacao}^{16} &= 0.9. \\
\alpha^{12} &= 0.002;
\end{aligned}$$

Sendo n o número da partida sendo jogada.

3.4.1.2 Comportamentos Utilizados

Os comportamentos utilizados foram: parar, comer e fugir. Ou seja:

$$\begin{aligned}
b \in B &= \{Ficar_Parado, Comer, Fugir\}, \\
s_b \in S_b &= \{Estado_Parar, Estado_Comer, Estado_Fugir\}.
\end{aligned}$$

Como explicado no tópico 3.2.3 agora são usados três modelos de seleção de ação motora:

$$P(M^t | S^t S_b^t M^{t-1} \pi_f) = \begin{cases} P(M^t | S^t [S_b^t = Estado_Parar] M^{t-1} \pi) \\ P(M^t | S^t [S_b^t = Estado_Comer] M^{t-1} \pi) \\ P(M^t | S^t [S_b^t = Estado_Fugir] M^{t-1} \pi) \end{cases}. \quad (3.34)$$

A seguir, cada modelo é apresentado de forma detalhada.

Parar

Esse é o comportamento mais simples, tendo como modelo de seleção de ação:

$$P(m^t | S^t [S_b^t = Estado_Parar] M^{t-1} \pi_f) = \begin{cases} 0.99 & \text{se } m^t = Parar \\ 0.0025 & \text{senão} \end{cases}. \quad (3.35)$$

Comer

Para esse comportamento utiliza-se um modelo mais complexo em que, para um dado estado de S^t se calcula uma ação com o algoritmo 3.

⁸Número de partidas executadas

⁹Número de partidas usadas para treinamento

¹⁰Número de partidas em que foi utilizada exploração gulosa. Como explicado no tópico 2.6.2

¹¹Fator de exploração gulosa utilizado.

¹²Fator de Aprendizagem

¹³Fator de Desconto

¹⁴Desvio padrão do erro gaussiano da posição recebida do Pacman

¹⁵Desvio padrão do erro gaussiano da distância recebida para os Fantasmas

¹⁶Fator de Acerto na movimentação do Pacman. Descrito no algoritmo 2 na seção 3.3

Algorithm 3 Escolher Ação Comer

```
1: procedure ESCOLHER_AÇÃO_COMER(estado)
2:   pos_comida  $\leftarrow$  posição comida_mais_proxima
3:   pos_pacman  $\leftarrow$  posição pacman
4:   dist_atual  $\leftarrow$  dist (pos_comida, pos_pacman)
5:   for Ação in Ações do
6:     nova_pos_pacman  $\leftarrow$  executa (pos_pacman, Ação)
7:     if dist (pos_comida, nova_pos_pacman)  $<$  dist_atual then
8:       return Ação                                 $\triangleright$  Retorna ação escolhida
9:   return Ação                                 $\triangleright$  Nenhuma ação boa, mas precisa retornar uma
```

Depois de calculada essa ação para um estado $s^t \in S^t$, utiliza-se um modelo parecido com o para o comportamento parado, apresentado na equação 3.36.

$$P(m^t | s^t [S_b^t = \text{Estado_Comer}] M^{t-1} \pi_f) = \begin{cases} 0.99 & \text{se } m^t = \text{Ação} \\ 0.0025 & \text{senão} \end{cases}. \quad (3.36)$$

Fugir

Para esse comportamento é utilizado um formato parecido com o de comer, mas com uma escolha de ação diferente. Para um dado estado de S^t se calcula uma ação usando o algoritmo 4.

Algorithm 4 Escolher Ação Fugir

```
1: procedure ESCOLHER_AÇÃO_FUGIR(estado)
2:   pos_fantasma  $\leftarrow$  posição fantasma_mais_proximo
3:   pos_pacman  $\leftarrow$  posição pacman
4:   dist_atual  $\leftarrow$  dist (pos_fantasma, pos_pacman)
5:   for Ação in Ações do
6:     nova_pos_pacman  $\leftarrow$  executa (pos_pacman, Ação)
7:     if dist (pos_fantasma, nova_pos_pacman)  $>$  dist_atual then
8:       return Ação                                 $\triangleright$  Retorna ação escolhida
9:   return Ação                                 $\triangleright$  Nenhuma ação boa, mas precisa retornar uma
```

Depois de calculado essa ação para um estado $s^t \in S^t$, assim como para o comportamento anterior, utiliza-se um modelo parecido com o modelo utilizado pelo comportamento parado.

$$P(m^t | s^t [S_b^t = \text{Estado_Fugir}] M^{t-1} \pi_f) = \begin{cases} 0.99 & \text{se } m^t = \text{Ação} \\ 0.0025 & \text{senão} \end{cases} \quad (3.37)$$

3.4.1.3 Vetor de Características Utilizado

Bias

Essa é uma característica que sempre está presente nesse vetor. Ela tem um valor padrão igual a 1.0 e seu peso ω indica quão bom esse comportamento é, independente da situação atual.

$$f_1 = 1.0$$

Distância para Provável Comida mais Próxima

Essa característica é proporcional à distância até a posição mais próxima em que é provável existir uma comida. Ela pode ser obtida a partir do algoritmo 5.

Algorithm 5 Obter Característica Distancia Comida

```
1: procedure OBTERCARACTERÍSTICADISTANCIACOMIDA(probabilidades_estados)
2:   max_prob  $\leftarrow 0.0$ 
3:   for Posição in Posições do
4:     prob_comida  $\leftarrow$  probabilidade de comida (Posição)
5:     if prob_comida  $>$  max_prob then
6:       max_prob  $\leftarrow$  prob_comida
7:   for Posição in Posições do
8:     prob_comida  $\leftarrow$  probabilidade de comida (Posição)
9:     if prob_comida  $>$   $\frac{\text{max\_prob}}{2}$  then
10:      Prob_Posições append Posição
11:   pos_comida  $\leftarrow$  mais proxima posição  $\in$  Prob_Posições
12:   pos_pacman  $\leftarrow$  posição pacman
13:   return dist (pos_comida, pos_pacman)
```

Tendo um conjunto de probabilidades $a \in A$ para os estados:

$$f_2 = \frac{\text{ObterCaracteristicaDistanciaComida}(a)}{\text{área_mapa}}$$

Soma das Probabilidades de Fantasmas a menos de 4 Movimentos

Essa característica é obtida a partir da soma das probabilidades de haver fantasmas a 3 movimentos de distância, ou menos, podendo então ter valor no intervalo $[0, \text{num_fantasmas} \cdot 100\%]$. O valor dessa característica é obtida utilizando o algoritmo 6.

Algorithm 6 Obter Característica Probabilidades Fantasmas

```
1: procedure OBTERCARACTERÍSTICAPROBFANTASMAS(probabilidades_estados)
2:   total  $\leftarrow$  0.0
3:   for Posição in Posições do
4:     for Posição2 in Posições do
5:       if dist(Posição, Posição2)  $<$  4 then
6:         prob_pacman  $\leftarrow$  probabilidade pacman (Posição)
7:         for Fantasma in Fantasmas do
8:           prob_fantasma  $\leftarrow$  probabilidade Fantasma (Posição2)
9:           prob_normal  $\leftarrow$  probabilidade estar normal (Fantasma)       $\triangleright$  Não branco
10:          total  $\leftarrow$  total + prob_pacman · prob_fantasma · prob_normal
11:   return total
```

$$f_3 = \text{ObterCaracteristicaProbFantasmas}(a)$$

3.4.2 3 Comportamentos no mapa original (Teste 2)

Esse teste é realizado no mapa clássico do jogo, mostrado na figura 3.5, e nele são utilizados apenas três comportamentos:

- parar;
- comer;
- fugir.

3.4.2.1 Parâmetros Utilizados

$$\begin{aligned} \text{Num. partidas} &= 1000; & \gamma &= 0.99; \\ \text{Num. treinos} &= 700; & \sigma_{pacman} &= 0.5; \\ \text{Num. Greedy} &= 500; & \sigma_{dist_fant} &= 0.5; \\ \beta &= 1 - \left(\frac{n}{\text{Num.Greedy}} \right); & \nu_{atuaacao} &= 0.9. \\ \alpha &= 0.001; \end{aligned}$$

Sendo n o número da partida sendo jogada.

3.4.2.2 Comportamentos Utilizados

Os comportamentos utilizados foram os mesmos que para o teste anterior e estão descritos no tópico 3.4.1.2.

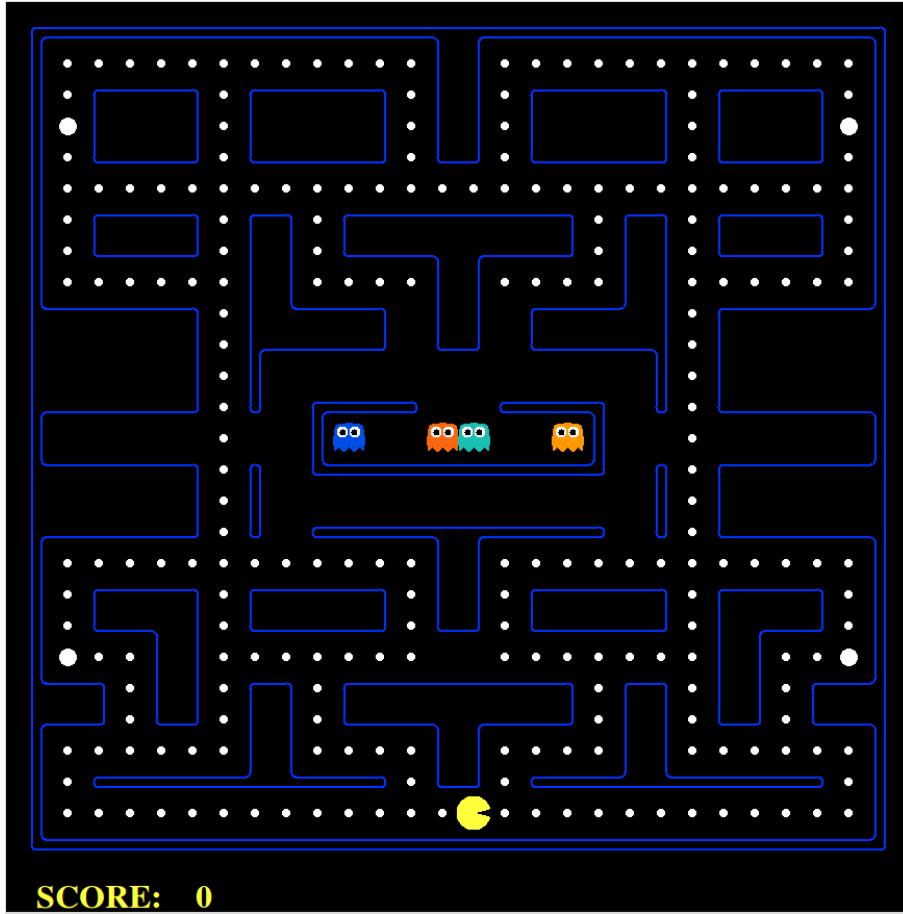


Figura 3.5: Mapa clássico do jogo Pacman. Usado nos testes 2 e 4.

3.4.2.3 Vetor de Características Utilizado

Para o vetor de características f , são utilizados parâmetros semelhantes aos do teste anterior, sendo o único diferente o uso da *Proximidade para Provável Comida mais Próxima* no lugar da *Distância para Provável Comida mais Próxima*.

Bias

$$f_1 = 1.0$$

Proximidade para Provável Comida mais Próxima

Essa é a única característica alterada do teste anterior para esse. Essa proximidade é utilizada, no lugar da distância, por ter um valor mais alto quando recompensas são recebidas e mais baixo no caso contrário, o que facilita a aprendizagem. Isso pode ser observado na equação 3.14, com um valor $f_i(a, b)$ maior, para um mesmo *erro*, tem-se uma variação maior de ω_i .

$$f_2 = \frac{1}{ObterCaracteristicaDistanciaComida(a)}$$

Soma das Probabilidades de Fantasmas a menos de 4 Movimentos

$$f_3 = ObterCaracteristicaProbFantasmas(a)$$

3.4.3 5 Comportamentos no mapa pequeno (Teste 3)

Esse teste é feito no mapa pequeno, mostrado na Figura 3.4, e nele são utilizados todos os cinco comportamentos:

- parar;
- comer;
- fugir;
- comer cápsula;
- caçar.

3.4.3.1 Parâmetros Utilizados

$$\begin{aligned} \text{Num. partidas} &= 3000; & \gamma &= 0.99; \\ \text{Num. treinos} &= 2700; & \sigma_{pacman} &= 0.5; \\ \text{Num. Greedy} &= 1500; & \sigma_{dist_fant} &= 0.5; \\ \beta &= 1 - \left(\frac{n}{\text{Num.Greedy}} \right); & \nu_{atuaacao} &= 0.9. \\ \alpha &= 0.002; \end{aligned}$$

Sendo n o número da partida sendo jogada.

3.4.3.2 Comportamentos Utilizados

Os comportamentos utilizados são: parar, comer, fugir, comer cápsula e caçar. Ou seja:

$$b \in B = \{Ficar_Parado, Comer, Fugir, Comer_Cápsula, Caçar\},$$

$$s_b \in S_b = \left\{ \begin{array}{l} Estado_Parar, \\ Estado_Comer, \\ Estado_Fugir, \\ Estado_Comer_Cápsula, \\ Estado_Caçar \end{array} \right\}.$$

Como explicado no tópico 3.2.3, e analogamente ao visto para três comportamentos, agora se têm cinco modelos de seleção de ação motora:

$$P(M^t | S^t S_b^t M^{t-1} \pi_f) = \begin{cases} P(M^t | S^t [S_b^t = Estado_Parar] M^{t-1} \pi) \\ P(M^t | S^t [S_b^t = Estado_Comer] M^{t-1} \pi) \\ P(M^t | S^t [S_b^t = Estado_Fugir] M^{t-1} \pi) \\ P(M^t | S^t [S_b^t = Estado_Comer_Cápsula] M^{t-1} \pi) \\ P(M^t | S^t [S_b^t = Estado_Caçar] M^{t-1} \pi) \end{cases}. \quad (3.38)$$

Os modelos para os estados *Estado_Parar*, *Estado_Comer* e *Estado_Fugir* foram os mesmo utilizados no primeiro experimento e estão descritos no tópico 3.4.1.2. Os outros dois modelos, para os estados de comportamento *Estado_Comer_Cápsula* e *Estado_Caçar*, estão descritos a seguir.

Comer Cápsula

O modelo para o estado *Estado_Comer_Cápsula* é análogo ao do estado *Estado_Comer*. Para um dado estado $s^t \in S^t$ calcula-se uma ação utilizando o Algoritmo 7.

Algorithm 7 Escolher Ação Comer Cápsula

```

1: procedure ESCOLHERAÇÃOCOMERCÁPSULA(estado)
2:   pos_capsula  $\leftarrow$  posição capsula_mais_proxima
3:   pos_pacman  $\leftarrow$  posição pacman
4:   dist_atual  $\leftarrow$  dist (pos_comida, pos_pacman)
5:   for Ação in Ações do
6:     nova_pos_pacman  $\leftarrow$  executa (pos_pacman, Ação)
7:     if dist (pos_capsula, nova_pos_pacman)  $<$  dist_atual then
8:       return Ação                                 $\triangleright$  Retorna ação escolhida
9:   return Ação                                 $\triangleright$  Nenhuma ação boa, mas precisa retornar uma

```

Depois de calculada essa ação para um estado $s^t \in S^t$, o modelo probabilístico da Equação 3.39 é utilizado.

$$P(m^t | s^t [S_b^t = Estado_Comer_Cápsula] M^{t-1} \pi_f) = \begin{cases} 0.99 & \text{se } m^t = Ação \\ 0.0025 & \text{senão} \end{cases}. \quad (3.39)$$

Caçar

O modelo para o estado *Estado_Caçar* é análogo ao do estado *Estado_Fugir*, exceto por ir em direção ao fantasma e não para longe dele. Para um dado estado $s^t \in S^t$ se calcula uma ação com o seguinte algoritmo:

Algorithm 8 Escolher Ação Caçar

```

1: procedure ESCOLHERAÇÃOCAÇAR(estado)
2:   pos_fantasma  $\leftarrow$  posição fantasma_mais_proximo
3:   pos_pacman  $\leftarrow$  posição pacman
4:   dist_atual  $\leftarrow$  dist (pos_fantasma, pos_pacman)
5:   for Ação in Ações do
6:     nova_pos_pacman  $\leftarrow$  executa (pos_pacman, Ação)
7:     if dist (pos_fantasma, nova_pos_pacman)  $<$  dist_atual then
8:       return Ação                                 $\triangleright$  Retorna ação escolhida
9:   return Ação                                 $\triangleright$  Nenhuma ação boa, mas precisa retornar uma

```

Uma vez calculada a ação para um estado $s^t \in S^t$, utiliza-se o seguinte modelo probabilístico:

$$P(m^t | s^t [S_b = \text{Estado_Caçar}] M^{t-1} \pi_f) = \begin{cases} 0.99 & \text{se } m^t = \text{Ação} \\ 0.0025 & \text{senão} \end{cases}. \quad (3.40)$$

3.4.3.3 Vetor de Características Utilizado

O vetor de características f para esse experimento tem sete parâmetros, sendo 3 deles os usados no experimento anterior (3.4.2) *Bias*, *Proximidade para Provável Comida mais Próxima* e *Soma das Probabilidades de Fantasmas a menos de 4 Movimentos*. Os outros quatro são:

- *Proximidade para Provável Cápsula mais Próxima*;
- *Probabilidade de ainda Existir uma Cápsula*;
- *Probabilidade de Existir Fantasma Branco*;
- *Soma das Probabilidades de Fantasmas Brancos a menos de 4 Movimentos*.

Cada uma dessas características está descrita a seguir.

Bias

$$f_1 = 1.0$$

Proximidade para Provável Comida mais Próxima

$$f_2 = \frac{1}{\text{ObterCaracteristicaDistanciaComida}(a)}$$

Proximidade para Provável Cápsula mais Próxima

Essa característica indica qual a proximidade até a posição mais próxima em que é provável existir uma cápsula. Ela pode ser obtida a partir do algoritmo 9, que é análogo ao algoritmo 5 do tópico 3.4.1.3.

Algorithm 9 Obter Característica Distancia Cápsula

```
1: procedure OBTERCARACTERÍSTICADISTANCIACÁPSULA(probabilidades_ estados)
2:   max_prob  $\leftarrow 0.0$ 
3:   for Posição in Posições do
4:     prob_capsula  $\leftarrow$  probabilidade de capsula (Posição)
5:     if prob_capsula  $>$  max_prob then
6:       max_prob  $\leftarrow$  prob_capsula
7:   for Posição in Posições do
8:     prob_capsula  $\leftarrow$  probabilidade de capsula (Posição)
9:     if prob_capsula  $>$   $\frac{\text{max\_prob}}{2}$  then
10:      Prob_Posições append Posição
11:   pos_capsula  $\leftarrow$  mais proxima posição  $\in$  Prob_Posições
12:   pos_pacman  $\leftarrow$  posição pacman
13:   return dist (pos_capsula, pos_pacman)
```

Tendo um conjunto de probabilidades $a \in A$ para os estados:

$$f_3 = \frac{1}{\text{ObterCaracteristicaDistanciaCapsula}(a)}$$

Probabilidade de ainda Existir uma Cápsula

Essa característica é obtida a partir da probabilidade de ainda existir uma cápsula no mapa.

$$f_4 = P(\text{capsula} | a)$$

Probabilidade de Existir Fantasma Branco

Essa característica é obtida a partir da probabilidade de existir um fantasma branco no ambiente. Um fantasma fica branco após se pegar uma cápsula e eles podem ser comidos pelo Pac-Man.

$$f_5 = P(\text{fantasma_branco} | a)$$

Soma das Probabilidades de Fantasmas a menos de 4 Movimentos

$$f_6 = \text{ObterCaracteristicaProbFantasmas}(a)$$

Soma das Probabilidades de Fantasmas Brancos a menos de 4 Movimentos

Essa característica é obtida a partir da soma das probabilidades de haver fantasmas a 3 movimentos de distância, ou menos, podendo então ter valor entre $[0, num_fantasmas \cdot 100\%]$. Ela pode ser obtida a partir do Algoritmo 10.

Algorithm 10 Obter Característica Probabilidades Fantasmas Brancos

```
1: procedure OBTERCARACTERÍSTICAPROBFANTASMASBRANCOS(probabilidades_estados)
2:   total  $\leftarrow 0.0$ 
3:   for Posição in Posições do
4:     for Posição2 in Posições do
5:       if dist(Posição, Posição2)  $< 4$  then
6:         prob_pacman  $\leftarrow$  probabilidade pacman (Posição)
7:         for Fantasma in Fantasmas do
8:           prob_fantasma  $\leftarrow$  probabilidade Fantasma (Posição2)
9:           prob_branco  $\leftarrow$  probabilidade estar branco (Fantasma)
10:          total  $\leftarrow$  total + prob_pacman  $\cdot$  prob_fantasma  $\cdot$  prob_branco
11:    return total
```

$$f_7 = \text{ObterCaracteristicaProbFantasmasBrancos}(a)$$

3.4.4 5 Comportamentos no mapa original (Teste 4)

Esse teste é feito no mapa clássico do jogo, apresentado na Figura 3.5, e nele, como no teste anterior (tópico 3.4.3), são utilizados todos os cinco comportamentos:

- parar;
- comer;
- fugir;
- comer cápsula;
- caçar.

3.4.4.1 Parâmetros Utilizados

$$\begin{aligned}
\text{Num. partidas} &= 1000; & \gamma &= 0.99; \\
\text{Num. treinos} &= 700; & \sigma_{pacman} &= 0.5; \\
\text{Num. Greedy} &= 500; & \sigma_{dist_fant} &= 0.5; \\
\beta &= 1 - \left(\frac{n}{\text{Num.Greedy}} \right); & \nu_{atuacao} &= 0.9. \\
\alpha &= 0.001;
\end{aligned}$$

Sendo n o número da partida sendo jogada.

3.4.4.2 Comportamentos Utilizados

Os comportamentos utilizados são os mesmos do experimento anterior (3.4.3), descritos no tópico 3.4.3.2.

3.4.4.3 Vetor de Características Utilizado

O vetor de características f usado nesse experimento possui seis características, sendo cinco delas iguais às utilizadas no experimento anterior¹⁷. A última é uma combinação das duas restantes: *Proximidade para Provável Capsula mais Próxima* e *Probabilidade de ainda Existir uma Capsula*. O vetor possui, então, as seguintes características:

Bias

$$f_1 = 1.0$$

Proximidade para Provável Comida mais Próxima

$$f_2 = \frac{1}{\text{ObterCaracteristicaDistanciaComida}(a)}$$

Probabilidade e Proximidade para Provável Cápsula mais Próxima

Essa é a característica nova inserida para esse experimento e simplifica as duas características anteriores *Proximidade para Provável Capsula mais Próxima* e *Probabilidade de ainda Existir uma Capsula*, sendo o produto das duas.

$$f_3 = \frac{P(\text{capsula} | a)}{\text{ObterCaracteristicaDistanciaCapsula}(a)}$$

¹⁷Essas características estão melhor descritas no tópico 3.4.3.3.

Probabilidade de Existir Fantasma Branco

$$f_4 = P(fantasma_branco \mid a)$$

Soma das Probabilidades de Fantasmas a menos de 4 Movimentos

$$f_5 = ObterCaracteristicaProbFantasmas(a)$$

Soma das Probabilidades de Fantasmas Brancos a menos de 4 Movimentos

$$f_6 = ObterCaracteristicaProbFantasmasBrancos(a)$$

Capítulo 4

Análise de Dados

“I used to want to change the world. Now I’m open to letting it change me.” – Po Bronson

4.1 Introdução

Nesse capítulo são apresentados brevemente os experimentos realizados, mais detalhadamente descritos na seção 3.4. São também apresentados seus resultados e analisados. Reiterando, os experimentos realizados são:

- Aprendizagem de 3 comportamentos em um mapa pequeno
- Aprendizagem de 3 comportamentos no mapa clássico
- Aprendizagem de 5 comportamentos em um mapa pequeno
- Aprendizagem de 5 comportamentos no mapa clássico

4.2 3 Comportamentos no mapa pequeno (Teste 1)

Nesse experimento¹ apenas 3 comportamentos são utilizados, $B = \{Ficar_Parado, Comer, Fugir\}$, e é usado como vetor de características f , dado por:

$$\begin{aligned} Bias : f_1(a, u) &= 1.0 \\ Dist.Comida : f_2(a, u) &= ObterCaracteristicaDistanciaComida(a) \\ Prob.Fantasma : f_3(a, u) &= ObterCaracteristicaProbFantasmas(a) \end{aligned} \quad (4.1)$$

O treinamento é realizado ao longo de 2700 partidas, sendo que a exploração epsilon gulosa é executada até a partida 1500. Após terminado o treinamento, 300 partidas são utilizadas para avaliar o algoritmo treinado.

¹Os parâmetros e o setup desse experimento estão melhor descritos no tópico 3.4.1.

4.2.1 Resultados e Análise

A evolução da aprendizagem se dá a partir da mudança dos pesos ω_i ao longo do tempo, devido à sua relação com o comportamento selecionado via a Equação 3.29. Nas Figuras 4.1 e 4.2 são apresentados os gráficos² de como os valores desses pesos evoluem com o tempo, para cada um dos comportamentos.

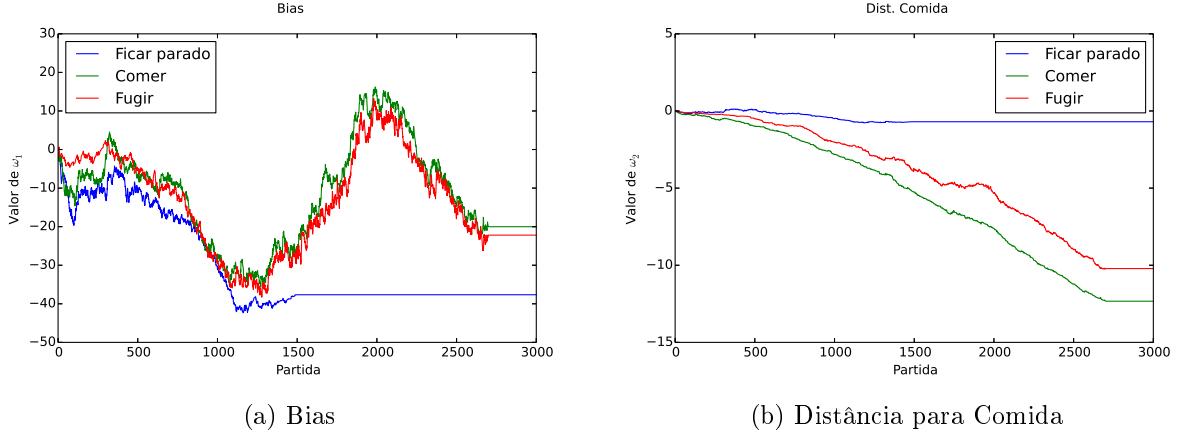


Figura 4.1: Evolução dos pesos ω_1 e ω_2

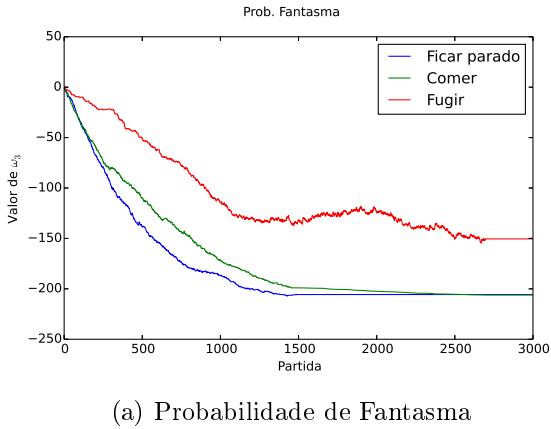


Figura 4.2: Evolução do peso ω_3

Percebe-se que o peso ω_3 , relativo à característica probabilidade de haver um fantasma normal perto, f_3 , é o que cresce mais rápido e que atinge valor absoluto final maior, entre -140 e -210, enquanto os outros dois ficam entre -50 e 20. Isso acontece porque f_3 sempre tem um valor alto quando o Pac-Man é morto, recebendo uma recompensa de -500 pontos. Como os pesos são atualizados seguindo as Equações 3.13 e 3.14, esse peso ω_3 tem grande variação a cada morte do Pac-Man. Isso mostra também que ele aprende que essa é uma situação ruim, dando um valor menor para estados em que ela tem valor mais alto.

Outra informação que pode se extrair desses gráficos é que estar perto de fantasmas é melhor (menos ruim) no caso de se escolher o comportamento fugir, do que para comer ou ficar parado.

²Todos os gráficos foram criados usando [31]

Analizando esses valores é possível ver que o comportamento *Ficar_Parado* nunca será executado³, o comportamento *Comer* será o normalmente utilizado e o comportamento *Fugir* será utilizado quando a probabilidade de haver um fantasma por perto for alta.

Como a seleção de comportamentos é aprendida com esse algoritmo, o comportamento escolhido a cada instante e o número de vezes que cada um é escolhido são fatores importantes para avaliá-lo. A nuvem de pontos presente na Figura 4.3 apresenta o número de vezes que cada comportamento é escolhido por partida.

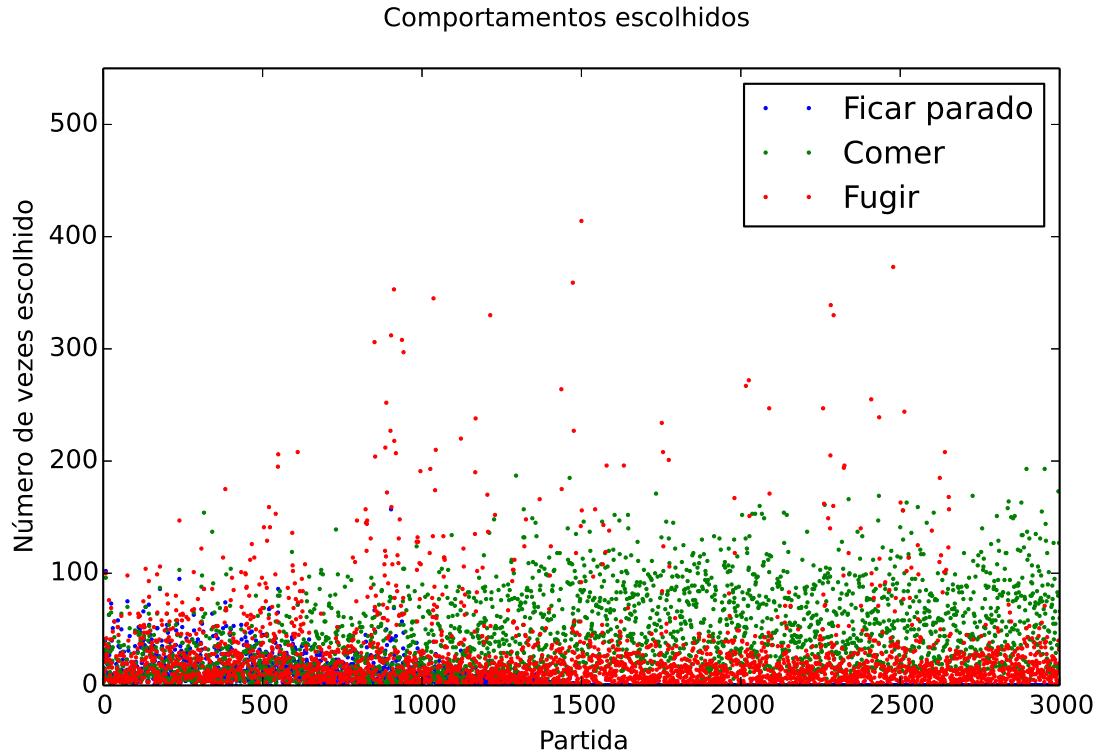


Figura 4.3: Escolha de comportamentos por partida.

Essa nuvem de dados permite avaliar como essa escolha de comportamento é feita ao longo do tempo. Para ter uma melhor visualização dos dados, é possível aproximar um polinômio para essa nuvem de pontos, utilizando o método dos mínimos quadrados. Para um polinômio de quarto grau essa curva fica como a descrita na Figura 4.4.

Inicialmente o comportamento *Comer* diminui em números de execução, enquanto *Fugir* aumenta. Isso acontece porque, inicialmente, não se sabe nada sobre o mundo, o agente “percebe” que quando ele executa o comportamento *Comer*, ele tem muito mais probabilidade de morrer. Após aproximadamente 500 partidas, esse comportamento começa a aumentar em número de execuções, enquanto o *Fugir* diminui.

Outro dado relevante para a análise é a pontuação obtida por jogo, que é o que se tenta maximizar durante a aprendizagem. A pontuação para cada partida é apresentada na Figura 4.5. Assim como para a evolução dos comportamentos escolhidos ao longo do tempo, esses dados

³Lembrando que, como pode ser visto no tópico 3.4.1, o valor de $f_2(A, B)$ estará sempre entre 0 e 1.

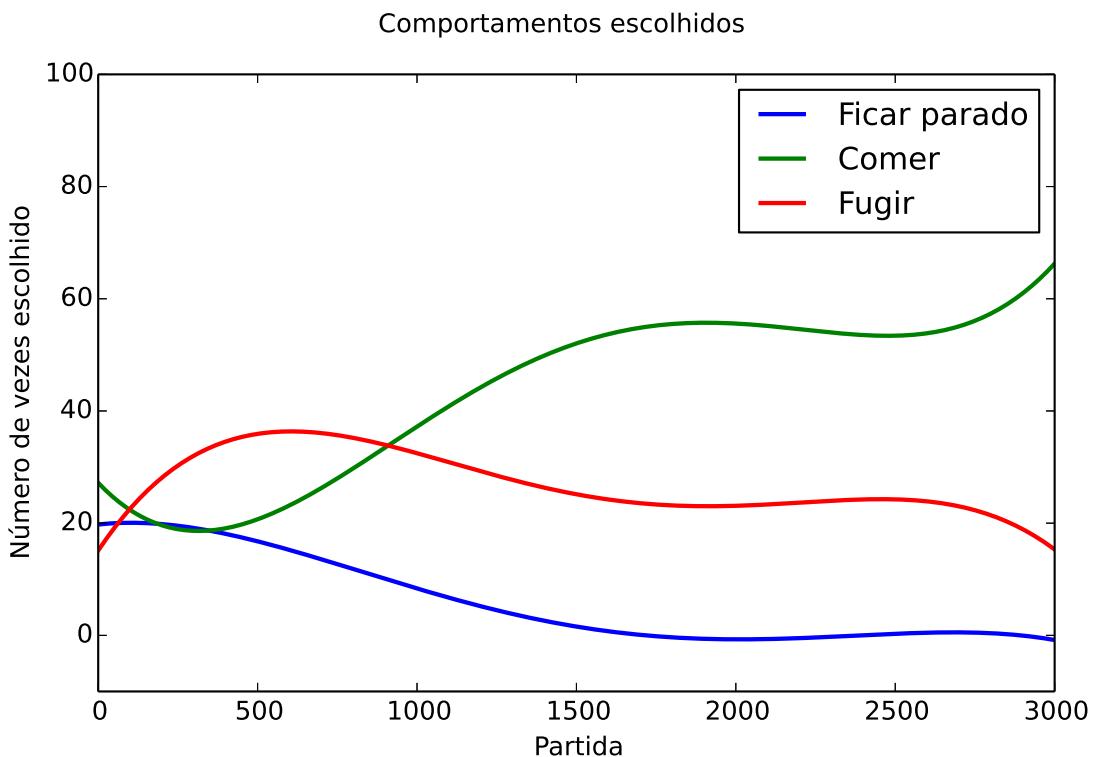


Figura 4.4: Polinômios referentes à escolha de comportamentos por partida.

também são aproximados por um polinômio de quarto grau utilizando o método dos mínimos quadrados.

É perceptível que essa função é crescente e que segue, aproximadamente, o formato da curva do número de execuções do comportamento *Comer*. Isso acontece pois, inicialmente, ao aumentar o número de execuções de *Fugir*, se diminui a pontuação geral⁴. A média de pontos feita, após a conclusão do treinamento é:

$$\text{média(score)} = 50.04.$$

Sendo a média de escolha dos comportamentos por partida pelo algoritmo de aprendizagem:

Tabela 4.1: Média da escolha dos comportamentos (Teste 1).

$\text{média(Ficar_Parado)}$	0.0
média(Comer)	61.77
média(Fugir)	17.21

⁴Se não pegar uma comida, se perde um ponto por movimento.

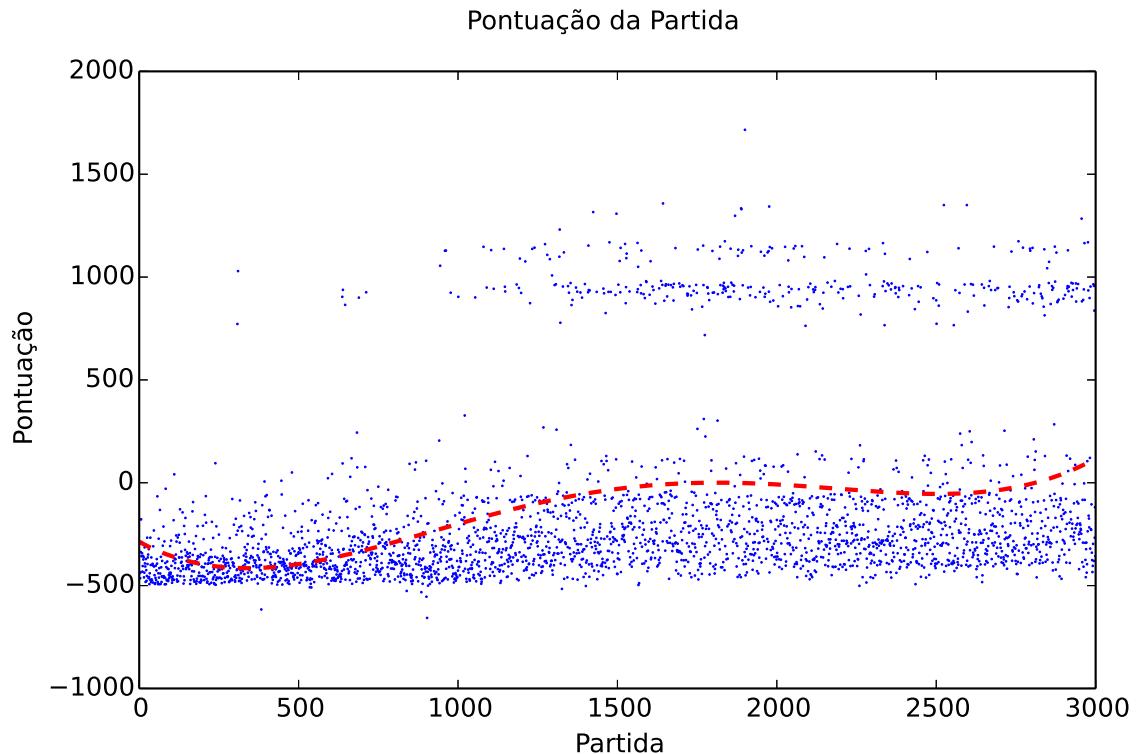


Figura 4.5: Pontuação por partida.

4.2.2 Discussão

O algoritmo teve o resultado esperado para o mapa e os comportamentos propostos, escolhendo sempre fugir, quando tendo alta probabilidade de perigo por perto, e comer no caso contrário.

4.3 3 Comportamentos no mapa original (Teste 2)

Assim como no teste anterior, para esse experimento⁵ são utilizados apenas 3 comportamentos, $B = \{Ficar_Parado, Comer, Fugir\}$. Usa-se também como vetor de características f , sendo ele:

$$Bias : f_1(a, u) = 1.0$$

$$Prox.Comida : f_2(a, u) = \frac{1}{ObterCaracteristicaDistanciaComida(a)} . \quad (4.2)$$

$$Prob.Fantasma : f_3(a, u) = ObterCaracteristicaProbFantasmas(a)$$

O treinamento foi realizado ao longo de 700 partidas, sendo que a exploração gulosa (*greedy exploration*) é executada até a partida 500. Após terminado o treinamento, 300 partidas são analisadas para avaliar o algoritmo treinado.

⁵Os parâmetros e a configuração desse experimento estão melhor descritos no tópico 3.4.2.

4.3.1 Resultados e Análise

Nas figuras a seguir, 4.6 e 4.7, são apresentados os gráficos de como os valores dos pesos ω_i evoluem com o passar das partidas, para cada um dos comportamentos.

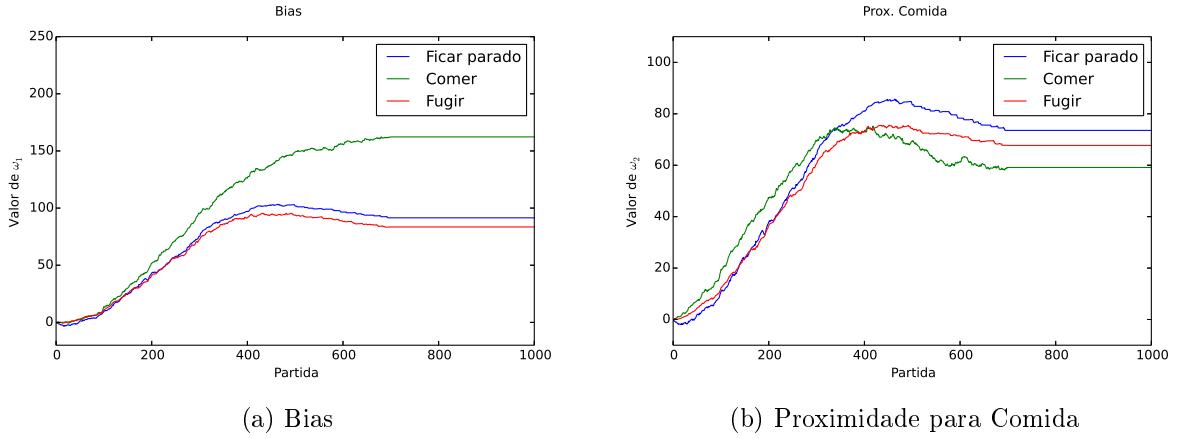


Figura 4.6: Evolução dos pesos ω_1 e ω_2

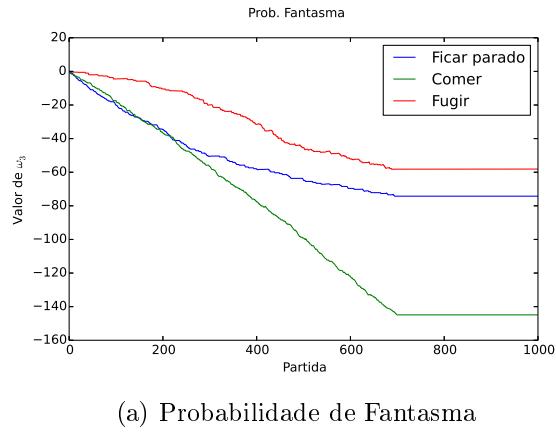


Figura 4.7: Evolução do peso ω_3

Para esse experimento todos os pesos, ω_1 , ω_2 e ω_3 , tem valores relativamente altos. Isso se dá, em boa parte, por esse ser um mapa “fácil”. Como se demora mais para morrer o peso da característica *Prob.Fantasma* cresce (ou diminui, já que é negativa) mais lentamente. Isso ocorre pois o mapa é grande e se tem menos chance de encontrar um fantasma. Além disso, como se morre menos vezes, os pesos das outras características tendem a aumentar mais do que diminuir.

Novamente, ele aprende que situações com valor alto para a característica f_3 , *Prob.Fantasma*, são ruins para se escolher o comportamento *Comer*, pois o peso ω_3 é negativo e com valor absoluto alto para esse comportamento. Também percebe-se pelo gráfico de ω_3 que estar perto de fantasmas é melhor no caso de se escolher o comportamento *fugir*, do que para *comer* ou *ficar parado*.

O fato de o peso ω_2 , da característica *Prox.Comida*, ser maior para ficar parado e menor para comer é contra-intuitivo. Isso ocorre pois os pesos devem se balancear pela equação de cálculo do erro 3.13 e, como *Comer* tem um peso ω_1 , da característica *Bias*, alto, seus outros pesos acabam

sendo afetados.

Pelo valor de ω_1 , percebe-se que o comportamento *Comer* é o normalmente utilizado. O comportamento *Fugir* é utilizado quando a probabilidade de haver um fantasma por perto é alta e o comportamento *Ficar_Parado* é escolhido raramente, quando se encontra numa situação intermediária.

O gráfico presente na Figura 4.8 apresenta o número de vezes que cada comportamentos é escolhido por partida, como uma nuvem de pontos.

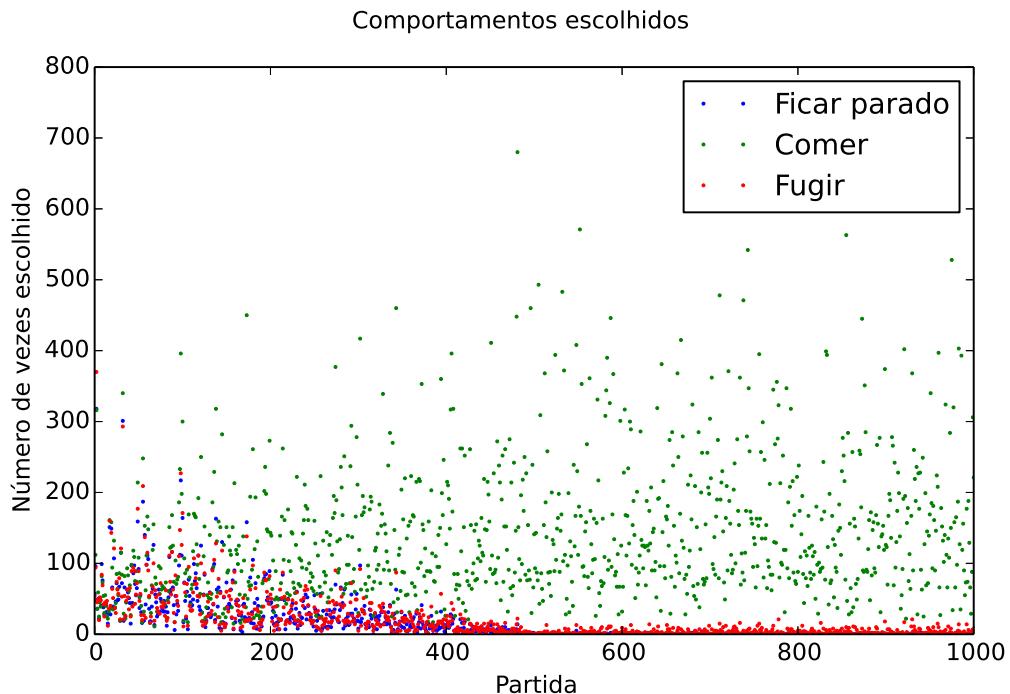


Figura 4.8: Escolha de comportamentos por partida.

Novamente, para uma visualização melhor, polinômios que representem essa nuvem de pontos é calculado, utilizando o método dos mínimos quadrados. A Figura 4.9 apresenta um polinômio de quarto grau assim obtido.

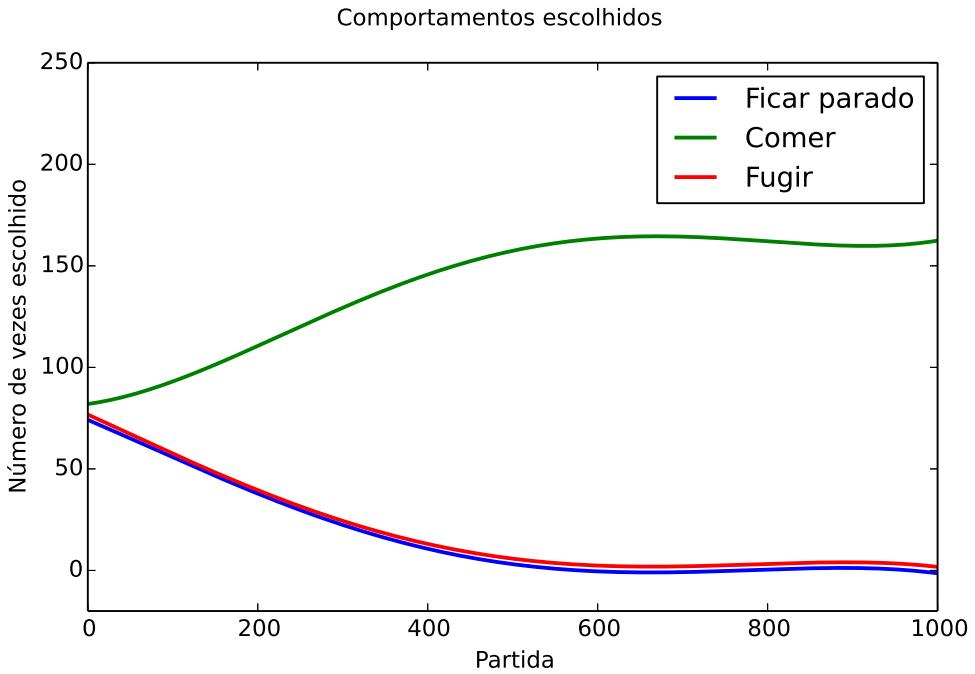


Figura 4.9: Polinômios referentes à escolha de comportamentos por partida.

Nesse mapa, o comportamento *Comer* nunca é considerado ruim e sobe em número de execuções rapidamente, enquanto os dois outros, *Fugir* e *Ficar_Parado*, diminuem em número de execuções até alcançarem valores relativamente constantes, perto de três e zero execuções por partida, respectivamente.

A pontuação para cada partida pode ser vista na Figura 4.10. Novamente aproxima-se esses dados por um polinômio de quarto grau, utilizando o método dos mínimos quadrado.

Essa função é crescente durante todo treinamento, tendendo a uma constante. A média de pontos feita, após a conclusão do treinamento é:

$$\text{média}(\text{score}) = 536.76.$$

Sendo a média de escolha pelo algoritmo de aprendizagem dos comportamentos:

Tabela 4.2: Média da escolha dos comportamentos (Teste 2).

$\text{média}(\text{Ficar_Parado})$	0.18
$\text{média}(\text{Comer})$	163.91
$\text{média}(\text{Fugir})$	3.32

4.3.2 Discussão

Percebe-se que o algoritmo, nesse experimento, deu preferência à ação de comer, sendo mais confiante. Isso pode ser explicado pela facilidade que o agente tem de receber pontos nesse ambiente, tendo menos encontros com fantasmas. Além disso, nesse mapa ele muitas vezes pega uma

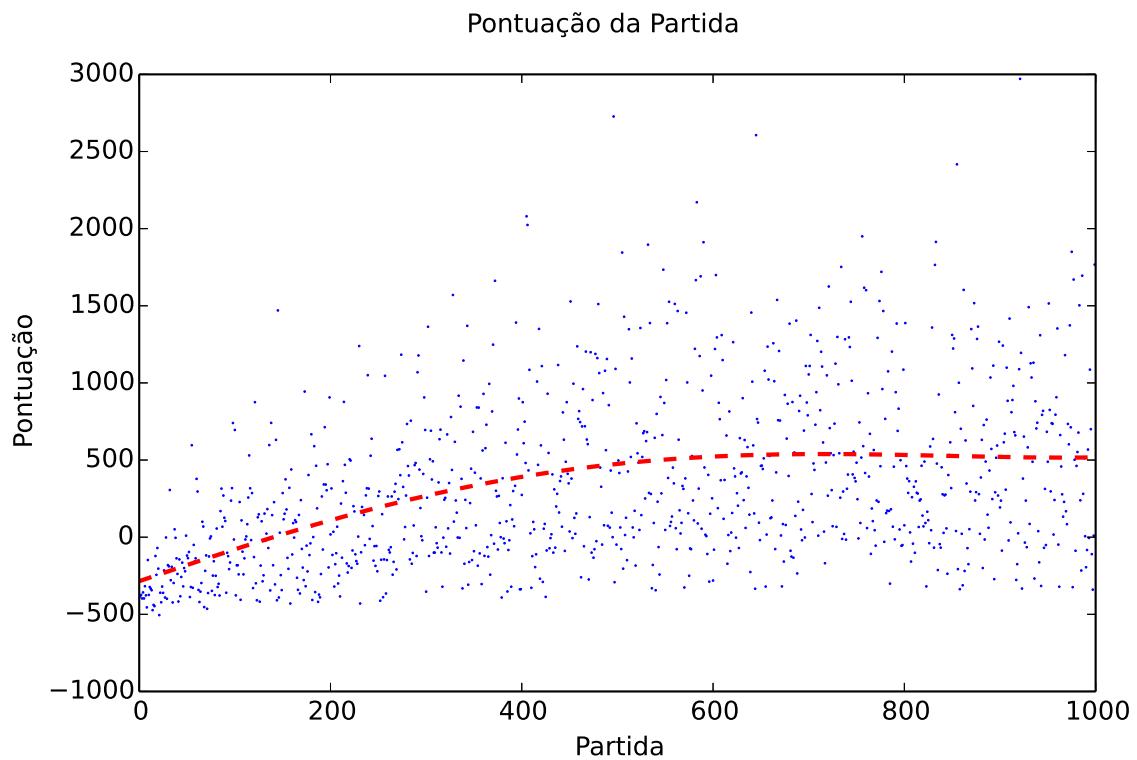


Figura 4.10: Pontuação por partida.

cápsula enquanto está executando o comportamento comer. Isso o permite comer fantasmas e receber pontos, valorizando essa ação.

4.4 5 Comportamentos no mapa pequeno (Teste 3)

Nesse experimento⁶, de forma diferente dos anteriores, 5 comportamentos são utilizados.

$$B = \{Ficar_Parado, Comer, Fugir, Comer_Cápsula, Caçar\}$$

⁶Os parâmetros e a configuração desse experimento estão melhor descritos no tópico 3.4.3.

e usa-se como vetor de características f :

$$\begin{aligned}
 Bias : \quad f_1(a, u) &= 1.0 \\
 Prox.Comida : \quad f_2(a, u) &= \frac{1}{ObterCaracteristicaDistanciaComida(a)} \\
 Prox.Capsula : \quad f_3(a, u) &= \frac{1}{ObterCaracteristicaDistanciaCapsula(a)} \\
 Prob.CapsulaExistir : \quad f_4(a, u) &= ObterCaracteristicaProbExistirCapsula(a) \\
 Prob.FantasmaBrancoExistir : \quad f_5(a, u) &= ObterCaracteristicaProbExistirFantasmaBranco(a) \\
 Prob.FantasmaPorPerto : \quad f_6(a, u) &= ObterCaracteristicaProbFantasmas(a) \\
 Prob.FantasmaBrancoPorPerto : \quad f_7(a, u) &= ObterCaracteristicaProbFantasmasBrancos(a)
 \end{aligned} \tag{4.3}$$

O treinamento é realizado ao longo de 2700 partidas, sendo que a exploração gulosa (*greedy exploration*) é executada até a partida 1500. Após terminado o treinamento, 300 partidas foram analisadas.

4.4.1 Resultados e Análise

Nas Figuras 4.11 a 4.14, os gráficos apresentam a evolução dos valores dos pesos ω_i com o tempo, para cada um dos comportamentos.

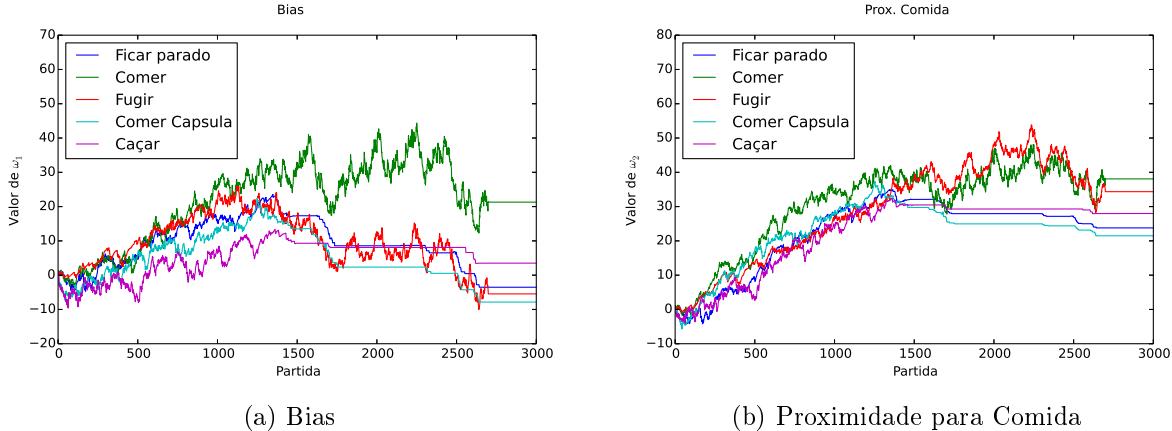


Figura 4.11: Evolução dos pesos ω_1 e ω_2 .

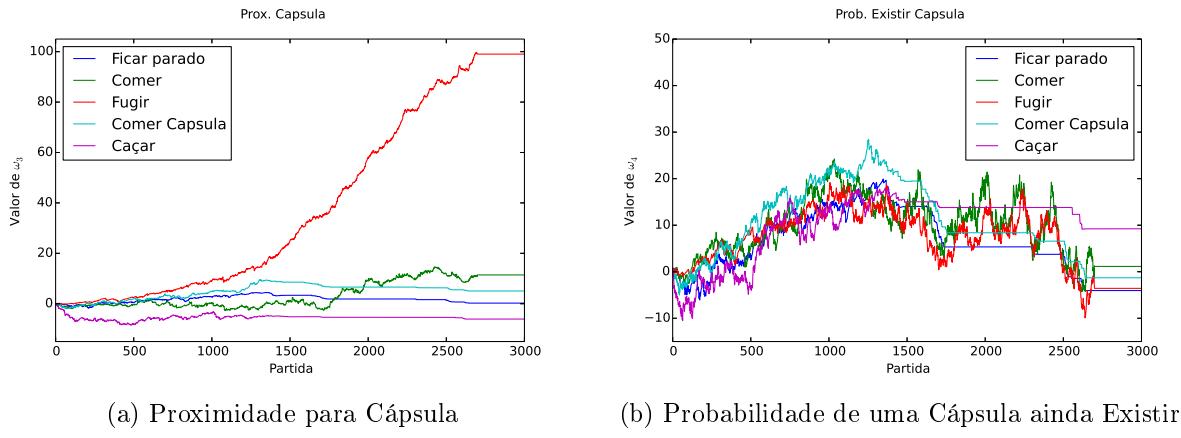


Figura 4.12: Evolução dos pesos ω_3 e ω_4 .

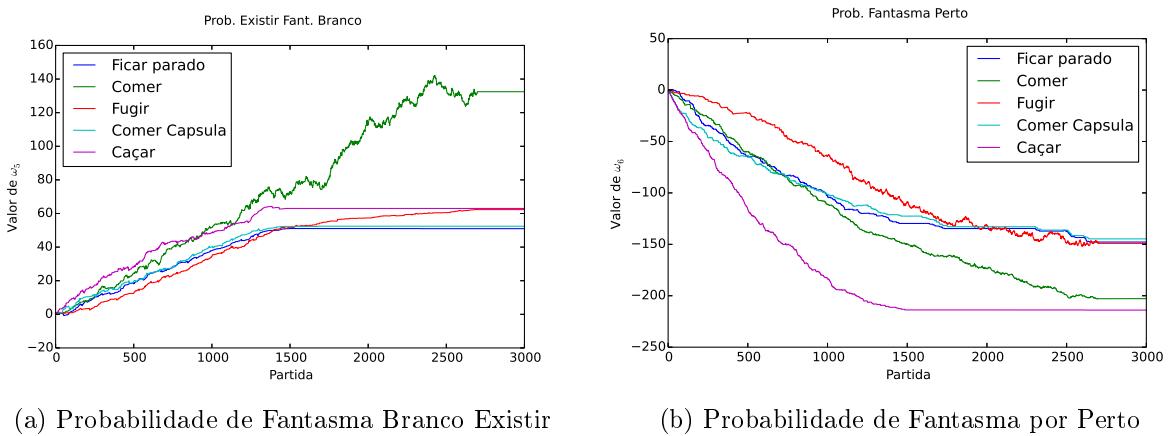


Figura 4.13: Evolução dos pesos ω_5 e ω_6 .

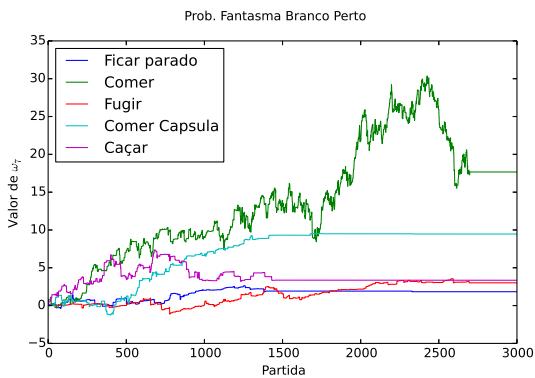


Figura 4.14: Evolução do peso ω_7 .

Para esse experimento, a análise dos pesos é mais complexa, pois existem agora 7 pesos a serem comparados. Algumas características ainda são simples de entender, por exemplo, ter um fantasma normal por perto é ruim quando se está comendo, e pior quando se está caçando (peso ω_6 negativo com valor absoluto alto). Por outro lado *Fugir* ou *Comer_Cápsula* são boas alternativas nesse

caso.

Para outras características, é mais complexo entender o comportamento de seus pesos. A característica *ProbFantasmaBrancoPerto*, por exemplo, tem peso maior para *Comer* do que para *Caçar*. Isso se dá, em boa parte, pela natureza probabilística do sistema. Como há uma probabilidade de esse fantasma não estar realmente branco, caçar pode não ser uma solução tão boa quanto ignorar os fantasmas, escolhendo o comportamento *Comer*.

Ao criar um gráfico que relaciona o número de vezes que cada comportamentos é escolhido por partida para esse experimento, se obtém a nuvem exposta na Figura 4.15.

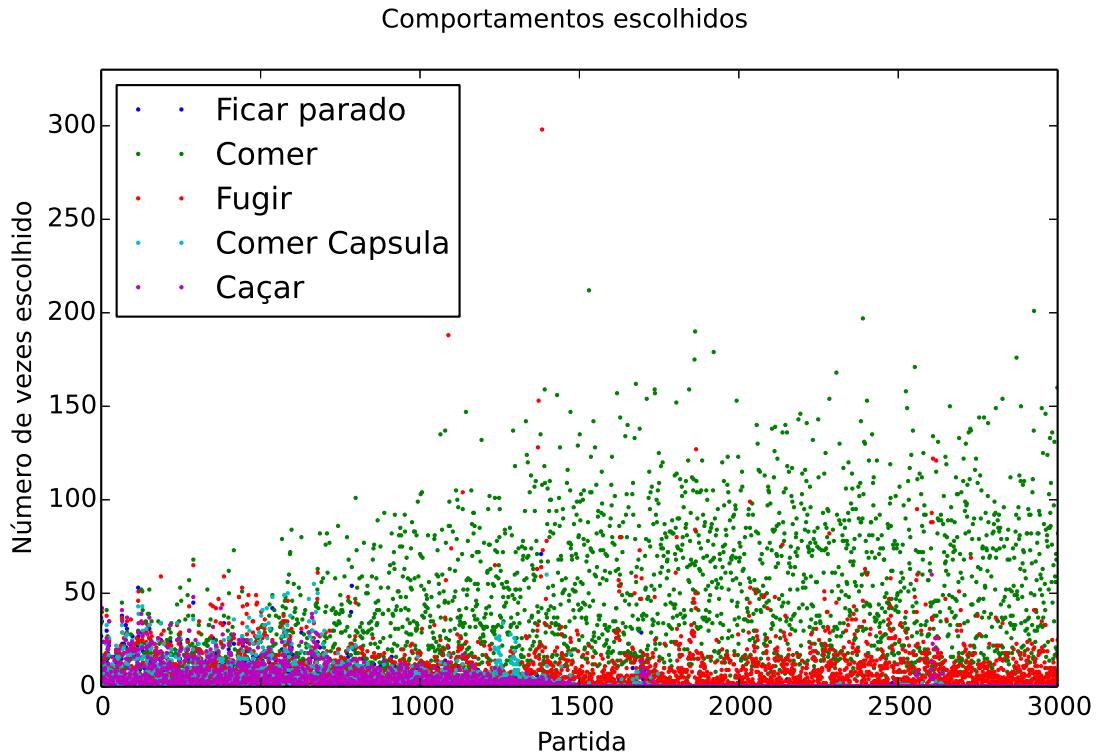


Figura 4.15: Escolha de comportamentos por partida.

Novamente, para ter uma visualização melhor desses dados calcula-se um polinômio que represente essa nuvem de pontos, utilizando o método dos mínimos quadrados. Para um polinômio de quarto grau essa curva é apresentada na figura 4.16.

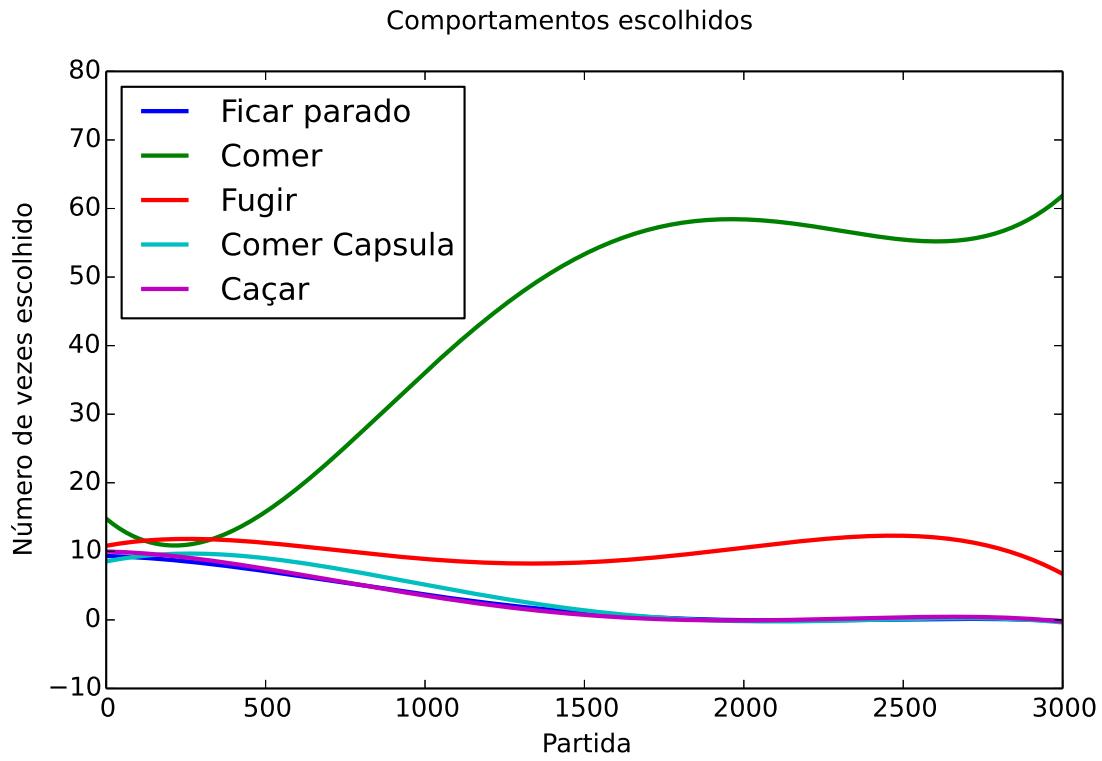


Figura 4.16: Polinômios referentes à escolha de comportamentos por partida.

Nesse experimento, o comportamento *Comer* novamente tem o maior número de execuções, sendo *Fugir* o único que continua sendo executado até o final. Isso é justificado por um fato não esperado, observado no experimento. O algoritmo aprendeu que muitas vezes em que o comportamento *Fugir* é executado, o agente foge em direção a Cápsula, não só evitando o fantasma, mas servindo de substituto ao comportamento *Comer_Capsula*. Por isso o peso ω_3 , da característica *Prox.Capsula*, é máximo para esse comportamento. Além disso, o comportamento *Comer*, não só vai atrás da comida, mas muitas vezes acaba indo em direção a um fantasma branco nesse mapa pequeno.

A pontuação para cada partida pode ser vista na Figura 4.17. Mais uma vez aproxima-se esses dados por um polinômio de quarto grau, utilizando o método dos mínimos quadrados.

Exceto pelo inicio do experimento, aproximadamente as primeiras trezentas partidas, a pontuação aumenta durante todo o treinamento. A média de pontos, após a conclusão do treinamento, é:

$$\text{média(score)} = 143.93.$$

Temos como média do número de escolhas dos comportamentos por partida:

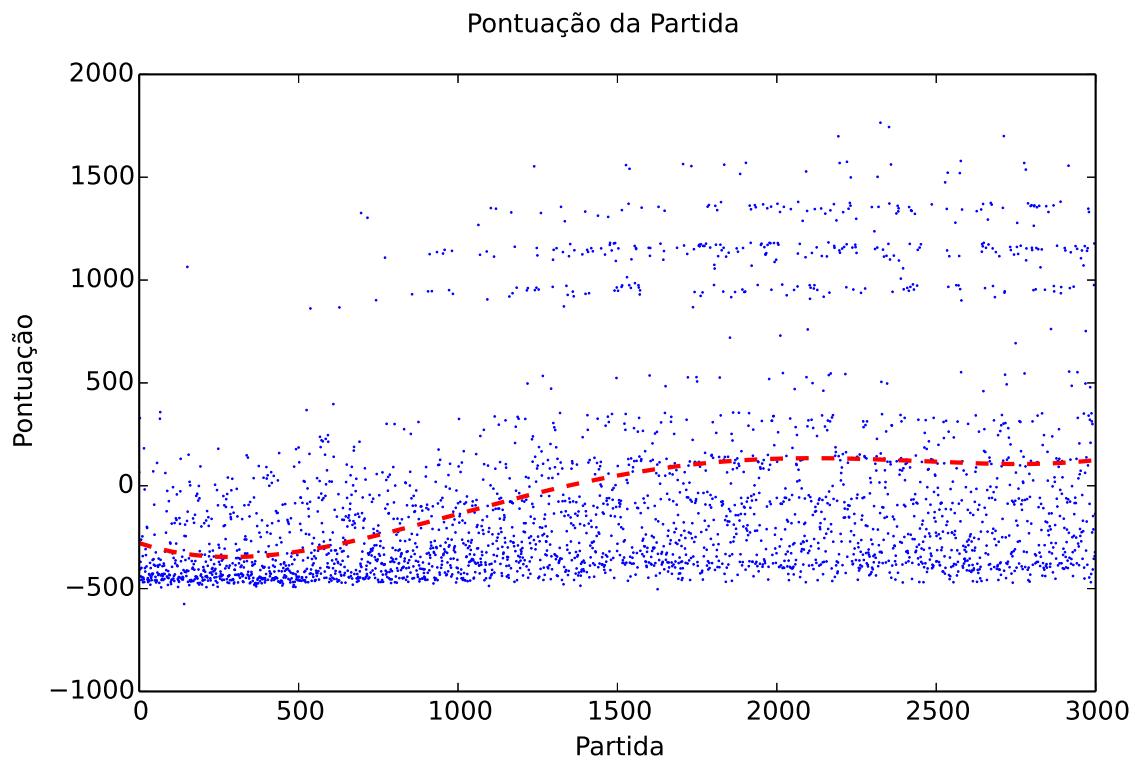


Figura 4.17: Pontuação por partida.

Tabela 4.3: Média da escolha dos comportamentos (Teste 3).

<i>média (Ficar_Parado)</i>	0.0
<i>média (Comer)</i>	59.10
<i>média (Fugir)</i>	8.35
<i>média (Comer_Cápsula)</i>	0.0
<i>média (Caçar)</i>	0.0

4.4.2 Discussão

O algoritmo, nesse experimento, após completo o treinamento, escolheu somente os comportamentos *Comer* e *Fugir*, mesmo em posse dos outros. Isso é explicado, em parte, devido a particularidades encontradas no mapa, que não eram esperadas a princípio, e em parte pela natureza probabilística do sistema, o que diminui a certeza de os fantasmas estão brancos.

Outro fato que explica a ausência de escolha do comportamento *Caçar* é a dificuldade do algoritmo de superar máximos locais. Devido a ele rapidamente aprender que estar perto de um fantasma é algo negativo, ele tem dificuldade de aprender a utilizar esse comportamento que o leva em direção a um fantasma.

4.5 5 Comportamentos no mapa original (Teste 4)

Nesse experimento⁷, como no anterior, foram utilizados 5 comportamentos.

$$B = \{Ficar_Parado, Comer, Fugir, Comer_Cápsula, Caçar\}$$

e como vetor de características f :

$$Bias : f_1(a, u) = 1.0$$

$$Prox.Comida : f_2(a, u) = \frac{1}{ObterCaracteristicaDistanciaComida(a)}$$

$$Prob.eProx.Capsula : f_3(a, u) = \frac{ObterCaracteristicaProbExistirCapsula(a)}{ObterCaracteristicaDistanciaCapsula(a)}$$

Prob.FantasmaBracoExistir :

$$f_4(a, u) = ObterCaracteristicaProbExistirFantasmaBranco(a)$$

$$Prob.FantasmaPorPerto : f_5(a, u) = ObterCaracteristicaProbFantasmas(a)$$

Prob.FantasmaBracoPorPerto :

$$f_6(a, u) = ObterCaracteristicaProbFantasmasBrancos(a)$$

(4.4)

O treinamento é realizado ao longo de 700 partidas, sendo que a exploração gulosa (*greedy exploration*) é executada até a partida 500. Após terminado o treinamento 300 partidas foram utilizadas para avaliar o algoritmo treinado.

4.5.1 Resultados e Análise

Nas figuras 4.18 a 4.20, os gráficos apresentam a evolução dos pesos ω_i com o tempo, para cada um dos comportamentos.

Nesse experimento existem 6 pesos diferentes e, em posse deles, percebe-se que os pesos da *Bias* alcançam os maiores valores, superando o valor absoluto do peso de *Prob.FantasmaPerto*. Isso se deve a esse mapa ser mais “fácil”, o que faz com que todos os comportamentos tenham valor maior para essa característica, que indica quão bom um comportamento é, independente da situação atual, como foi explicado no tópico 3.4.1.3.

Assim como no teste 3, também com 5 comportamentos, há pesos com valores diferentes do esperado para a característica *Proximidade Cápsula*, mas dessa vez o comportamento que tem maior valor para ela é *Comer*. Isso ocorre pois, nesse mapa, *Comer*, quando escolhido próximo a uma cápsula, geralmente leva o agente a comê-la.

Vemos também que os pesos ω_4 e ω_6 , *Prob.ExistirFantasmaBranco* e *Prob.FantasmaBracoPerto*, também têm maior valor para *Comer*. O que reflete que *Comer* nesse mapa, sendo indiferente aos fantasmas, é preferível a procurar ativamente por eles.

⁷Os parâmetros e a configuração desse experimento estão melhor descritos no tópico 3.4.4.

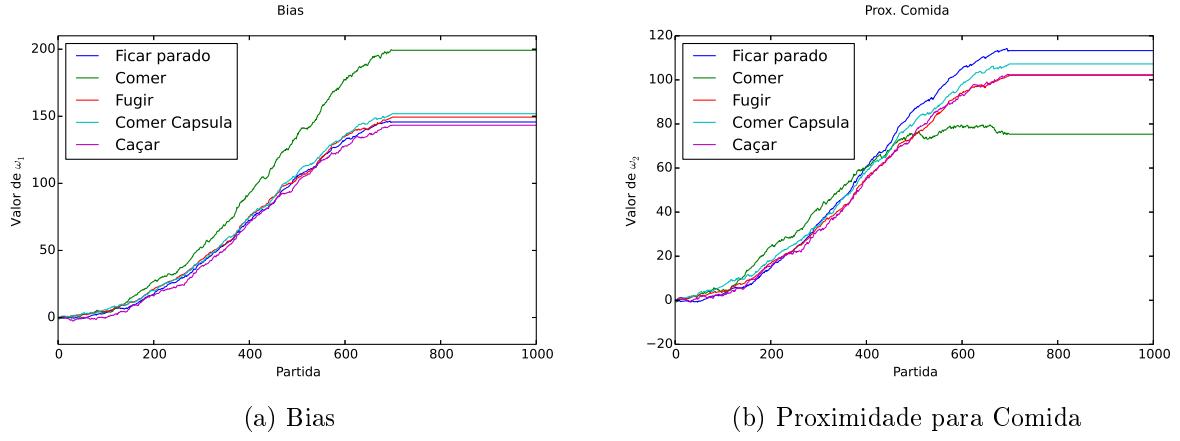


Figura 4.18: Evolução dos pesos ω_1 e ω_2 .

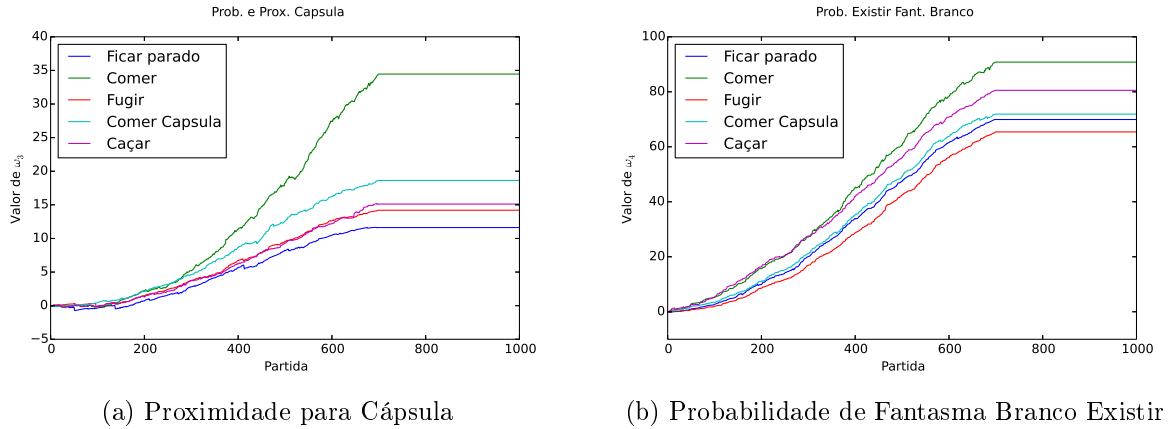
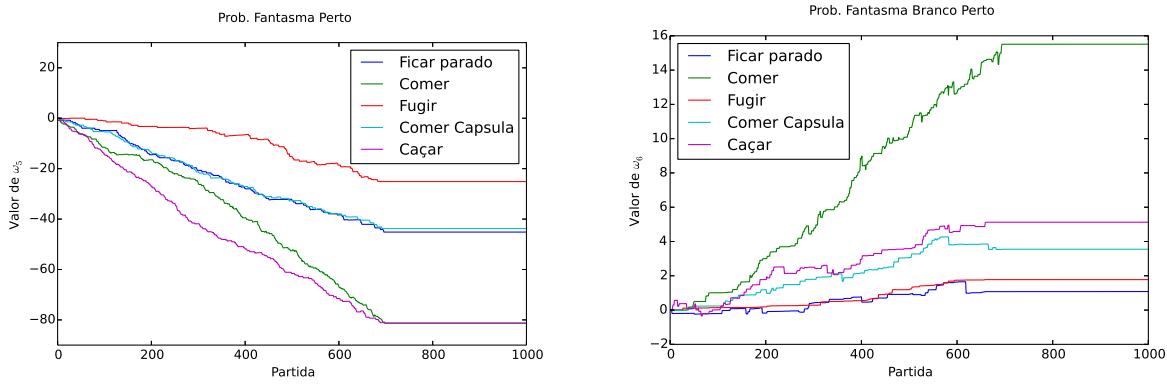


Figura 4.19: Evolução dos pesos ω_3 e ω_4 .

Como para os outros experimentos, o número de vezes que cada comportamento é escolhido por partida para esse experimento é apresentado na Figura 4.21.

Novamente, para ter uma visualização melhor dos dados, calcula-se um polinômio que represente essa nuvem de pontos. Para um polinômio de quarto grau essa curva fica como a descrita na Figura 4.22.



(a) Probabilidade de Fantasma por Perto

(b) Probabilidade de Fantasma Branco por Perto

Figura 4.20: Evolução dos pesos ω_5 e ω_6 .

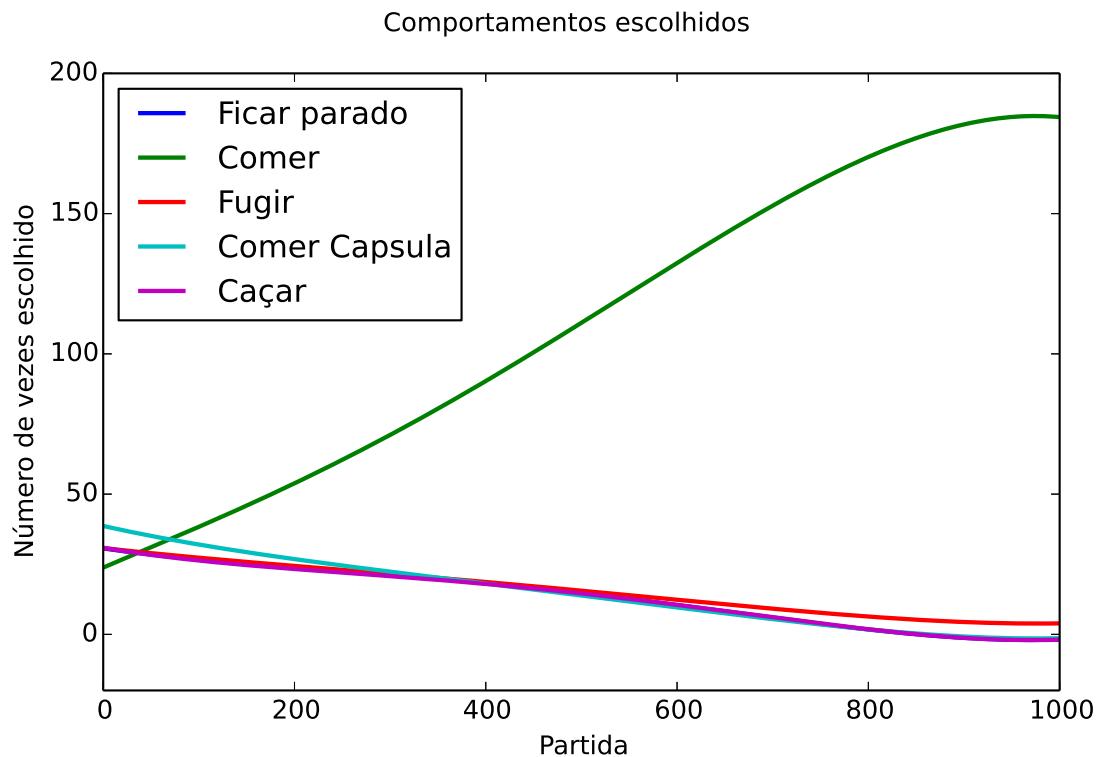


Figura 4.22: Polinômios referentes à escolha de comportamentos por partida.

Como para os outros experimentos, o comportamento *Comer* foi o mais escolhido, seguido por *Fugir*. Assim como para o outro experimento com 5 comportamentos, após completo o treinamento somente esses dois comportamentos são utilizados. Isso novamente se deve a uma característica do mapa, em geral quando o agente está a uma pequena distância de uma cápsula, o comportamento *Comer* o leva na mesma direção que o *Comer_Cápsula*.

A pontuação para cada partida pode ser vista na figura 4.23. Como anteriormente, aproxima-se esses dados por um polinômio de quarto grau.

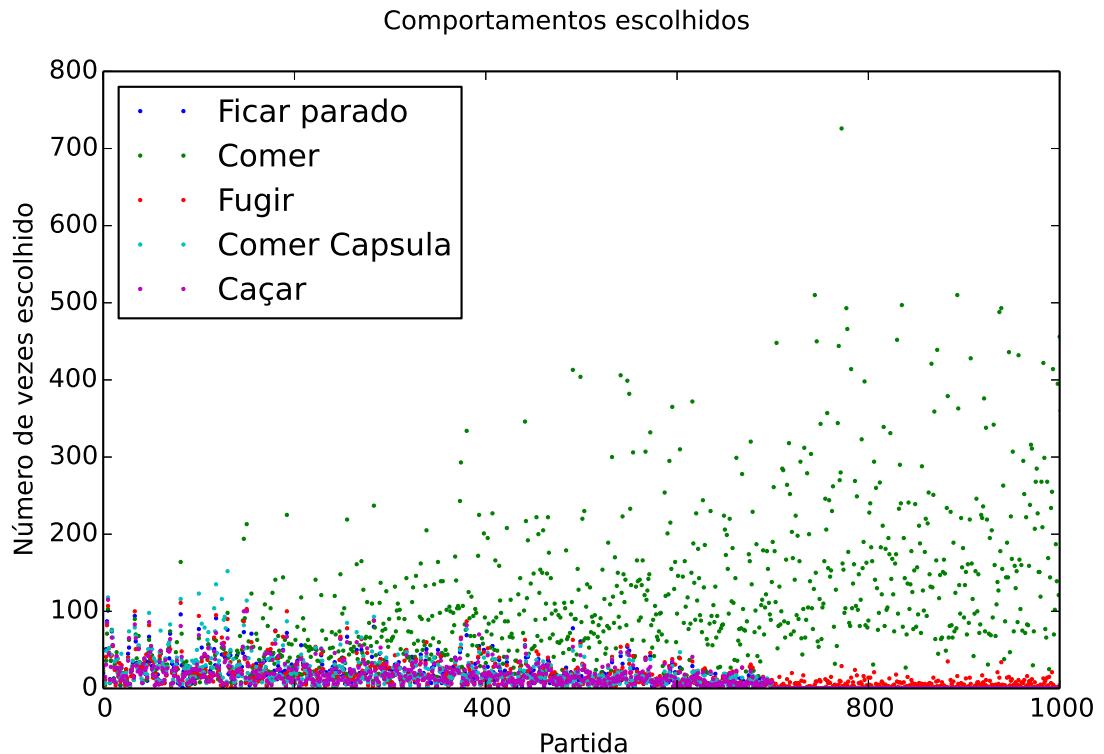


Figura 4.21: Escolha de comportamentos por partida.

Essa curva mostra uma melhora na pontuação ao longo de todo o treinamento. A média de pontos, após a conclusão do treinamento, é:

$$\text{média}(\text{score}) = 619.20,$$

sendo a média de escolha dos comportamentos por partida:

Tabela 4.4: Média da escolha dos comportamentos (Teste 4).

$\text{média}(Ficar_Parado)$	0.0
$\text{média}(Comer)$	181.04
$\text{média}(Fugir)$	5.29
$\text{média}(Comer_Cápsula)$	0.0
$\text{média}(Caçar)$	0.0

4.5.2 Discussão

Assim como para o teste anterior, o algoritmo escolheu somente os comportamentos *Comer* e *Fugir*, após o treinamento estar completo. Isso é explicado em parte devido a particularidades encontradas no mapa, que não eram esperadas a princípio, e em parte pela natureza probabilística do sistema, o que dificulta se ter uma certeza de os fantasmas estarem brancos.

Observa-se também que existe uma dificuldade do algoritmo de sair de um máximo local para

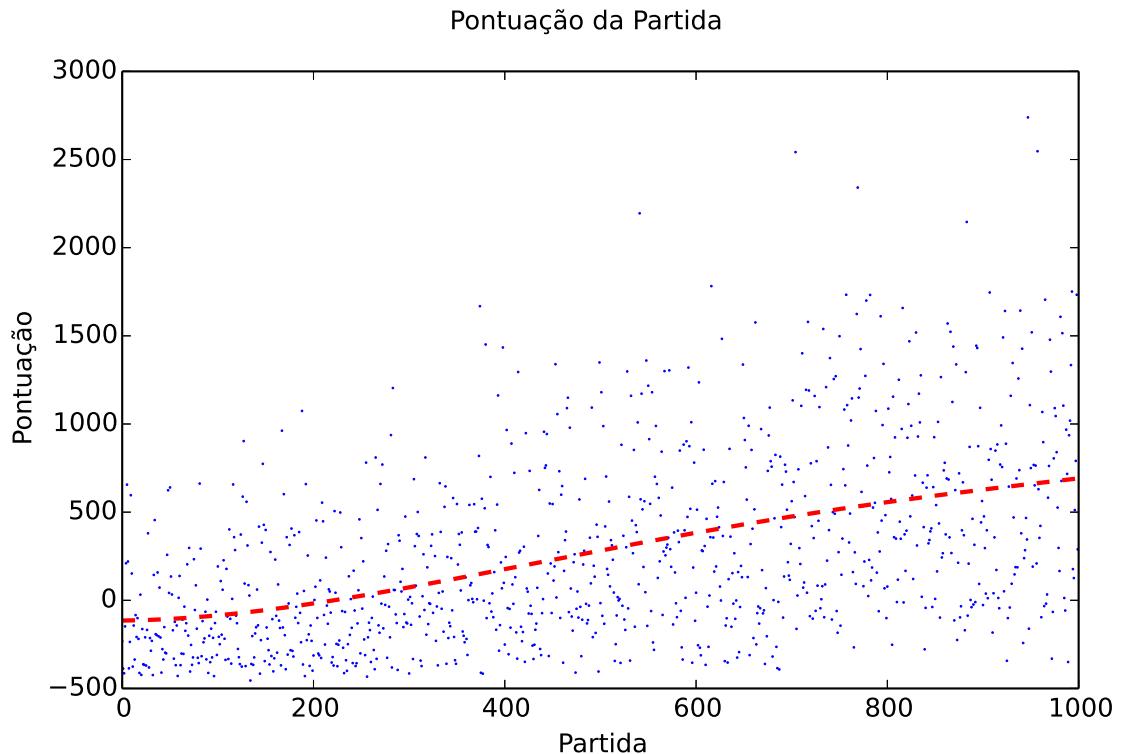


Figura 4.23: Pontuação por partida.

alcançar um máximo global. Pelo fato de ele inicialmente perceber que é ruim ter uma probabilidade alta de possuir fantasmas normais por perto, ele não consegue aprender a escolher o comportamento *Caçar* como seria desejado, mesmo para casos com alta probabilidade de os fantasmas estarem brancos.

4.6 Discussão Geral

O algoritmo, como esperado, teve comportamentos diferentes para diferentes mapas, comportamentos e parâmetros.

Alguns fatos inesperados foram observados, por exemplo, nos terceiro e quarto casos de teste, mesmo só executando, ao final, dois comportamentos, *Comer* e *Fugir*, os mesmos executados pelos dois primeiros experimentos, eles conseguiram uma média de pontos superior. Isso se deve a que uma característica, *Proximidade Cápsula*, que parecia ter pouca relevância para esses dois comportamentos, é mais importante que o esperado. Isso reforça a informação dada na Fundamentação Teórica, no tópico 2.6.1, que afirma que devemos escolher essas características com cuidado, ou podemos ter dois estados-comportamentos com valores parecidos e, consequentemente, valores de $Q(S, U)$ também parecidos, mas que são muito diferentes. Deve-se então escolher as características f_i com cuidado, testando várias combinações diferentes.

Deve-se, também, tomar cuidados com máximos locais, que podem dificultar o algoritmo a alcançar uma seleção de comportamentos ótima. Esse algoritmo pode inicialmente aprender que

uma situação é ruim e não conseguir superar essa percepção inicial.

O modelo obteve resultados satisfatórios, conseguindo fazer escolhas coerentes mesmo num sistema com alto erro de atuação e sensoriamento e sem ter qualquer informação previa de como os comportamentos influenciariam seu desempenho ou atuação. Mesmo com algumas características inesperadas encontradas, o algoritmo se comportou como desejado. Nos experimentos com o mapa clássico, por exemplo, ele aprende e valoriza mais o comportamento de comer, enquanto nos mapas pequenos ele é mais cauteloso, executando mais vezes o comportamento *Fugir*.

Capítulo 5

Conclusões

“You only live once, but if you do it right, once is enough.” – Mae West

Esse trabalho se propõe a desenvolver uma ferramenta para seleção de comportamentos num sistema parcialmente observável utilizando redes bayesianas e aprendizagem por reforço.

O algoritmo teve desempenho satisfatório e conseguiu aprender como utilizar os diferentes parâmetros para fazer boas escolhas num sistema estocástico. Ele apresentou, ainda, curvas de aprendizagem diferentes para mapas diferentes, como é esperado.

Os testes mostram, também, que esse algoritmo consegue aprender características específicas dos mapas. Um exemplo é a percepção que um ou outro comportamento, ao ser invocado em uma parte específica do mapa, consegue ganhos que não são inerentes a ele.

O algoritmo, no entanto, possui algumas limitações, como, por exemplo, sua dificuldade para sair de um máximo local, para certas circunstâncias, como a dificuldade de alterar uma informação que ele aprendeu como ruim. Outra limitação é ter de se escolher os parâmetros manualmente, de forma que eles consigam descrever bem o sistema, tendo valores linearmente separáveis para diferentes comportamentos.

De forma geral o algoritmo conseguiu aprender a utilizar os comportamentos de forma intuitiva e lógica, os combinando para realizar tarefas complexas.

5.1 Perspectivas Futuras

Uma proposta que pode ser feita para evoluir esse trabalho é a utilização de uma rede neural no lugar da função de ganho atualmente utilizada. Com uma rede neural, as características não teriam de ser escolhidos manualmente, embora fosse necessário escolher outros parâmetros.

Outro trabalho seria o teste de várias características diferentes utilizando esse mesmo sistema para tentar superar a limitação de máximo local encontrado nesse trabalho, o que impediu a aprendizagem de alguns comportamentos. Ou a implementação desse algoritmo em um sistema diferente, com suas próprias particularidades, para testar diferentes possíveis aplicações dele.

Um terceiro trabalho que pode ser feito é a criação de uma plataforma física em que esse sistema

poderia ser testado fora de simulação, dando suporte para os robôs não se danificarem durante o processo de aprendizagem.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] KOIKE, C. M. C. e C. *Bayesian Approach to Action Selection and Attention Focusing*. Tese (Doutorado) — Institut National Polytechnique De Grenoble, 2005.
- [2] FIéVET, C. *Les Robots: Que sais-je?* Paris, France: Presses universitaires de France, 2002.
- [3] MERRIAM-WEBSTER Online Dictionary. Dez 2014. Disponível em: <<http://www.merriam-webster.com/dictionary/robot>>.
- [4] THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. Cambridge, MA, USA: The MIT Press, 2005. ISBN 0262201623.
- [5] FREDA, L.; ORIOLO, G. Frontier-based probabilistic strategies for sensor-based exploration. In: *ICRA*. Barcelona, Spain: IEEE, 2005. p. 3881–3887.
- [6] YAMAUCHI, B. Frontier-based exploration using multiple robots. In: *Proceedings of the Second International Conference on Autonomous Agents*. New York, NY, USA: ACM, 1998. (AGENTS '98), p. 47–53. ISBN 0-89791-983-1. Disponível em: <<http://doi.acm.org/10.1145/280765.280773>>.
- [7] YAMAUCHI, B. A frontier-based approach for autonomous exploration. In: *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*. Washington, DC, USA: IEEE Computer Society, 1997. (CIRA '97), p. 146–151. ISBN 0-8186-8138-1. Disponível em: <<http://dl.acm.org/citation.cfm?id=523996.793157>>.
- [8] VAHRENKAMP, N. et al. Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '09)*. St. Louis, USA: IEEE, 2009.
- [9] NORRIS, J. R. *Markov chains*. Cambridge, United Kingdom: Cambridge University Press, 1998. I-XVI, 1-237 p. (Cambridge series in statistical and probabilistic mathematics). ISBN 978-0-521-48181-6.
- [10] PINEAU, J.; THRUN, S. *Hierarchical POMDP Decomposition for A Conversational Robot*. ICML Workshop on Hierarchy and Memory, 2001. Disponível em: <<http://citeseer.ist.psu.edu/454010.html>>.

- [11] LIDORIS, G.; WOLLHERR, D.; BUSS, M. Bayesian state estimation and behavior selection for autonomous robotic exploration in dynamic environments. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Nice, France: IEEE, 2008.
- [12] LIDORIS, G. *State Estimation, Planning, and Behavior Selection Under Uncertainty for Autonomous Robotic Exploration in Dynamic Environments*. Kassel University Press, 2011. ISBN 9783862190638. Disponível em: <<http://books.google.com.br/books?id=3PjJwKvQcnYC>>.
- [13] CALVET LAURENT, E.; Czellár, V. *Efficient Estimation of Learning Models*. Paris, France, fev. 2012. Mimeo, 2010. Disponível em: <<https://hal-hec.archives-ouvertes.fr/hal-00674226>>.
- [14] GUTMANN, M.; HYVÄRINEN, A. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In: *International Conference on Artificial Intelligence and Statistics*. [S.l.: s.n.], 2010. p. 297–304.
- [15] SUTTON, R. S. Learning to predict by the methods of temporal differences. *Mach. Learn.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 3, n. 1, p. 9–44, ago. 1988. ISSN 0885-6125. Disponível em: <<http://dx.doi.org/10.1023/A:1022633531479>>.
- [16] WENDEMUTH, A. Dynamics of temporal difference learning. International Joint Conferences on Artificial Intelligence, Magdeburg, Germany, p. 1107–1112, 2007.
- [17] SUTTON, R.; BARTO, A. *Reinforcement learning: An introduction*. Cambridge, MA: Cambridge Univ Press, 1998.
- [18] KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. P. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, v. 4, p. 237–285, 1996. Disponível em: <<http://people.csail.mit.edu/lpk/papers/rl-survey.ps>>.
- [19] RICHTER, S.; ABERDEEN, D.; YU, J. Natural actor-critic for road traffic optimization. In: Schölkopf, B.; PLATT, J.; HOFMANN, T. (Ed.). *Advances in Neural Information Processing Systems 19*. Cambridge, MA: MIT Press, 2007.
- [20] BHATNAGAR, S. et al. Incremental natural actor-critic algorithms. In: PLATT, J. C. et al. (Ed.). *NIPS*. Quebec, Canada: NIPS, Inc., 2007.
- [21] KONDA, V. R.; TSITSIKLIS, J. N. Actor-critic algorithms. In: *SIAM JOURNAL ON CONTROL AND OPTIMIZATION*. Cambridge, MA: MIT Press, 2001. p. 1008–1014.
- [22] GASKETT, C. *Q-Learning for Robot Control*. Tese (Doutorado) — The Australian National University, 2002.
- [23] GHEREGA, A.; RADULESCU, M.; UDREA, M. A q-learning approach to decision problems in image processing. In: *SIAM JOURNAL ON CONTROL AND OPTIMIZATION*. Chamonix, France: IARIA, 2012. p. 60–66. ISBN 978-1-61208-195-3. ISSN 2308-4448.

- [24] CORAZZA, M.; BERTOLUZZO, F. *Q-Learning-based financial trading systems with applications*. Venice, Italy, 2014. Disponível em: <<http://EconPapers.repec.org/RePEc:ven:wpaper:2014:15>>.
- [25] TSITSIKLIS, J. N. Asynchronous stochastic approximation and q-learning. *Machine Learning*, v. 16, n. 3, p. 185–202, 1994.
- [26] JAAKKOLA, T.; JORDAN, M. I.; SINGH, S. P. Convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, v. 6, p. 1185–1201, 1994.
- [27] WATKINS, C. J. C. H. *Learning from Delayed Rewards*. Tese (Doutorado) — King's College, Cambridge, UK, May 1989.
- [28] HAYKIN, S. *Neural Networks: A Comprehensive Foundation*. 2nd. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. ISBN 0132733501.
- [29] PRIDDY, K.; KELLER, P. *Artificial Neural Networks: An Introduction*. SPIE Press, 2005. (Tutorial Text Series). ISBN 9780819459879. Disponível em: <<http://books.google.com.br/books?id=BrnHR7esWmkC>>.
- [30] NGUYEN, H. et al. Ros commander: Flexible behavior creation for home robots. In: *International Conference on Robotics and Automation*. Karlsruhe, Germany: ICRA, 2013.
- [31] HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.

APÊNDICES

I. DESCRIÇÃO DO CONTEÚDO DO CD

O CD segue a seguinte estrutura:

- leiam.pdf : Essa descrição do CD
- resumo.pdf : Arquivo com o resumo e palavras chaves em pdf.
- \Relatório : Pasta com relatório final em pdf.
- \Código
 - \pacman_game : Pasta com código em Python da plataforma de Pac-Man utilizada para testes.
 - \bayesian_q_learning : Pasta com código desenvolvido nesse trabalho para implementar o filtro Bayesiano e a aprendizagem por reforço.
 - \pacman_msgs : Pasta com as mensagens trocadas, utilizando ROS, especificando seus formatos.
 - \pacman_abstract_classes : Pasta com código em C++, contendo classes abstratas que representam o agente (Pac-Man) e os fantasmas, além de funções globalmente utilizadas.
 - \q_learning_pacman : Pasta com código em C++ contendo uma versão mais primitiva do algoritmo de aprendizagem para casos determinísticos.